Computer Organization and Architecture (EET 2211)

Chapter 3
A Top-Level View of Computer Function and Interconnection

Interrupts

- Interrupt is a mechanism by which other modules (I/O, memory) may **interrupt** the normal processing of the processor.

- Interrupts are provided primarily as a way to improve processing efficiency.

- For example, most external devices are much slower than the processor. Suppose that the processor is transferring data to a printer using the instruction cycle scheme. After each write operation, the processor must pause and remain idle until the printer catches up. The length of this pause may be on the order of many hundreds or even thousands of instruction cycles that do not involve memory. Clearly, this is a very wasteful use of the processor.

Classes of Interrupts

Program	Generated by some condition that occurs as a result of an instruction execution, such as arithmetic overflow, division by zero, attempt to execute an illegal machine instruction, or reference outside a user's allowed memory space.
Timer	Generated by a timer within the processor. This allows the operating system to perform certain functions on a regular basis.
I/O	Generated by an I/O controller, to signal normal completion of an operation, request service from the processor, or to signal a variety of error conditions.
Hardware Failure	Generated by a failure such as power failure or memory parity error.

Fig.1. Classes of Interrupts

Instruction Cycle with Interrupts

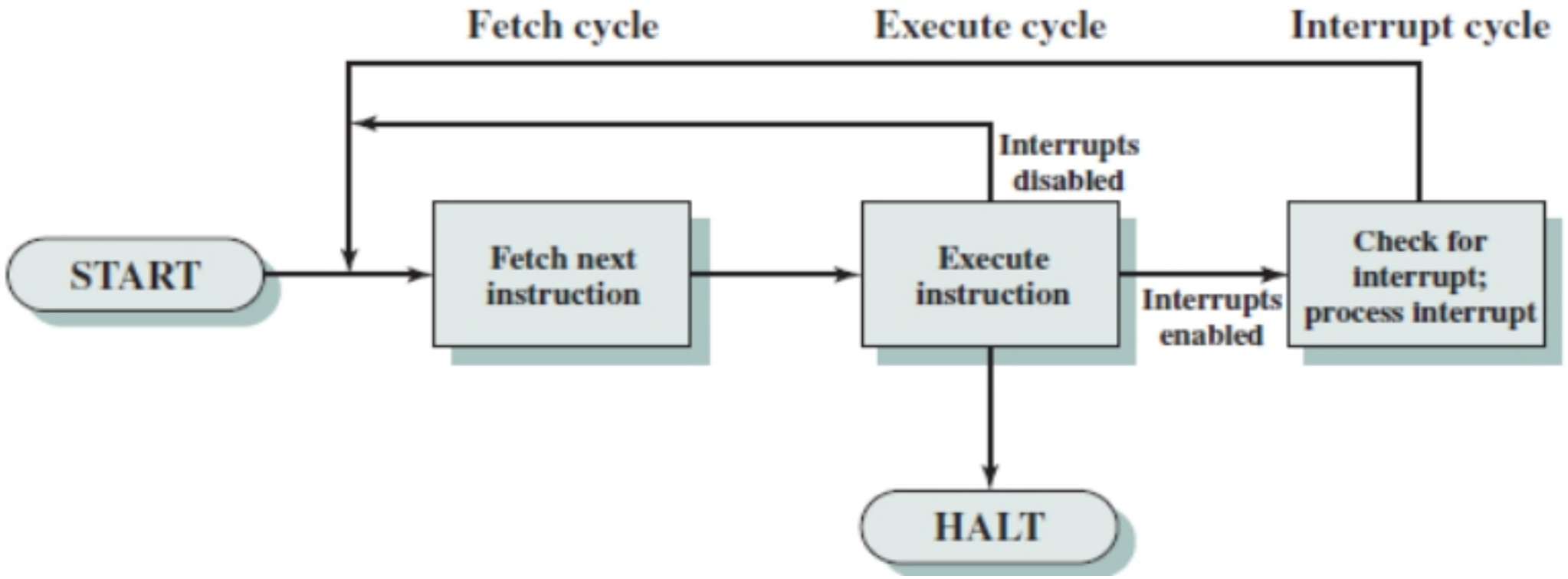


Fig.2. Instruction Cycle with Interrupts

- To accommodate interrupts, an *interrupt cycle* is added to the instruction cycle, as shown in fig.2.

