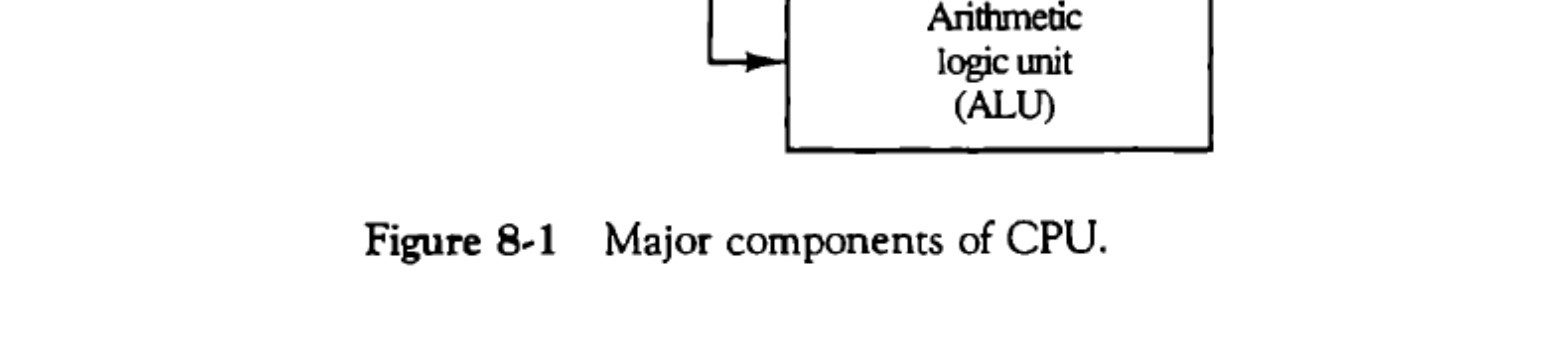
Unit 5 Central Processing Unit

# Introduction

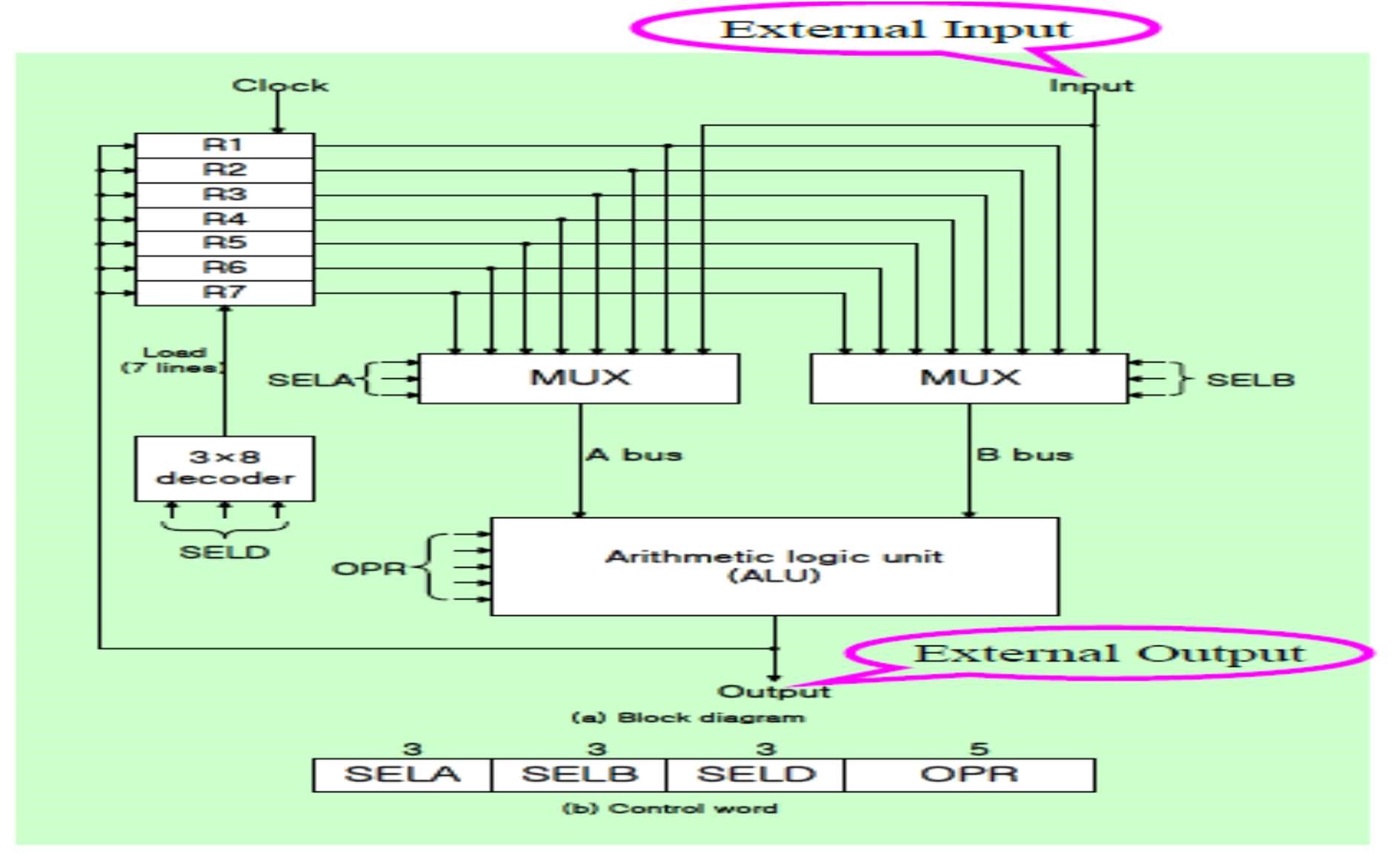
* Part of computer that performs bulk data processing
* Three major parts of CPU :
* 1) Register Set :- store intermediate Data used during the instruction execution
* 2) ALU:- performs required microoperations to execute an instructions
* 3) Control: - supervise transfer of data among registers and instruct ALU which operation to be performed
* Describe the organization and architecture of the CPU with an emphasis on the user’s view of the computer
* The design of CPU is task that involves choosing hardware for implementing machine instructions
* User who programs the computer in machine/assembly language must be aware of
* 1) Instruction Formats
* 2) Addressing Modes
* 3) Register Set
* The last section presents the concept of Reduced Instruction Set Computer(RISC)



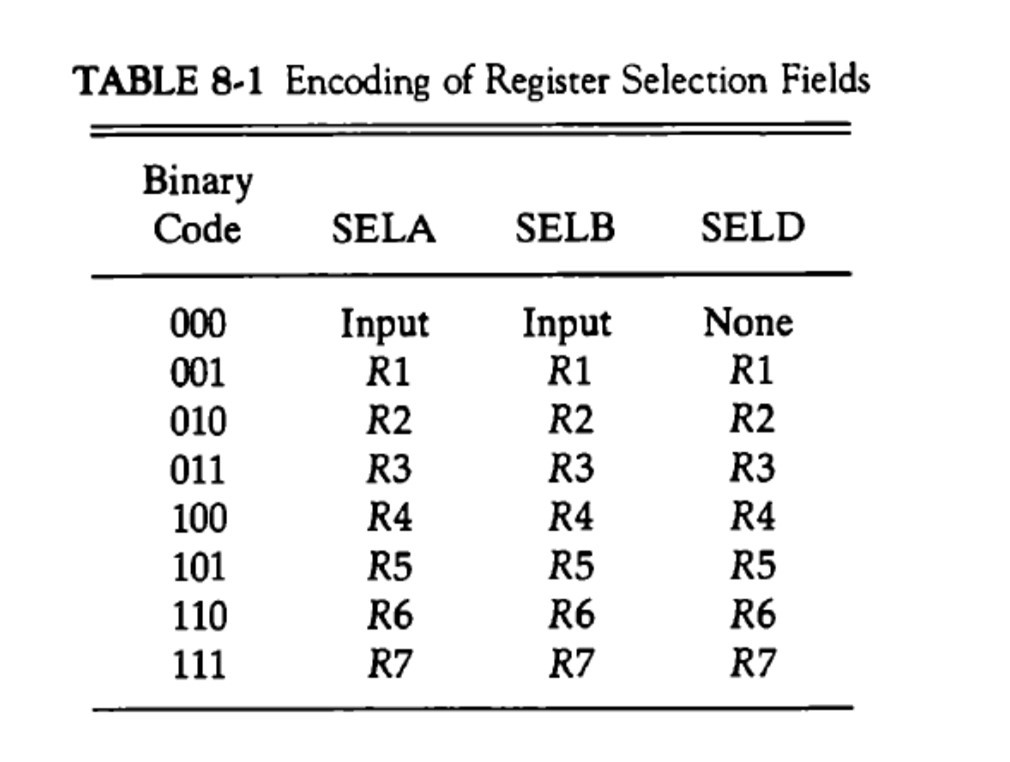
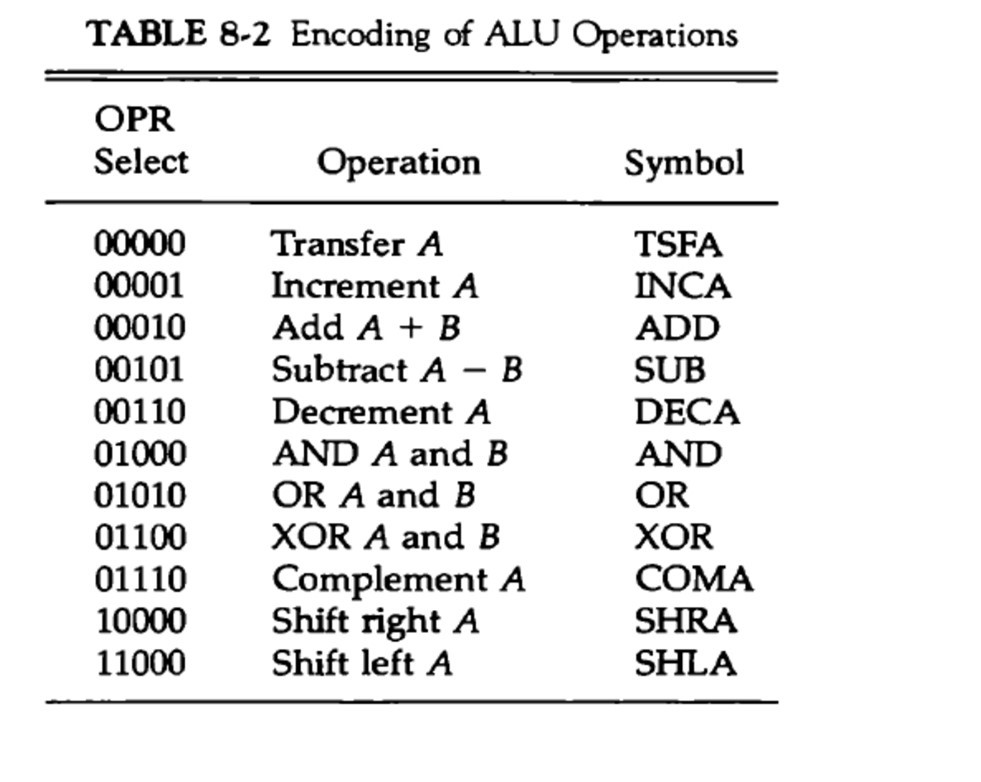
## General Register Organization

