We are now in a position to detail the full series of steps involved in an interrupt sequence on the CPU:

- Device requests an interrupt by placing its priority on the IRQ lines (IRQ1...7).
- CPU accepts interrupt if IRQ priority > current IPM.
- An internal copy of the SR is taken (i.e.: not saved on the stack).

Full Interrupt Sequence

4. IPM is set to accept new IRQ level of device.
   - This disables the interrupt (the S bit is set to 1 and the T bits are cleared).
5. CPU acknowledges the interrupt by setting the function code (FC) lines and places the current interrupt level on the address bus, a1...a3.
6. Device responds by placing the handler vector number on the data bus (or selects auto-vectoring by asserting AVEC).

Request -> Acknowledge -> Handle

This is a fairly complicated sequence of operations, but stated simply it can be expressed as 3 phases:

- Interrupt Request (stages 1...4).
- Interrupt Acknowledge (stages 5 and 6).
- Interrupt Handling (stages 7...10).

After stage 10, the handler is executing, and presumably taking action appropriate to the interrupt and device.
**Why is Stage 4 Important?**

- If the IPM is not updated to the new priority level, the device will interrupt the CPU again, and will continue to do so.
  - This would result in an infinite loop.
- All the stages 1...10 represent actions taken by the CPU outside of the normal F->D->E cycle.
  - No instructions are actually being executed.
- In stage 4, the T bit is cleared:
  - Prevents tracing of the trace handler itself.

**Clearing the Interrupt**

- As part of the handler, the device must be told to stop interrupting (i.e.: stop asserting the IRQ lines).
- This is not done as part of the interrupt sequence.
- The way in which this is done is usually device specific.
  - If the handler does not clear the source of the interrupt, the IRQ lines will still be asserted after the interrupt is handled, thus the interrupt will still be pending.

**Restoring the IPM**

- After the interrupt has been handled, the IPM must be restored to its original value (i.e.: return to original priority level).
- The RTE instruction retrieves the saved value of the SR.
  - IPM value is reset to the priority prior to the previous interrupt.
- Why was stage 3 required in the interrupt sequence?
  - The IPM prior to the interrupt must be saved, not the IPM that has been adjusted to the IRQ level of the current interrupt.

**NMI (Non Maskable Interrupt)**

- Priority level 7 interrupts are a special case, in that they cannot be masked using the IPM.
- If IPM = 7 and an IRQ of level 7 is requested by a device, the interrupt will still be handled.
- To prevent a device interrupting itself, all level 7 interrupts are edge triggered rather than level triggered.
  - Level 7 interrupt is only generated when the IRQ lines are raised to 7 from a lower level.
**NMI (Non Maskable Interrupt)**

- IPM raised to Current IRQ level
- Handler A Interrupted
- Handler B Running

**Auto Vectoring**

- In stage 6, devices react after the CPU's ACK by either:
  - Placing the IRQ vector number on the data bus, or
  - Selecting auto-vectoring.
- Simpler devices (e.g., 6800 series of devices used with the 8-bit MC6800 CPU) cannot supply a vector number.
- Instead, the IRQ number itself is used to determine the vector number:
  - IRQ n → Auto Vector number n required.

**AVEC Line**

- The CPU must be told that the device required auto-vectoring, so such devices must assert the AVEC line.
- This tells the CPU that no vector number will be supplied on the data bus, and that the IRQ value itself is to be used.
- There are 7 possible auto-vectors, and these are mapped to the vector number 25...31.

**Exceptions**

- Interrupts represent only one subset of the general family of 68332 exceptions.
- Interrupts are exception that have been raised in response to external events.
- Exceptions processing on the 68332 is very similar to interrupt processing.
Exception Grouping

Exceptions may be classified according to priority. On the 68332, there are 4 exception groupings:

<table>
<thead>
<tr>
<th>Group</th>
<th>Exception</th>
<th>Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reset</td>
<td>Resets all processing</td>
</tr>
<tr>
<td>1.1</td>
<td>Address Error</td>
<td>Exception processing begins at the next minor cycle</td>
</tr>
<tr>
<td>1.2</td>
<td>Bus Error</td>
<td>Exception processing begins before the next instruction</td>
</tr>
<tr>
<td>2</td>
<td>TRAP, TRAPV, CHK</td>
<td>Exception processing is started by normal instruction execution</td>
</tr>
<tr>
<td>3</td>
<td>Illegal Instruction</td>
<td>Exception processing begins before the next instruction</td>
</tr>
<tr>
<td></td>
<td>Privilege Violation</td>
<td></td>
</tr>
<tr>
<td>4.1</td>
<td>Trace</td>
<td>Exception processing begins before the next instruction</td>
</tr>
<tr>
<td>4.2</td>
<td>Hardware Breakpoint</td>
<td></td>
</tr>
<tr>
<td>4.3</td>
<td>Interrupt</td>
<td></td>
</tr>
</tbody>
</table>