- In the interrupt cycle, the processor checks to see if any interrupts have occurred, indicated by the presence of an interrupt signal.

- If no interrupts are pending, the processor proceeds to the fetch cycle and fetches the next instruction of the current program.

- If an interrupt is pending, the processor does the following:

- It suspends execution of the current program being executed and saves its context. This means saving the address of the next instruction to be executed (current contents of the program counter) and any other data relevant to the processor's current activity.

- It sets the program counter to the starting address of an *interrupt handler* routine.

Interrupt handler

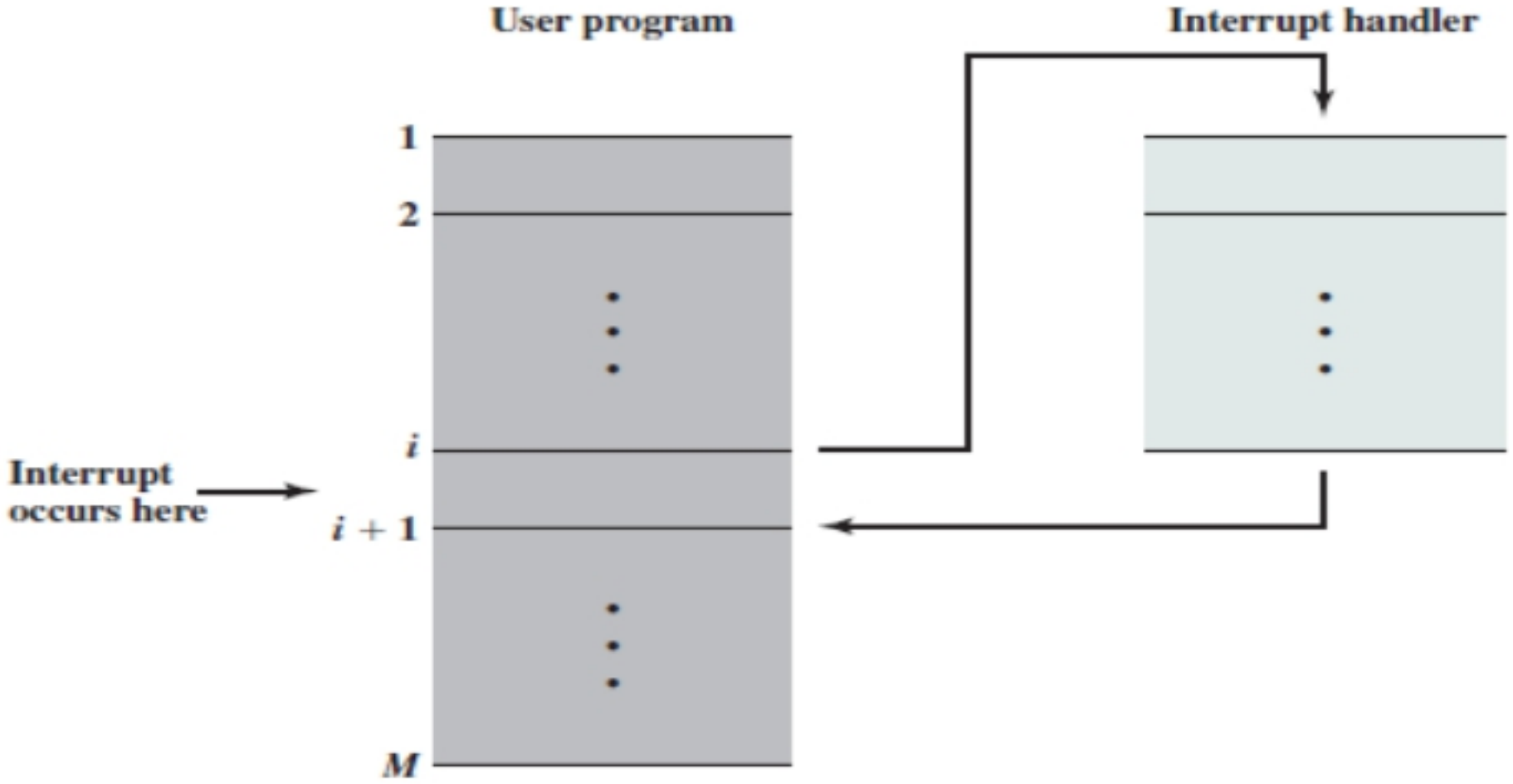


Fig.3. Transfer of Control via Interrupts

- From the user program's point of view, an interrupt is an interruption of the normal sequence of execution.

- When the interrupt processing is completed, execution resumes as shown in fig.3.

- The user program does not contain any special code to accommodate interrupts; the processor and the operating system are responsible for suspending the user program and then resuming it at the same point.

- When the processor proceeds to the fetch cycle, it fetches the first instruction in the interrupt handler program, which will service the interrupt.

- The interrupt handler program is generally part of the operating system which determines the nature of the interrupt and performs whatever actions are needed.

- In fig.3. the handler determines which I/O module generated the interrupt and may branch to a program that will write more data out to that I/O module.

- When the interrupt handler routine is completed, the processor can resume execution of the user program at the point of interruption.

Program Flow of Control and program timing without Interrupts

- Fig.4. shows the program flow of control with no interrupts.
- The user program performs a series of WRITE calls interleaved with processing.
- Code segments 1, 2, and 3 refer to sequences of instructions that do not involve I/O.
- The **WRITE** calls are to an I/O program that is a system utility and that will perform the actual I/O operation.