* Memory locations are needed to store pointers, return address, temporary results etc
* As memory access is most time consuming operation in the computer So we can store intermediate result in process registers
* It is efficient to connect large number of registers with common bus
* Its is necessary to provide common unit that performs data transfer, arithmetic, logical and shift microoperations
* Major components of CPU
* Storage components(registers, flipflops), execution components(ALU), transfer component (Bus), control component(control unit)
* Bus organization for 7 CPU registers :
* 2 MUX: select one of 7 register or external data input by SELA and SELB
* BUS A and BUS B: form the inputs to a common ALU
* ALU: OPR determine the arithmetic or logic microoperation
* The result of the microoperation is available for external data output and also goes into the inputs of all the registers
* 3 X 8 Decoder: select the register (by SELD) that receives the information from

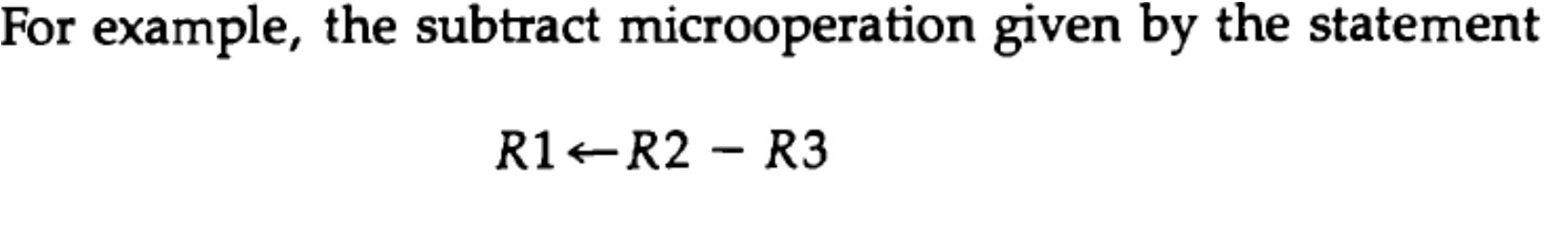
ALU

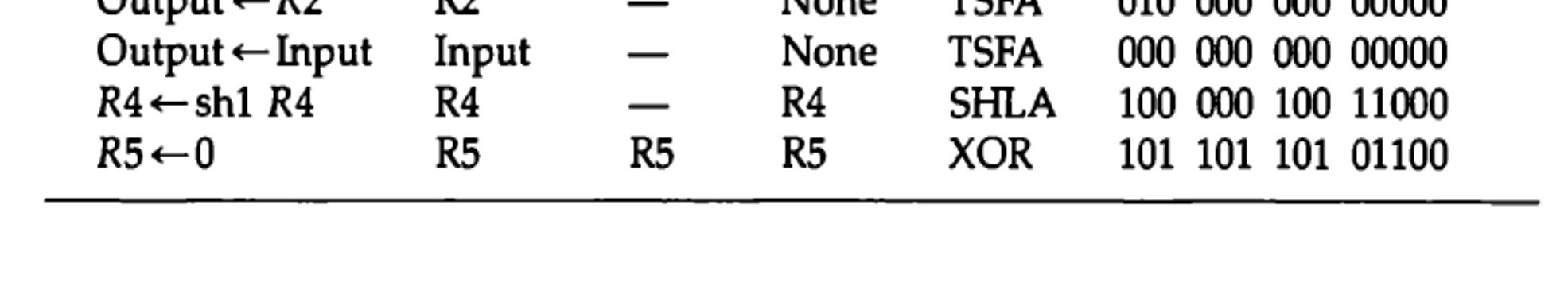
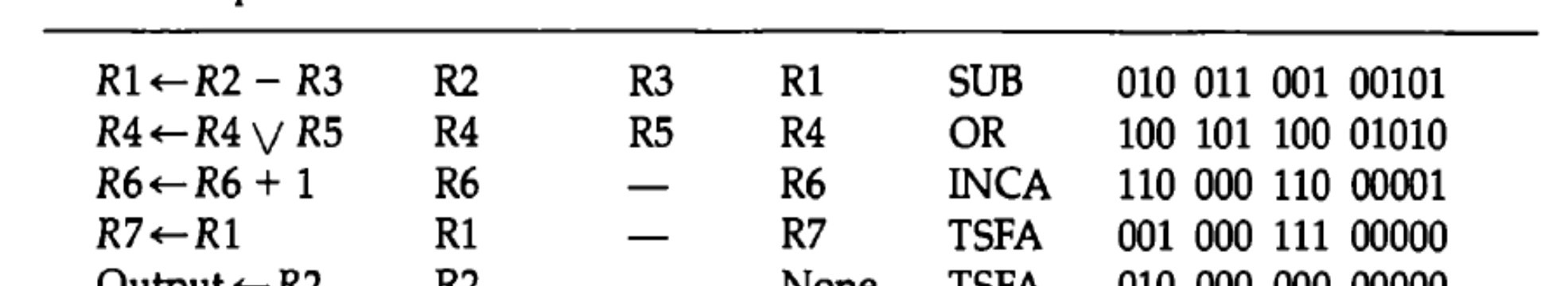
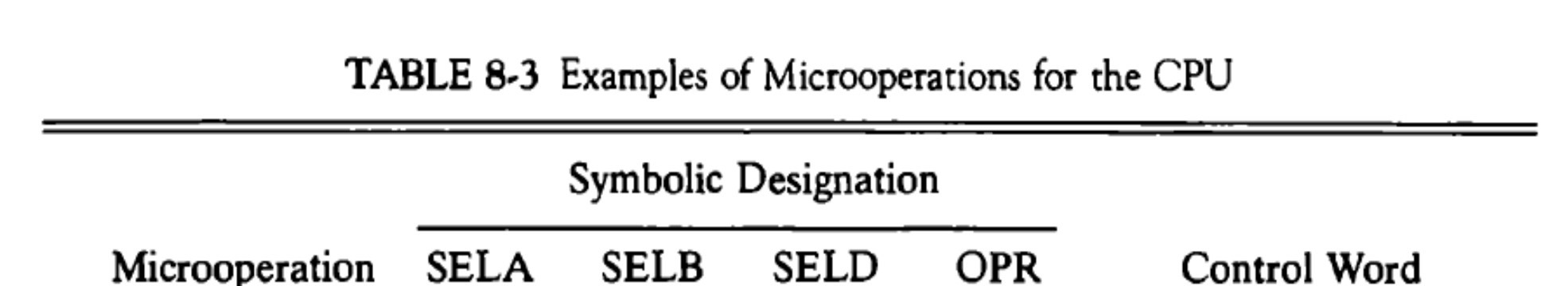
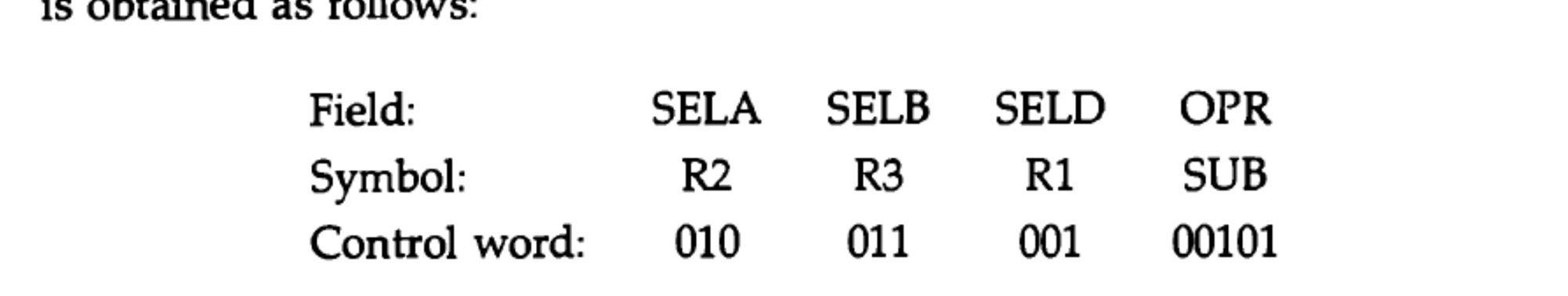
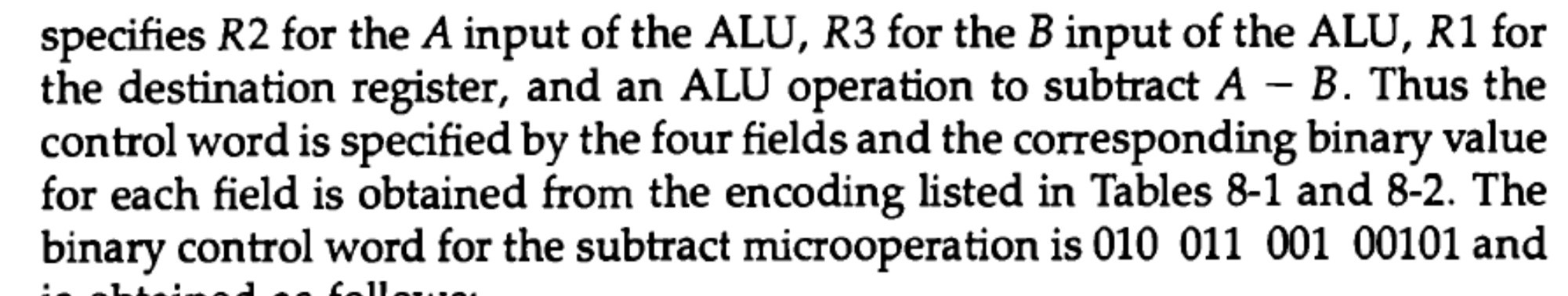


