Lecture 9: Exception processing

- Privilege states and exceptions
- Exception taxonomy
- Exception processing in detail
- Hardware-initiated exceptions
- Interrupts
  - Vectored interrupts
  - Auto-vectored interrupts
- Software-initiated exceptions
Privilege states

- **The MC68000 provides two states of privilege**
  - Supervisor (S-bit in SR is 1)
  - User (S-bit in SR is 0)

- **Each state has its own stack pointer and stack**
  - User Stack Pointer
  - Supervisor Stack Pointer

- **A few instructions are only available in supervisor mode**
  
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND #data,SR</td>
<td>MOVE An, SSP</td>
</tr>
<tr>
<td>EOR #data,SR</td>
<td>RESET</td>
</tr>
<tr>
<td>OR #data,SR</td>
<td>RTE</td>
</tr>
<tr>
<td>MOVE &lt;ea&gt;, SR</td>
<td>STOP</td>
</tr>
<tr>
<td>MOVE SSP, An</td>
<td></td>
</tr>
</tbody>
</table>

Transition may occur during exception processing: MOVE to SR, ANDI to SR, EOR to SR, RTE.

Apart from exception processing, transitions may occur through four instructions:
- MOVE to SR
- ANDI to SR
- EOR to SR
- RTE
**Function code output pins**

- The state of the CPU is indicated to external circuitry by three *function code output pins*: FC0, FC1, FC2

<table>
<thead>
<tr>
<th>FC2</th>
<th>FC1</th>
<th>FC0</th>
<th>Memory access type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Undefined --reserved</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>User data</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>User Program</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Undefined --reserved</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Undefined --reserved</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td><strong>Supervisor data</strong>*</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td><strong>Supervisor program</strong>**</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1ACK space (CPU space)</td>
</tr>
</tbody>
</table>

* System is in supervisor mode and accessing data from memory
** System is in supervisor mode and accessing an instruction from memory
Exceptions

- Exceptions can be viewed as *operating system calls*
- Exceptions can be made by either
  - the programmer requesting an OS service (I/O for instance) or
  - the 68000 in response to certain kinds of SW or HW errors
- Each type of exception has its own *exception handler* to deal with the condition that triggered the exception
- Exception handlers are *very similar* to subroutines BUT
  - Exceptions don’t require an explicit address (it is determined by a vector #)
  - Exceptions save PC and SR (and more) to the Supervisor Stack
  - Exceptions set the S-bit to 1, and subroutines do not alter the SR
  - A RTE is used to return instead from the usual RTS
  - Nesting of exceptions is prioritized
  - Exceptions are typically written by the *systems programmer*
- Exceptions can be *external* or *internal*
Exception taxonomy

All exceptions

Hardware (external)
- RESET
- Interrupts
- Bus error

Software (internal)
- Errors
  - Address error
  - Privilege violation
  - Illegal instruction
  - Divide-by-zero
- Programmer initiated
- Coprocessor

Vectored
- Normal
- Spurious

Auto-vectored
- Normal
- Spurious

Errors
- CHK
- TRAPV
- TRAP #n
- Line A
- Line F

Programmer initiated

Interrupts

RESET

Un-initialized

Hardware (external)

Software (internal)

All exceptions
Exception processing

Exception request occurs while instruction i is being executed

Jump to the exception handler routine after finishing* execution of instruction i

Resume program execution after returning from exception handler routine

*With the exception of RESET, BUS ERROR or ADDRESS ERROR exceptions, which initiate exception processing within two clock cycles
Exception processing in more detail

- When the 68000 receives an exception, the following procedure is performed
  - Save PC and SR to the Supervisor Stack
  - Determine the address of the exception handler
  - Execute the exception
  - Restore PC and SR from the Supervisor Stack

Jump to an exception handler

\[
\begin{align*}
[temp\_register] &\leftarrow [SR] \\
[S] &\leftarrow [1] \\
[T] &\leftarrow [0] \\
\text{get\_vector\_number} &\leftarrow \text{vector\_number} \times 4 \\
\text{exception\_vector} &\leftarrow \text{vector\_number} \times 4 \\
\text{handler\_address} &\leftarrow [\text{M(exception\_vector)}] \\
\text{[SSP]} &\leftarrow [\text{SSP}] - 4 \\
[\text{M(SSP)}] &\leftarrow [\text{PC}] \\
[\text{SSP}] &\leftarrow [\text{SSP}] - 2 \\
[\text{M(SSP)}] &\leftarrow [\text{temp\_register}] \\
\text{[PC]} &\leftarrow \text{handler\_address}
\end{align*}
\]

