Program Interrupt

- Program interrupt defines the transfer of program control from a currently running program to another service program as a result of an external or internal created request.

- Control returns to the initial program after the service program is implemented.

Types of program interrupts

External Interrupts

Internal Interrupts

Software Interrupts

External Interrupts

- External interrupts come from input-output (I/0) devices, from a timing device, from a circuit monitoring the power supply, or from any other external source.

- The timeout interrupt can result from a program that is in an endless loop and thus exceeded its time allocation.

Internal Interrupts

- Internal interrupts arise from illegal or erroneous use of an instruction or data. Internal interrupts are also called traps.

- These error conditions generally appear as a result of premature termination of the instruction execution.

- The service program that processes the internal interrupt determines the corrective measure to be taken.

Difference between internal and external interrupts

- The main difference between internal and external interrupts is that the internal interrupt is initiated by some exceptional condition caused by the program itself rather than by an external event.

- Internal interrupts are synchronous with the program while external interrupts are asynchronous. If the program is rerun, the internal interrupts will appear in the same place each time.

Difference between internal and external interrupts

- External interrupts depend on external conditions that are independent of the program being executed at the time.

Software Interrupts

- A software interrupt is initiated by executing an instruction.
- A 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.

Data Transfer and Manipulations

- Computers provide an extensive set of instructions to give the user the flexibility to carry out various computational tasks.

- Most computer instructions can be classified into three categories.

- Data transfer instructions
- Data manipulation instructions
- Program control instructions

Data Transfer Instructions

- Data transfer instructions transfer the data between memory and processor registers, processor registers and I/O devices, and from one processor register to another.

- There are eight commonly used data transfer instructions. Each instruction is represented by a mnemonic symbol.

Data Transfer Instructions

Data Transfer Instructions

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

Data Transfer Instructions

- **Load** – The load instruction is used to transfer data from the memory to a processor register, which is usually an accumulator.
- **Store** – The store instruction transfers data from processor registers to memory.
- **Move** – The move instruction transfers data from processor register to memory or memory to processor register or between processor registers itself.
- **Exchange** – The exchange instruction swaps information either between two registers or between a register and a memory word.

Data Transfer Instructions

Input – The input instruction transfers data between the processor register and the input terminal.

Output – The output instruction transfers data between the processor register and the output terminal.

Push and Pop – The push and pop instructions transfer data between a processor register and memory stack.

Data Transfer Instructions

- All these instructions are associated with a variety of addressing modes.
- Some assembly language instructions use different mnemonic symbols just to differentiate between the different addressing modes.

Data Manipulation Instructions

Data Manipulation Instructions

- Data manipulation instructions are processor-level commands that perform operations on data stored in memory or registers.
- They enable tasks like mathematical computations, logical operations, and bit-level manipulations.
- These instructions modify data to execute program requirements efficiently.

Types of **Data Manipulation Instructions**

- Arithmetic instructions
- Logical and bit manipulation instructions
- Shift instructions

Arithmetic instructions

- The four basic operations are addition, subtraction, multiplication, and division.

- Most computers provide instructions for all four operations.

Typical Arithmetic Instructions

Name	**Mnemonic**
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL

Typical Arithmetic Instructions

Divide	DIV
Add with carry	ADDC
Subtract with borrow	SUBB
Negate (2's complement)	NEG

Logical and bit manipulation instructions

- Logical instructions perform binary operations on strings of bits stored in registers.

- They are helpful for manipulating individual bits or a group of bits.

Logical and bit manipulation instructions

Name	**Mnemonic**
Clear	CLR
Compleme nt	COM
AND	AND
OR	OR
Exclusive - OR	XOR

Logical and bit manipulation instructions

Name	Mnemonic
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC
Enable interrupt	E1
Disable interrupt	D1

Shift Instructions

- Shifts are operations in which the bits of a word are moved to the left or right.

- Shift instructions may specify either logical shifts, arithmetic shifts, or rotate-type operations.