* Binary selector input : R1 R2+R3
* 1) MUXAselector (SELA) : to place the content of R2 into BUS
* 2) MUX B selector (SELB) : to place the content of R3 into BUS
* 3)ALU operation selector (OPR) : to provide the arithmetic addition R2 + R3
* 4) Decoder selector (SELD) : to transfer the content of the output bus into R1
* Control Word
* 14 bit control word (4 fields) (14 selection lines)
* SELA(3 bits) : select a source register for the Ainput of theALU
* SELB(3 bits) : select a source register for the B input of theALU
* SELD(3 bits) : select a destination register using the 3 X 8 decoder
* OPR(5 bits) : select one of the operations in theALU



* Encoding of Register Selection Fields :
* SELAor SELB= 000 (Input): MUX selects the external input data
* SELD= 000 (None): no destination register is selected but the contents of the output bus are available in the external output





# Stack Organization

* Stack or LIFO(Last-In, First-Out)
* A storage structure that stores information
* The item stored last is the first item retrieved = a stack of tray
* 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 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.
* Stack Pointer (SP)»The register that holds the address for the stack
* SP always points at the top item in the stack
* Two Operations of a stack : Insertion and Deletion of Items
* PUSH : Push-Down = Insertion
* POP : Pop-Up = Deletion
* Stack
* 1) Register Stack (Stack Depth):- a finite number of memory words or register(stand alone) (build using register)
* 2) Memory Stack (Stack Depth):- logical memory part of memory allocated as stack

# Register Stack

* A stack can be placed in a portion of a large memory or it can be organized as a collection of a finite number of memory words or 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.
* Full and empty register
* Data register to transfer data to and FRO from 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.

# Register Stack



* 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. Note that item C has been read out but not physically removed.
* This does not matter because when the stack is pushed, a new item is written in its place.
* In a 64-word stack, the stack pointer contains 6 bits because 26 = 64.
* Since SP has only six bits, it cannot exceed a number greater than 63 (111111 in binary). When 63 is incremented by 1, the result is 0 since 111111 + 1 = 1000000 in binary, but SP can accommodate only the six least significant bits.

•

* Similarly, when 000000 is decremented by 1, the result is 111111.
* The one-bit register FULL is set to 1 when the stack is full, and 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.
* Initially, SP is cleared to 0, EMTY is set to 1, and FULL is cleared to 0, so that SP points to the word at address 0 and the stack is marked empty and not full. If the stack is not full (if FULL = 0), a new item is inserted with a push operation.

# Push Operation

* The stack pointer is incremented so that it points to the address of the next-higher word.
* A memory write operation inserts the word from DR into the top of the stack. Note that SP holds the address of the top of the stack and that M[SP] denotes the memory word specified by the address presently available in SP.
* The first item stored in the stack is at address L The last item is stored at address

0.

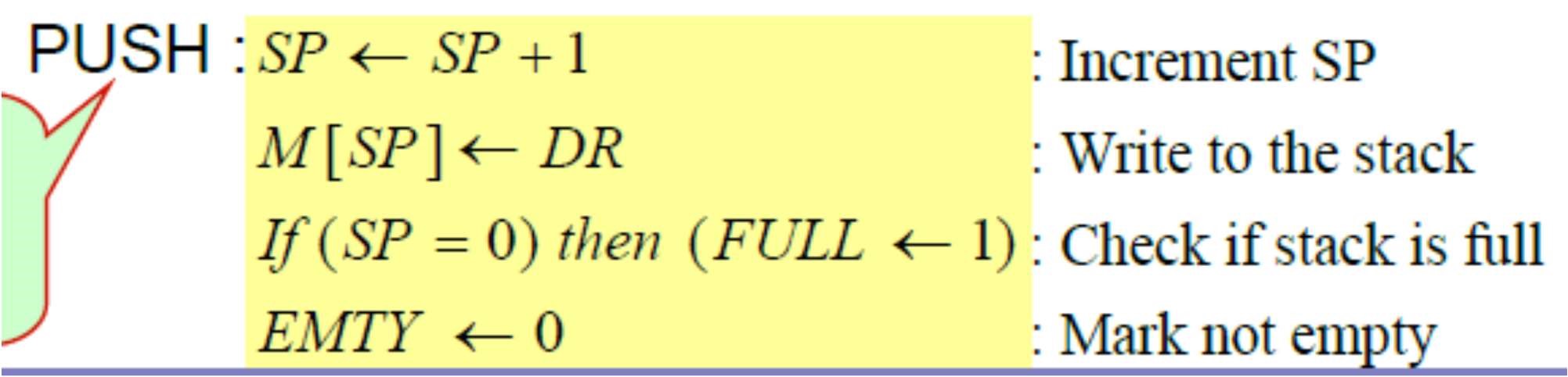
* If SP reaches 0, the stack is full of items, so FULL is set to L This condition is reached if the top item prior to the last push was in location 63 and, after incrementing SP, the last item is stored in location 0.
* Once an item is stored in location 0, there are no more empty registers in the stack. If an item is written in the stack, obviously the stack cannot be empty, so EMTY is cleared to 0.

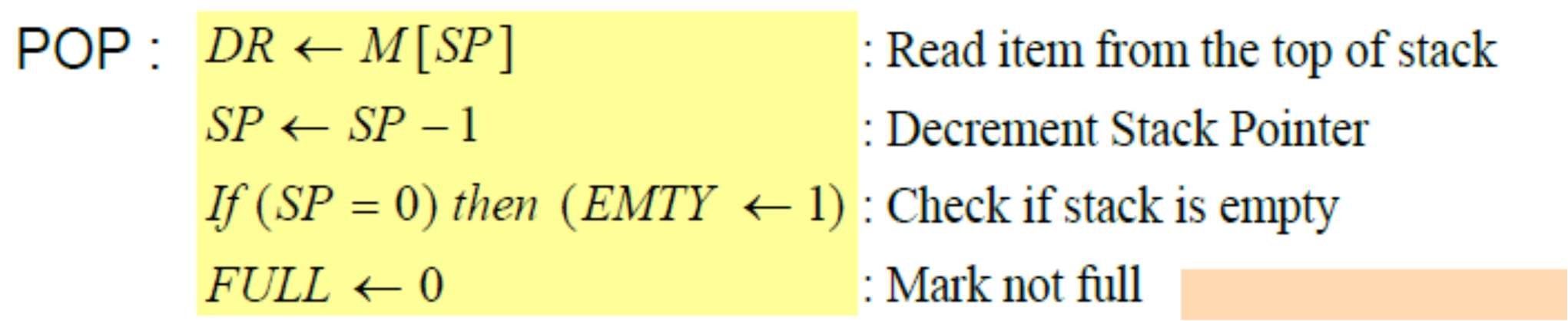
# POP operation

* 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.

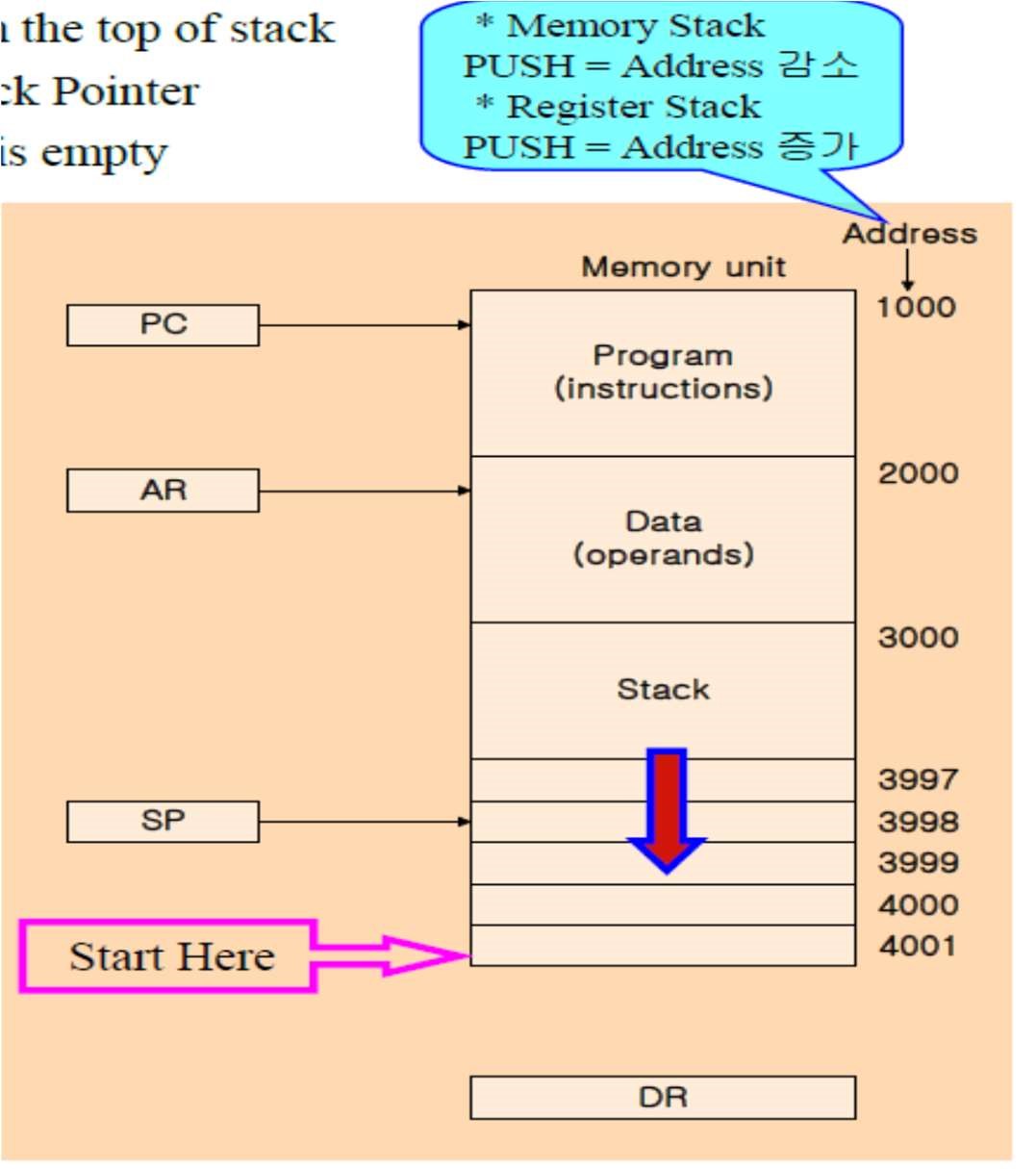
* This condition is reached if the item read was in location 1.
* Once this item is read out, SP is decremented and reaches the value 0, which is the initial value of SP.
* Note that if a pop operation reads the item from location 0 and then SP is decremented, SP changes to 111111, which is equivalent 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