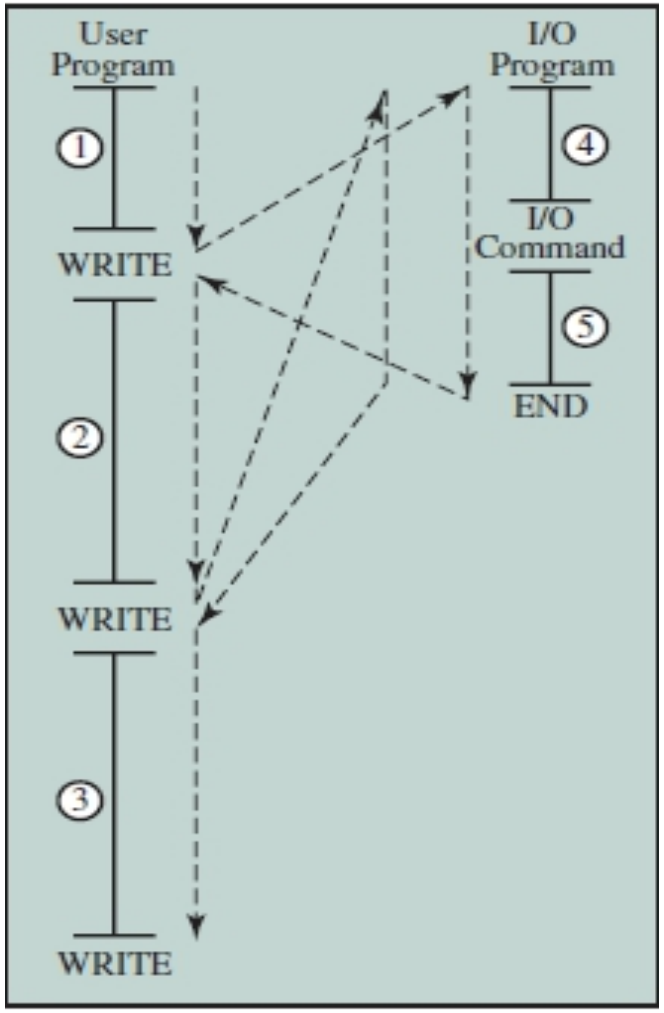


Fig.4(a) Flow control

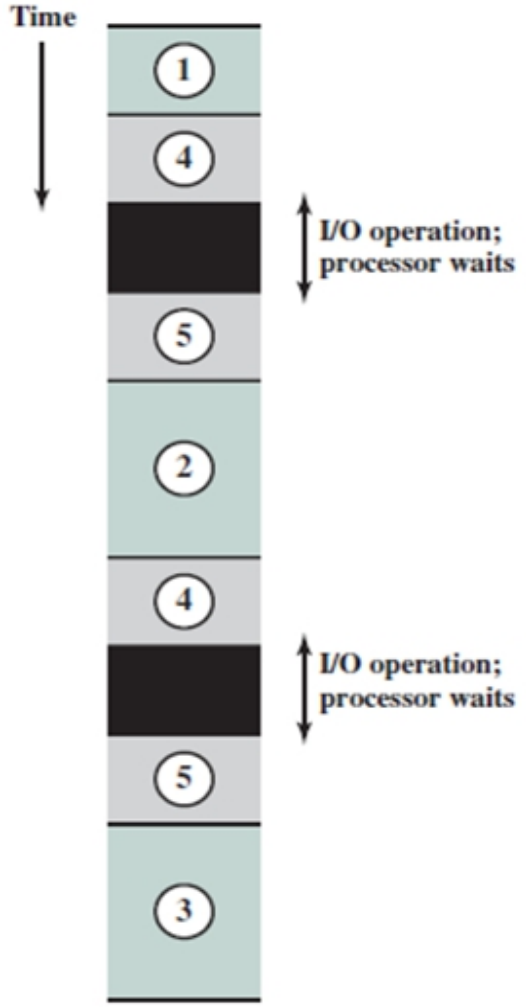


Fig.4(b) Program Timing

- The I/O program consists of three sections:

- A sequence of instructions, labeled 4 in the figure, to prepare for the actual I/O operation. This may include copying the data to be output into a special buffer and preparing the parameters for a device command.

- The actual I/O command. Without the use of interrupts, once this command is issued, the program must wait for the I/O device to perform the requested function (or periodically poll the device). The program might wait by simply repeatedly performing a test operation to determine if the I/O operation is done.

- A sequence of instructions, labeled 5 in the figure, to complete the operation. This may include setting a flag indicating the success or failure of the operation.

*Because the I/O operation may take a relatively long time to complete, the I/O program is hung up waiting for the operation to complete; hence, the user program is stopped at the point of the WRITE call for some considerable period of time.

Program Flow of Control and Program Timing with Interrupts: Short I/O wait

- With interrupts, the processor can be engaged in executing other instructions while an I/O operation is in progress.
- The I/O program that is invoked in this case consists only of the preparation code and the actual I/O command.
- After these few instructions have been executed, control returns to the user program.
- Meanwhile, the external device is busy accepting data from computer memory and printing it.
- This I/O operation is conducted concurrently with the execution of instructions in the user program.

