OS has 4 components: Processor(CPU)-controls the op.s of comp., main mem-volatile(only available when powered) I/O-modules, system bus- communication among processors, main mem, and I/O. other resources: **Microprocessor:** A processor on a single chip. Essentially the same thing as a regular processor but a microprocessor refers to a CPU of a smaller computer system, such as a personal computer or an embedded device like a phone.**Graphical Processing Unit:** (GPU) Used for general numerical processing.**Digital Signal Processor:** (DSP) Deal with streaming signals such as audio/video. Found in things like headphones, speakers, and smartphones. Also support for encryption and security as well.**System on a chip:** (SOC) To make handheld devices, manufacturers will sometimes put CPUs (microprocessors), GPUs, codecs, and main memory on the same chip.**Program Counter:**while proc. Is fetching instr. From main PC holds address of next instr. Then increments. **Instruction Reg.(IR):**Processor interprets fetched instr. And performs action. **Interrupts :**stop sequencing of proc. Done to utilize proc. b/c I/O devices are slower than proc. Common interrupts: Program- occurs due to instruction exec. i.e overflow, divide by 0, ref. out of mem space. Timer- generated by timer w/I proc. Allows OS to perform certain fns regularly. I/O- generated by I/O controller to signal completion of op. or error. Hardware failure- i.e. no power, mem. Parity Clock interrupt- oS determines process has been running too long, memory fault- processor encounters a virual mem address that's not in main. **Mem Hier.:** going down-> decreasing cost per bit, increasing capacity, increasing access time, decreasing freq. of access to mem. By proc. **Performance of a Simple Two-Level Memory:** Avg. Access time- T1*Hit ratio + (T2+T1)*miss ratio. **Cache mem.:**high speed mem. used to store freq. used data and instr, closer to proc. than main mem. allows faster access so keeps copy of most freq. used data from main, proc. First checks cache before main if not found data read into cache, larger cache size is better perf. Size of Cache form.: M=2^n/K where M is cache size, n is # of cache lines/blocks, and K is size of each cache line in bytes. Caches can use LRU or more advanced replacement algo. But more advanced req. more hardware. When writing sometimes need to write from cache to main if data altered, can happen when block updated or when replaced. **I/O techniques:**prog. I/O- IO perf. Action then sets appropriate bits in reg. but doesn't alert proc. So it takes time det. When IO instr. Is done by checking periodically. Interrupt driven IO- IO will interrupt proc. when done but drawbacks are IO transfer rate is limited by speed proc. can test device and proc, is tied up managing IO trans. Direct mem. access(DMA)-use separate module to manage, proc. sends DMA mod. whether read or write is wanted, address of IO dev. starting loc. in mem to read or write from, # of words to be trans. Block transferred directly to mem so it doesn't use proc. **Symmetric Multiprocessors(SMP):**proc. share same mem. and are connected via system bus, share IO dev.s, can all perf. same fns., system controlled by integrated OS. Advantages: perf. Scaling, Availability(no stop if one proc. not working), incremental growth(add more proc.s). **Kernel** - Central component of an operating system that manages operations of computer and hardware; primary interface between the hardware and the processes of a computer. **Batch systems: simple batch sys.:** introduced monitors so user no longer has direct access to proc., job is submitted to computer operator who batches together and places on input device, each prog. Is constructed to branch back to the monitor when complete. Monitor must always be in main mem and available for exec. monitor reads jobs one at a time from input where control is given, when job done return control to monitor which reads next job. **Job Control Language (JCL) -** The programming language used to provide instructions to the monitor; specifies what compiler/data to use. **Hardware features:** mem. protection- while user prog is exec must not alter area containing monitor, timer, privileged instr. That can only be exec by monitor, interrupts. **Multiprogrammed Batch Systems: Uniprogramming:** proc spends certain time executing until reaches IO instr. then must wait until IO done. **Multiprogramming:**when one job must wait for IO proc can switch to another job not waiting for IO. **Processes:** can be prog exec, instance of prog running on comp, entity that can be assigned and exec on proc, unit of activity characterized by a single sequential thread of exec, a current state, and a set of sys resources. **Execution context/process state:**internal data used by OS to supervise process, includes contents of various proc reg.s such as prog. counter and data regs and info useful to OS such as priority and IO. **Virtual mem.:** facility that allows progs to address mem from logical POV w/o regard to amt. of main mem available, temp transfers to and from RAM. **Paging:** allows process to be comprised of # of fixed blocks called pages. Prog accesses word using virtual addres that is page # and offset in the page. **Multithreading:** tech. in which a process, executing an app is divided into threads that run at same time. **Thread:**dispatchable unit of work that execs sequentially and is inter. so proc can turn to another thread. A process is a collection of one or more threads and assoc. resources. **Virtualization:** enables single PC to simultaneously run mult OS. **Process Control Block(PCB):** is a data structure, created and managed by the OS, which stores all the information about a process. Most important it contains enough info so that it can interrup process and later resume exec as if no interrupt happened. PCB key tool that allows OS to support mult processes. **Trace**:the listing of a sequence of instructions that execute for a given process; can characterize the behavior of the processor by showing how the traces of the various processes are interleaved. **Dispatcher**: switches the processor from one process to another. **Two-State process model:** either a process is being exec or not being exec. When OS creates process it creates PCB then enters not running state, process waits to get exec and then goes into running state and then dispatcher chooses what other processes will run from a queue. **Process Creation:** steps: 1. Assign unique process identifier 2. Allocate space for process 3. Initialize PCB 4. Set up appropriate linkages 5. Create/expand other data structures common events leading to creation of process 1. New batch job 2. Interactive logo on 3. Created by OS to provide service 4. Spawned by existing process. **Process Spawning** - when the OS creates a process at the explicit request of another process. **Parent Process:**one process spawning another. **Child Process:** spawned process. **Five-State Model:** splits not running into 2 states(ready and blocked). **Mem. Tables:** used to track main and virtual mem. Must include: alloc of main mem to processes, alloc of virtual mem to processes, protection attributes of blocks of mem, info needed to manage virtual mem. **File Tables:** provide info about existence of files and location etc. **process tables:** must be maintained to manage processes, must reference mem, I/O and files. **Process Image** - collection of program, data, stack, and attributes. **Threads:** processes have two characteristics: resource ownership, scheduling/exec. We call unit of dispatching a thread or lightweight process, unit of resource ownership is a process or task. It is important for threads to synchronize activities with other threads as any resource altered by one thread can affect others. **User Level Threads:** all work of thread manag is done by apps so kernel not aware of threads. Advantages: doesn't require kernel provileges, scheduling can be app specific, ULTs can run on any OS. Disadvantages: cant use multiprocessing. **Kernel Level Threads:** all thread management is done by kernel. Advantages: kernel can use multiprocessing, if one thread is blocked kernel can schedule another thread of same process, can be multithreaded. **Multithreading patterns:** dispatcher worker pattern- dispatcher thread reads requests from mailbox, worker thread performs task. Can lead to bottle necking in dispatcher thread. Pipeline pattern: threads assigned one subtask in system but entire task is pipeline of threads. Threads run concurrently in different pipeline stages, share buffer based communication b/t stages. Performance depends on weakest link.