* A stack can exist as a stand-alone unit or can be implemented in a randomaccess memory attached to a CPU.
* 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 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.
* The address register AR points at an array of data.



* The stack pointer SP points at the top of the stack. The three registers are connected to a common address bus, and either one can provide an address for memory.
* PC is used during the fetch phase to read an instruction.
* AR is used during the execute phase to read an operand.
* SP is used to push or pop items into or the stack. As shown in Fig, the initial value of SP is 4001 and the stack grows with decreasing addresses.
* Thus the first item stored in the stack is at address 4000, the second item is stored at address 3999, and the last address that can be used for the stack Is 3000.
* No provisions are available for stack limit checks.
* We assume that the items in the stack communicate with a data register DR . A new item is inserted with the push operation as follows:

SP ← SP - 1

M[SP] ← DR

* The stack pointer is decremented so that it points at the address of the next word. A memory write operation inserts the word from DR into the top of the stack. A new item is deleted with a pop operation as follows:

DR ← M[SP]

SP ← SP + 1

* The top item is read from the stack into DR. The stack pointer is then incremented to point at the next item in the stack.
* Most computers do not provide hardware to check for stack overflow (full stack) or underflow (empty stack).
* The stack limits can be checked by using two processor registers: one to hold the upper limit (3000 in this case), and the other to hold the lower limit (4001 in this case).
* After a push operation, SP is compared with the upper-limit register and after a pop operation, SP is compared with the lower-limit register.
* The two microoperations needed for either the push or pop are (1) an access to memory through SP, and (2) updating SP. 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 Fig. the stack grows by decreasing the memory address.
* The stack may be constructed to grow by increasing the memory address.
* In such a case, SP is incremented for the push operation and decremented for the pop operation. A stack may be constructed so that SP points at the next empty location above the top of the stack.
* In this case the sequence of microoperations must be interchanged. A stack pointer is loaded with an initial value. This initial value must be the bottom address of an assigned stack in memory. Henceforth, SP is automatically decremented or incremented with every push or pop operation.
* 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.

# Instruction Formats

* A computer will usually have a variety of instruction code formats.
* It is the function of the control unit within the CPU to interpret each instruction code and provide the necessary control functions needed to process the instruction.
* The format of an instruction is usually depicted in a rectangular box symbolizing the bits of the instruction as they appear in memory words or in a control register.
* The bits of the instruction are divided into groups called fields. The most common fields found in instruction formats are:

•1. An operation code field that specifies the operation to be performed.

•2. An address field that designates a memory address or a processor register.

•3. A mode field that specifies the way the operand or the effective address is determined

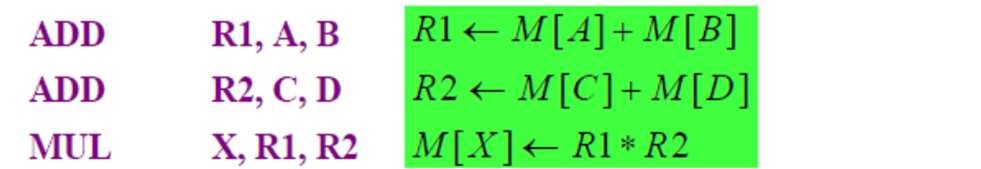
* Other special fields are sometimes employed under certain circumstances, as for example a field that gives the number of shifts in a shift-type instruction.
* The operation code field of an instruction is a group of bits that define various processor operations, such as add, subtract, complement, and shift.
* The bits that define the mode field of an instruction code specify a variety of alternatives for choosing the operands from the given address.
* Operations specified by computer instructions are executed on some data stored in memory or processor registers.
* Operands residing in memory are specified by their memory address.
* Operands residing in processor registers are specified with a register address.
* A register address is a binary number of k bits that defines one of '2k' registers in the CPU. Thus a CPU with 16 processor registers R0 through R15 will have a register address field of four bits.
* The binary number 0101, for example, will designate register RS.
* Computers may have instructions of several different lengths containing varying number of addresses.
* The number of address fields in the instruction format of a computer depends on the internal organization of its registers. Most computers fall into one of three types of CPU organizations:
* 1. Single accumulator organization.
* 2. General register organization.
* 3. Stack organization.
* In an accumulator-type organization all operations are performed with an implied accumulator register. The instruction format in this type of computer uses one address field.
* For example, the instruction that specifies an arithmetic addition is defined by an assembly language instruction as ADD X where X is the address of the operand. The ADD instruction in this case results in the operation AC ←

AC + M [X].

* AC is the accumulator register and M [X] symbolizes the memory word located at address X.
* In a general register type of organization the instruction format in this type of computer needs three register address fields.
* Thus the instruction for an arithmetic addition may be written in an assembly language as ADD R1 , R2 , R3 to denote the operation R1 ← R2 + R3.
* The number of address fields in the instruction can be reduced from three to two if the destination register is the same as one of the source registers.
* Thus the instruction ADD R1 , R2 would denote the operation R1 ← R1 + R2. Only register addresses for R1 and R2 need be specified in this instruction.
* Computers with multiple processor registers use the move instruction with a mnemonic MOV to symbolize a transfer instruction. Thus the instruction MOV R1 , R2 denotes the transfer R1 ← R2 (or R2 ← R1, depending on the particular computer).
* Thus transfer-type instructions need two address fields to specify the source and the destination.
* General register-type computers employ two or three address fields in

# Three-Address Instructions

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

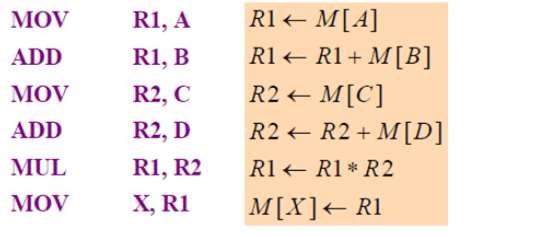
•The program in assembly language that evaluates X = (A + B) \* (C + D) is shown below, together with comments that explain the register transfer operation of each instruction. 

•It is assumed that the computer has two processor registers, R1 and R2. The symbol M[A] denotes the operand at memory address symbolized by A.

•The advantage of the three-address format is that it results in short programs when evaluating arithmetic expressions.

•The disadvantage is that the binary-coded instructions require too many bits to specify three addresses.

## Two-Address Instructions

•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) is as follows: 

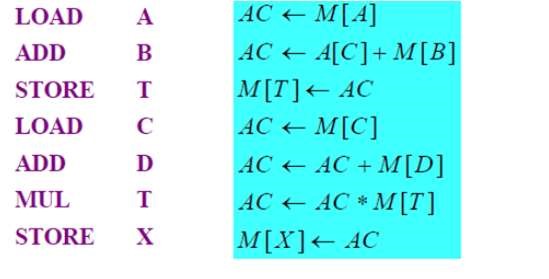
•The MOV instruction moves or transfers the operands to and from memory and processor registers.

•The first symbol listed in an instruction is assumed to be both a source and the destination where the result of the operation is transferred.

# One-Address Instructions

•One-address instructions use an implied accumulator (AC) register for all data manipulation. For multiplication and division there is a need for a second register. However, here we will neglect the second register and assume that the AC contains the result of all operations.

•The program to evaluate X = (A + B) \* (C + D) is

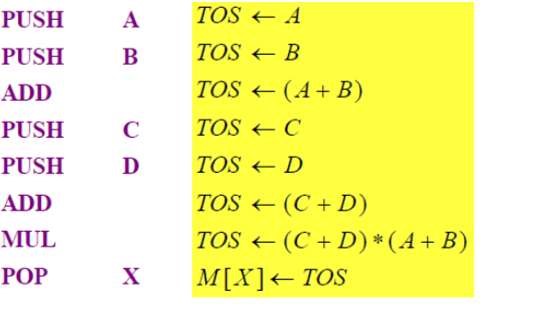


•All operations are done between the AC register and a memory operand. T is the address of a temporary memory location required for storing the intermediate result.

# Zero-Address Instructions

•A stack-organized computer does not use an address field for the instructions ADD and MUL. The PUSH and POP instructions, however, need an address field to specify the operand that communicates with the stack.

•The following program shows how X = (A + B) \* (C + D) will be written for a stack organized computer. (TOS stands for top of stack.)