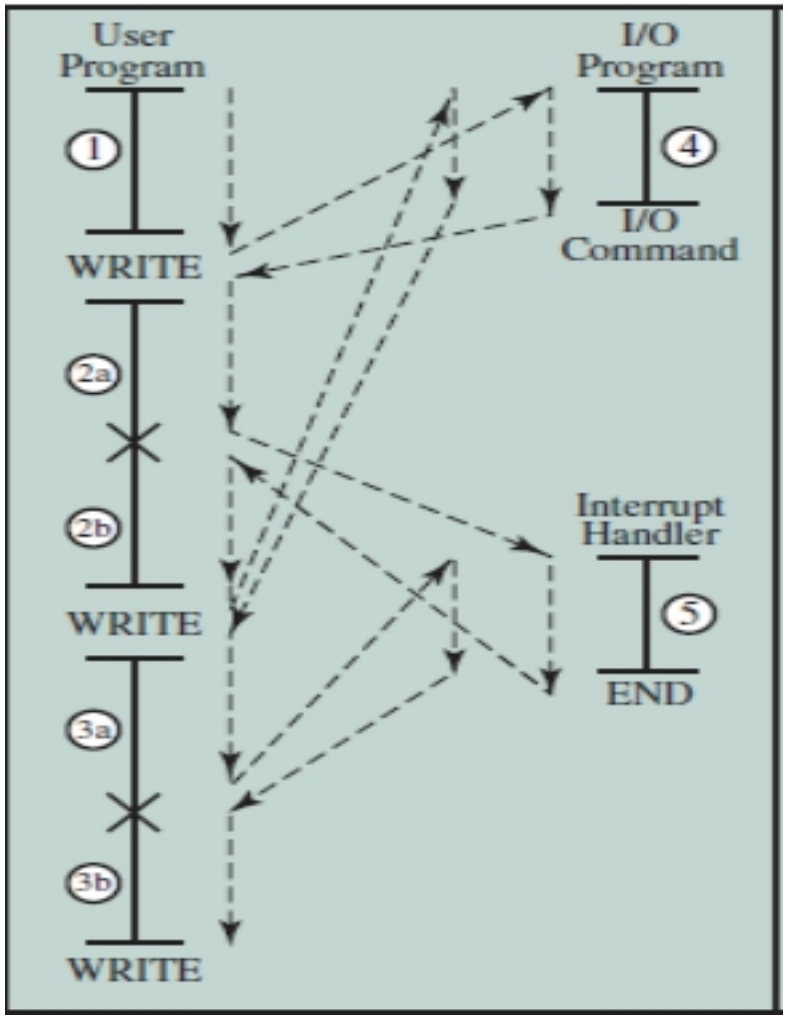


Fig.5(a) Flow Control

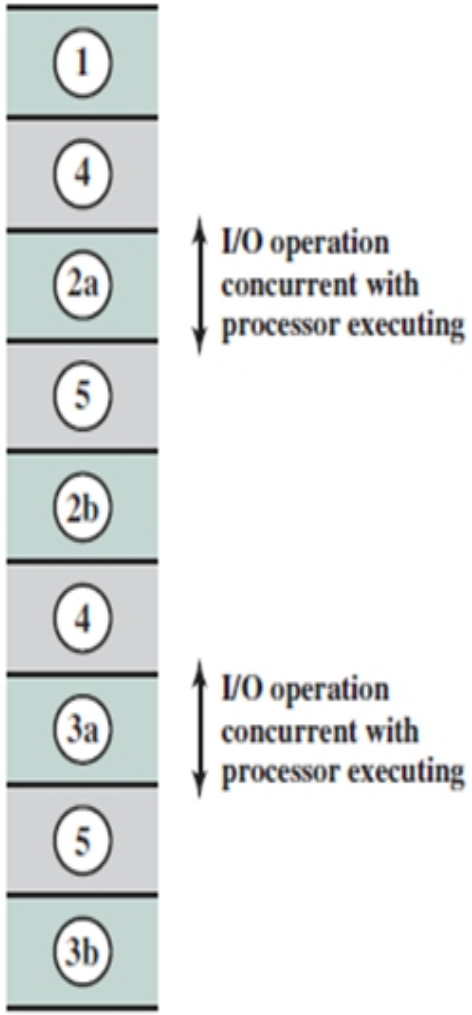


Fig.5(b) Program Timing

- When the external device becomes ready to be serviced—that is, when it is ready to accept more data from the processor—the I/O module for that external device sends an *interrupt request* signal to the processor.

- The processor responds by suspending operation of the current program, branching off to a program to service that particular I/O device, known as an **interrupt handler**, and resuming the original execution after the device is serviced.

- The points at which such interrupts occur are indicated by an asterisk(x) in fig.5.

- Fig- 5(a) and 5(b) assume that the time required for the I/O operation is relatively short: less than the time to complete the execution of instructions between write operations in the user program.

- In this case, the segment of code labeled code segment 2 is interrupted.

