Part I – Concepts [2 marks]

a) **[0.1 mark] Explain, in detail, the complete Interrupt mechanism, starting from an external signal until completion. Differentiate clearly what part of the process is carried out by hardware components, and what is done by software.**

   Say the external signal is a Button. When the button is pressed, Hardware triggers an interrupt by sending a signal to the CPU using the system bus. Then, the Software triggers an interrupt by executing a special operation called a system call or monitor call. When the CPU is interrupted, it stops what it is doing and immediately transfers execution to a fixed location. The fixed location usually contains the starting address where the service routine for the interrupt is located. The interrupt service routine executes; on completion, the CPU resumes the interrupted computation.

b) **Explain, in detail, what a System Call is, give at least three examples of known system calls. Additionally, explain how system Calls are related to Interrupts and explain how the Interrupt hardware mechanism is used to implement System Calls.** [Student 2 Ibrahim Komeha]

   A system call is the interface between a user program and the operating system. Instead or letting user programs directly access hardware, system calls provide controlled entry points into the kernel.
   Examples of system calls:
   read() – read data from a file or input device
   write() – write data to a file or output device
   fork() – creates a new process
   When a program executes a system call, it triggers a software interrupt, which the instructions switches the CPU from user mode to kernel mode. The hardware interrupt mechanism is then used to transfer control to the appropriate system call handler in the OS Kernal. After the system call finishes, the kernel executes a return-from-interrupt instruction, resuming user program execution.

c) **[0.1 marks] In class, we showed a simple pseudocode of an output driver for a printer. This driver included two generic statements:**
   i. check if the printer is OK
   ii. print (LF, CR) [student 1 David Mea Andjou]

Discuss in detail all the steps that the printer must carry out for each of these two items

i - check if the printer is OK:

That printer model was taking punch cards as inputs and reads from it. the check_if_printer_OK() is to mainly check if the mechanism of printing is in place and this includes checking:

- if there's no card jam in the printer (maybe flags are raised)
- run the typical mechanism before hand
- only after that it would print the character at position [i++]

ii: print (LF, CR):

LF stands for line feed and CR for Carried return

This command is used to align the printer for the next line to be printed by:

- Moving horizontally (CR) back to position 0
- and moving vertically down (LF) to the next line

d) **0.4 marks] Explain briefly how the off-line operation works in batch OS. Discuss advantages and disadvantages of this approach.**

Input is first read onto tape/disks by separate I/O devices, then later fed to the CPU for processing. Similarly, output is written to tape and printed offline.

- **Advantage:** CPU is not held up by slow I/O, so it can process jobs faster. It's less job for the operators, it doesn't require a human to consistently monitor what is happening with the jobs

- **Disadvantage:** Requires extra hardware and introduces delay between input and execution.

e) **next question**

i. [0.2 marks] Explain what would happen if a programmer wrote a driver and forgot to parse the "$" in the cards read. How do we prevent that error?

Parsing ensures that control symbols like "$" are recognized as special commands. If the programmer forgets to parse it, the system will treat "$RUN",

"$END", etc. as normal text and jobs will not execute correctly. To prevent this, the driver must explicitly check for "$" during parsing.

ii. [0.2 marks] Explain what would happen if, in the middle of the execution of the program (i.e., after executing the program using $RUN), we have a card that has the text "$END" at the beginning of the card. What should the Operating System do in that case?

If $END appears in the middle of execution, the OS interprets it as the end of the current job. It stops the program, discards any remaining cards for that job, and prepares to load the next job in the batch.

f) [0.2 marks] Write examples of four privileged instructions and explain what they do and why they are privileged (each student should submit an answer for two instructions, separately, by the first deadline).

If an attempt is made to execute a privileged instruction in user mode, the hardware does not execute the instruction but rather treats it as illegal and traps it to the operating system. Examples of instructions are: switching to kernel mode, I/O control, timer management, and interrupt management.

I/O control:
I/O control instructions allow the operating system to communicate with hardware devices. They are privileged because giving user programs direct access to I/O could cause corruption, security risks, or device conflicts.

Timer Management:
Timer management instructions configure the system timer, which is essential for process scheduling. They are privileged because, if misused by a user program, they could monopolize CPU time and disrupt multitasking.