a) HEX notation

b) IR = OPCode + Mem Addr;

c)IR=16bits;

d) PC = 3 HEX digits; and

e) Mem word size = Data (unsigned integer) = IR

answers: Number of different opcodes: 16

Mem size in bytes: 8192

Mem range: [000 - FFF]

Data range: [0000 - FFFF]

```
#include <iostream>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>


int main()
{
    pid_t pid;
    std::cout << "I am the parent process" << std::endl;
    for(int i=0;i<3;i++)
    {
        pid = fork();
        if (pid == 0)
        {
            std::cout << "I am the child process " << i << std::endl;
            if (i==1)
            {
                pid = fork();
                if (pid == 0)
                {
                    std::cout << "I am a grandchild process from child process " << i << std::endl;
                    _exit(0);
                }
                wait(0);
            }
            _exit(0);
        }
        wait(0);
    }
    return 0;
}
```

```
{
     struct operation *pos_ptr = (struct operation *)pos_void_ptr;
   switch(pos_ptr->op)
   {
       case 0: pos_ptr->result = pos_ptr->val1 + pos_ptr->val2;
               break;
       case 1: pos_ptr->result = pos_ptr->val1 - pos_ptr->val2;
               break;
       case 2: pos_ptr->result = pos_ptr->val1 * pos_ptr->val2;
               break;
       case 3: if (pos_ptr->val2 != 0)
                   pos_ptr->result = (double) pos_ptr->val1 / pos_ptr->val2;
               else
                   pos_ptr->result = 0;
               break;
   }
     return NULL;
}

int main()
{
      static struct operation operations[NOPER];
      pthread_t tid[NOPER];

      for(int i=0;i<NOPER;i++)
      {
            operations[i].op = i;
        std::cin >> operations[i].val1;
        std::cin >> operations[i].val2;
            if(pthread_create(&tid[i], NULL, calculator, &operations[i]))
            {
                  fprintf(stderr, "Error creating thread\n");
                  return 1;

            }
      }
      // Wait for the other threads to finish.
      for (int i = 0; i < NOPER; i++)
            pthread_join(tid[i], NULL);
   for (int i = 0; i < NOPER; i++)
   {
       switch(operations[i].op)
       {
           case 0: std::cout << operations[i].val1 << " + " << operations[i].val2 << " = " << std::fixed << std::setprecision(2) << operations[i].result << std::endl;
                   break;
           case 1: std::cout << operations[i].val1 << " - " << operations[i].val2 << " = " << std::fixed << std::setprecision(2) << operations[i].result << std::endl;
                   break;
           case 2: std::cout << operations[i].val1 << " * " << operations[i].val2 << " = " << std::fixed << std::setprecision(2) << operations[i].result << std::endl;
                   break;
           case 3: std::cout << operations[i].val1 << " / " << operations[i].val2 << " = " << std::fixed << std::setprecision(2) << operations[i].result << std::endl;
                   break;
       }
   }
   return 0;
}
```