- A portion of the code (2a) executes (while the I/O operation is performed) and then the interrupt occurs (upon the completion of the I/O operation).

- After the interrupt is serviced, execution resumes with the remainder of code segment 2 (2b).

Program Flow of Control and Program Timing with Interrupts: Long I/O wait

- Let us consider a typical case where the I/O operation will take much more time than executing a sequence of user instructions (especially for a slow device such as a printer) as shown in fig.6(a).
- In this case, the user program reaches the second WRITE call before the I/O operation spawned by the first call is complete.
- The result is that the user program is hung up at that point.

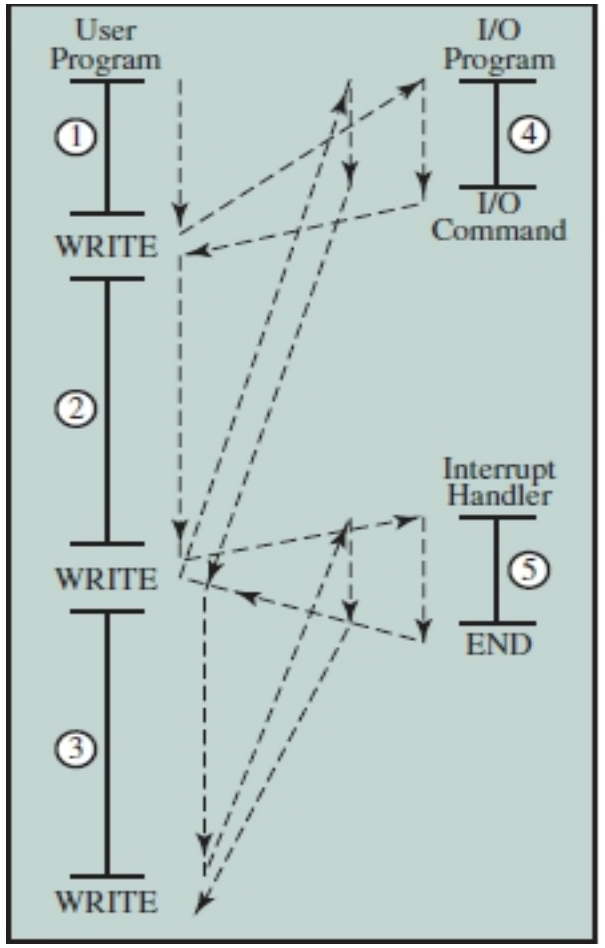


Fig.6(a) Flow Control

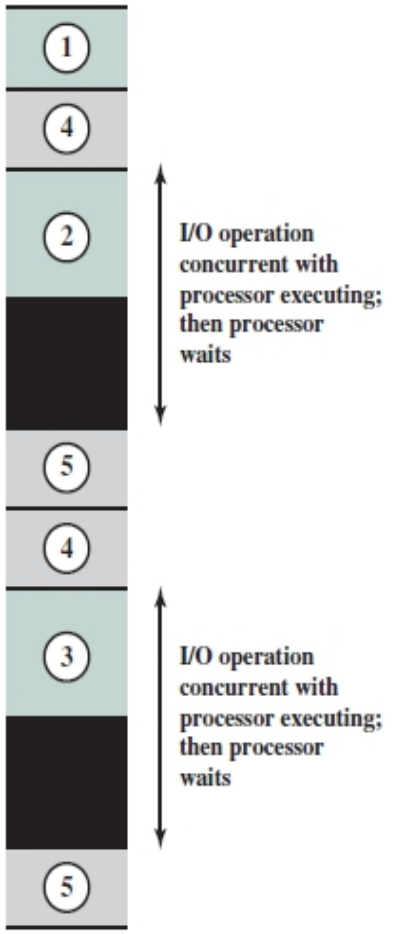
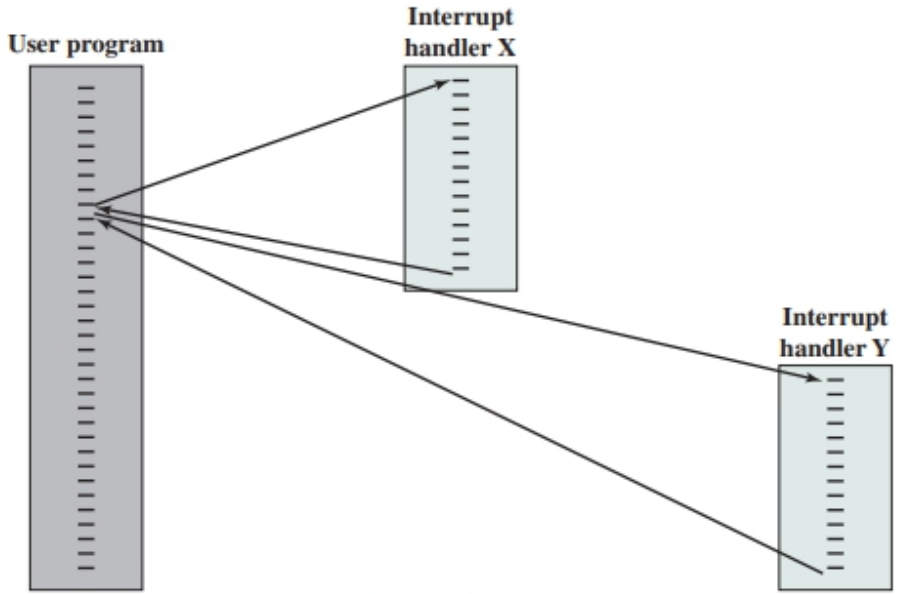