Disable interrupts / Enable
These instructions turn off or turn on the hardware interrupts**.** If user programs could disable interrupts, they could potentially block the OS from regaining control which leads to hangs, and waste of CPU time, so therefore only the OS Kernel can control interrupt enabling / disabling

LOAD / STORE
Lets the CPU set or change control registers, such as the base and limit registers used for memory protection and relocation. Allowing a user program to change these would let it access or overwrite other processes memory or the OS itself, breaking isolation and security. Only the kernel can safely modify these registers.

g) **Suppose that you have to run a program whose executable is stored in a tape. The command $LOAD TAPE1: will activate the loader and will load the first file found in TAPE1: into main memory (the executable is stored in the User Area of main memory). The $RUN card will start the execution of the program.**

$LOAD TAPE1:

The **Control Language Interpreter (CLI)** reads this command and activates the **loader**.
The loader uses the **Tape Device Driver** to read the first file from *TAPE1* and loads it into the **User Area** of main memory.
Once loading is complete, control returns to the **monitor** to await the next command.

$RUN:

The **CLI** interprets **$RUN** and transfers control from the monitor to the starting address of the loaded user program.
The **CPU** begins executing the user program's instructions directly from memory.

During execution:

Interrupt Processing handles any I/O or error interrupts that occur while the user program runs, temporarily returning control to the monitor.
The **Device Drivers** manage all I/O operations requested by the user program.

At program termination:

Control returns to the **monitor**, which may invoke **Job Sequencing** to select and start the next job in the batch.

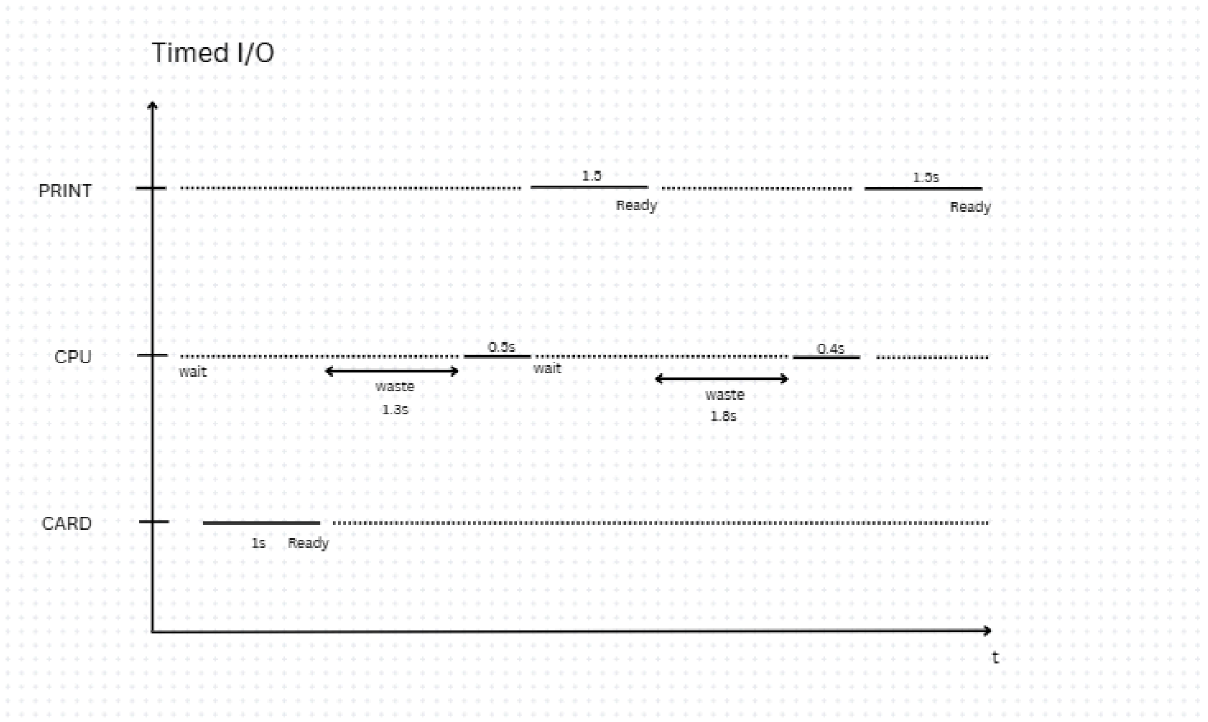
h) [0.3 marks] Consider the following program:

```
Loop 284 times {
        x = read_card(); 1s
        name = find_student_Last_Name (x); // 0.5s
        print(name, printer); //1.5s
        GPA = find_student_marks_and_average(x); // 0.4s
        print(GPA, printer); //1.5s
}
```

