**Operating System**

• Exploits the hardware resources of one or more processors

• provides a set of services to system users

• manages secondary memory and I/O devices

**Basic elements**

• Processor (if only one CPU)

• has at least two internal registers

⁃ Memory address register

• Main memory

⁃ volatile

• I/O modules

⁃ secondary memory devices

⁃ communications equip

⁃ terminals

• system bus

⁃ communication among processors, main memory, I/O modules

**Processor Registers**

• User-visible registers

⁃ enables programmer to minimize main memory references by optimizing register use

• Control and status registers

⁃ used by processor to control operating of the processor

⁃ used by privileged OS routines to control execution of programs

**User-Visible Registers**

• May be referenced by machine language

• Available to all programs

**Instruction execution**

• Two steps in a loop (simplest)

⁃ processor fetches instruction from memory

⁃ address of next instruction is contained in PC register

⁃ Processor executes fetched instruction (execution stage)

• When does it end?

⁃ Only if turned off, unrecoverable error, or an explicit instruction to halt

**Interrupts**

• interrupts are a mechanism to interrupt a normal sequence of a processor

⁃ Event that alters the sequences of instructions executed by CPU

⁃ Correspond with electrical signals generated by hardware circuits

• helps to improve CPU utilization

⁃ **WE WILL HAVE TO KEEP THIS AS A STATISTIC DURING OUR PROGRAMS!**

**Classes of interrupts**

• Program

• Timer

• I/O

• Hardware Failure

**CPU interactions**

• Two types of interactions between CPU and hardware

⁃ CPU giving orders

• Hardware many tell something to CPU and hardware

⁃ CPU giving orders

**Interrupts and execution cycle**

• Remember we have synch or asynchronous execution of instructions and I/O

• when external device is ready, I/O module sends interrupt request to CPU

• CPU responds by suspending current process

**Interrupt Process**

• If there is an interrupt (flag is set), CPU goes to interrupt handler

⁃ Interrupt handler looks at type of interrupt and executes appropriate function

⁃ some overhead from doing this

⁃ less than if CPU had to do everything

**Interrupt Constraints**

• Interrupt handlers have some constraints

⁃ must be very fast

⁃ deferrable part that can be handled later

⁃ block of data arrives on network line

⁃ Kernel marks presence of data **URGENT PART**

⁃ gives control back to process running before on CPU

⁃ Rest of processing can be done later (move data to buffer)

⁃ Interrupt handler uses interrupt vector

**Interrupt Processing**

• Device issues interrupt to CPU

• CPU finishes execution of current instruction

• CPU tests for pending interrupt requests

⁃ if so, inform device, this removes interrupt signal

• cpu saves program status word onto control stack

• cpu loads the location of interrupt handler into PC register

• **Save content of all registers from control stack to memory**

• **find out the cause of interrupt, invoke appropriate routine**

**• restore saved registers**

**• restore PC to last dispatched program**

**Interrupts in Linux**

• CPU receives interrupt, stops what it is doing

⁃ unless it is processing higher order interrupt

• Saves parameters on stack and calls interrupt handler

• System is in unknown state, so not all things allowed with handler

**Cache Principles**

• Slide

**SLIDES TO REVIEW!!!!!**

• **Hit ratio and access time —> \*Test Question\***

⁃ Slide 54

• **Cache Principles**

⁃ Slide 64

⁃ Replacement Algorithm

⁃ How to choose which block to replace in cache

• **Memory Mgmt**

⁃ slide 67

⁃ **Major areas of responsibility**

• **Memory Bandwidth**

⁃ NOTE: QPI —> Quick Path Interconnect is just a point-to-point link between caches.

**• Multiprocessor organization**

⁃ Array processing / SIMD

⁃ single instruction, multiple data

⁃ multiple processing elements that perform the same operation on multiple pieces of data at the same time

⁃ only a single instruction at any time

⁃ General multiprocessing / MIMD

⁃ processors that function asynchronously and independently

• **NOTE**: