Sairam INSTITUTIONS

Sri SAI RAM ENGINEERING COLLEGE
INSTITUTE OF TECHNOLOGY
West Tambaram, Chennai - 44

YEAR	SEM
II	IV

CS8392

OPERATING SYSTEMS
(Common to CSE, IT)

UNIT No. 1

1.2 Instruction Execution, interrupts

Version: 1.XX

Instruction Execution, interrupts

Instruction execution cycle and pipelining

It is to divide a typical instruction execution cycle into the following five phases:

Instruction Fetch: The instruction is fetched from memory into an instruction register.

Decode: The opcode is decoded and the input operand locations are determined.

Data Fetch: The operands are fetched from memory if necessary.

Execute: The operation is executed.

Write-back: The operation output results are stored in the appropriate locations.

The instruction or the operands may be in a cache memory instead of the primary memory. For many simple instructions, each phase typically takes one clock cycle, although this may differ depending on the CPU, the type of instruction, and the addressing modes for the operands.

A simple instruction would take five clock cycles from start to finish. In order to speed up program execution, most modern processors employ a strategy called **pipelining,** where successive instructions over- lap their execution phases.

For example, while one instruction is in its write-back phase, the next instruction would be in its execute phase, the following one in its data fetch phase, and so forth. As long as all instructions are executed in sequential order so that their order of execution is known in advance by the processor.

A speedup of instruction processing by a factor of five would be realized in this case. A pipelining processor would have to include provisions for instructions that change the order of execution, such as *branch* and *jump* instructions. A jump will terminate one execution pipeline and start another at a different instruction loca- tion. Instructions that have gone through some steps of their execution cycle may have to be cancelled (undone) if a branch is determined after their execution cycle is started. It is also sometimes necessary to delay the pipeline if an instruction needs as its input an operand that is being produced by the previous instruction. Hence, the speedup actually achieved by pipelining must be estimated by averaging the speedup achieved by many different programs.

Interrupts

An interrupt is usually an **asynchronous event,** which is an event that can occur at any time, and is hence not synchronized with the system clock and with processor instruction execution cycle. The interrupt signals to the processor that it needs to handle a high-priority event. The processor hardware typically includes one or more **interrupt registers,** which are set by the interrupting event.

Whenever an instruction finishes executing, the control circuitry automati- cally checks to see whether any event has placed a value in an interrupt register. Hence, interrupts cannot be serviced *during instruction execution*—only between instructions. If so, the **processor state**—which includes the contents of the pro- gram counter and any registers that will be used during interrupt processing—is saved into memory and a jump to execute the program code that handles interrupts is performed. Once the interrupt handler is done, the system will normally restore the processor state and resume processing the user program from the point at which it was interrupted. The OS may switch to run another program if the interrupt caused the current program to be terminated or suspended.

While processing an interrupt, it is usually the case that lower priority or less important interrupts are disabled until interrupt handling is completed. The OS does this by setting an **interrupt disable** (or **interrupt mask**) register. Depending on the value in that register the system will not check for interrupts for lower priority inter- rupt levels. Hence, the OS can set this register before starting interrupt processing, and reset it back after completing the interrupt processing.

We can categorize the events that cause interrupts into hardware events and soft- ware events. In general, hardware interrupts are asynchronous and software inter- rupts are synchronous. Typical of the **hardware events** that can cause interrupts are the following:

Some I/O user action has occurred, such as mouse movement or mouse button click or keyboard input. The interrupt handler would retrieve the information about the I/O action, such as mouse coordinates or which character was input from the keyboard.

A disk I/O transfer was completed. The interrupt handler would check to see if other disk I/O operations were pending, and if so initiate the next disk I/O transfer to or from main memory.A clock timer interrupt has occurred, which allows the OS to allocate the CPU to another program.

The **software events** that can cause interrupts may be further categorized into **traps,** which occur when a program error or violation happens, and **system calls,** which occur when a program requests services from the OS. (a sys- tem call interrupt is sometimes called a trap—.) Some events that cause traps are the following:

A memory protection violation, for example, a program executing in user mode tries to access an area of memory outside of its allowed memory space.

An instruction protection violation, for example, a program executing in user mode attempts to execute an instruction reserved for supervisor mode.

An instruction error such as division by zero.

An arithmetic error such as a floating point overflow.