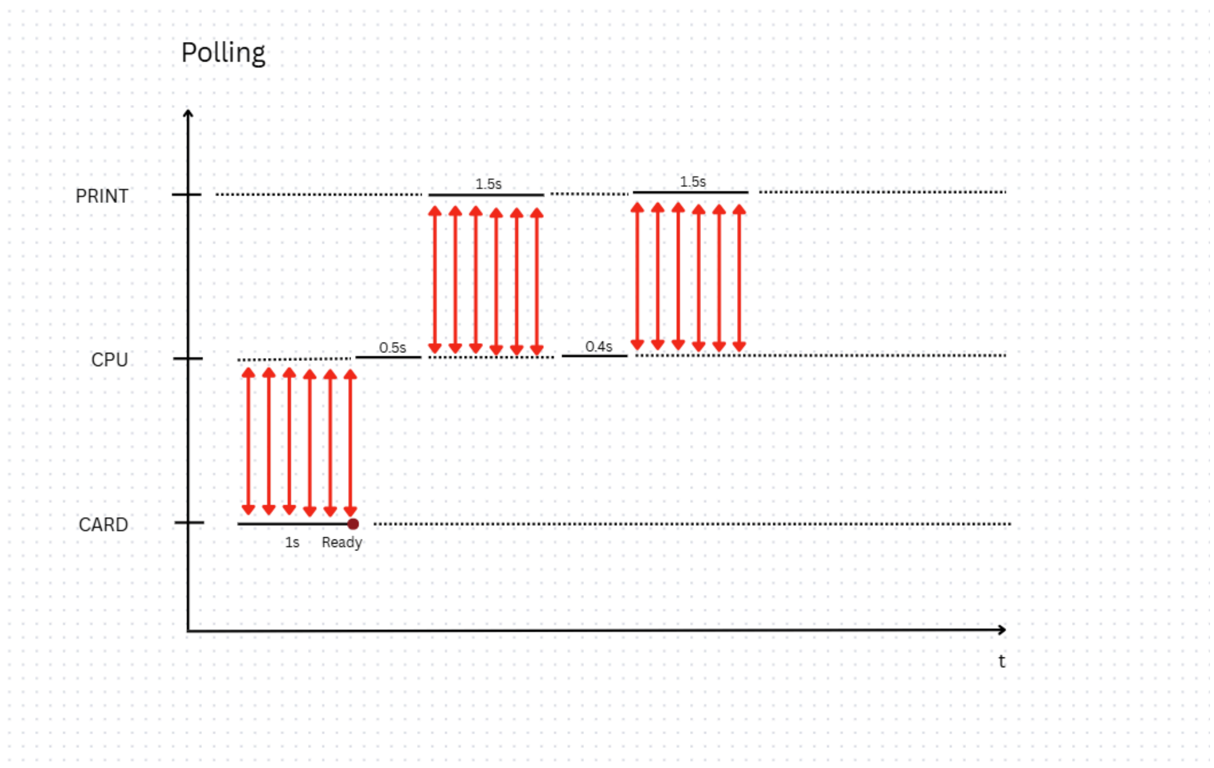
Reading a card takes 1 second, printing anything takes 1.5 seconds. When using basic timing I/O, we add an error of 30% for card reading and 20% for printing. Interrupt latency is 0.1 seconds.

For each of the following cases: create a Gantt diagram which includes all actions described above as well as the times when the CPU is busy/not busy, calculate the time for one cycle and the time for entire program execution, and finally briefly discuss the results obtained.