Typical Shift Instructions

Name	Mnemonic
Logical shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR

Typical Shift Instructions

Name	Mnemonic
Rotate left	ROL
Rotate right through carry	RORC
Rotate left through carry	ROLC

Program Control Instructions

- Instructions of the computer are always stored in consecutive memory locations. These instructions are fetched from successive memory locations for processing and executing.

- When an instruction is fetched from the memory, the program counter is incremented by 1 so that it points to the address of the next consecutive instruction in the memory

Program Control Instructions

- Once a data transfer and data manipulation instruction are executed, the program control along with the program counter, which holds the address of the next instruction to be fetched, is returned to the fetch cycle.

Program Control Instructions

- Data transfer and manipulation instructions specify the conditions for data processing operations, whereas the program control instructions specify the conditions that can alter the content of the program counter.

- The change in the content of the program counter can cause an interrupt/break in the instruction execution. However, the program control instructions control the flow of program execution and are capable of branching to different program segments.

Typical Program Control Instructions

Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	Call
Return	RET

Typical Program Control Instructions

Name	Mnemonic
Compare (by Subtraction)	CMP
Test (by ANDing)	TST

Program Control Instructions

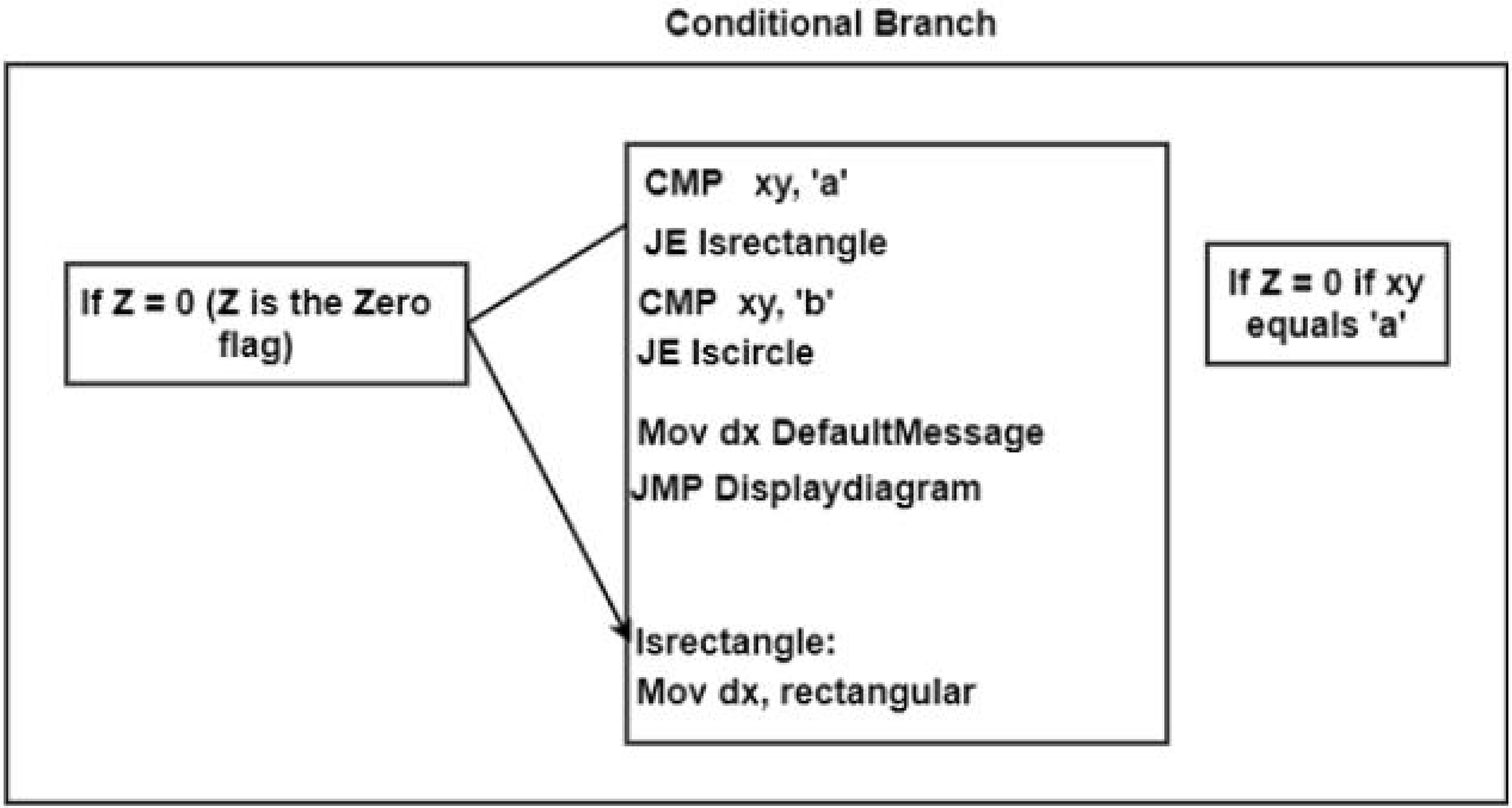
- The branch is a one-address instruction. It is represented as BR ADR, where ADR is a mnemonic for an address.
- The branch instruction transfers the value of ADR into the program counter.
- The branch and jump instructions are interchangeably used to mean the same. However, sometimes they denote different addressing modes.

