The following is a discussion of interrupt handling mechanism in which an Interrupt service routine(ISR) is called using an interrupt vector table.

It comprise 4 steps as discussed below:
1. Interrupt Vector: Each interrupt has a unique vector number associated with it.
2. Indexing IVT: The CPU uses this vector number to index into the IVT.
3. Fetching Address: The address of the interrupt handler corresponding to the vector number is fetched from the IVT.
4. Fetching the handler: Based on the interrupt handler address the correct ISR is fetched by the CPU.

CPU receives a unique vector number associated with that interrupt. This vector number is typically provided by the interrupting device or the interrupt controller.

Indexing IVT:
The Interrupt Vector Table is located at a specific base address in memory. This base address is known to the CPU.

The CPU calculates the address within the IVT by multiplying the vector number by the size of each entry in the IVT (usually the size of a memory address, e.g., 4 bytes).

The calculated entry address points the location of handler address. By using that handler address the cpu jumps to the handler (interrupt service routine) that is required.

Directly mapping the vector number to the handler address would require a very large IVT if the vector numbers are large, leading to inefficient use of memory. By using an index, the IVT can be kept compact and efficient.

Now I want to implement this entire concept interrupt handling mechanism with the use of IVT into a hardware.

For this I have considered a real time example of a library system.

Assuming I have created a vector table in the memory where each entry consists of a vector numbers (mapped with the each book) and the corresponding handler address of that book (shelf and row location)

I have to develop a Lookup Mechanism hardware logic to look up the address in the vector table using the vector number.

After address lookup the Jump Operation is executed: The CPU uses the retrieved address to execute a jump operation to the handler (ISR)

The vector table is below:

Vector Number	Handler Address (Shelf, Row)
1	S1R1
2	S2R1
3	S3R1
4	S1R2
5	S2R2
6	S3R2

The logic of lookup mechanism is as follows:

Multiply the vector number by the size of the entry (2 bits in this case- 1 bit for shelf S & 1 bit for row R) to calculate the entry address.

Retrieve the handler address from the vector table using the calculated entry address

execute the operation to locate the book (handler) in the library system.

Let us imagine 2 scenarios:
The book B1 first is located at location S1R1.

In the second scenario the location of book B1 is updated to other handler address S2R2.

Scenario 1: Initial Location (Book B1 at S1R1)

system needs to find the book B1.

vector number for B1 is 1,

Multiply the vector number (1) by the size of the entry (2 bits) to get the entry address: 1 * 2 = 2.

Use the entry address (2) to index into the vector table and retrieve the address S1R1

Use the handler address S1R1 to locate B1 in the library system.

Scenario 2: Location Changed (B1 at S2R2)

Update the vector table entry for vector number 1 to S2R2

The system needs to find the book B1 again.

vector number for B1 is 1

Multiply the vector number (1) by the size of the entry (2 bits) to get the entry address: 1 * 2 = 2.

Use the entry address (2) to index into the catalog table and retrieve the new address S2R2

Use the handler address S2R2 to locate B1 in the library system.

Now what I notice is that:
since in both the scenarios the calculated entry address is 2 which refers to S2R1 (which in fact not the location of book B1.

So I want to know that how this step should be corrected so that each time when the entry address is calculated, it refers to the correct location of the book B1 which is S1R1 & S2R2 respectively.

This is the hardware logic which I am trying to design.

Pls help?