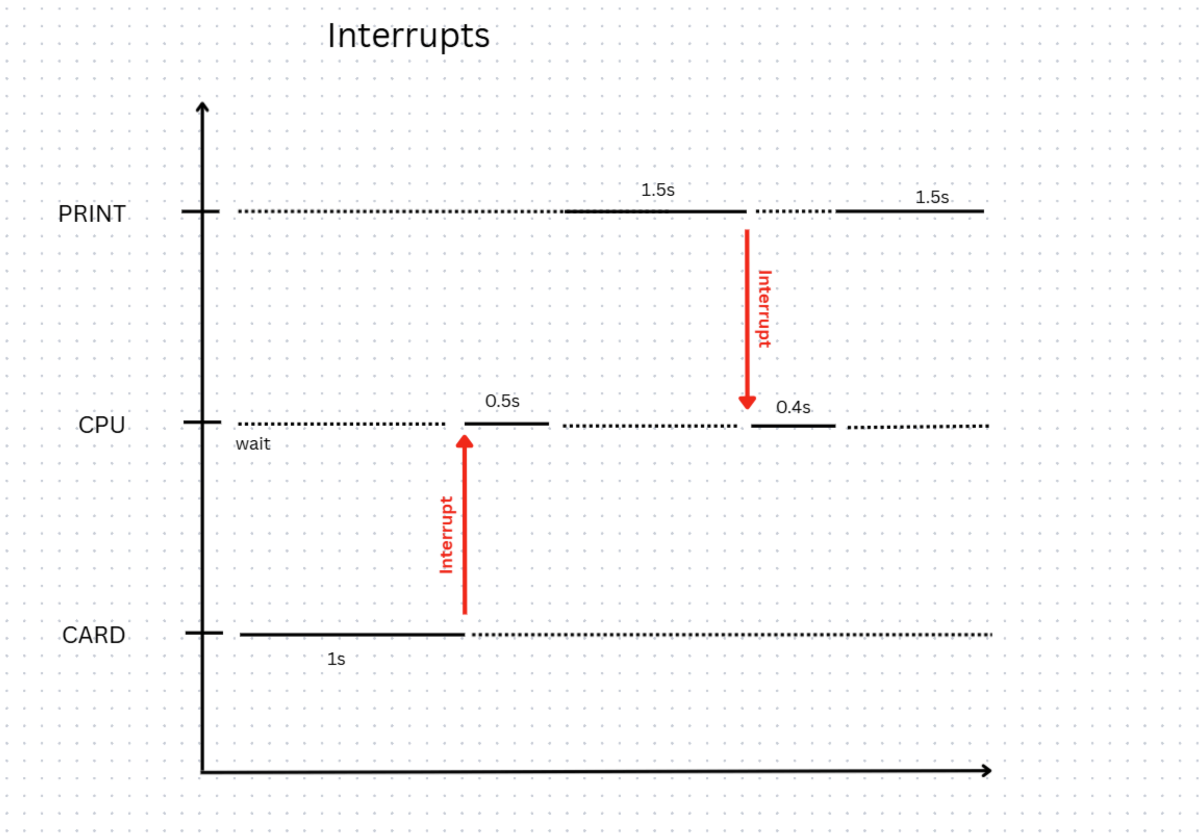
i) Timed I/O



ii) Polling

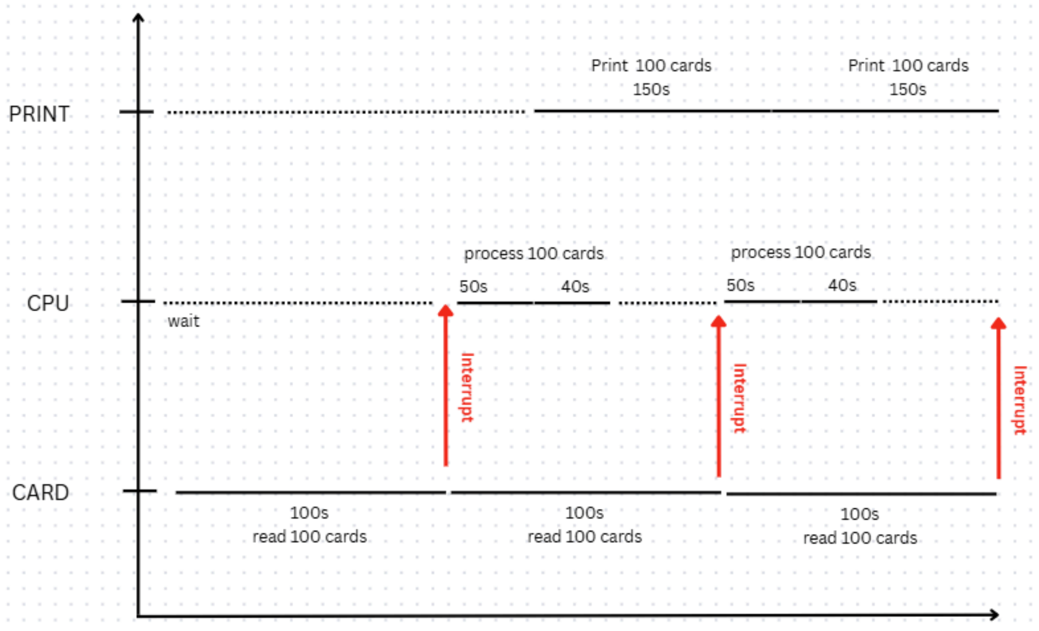


iii) Interrupts



iv) Interrupts + Buffering (Consider the buffer is big enough to hold one input or one output)

Interrupts + buffering


PRINT
CPU
CARD
Print 100 cards 150s
Print 100 cards 150s
process 100 cards
50s
40s
process 100 cards
50s
40s
wait
Interrupt
Interrupt
Interrupt
100s read 100 cards
100s read 100 cards
100s read 100 cards