Fig.6(b) Program Timing

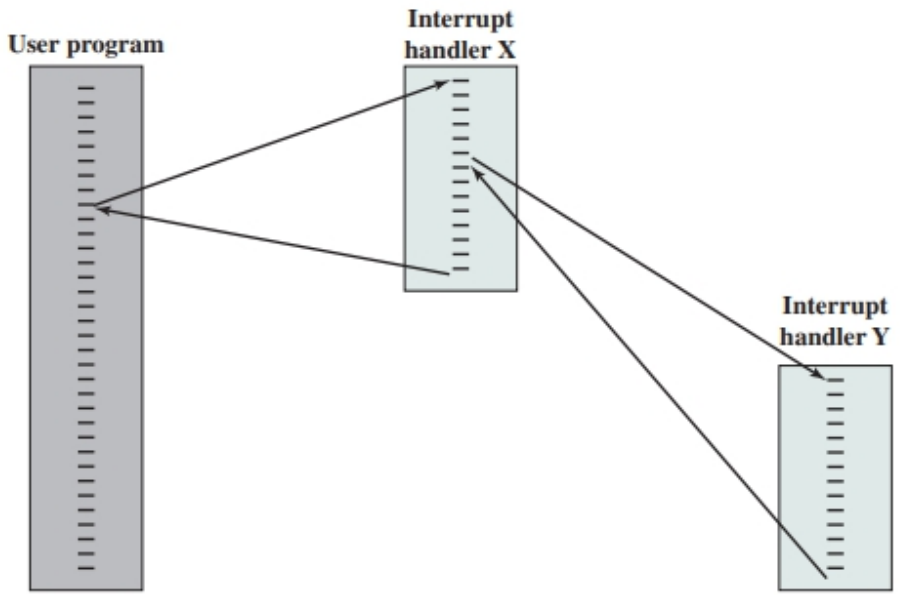
- When the preceding I/O operation is completed, this new WRITE call may be processed, and a new I/O operation may be started.

- Fig.6(b) shows the timing for this situation with the use of interrupts.

- We can see that there is still a gain in efficiency because part of the time during which the I/O operation is under way overlaps with the execution of user instructions.



(a) Sequential interrupt processing



(b) Nested interrupt processing

Figure 3.13 Transfer of Control with Multiple Interrupts

Instruction Cycle State Diagram with Interrupts

Fig.7 shows a revised instruction cycle state diagram that includes interrupt cycle processing.