Program Control Instructions

- The conditional branch instructions such as ‘branch if positive’, or ‘branch if zero’ specifies the condition to transfer the flow of execution.

- When the condition is met, the branch address is loaded in the program counter.

Conditional branch



Conditional branch

- The compare instruction performs an arithmetic subtraction. Here, the result of the operation is not saved; instead, the status bit conditions are set. The test instruction performs the logical AND operation on two operands and updates the status bits.

Conditions of Status Bits

Conditional Branch Instruction

Subroutines

Subroutines are programs that are used by other routines to accomplish a particular task.

A subroutine can be called from any point within the main body of the micro-program. Frequently, many micro-programs contain identical sections of code.

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

Subroutines

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

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

Subroutines

- Micro-programs that use subroutines 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 output from the control address register into a subroutine register and branching to the beginning of the subroutine.

Subroutines

- The subroutine register can then become the source for transferring the address for the return to the main routine.

- The best way to structure a register file that stores addresses for subroutines is to organize the registers in a last-in, first-out (LIFO) stack.

Subroutines

- The instruction that transfers program control to a subroutine is known by different names. The most common names used are called subroutine, jump to the subroutine, branch to the subroutine, or branch and save the address.

- A call subroutine instruction consists of an operation code together with an address that specifies the beginning of the subroutine.

Subroutines

- The instruction is executed by performing two operations are as follows −

- 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.

Subroutines

- The control is transferred to the beginning of the subroutine. The last instruction of every subroutine commonly called return from a subroutine transfers the return address from the temporary location into the program counter. This results in a transfer of program control to the instruction whose address was originally stored in the temporary location.

Subroutines

A subroutine call is implemented with the following micro-operations –

$SP \leftarrow SP - 1$	It can decrement the stack pointer.
$M[SP] \leftarrow PC$	It is used to push the content of the PC onto the stack.
$PC \leftarrow \text{Effective Address}$	It can transfer control to the subroutine.

Subroutines

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 –

$PC \leftarrow M[SP]$	It is used to pop the stack and transfer it to PC.
$SP \leftarrow SP + 1$	It can increment the stack pointer.

Subroutines

By using a subroutine stack, all return addresses are automatically stored by the hardware in one unit. The programmer does not have to be concerned or remember where the return address was stored.

Recursive subroutine

- A recursive subroutine is a subroutine that calls itself. If only one register or memory location can save the return address, and the recursive subroutine calls itself, it ends the previous return address.

- This is undesirable because vital information is destroyed. This problem can be solved if different storage locations are employed for each use of the subroutine while another lighter-level use is still active.

Recursive subroutine

When a stack is used, each return address can be pushed into the stack without destroying any previous values.

This solves the problem of recursive subroutines because the next subroutine to exit is always the last subroutine that was called.

Thank You