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

•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

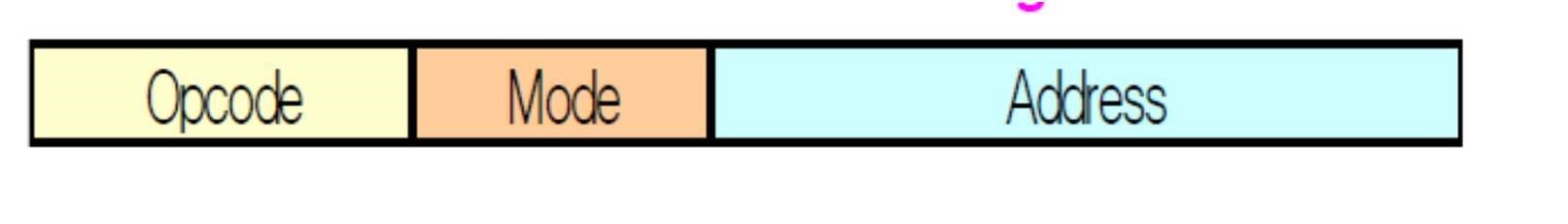
* The operation field of an instruction specifies the operation to be performed.
* This operation must be executed on some data stored in computer registers or memory words.
* The way the operands are chosen during program execution is dependent on the addressing mode of the instruction. The addressing 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:
* 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.
* To reduce the number of bits in the addressing field of the instruction.
* The availability of the addressing modes gives the experienced assembly language programmer flexibility for writing programs that are more efficient with respect to the number of instructions and execution time.
* To understand the various addressing modes to be presented in this section, it is imperative that we understand the basic operation cycle of the computer.

The control unit of a computer is designed to go through an instruction cycle that is divided into three major phases:

•1. Fetch the instruction from memory.

•2. Decode the instruction.

* 3. Execute the instruction.
* There is one register in the computer called the program counter or PC that keeps track of the instructions in the program stored in memory.
* PC holds the address of the instruction to be executed next and is incremented each time an instruction is fetched from memory.
* The decoding done in step 2 determines the operation to be performed, the addressing mode of the instruction, and the location of the operands.
* The computer then executes the instruction and returns to step 1 to fetch the next instruction in sequence.
* In some computers the addressing mode of the instruction is specified with a distinct binary code, just like the operation code is specified.
* Other computers use a single binary code that designates both the operation and the mode of the instruction.
* Instructions may be defined with a variety of addressing modes, and sometimes, two or more addressing modes are combined in one instruction.



## 1. Implied Mode

* In this mode the operands are specified implicitly in the definition of the instruction.
* For example, the instruction "complement accumulator" is an impliedmode instruction because the operand in the accumulator register is implied in the definition of the instruction.
* In fact, 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.
* Example :- CMA

## 2. Immediate Mode

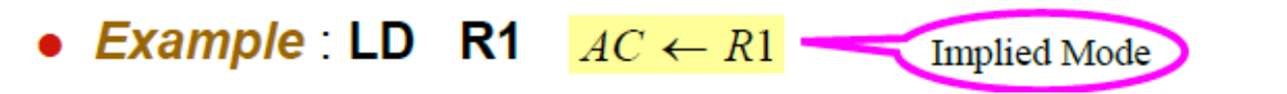
* In this mode the operand is specified in the instruction itself.
* In other words, an immediate-mode instruction has an operand field rather than an address field.
* The operand field contains the actual operand to be used in conjunction with the operation specified in the instruction.
* Immediate-mode instructions are useful for initializing registers to a constant value.
* It was mentioned previously that the address field of an instruction may specify either a memory word or a processor register.
* When the address field specifies a processor register, the instruction is said to be in the register mode.
* LDA 05H
* MOV R1,02H



## 3. Register Mode

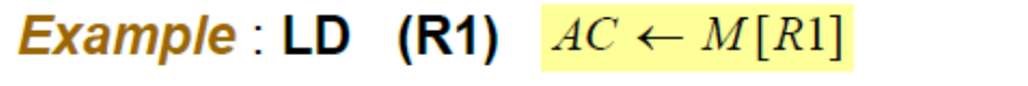
* In this mode the operands are in registers that reside within the CPU.
* The particular register is selected from a register field in the instruction.
* A k-bit field can specify any one of 2k registers.
* Ex: MOV R1,R2

•



## 4. Register Indirect Mode

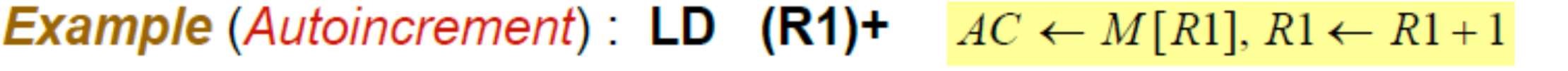
* In this mode the instruction specifies a register in the CPU whose contents give the address of the operand in memory.
* In other words, the selected register contains the address of the operand rather than the operand itself.
* Before using a register indirect mode instruction, the programmer must ensure that the memory address of the operand is placed in the processor register with a previous instruction.
* A reference to the register is then equivalent to specifying a memory address.
* The advantage of a register indirect mode instruction is that the address field of the instruction uses fewer bits to select a register than would have been required to specify a memory address directly.



## 5. Autoincrement or Autodecrement Mode

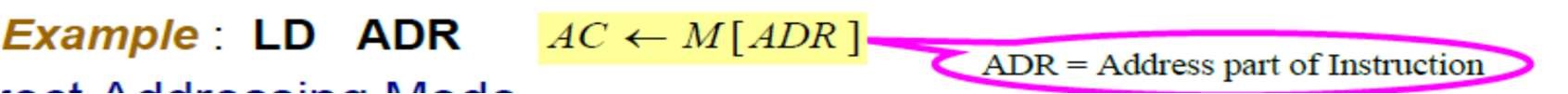
* This is similar to the register indirect mode except that the register is incremented or decremented after (or before) its value is used to access memory.
* When the address stored in the register refers to a table of data in memory, it is necessary to increment or decrement the register after every access to the table.
* This can be achieved by using the increment or decrement instruction.
* However, because it is such a common requirement, some computers incorporate a special mode that automatically increments or decrements the content of the register after data access. The address field of an instruction is used by the control unit in the CPU to obtain the operand from memory.
* Sometimes the value given in the address field is the address of the operand, but sometimes it is just an address from which the address of the operand is calculated.
* To differentiate among the various addressing modes it is necessary to distinguish between the address part of the instruction and the effective address used by the control when executing the instruction. The effective address is defined to be the memory address obtained from the computation dictated by the given addressing mode. The effective address is the address of the operand in a computational- type instruction.

It is the address where control branches in response to a branch-type instruction



### 6. Direct Address Mode

* In this 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.
* Example ADD 457 (operand resides at address 457)



## 7. Indirect Address Mode

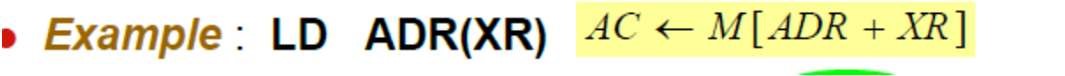
* In this mode the address field of the instruction gives the address where the effective address is stored in memory.
* Control fetches the instruction from memory and uses its address part to access memory again to read the effective address.
* A few addressing modes require that the address field of the instruction be added to the content of a specific register in the CPU.
* The effective address in these modes is obtained from the following computation: effective address = address part of instruction + content of CPU register
* The CPU register used in the computation may be the program counter, an index register, or a base register.
* In either case we have a different addressing mode which is used for a different application.



## 8. Relative Address Mode

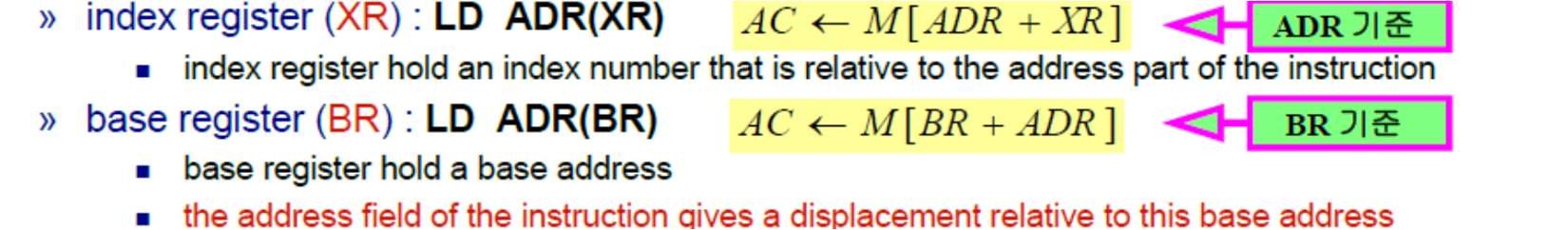
* In this mode the content of the program counter is added to the address part of the instruction in order to obtain the effective address.
* The address part of the instruction is usually a signed number (in 2' s complement representation) which can be either positive or negative. When this number is added to the content of the program counter, the result produces an effective address whose position in memory is relative to the address of the next instruction.
* To clarify with an example, assume that the program counter contains the number 825 and the address part of the instruction contains the number 24.
* Effective address = address part of instruction + content of PC
* 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.
* This is 24 memory locations forward from the address of the next instruction. Relative addressing is often used with branch-type instructions when the branch address is in the area surrounding the instruction word itself.
* It results in a shorter address field in the instruction format since the relative address can be specified with a smaller number of bits compared to the number of bits required to designate the entire memory address.