Highest and Lowest Priority

- Group 0 exception
  - Highest priority
- Group 4 exception
  - Lowest priority

- If both an interrupt (Group 4.3) and a bus error (Group 1.2) occur at the same time, the bus error is always processed first.
- Each exception type has an associated exception vector.

Exception Stack Frame

Each exception generates its own format of exception stack frame.

<table>
<thead>
<tr>
<th>Format</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status Register</td>
<td>0</td>
<td>Program Counter (high)</td>
</tr>
<tr>
<td>Program Counter (low)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vector Offset</td>
<td></td>
<td>Other processing state information (depending on exception)</td>
</tr>
</tbody>
</table>

Exception Processing

During simultaneous exception handling, handlers for lower priority exceptions are called first.

- Higher priority exceptions are processed first.
- Upon exit from CPU exception processing state, the lower priority exceptions are processed in turn.
- Each places its stack frame on the system stack.
- First instruction to execute is first of lowest priority handler.
- RTE from this handler causes execution of next higher priority.
Group 0 & 1 Exceptions

- Reset, Bus Error and Address Error are exceptions of the highest priority and need to be handled with **minimum delay**.
- Current instruction execution is aborted mid-instruction-cycle.
- Each instruction cycle is composed of multiple execution stages = minor-cycles, representing atomic processing steps of the CPU.
- Exception processing begins immediately.

Reset (one)

- A reset exception is caused when:
  - The machine is powered-on.
  - The reset button is pressed.
  - The reset is the highest priority exception and represents a catastrophic event.
  - It is handled differently from all other exceptions.

Reset (two)

1. The PC and SR are not saved on the stack.
2. The Reset vector (Vector 0) is 8-bytes. The first 4 bytes are loaded into the SSP and the next 4 into the PC = CPU state after reset.
3. The SR is initialised with the value $2700, disabling tracing and interrupts (IPM = 7) and VBR set to 0.
4. The reset handler is not terminated with an rte instruction.

Reset (three)

- The reset sequence is known as the bootstrap sequence in more sophisticated operating systems.
**Reset (four)**

- The reset does not cause the loading of a reset vector.
- The reset line from the CPU is asserted, causing all devices connected to the CPU to enter their reset cycles.
- The entire system is put into a consistent starting state whenever a reset or power-on occurs.

**Address Error**

- Address errors are a result of attempting to access a word or a long-word on an odd memory address.
- The current instruction cycle is aborted and SR and PC saved on the Stack and vector number $03$ is used.

**Bus Error**

- Whenever the external logic connected to the address bus requests it.
- Usually due to an attempt to access memory that does not exist.
- Exception processing is as for the Address Error, except vector number $02$ is used.

**Address/Bus Error Processing**

- Serious (current instruction execution cycle is aborted).
- So the processor saves more state on the stack to allow the handler to determine the cause of the error.
Address/Bus Error Processing

- The handler should **not** return control to the user program.
- The robot exception handlers print out the address where the error occurred and resumes execution of the monitor program by writing the address of the monitor program over the saved PC on the stack:
  - `move.l #START,2(a7)`
  - `rte`

Note: If a bus/address error occurs during the exception handler for a bus/address or reset exception handler:
- the CPU enters the halt state, terminating all further processing.

Group 2

- Software exceptions (processing is normal part of instruction execution).
  - `chk` instruction
  - Zero Divide
  - `trap` and `trapv` instruction
- Exception stack frame is the same as for interrupt processing but IPM is left unchanged.

CHK (vector $06$)

- Raised by executing a `chk` instruction:
  - `chk.w <ea>,Dn`
- Tests if the register Dn (word size only) is within the bounds $0 \leq Dn \leq (<ea>)$.
- If not, the exception is raised.
- Useful for performing bounds checking on array subscript registers.
**Zero Divide (vector $05)$**

- Raised if the `divs` or `divu` instruction are given a divisor value of 0.