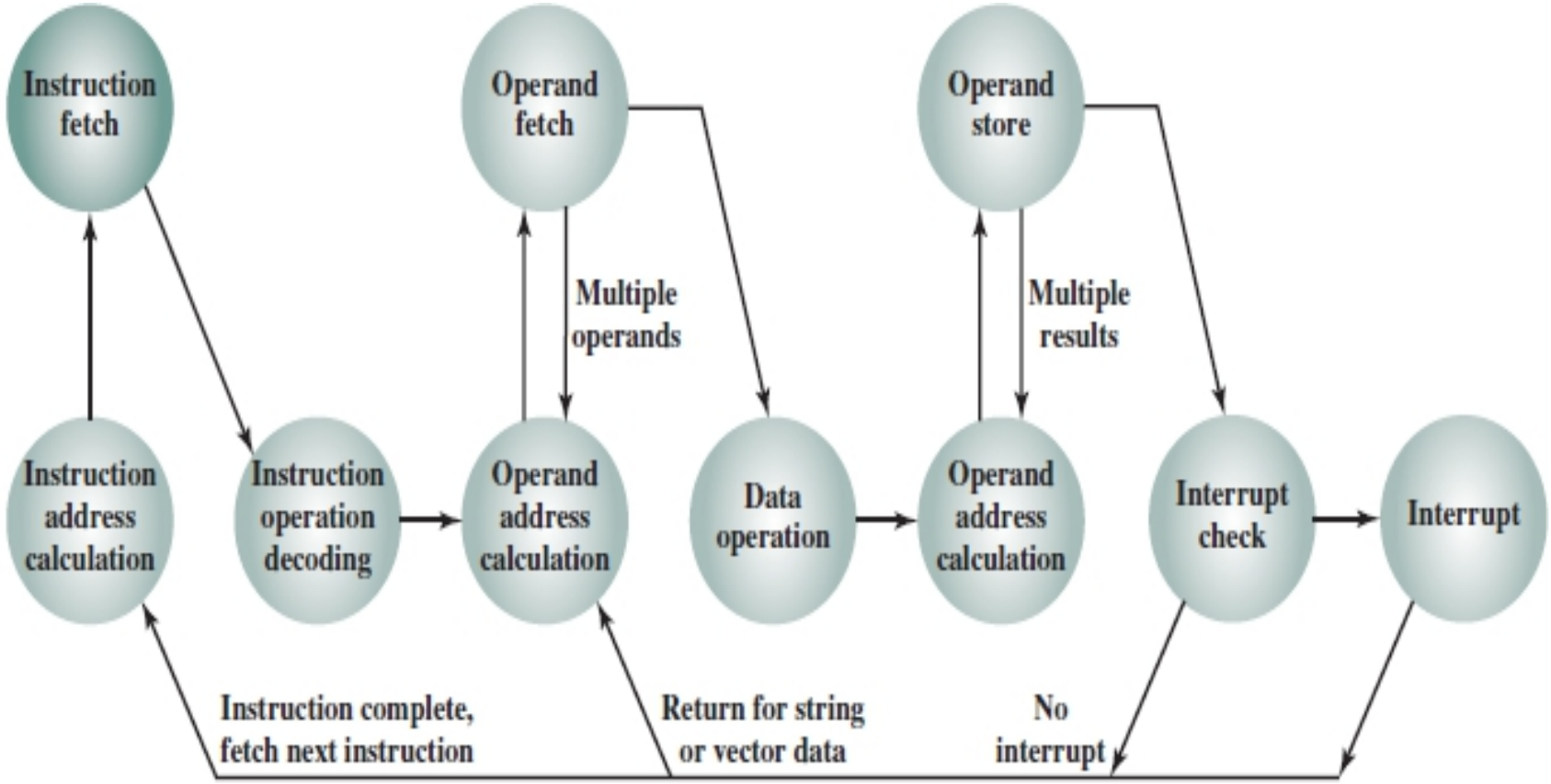


Fig.7. Instruction Cycle State Diagram with Interrupts

Thank you !