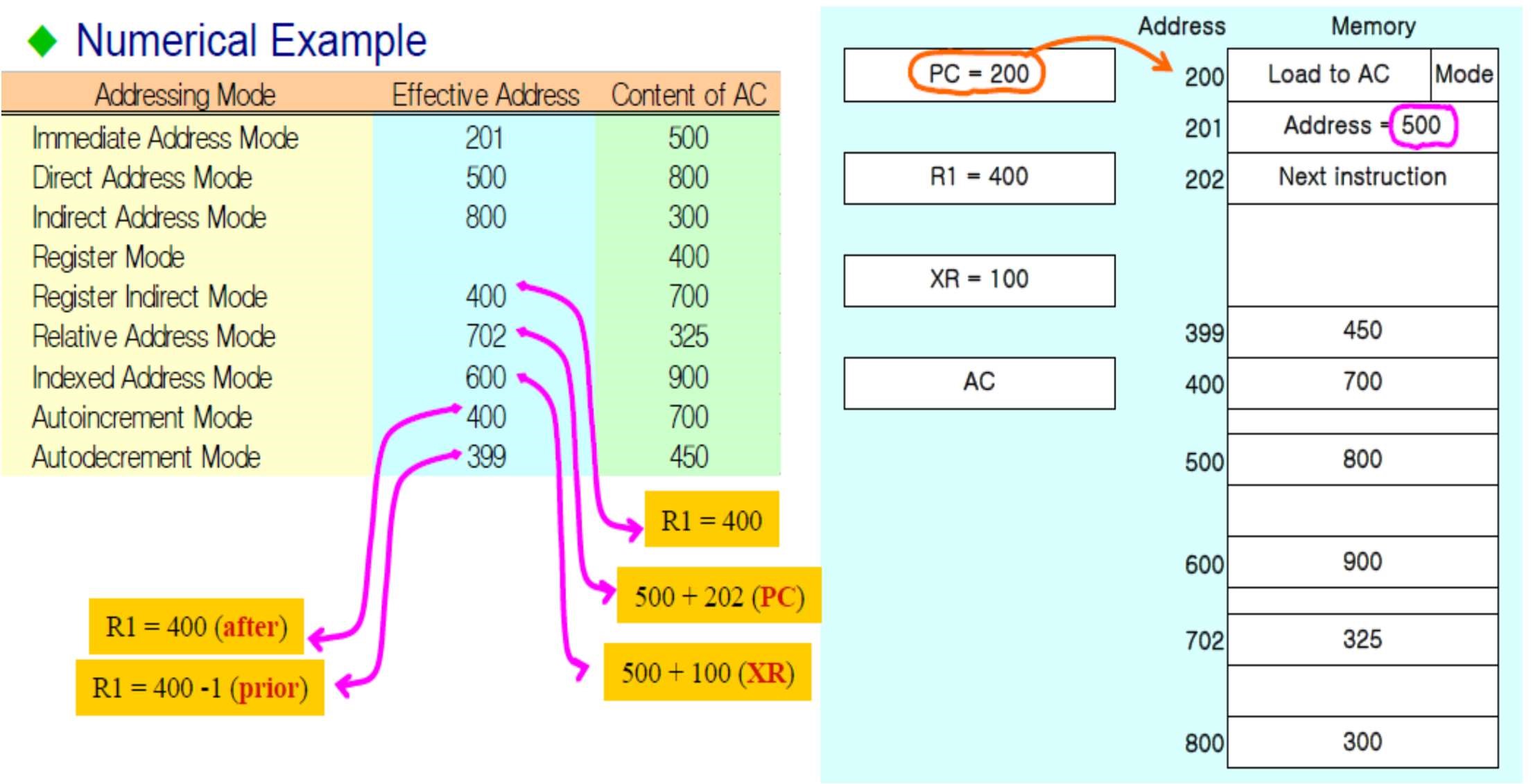
## 9. Indexed Addressing Mode

* In this mode the content of an index register is added to the address part of the instruction to obtain the effective address.
* 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.
* Each operand in the array is stored in memory relative to the beginning address. The distance between the beginning address and the address of the operand is the index value stored in the index register.
* Effective address = address part of instruction+ content of index register
* Any operand in the array can be accessed with the same instruction provided that the index register contains the correct index value.
* The index register can be incremented to facilitate access to consecutive operands. Note that if an index type instruction does not include an address field in its format, the instruction converts to the register indirect mode of operation.
* Some computers dedicate one CPU register to function solely as an index register.
* This register is involved implicitly when the index-mode instruction is used.
* In computers with many processor registers, any one of the CPU registers can contain the index number.
* In such a case the register must be specified explicitly in a register field within the instruction format.

## 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.
* This is similar to the indexed addressing mode except that the register is now called a base register instead of an index register.
* The difference between the two modes is in the way they are used rather than in the way that they are computed.
* An index register is assumed to hold an index number that is relative to the address part of the instruction.
* A base register is assumed to hold a base address and the address field of the instruction gives a displacement relative to this base address. The base register addressing mode is used in computers to facilitate the relocation of programs in memory.
* Effective address = address part of instruction+ content of base register(offset)
* 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.
* With a base register, the displacement values of instructions do not have to change. Only the value of the base register requires updating to reflect the beginning of a new memory segment.





### Data Transfer and Manipulation

* Most computer instructions can be classified into three categories:
* 1) Data transfer (transfer the data from one place to another)
* 2) Data manipulation (to perform arithmetic, logical and shift operation)
* 3) Program control instructions (for decision making capabilities and changing the path of the program)

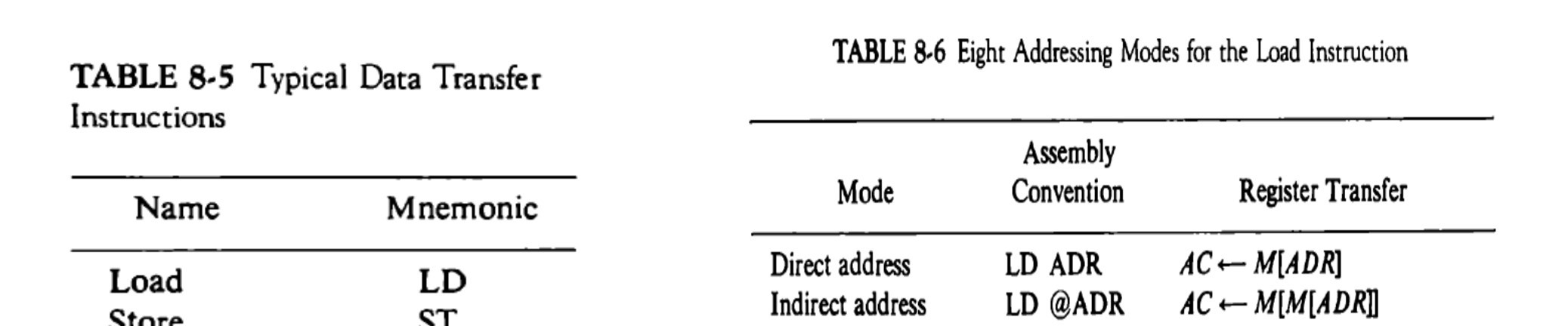
## Data Transfer Instruction

Data transfer instructions are to transfer data from memory to register, between registers or from input or output devices Typical Data Transfer Instruction :

* »Load: transfer from memory to a processor register, usually an AC (memory read)
* »Store: transfer from a processor register into memory (memory write)
* »Move: transfer from one register to another register
* »Exchange: swap information between two registers or a register and a memory word
* »Input/Output: transfer data among processor registers and input/output device
* »Push/Pop: transfer data between processor registers and a memory stack

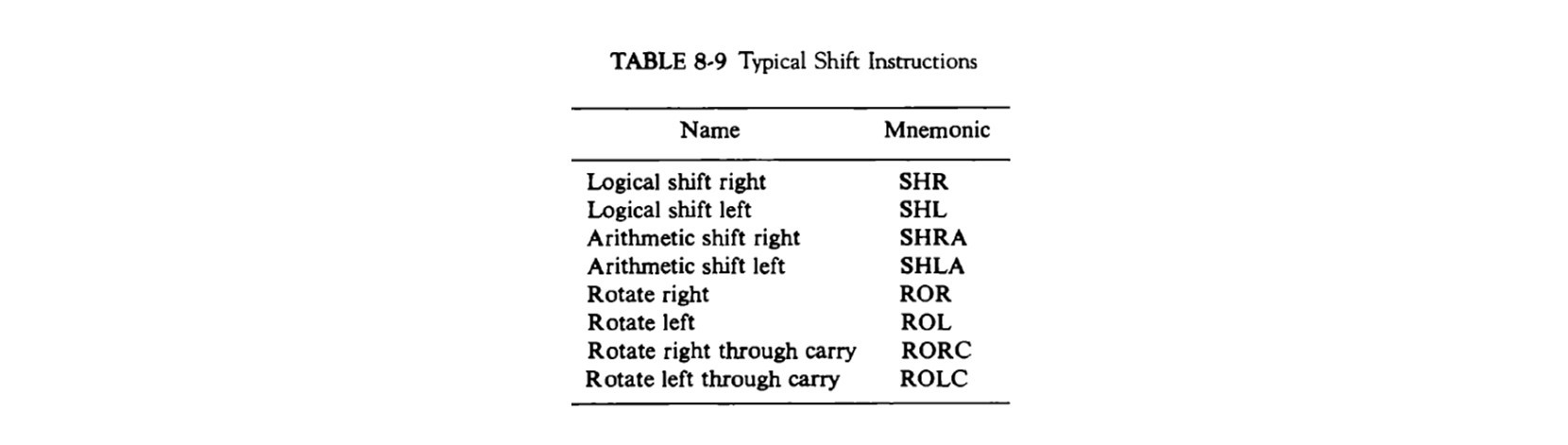
8 Addressing Mode for the LOAD Instruction :

* »@: Indirect Address
* »$: Address relative to PC
* »#: Immediate Mode
* »( ): Index Mode, Register Indirect, Autoincrement register