---

**Trap & Trapv**

- Certain instructions can only be executed in supervisor mode.
- Some hardware features may be inaccessible in user mode.
- We need to provide a way for users to call routines that operate in supervisor mode, allowing access to all instruction and hardware.
- Example: Disk I/O
- A trap is a user-generated exception. There are 16 trap vectors (vector 32...47) available for the 68332.

---

**Example**

To raise a trap vector, the trap instruction is used:

```assembly
move.l #1000,d0  
trap #13  *Delay, 1sec. 
trap #0  *Raise TRAP 0
```

---

**TRAPV**

- The `trapv` instruction is a conditional trap instruction that is raised only if the current overflow flag value is 1.

```assembly
move.b #$7f,d0  
move.b #1,d1  
add.b d0,d1  *Overflow 
trapv  *trap will be taken
```

- Vector used is number $07$.
- Used to provide convenient *error handling* during arithmetic operations.
Using Traps

- The address of the trap handler need not be known to the calling program/user.
- Address of the trap handler can change (by changing the vector table entry) without having to change the program invoking the trap.
- The trap handler becomes a numbered subroutine.

Protected Subroutines (1)

- The address of the trap handler need not be known to the calling program/user.
- Address of the trap handler can change (by changing the vector table entry) without having to change the program invoking the trap.
- The trap handler becomes a numbered subroutine.

Protected Subroutines (2)

- Traps allow access to instructions and resources available only in supervisor mode.
- A trap exception handler executes in supervisor mode, where user subroutines normally execute in user mode.
- Privilege violation whenever privileged instructions are executed.
- The trap handler is a protected subroutine.

TRAP - Parameter Passing

- Parameters may be passed to and from a trap handler just as you would for a subroutine with one exception:
  - You cannot pass back results in the CCR in the normal manner.
- Why?
  - The trap handler is terminated with an *rte* and not an *rts*.
  - The saved value of the SR is popped off the stack, thus destroying any parameter stored in the CCR.
Example - Page One

* This is a test program which uses trap#14 to output a value to the LCD.

```
org $4000
move.b #'A',d1
trap #14 *Write('A')
rts
```

Example - Page Two

* Trap#14 Handler
* Write the character in d1 directly to the LCD
* d1.b Char Value
* T14 move.b d1,DDRamt *write to DD RAM
  bsr Delay *wait for LCD
  rte *
* Install the trap handler vector table entry.
* org 14*4
  dc.l T14

Traps & Operating Systems

Traps are commonly used in operating systems as they provide:
- A convenient way to limit control to sensitive areas of the operating system's hardware and software.
- The Robot uses traps to implement its library subroutines.

Group 3

- Illegal instructions
- F-line and A-line opcodes
- Privilege violation exceptions
- Exception handling sequence is the same as interrupts
Illegal (Unimplemented) Instructions

Many instruction words do not represent legal instructions.

Before the execution of such instructions and illegal instruction exception is raised.

The PC saved on the stack is the address of the illegal instruction, not the address of the next instruction.

F-line & A-line Opcodes

$\text{fxxxx}$
- $\rightarrow$ F-line trap = vector 10

$\text{axxxx}$
- $\rightarrow$ A-line trap = vector 11

Used to extend the instruction set.

The Lower 12 bits may be used to store extra information for instruction.

Privilege Violation

There are 3 CPU execution modes:

- **Supervisor Mode (S-bit=1)**
  - In supervisor mode the SSP (System Stack Pointer) is used as opposed to the USP (User Stack Pointer).
  - The SSP and USP represent different internal registers that are both mapped to the a7 register.
  - Pins on the CPU inform external devices of the current mode of the CPU.
  - This allows devices (e.g. memory) to change access rights according to mode.

User and Supervisor Memory

i.e.: we might have both user and supervisor memory:
2. User Mode (s-bit=0)  
-> Normal mode of Operation

No access is permitted to supervisor memory/resources or instructions.

A special case of user mode is *stopped mode*. If the stop instruction is executed, all further memory accesses are stopped, and the processor halts until a reset, interrupt or trace exception occurs.

2. Halted

If a catastrophic error occurs (e.g., an address error exception in the address error handler), the CPU gives up and enters the halted state, where all further processing is suspended.

The CPU only restarts on a reset.

Supervisor mode

Certain instructions may only be executed in supervisor mode.

```
stop reset move == SR and immediate to SR set immediate to SR or immediate to SR move USP
```

If such an instruction is executed in user mode then a privilege violation exception is raised.

Processing is the same as for an illegal instruction.

That's all