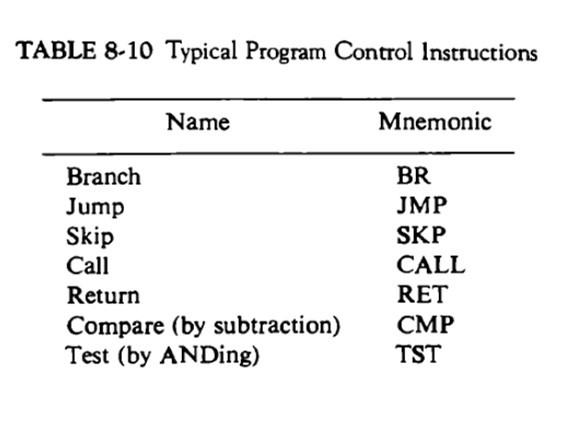


## Data Manipulation Instruction

* 1) Arithmetic
* 2) Logical and bit manipulation
* 3) Shift Instruction

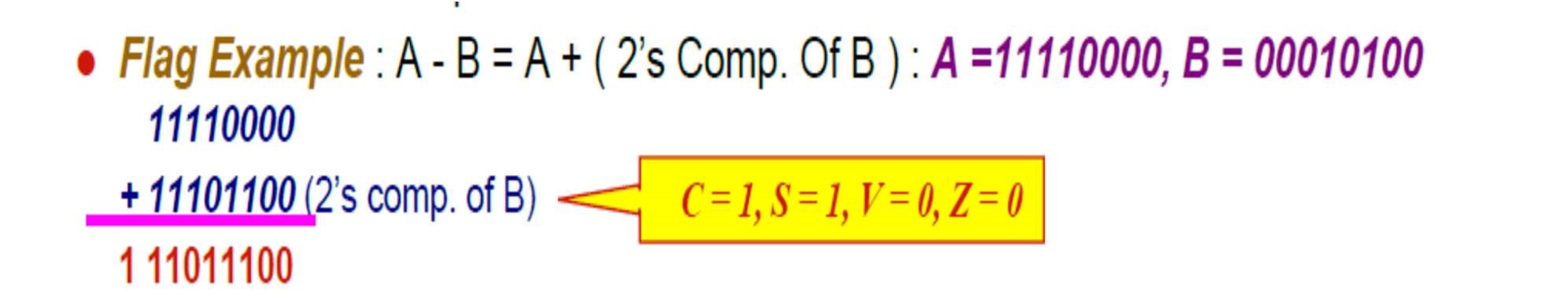


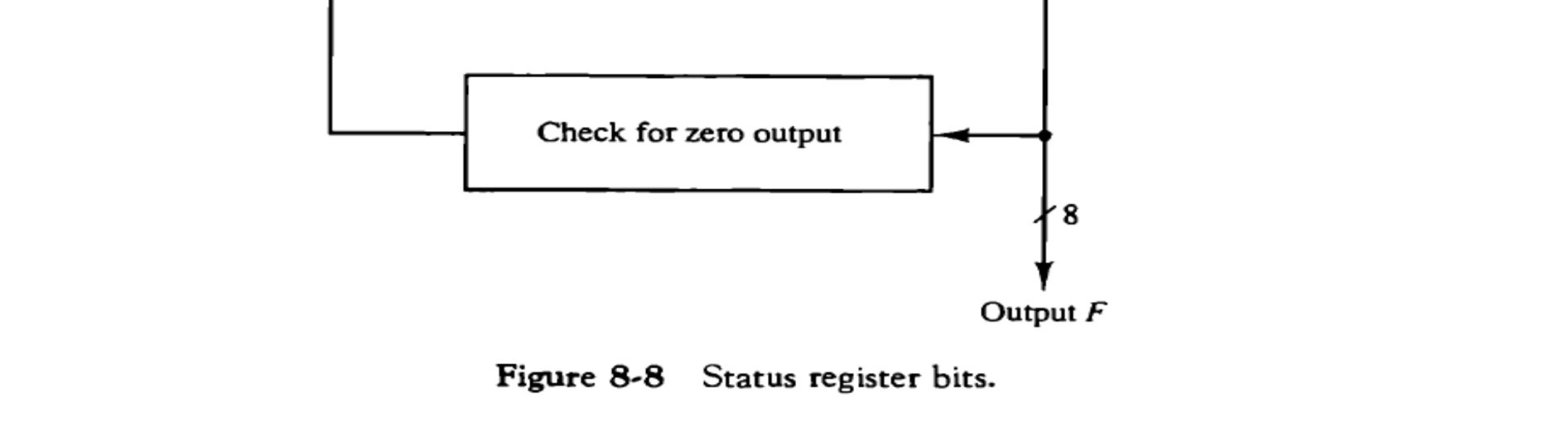
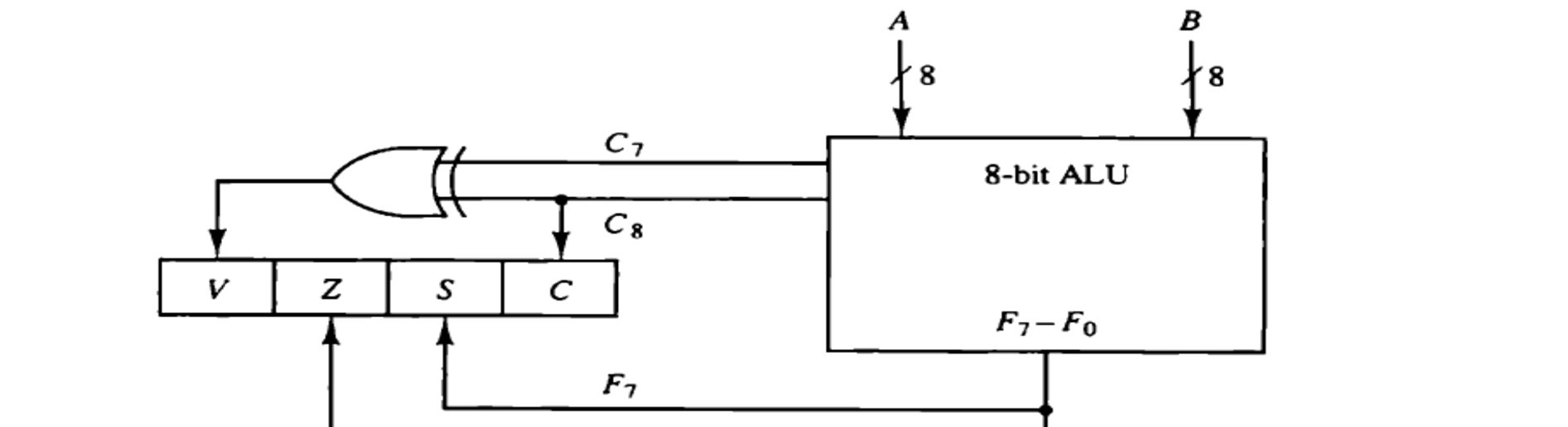
## Program Control Instructions

* A program control instruction is type of instruction when executed may change the value of program counter and alter the flow of execution
* Change in the value of program counter causes break in sequence of program execution
* These all are unconditional branching instruction 

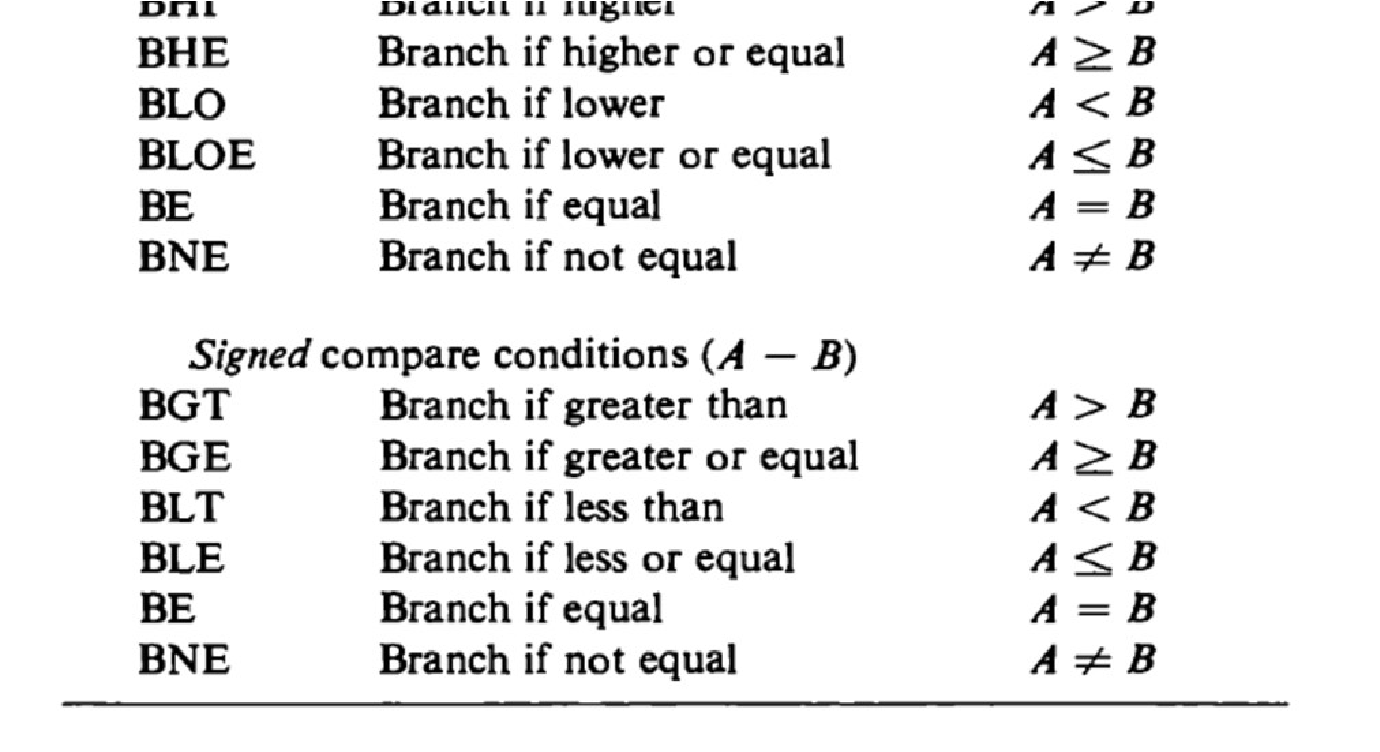
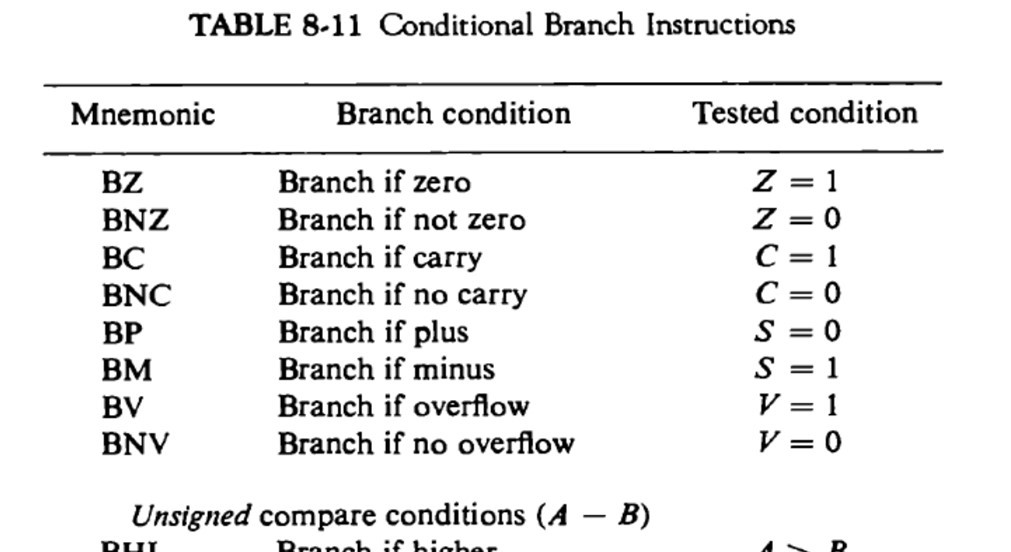
## Status bit conditions (flag registers)

* Condition Code Bit or Flag Bit»The bits are set or cleared as a result of an operation performed in the ALU
* 4-bit status register
* Bit C(carry) : set to 1 if the end carry C8 is 1, else 0
* S(sign) : set to 1 if F7 is 1 else 0 (for positive-0, negative-1)
* Bit Z(zero) : set to 1 if the output of the ALU contains all 0’s
* Bit V(overflow) : set to 1 if the exclusive-OR of the last two carries (C8and C7) is equal to 1



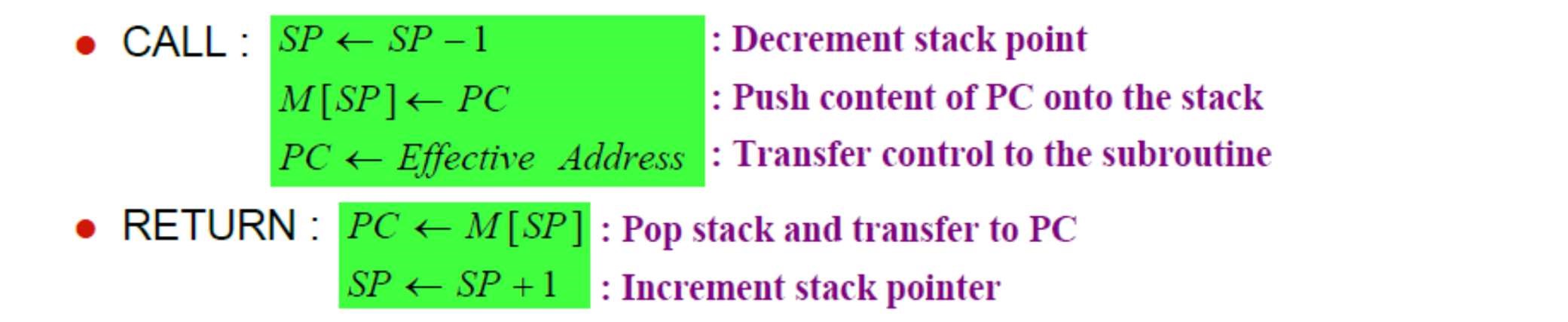


### Conditional Branch



* Some conditional branch instructions are based on status bits
* Some are based on comparisons of unsigned numbers
* Some are based on comparison of signed numbers
* So in HLL program when we use relational operators it will be converted in to these assembly level instructions

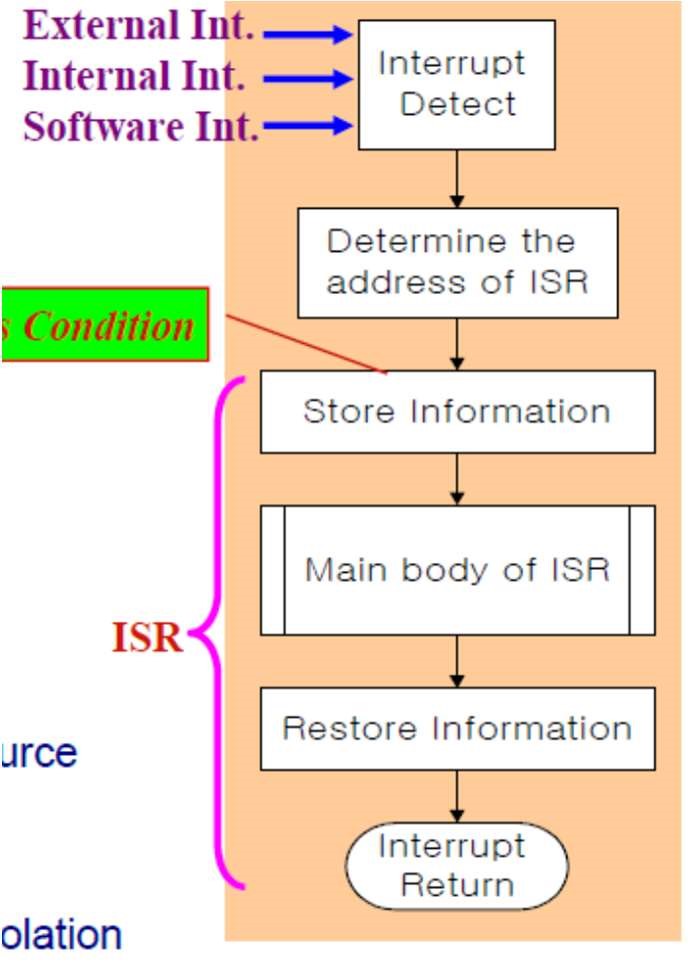
### Subroutine Call and Return



## Program Interrupt

* Program Interrupt
* Transfer 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
* It is quite similar to subroutine call except for three variations
* 1) An interrupt is initiated by an internal or external signal (except for software interrupt) where A subroutine call is initiated from the execution of an instruction (CALL)
* 2) The address of the interrupt service program is determined by the hardware where The address of the subroutine call is determined from the address field of an instruction
* 3) An interrupt procedure stores all the information necessary to define the state of the CPU where A subroutine call stores only the program counter (Return address)
* The collection of all status bit conditions in the CPU is called Program Status Word (PSW)
* PSW is stored in separate hardware registers and contains the status information that characterizes the state of CPU
* Two CPU Operating Modes
* Supervisor (System) Mode : Privileged Instruction
* When the CPU is executing a program that is part of the operating system
* User Mode : User program
* When the CPU is executing a user program





* Types of Interrupts
* 1) External Interrupts
* come from I/O device, from a timing device, from a circuit monitoring the power supply (control circuits), or from any other external source
* Examples of external interrupt:• I/O device request transfer of data
* I/O device finish transfer of data
* Elapsed time of an event (more time passed)
* Power failure
* These are asynchronous, not synchronous with CPU operations
* Depends on external conditions that are independent of program being executed at time
* 2) Internal Interrupts or TRAP
* Illegal or erroneous use of an instruction or data
* caused by register overflow, attempt to divide by zero, an invalid operation code, stack overflow, and protection violation
* Mainly occurs due to premature termination of instruction execution(before time execution)
* Synchronous (generated within CPU)
* 3) Software Interrupts
* Special call instruction that behaves like an interrupt rather than subroutine call
* initiated by executing an instruction (INT or RST)
* Used to switch from CPU user mode to CPU supervisor mode
* When input or output transfer is required , the supervisor mode is requested by means of supervisor call instruction. This instruction causes software interrupt that stores old CPU state and brings in new PSW that belongs to the supervisor mode
* used by the programmer to initiate an interrupt procedure at any desired point in the program

## RISC (Reduced Instruction Set Computer)

* Design instruction set for the processor
* Earlier computers had small instruction set as hardware became cheaper recent computers can have 100 or even 200 instruction in instruction set
* A computer with large number of instructions in their instruction set is called CISC(complex instruction set computer)
* Number of computer designers recommended that computer uses fewer instructions with simple construct so they can be executed much faster in CPU without having to refer memory much often
* These computers are called RISC (Reduced Instruction Set Computer)

### Complex Instruction Set Computer (CISC) Characteristics

* 1) A large number of instructions -typically from 100 to 250 instruction
* 2) Some instructions that perform specialized tasks and are used infrequently (for example:- MUL,SUB)
* 3) A large variety of addressing modes -typically from 5 to 20 different modes (for large number of instructions)
* 4) Variable-length instruction formats (as addressing modes are different we can have 2-byte, 3-byte instruction)
* 5) Instructions that manipulate operands in memory (which can directly change data in memory)

### Reduced Instruction Set Computer (RISC) Characteristics

* 1) Relatively few instructions
* 2) Relatively few addressing modes
* 3) Memory access limited to load and store instruction
* 4) All operations done within the registers of the CPU (no direct memory operations as in CISC)
* 5) Fixed-length, easily decoded instruction format
* 6) Single-cycle instruction execution (can be executed within single timing cycle so faster execution unlike CISC)
* 7) Hardwired rather than microprogrammed control
* Other characteristics of a RISC architecture
* 1) A relatively large number of registers in the processor unit (as all the operations are performed using registers, cant access memory directly)

2) Use of overlapped register windows to speed-up procedure call and return

* 3) Efficient instruction pipeline (makes execution faster)
* 4) Compiler support for efficient translation of high-level language programs into machine language programs Overlapped Register Windows
* Feature available in RISC type processor
* Time consuming operations during procedure call
* »Saving and restoring registers
* »Passing of parameters and results
* Overlapped Register Windows
* RISC processors uses overlap windows to provide the passing of parameters and avoid the need for saving and restoring register value byhardware
* Each procedure call results in the allocation of a new window consisting of a set of registers from register file for use by new procedure
* Windows for adjacent procedures have overlapping registers that are shared to provide the passing parameters and result