Return from an exception handler

\[
\begin{align*}
[S] &\leftarrow [\text{M(SSP)}] \\
[\text{SSP}] &\leftarrow [\text{SSP}] + 2 \\
[\text{PC}] &\leftarrow [\text{M(SSP)}] \\
[\text{SSP}] &\leftarrow [\text{SSP}] + 4
\end{align*}
\]

To determine the address of the exception handler, the 68000 uses a VECTOR TABLE

- The vector table is stored at $00\ 0000$ to $00\ 03\FF$
- Each element in the vector table consists of 4 bytes that specify the address of the exception handler
- Each exception is associated with an index of the vector table called a vector number
- The handler address is stored in a memory location pointed by the vector number \( \times 4 \)

---

<table>
<thead>
<tr>
<th>Vector number</th>
<th>Memory address</th>
<th>Memory contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$00\ 00$</td>
<td>Reset, initial supervisor stack pointer</td>
</tr>
<tr>
<td>1</td>
<td>$00\ 04$</td>
<td>Reset, initial program counter value</td>
</tr>
<tr>
<td>N</td>
<td>((N)\times 4)</td>
<td>Starting address of the (N)th handler</td>
</tr>
<tr>
<td>N+1</td>
<td>((N+1)\times 4)</td>
<td>Starting address of the ((N+1))th handler</td>
</tr>
<tr>
<td>N+2</td>
<td>((N+2)\times 4)</td>
<td>Starting address of the ((N+2))th handler</td>
</tr>
</tbody>
</table>
### The vector table

<table>
<thead>
<tr>
<th>vector number (Decimal)</th>
<th>address (Hex)</th>
<th>assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>RESET: initial supervisor stack pointer (SSP)</td>
</tr>
<tr>
<td>1</td>
<td>0004</td>
<td>RESET: initial program counter (PC)</td>
</tr>
<tr>
<td>2</td>
<td>0008</td>
<td>bus error</td>
</tr>
<tr>
<td>3</td>
<td>000C</td>
<td>address error</td>
</tr>
<tr>
<td>4</td>
<td>0010</td>
<td>illegal instruction</td>
</tr>
<tr>
<td>5</td>
<td>0014</td>
<td>zero divide</td>
</tr>
<tr>
<td>6</td>
<td>0018</td>
<td>CHK instruction</td>
</tr>
<tr>
<td>7</td>
<td>001C</td>
<td>TRAPV instruction</td>
</tr>
<tr>
<td>8</td>
<td>0020</td>
<td>privileged violation</td>
</tr>
<tr>
<td>9</td>
<td>0024</td>
<td>trace</td>
</tr>
<tr>
<td>10</td>
<td>0028</td>
<td>1010 instruction trap</td>
</tr>
<tr>
<td>11</td>
<td>002C</td>
<td>1111 instruction trap</td>
</tr>
<tr>
<td>12*</td>
<td>0030</td>
<td>not assigned, reserved by Motorola</td>
</tr>
<tr>
<td>13*</td>
<td>0034</td>
<td>not assigned, reserved by Motorola</td>
</tr>
<tr>
<td>14*</td>
<td>0038</td>
<td>not assigned, reserved by Motorola</td>
</tr>
<tr>
<td>15</td>
<td>003C</td>
<td>uninitialized interrupt vector</td>
</tr>
<tr>
<td>16-23*</td>
<td>0040-005F</td>
<td>not assigned, reserved by Motorola</td>
</tr>
<tr>
<td>24</td>
<td>0060</td>
<td>spurious interrupt</td>
</tr>
<tr>
<td>25</td>
<td>0064</td>
<td>Level 1 interrupt autovector</td>
</tr>
<tr>
<td>26</td>
<td>0068</td>
<td>Level 2 interrupt autovector</td>
</tr>
<tr>
<td>27</td>
<td>006C</td>
<td>Level 3 interrupt autovector</td>
</tr>
<tr>
<td>28</td>
<td>0070</td>
<td>Level 4 interrupt autovector</td>
</tr>
<tr>
<td>29</td>
<td>0074</td>
<td>Level 5 interrupt autovector</td>
</tr>
<tr>
<td>30</td>
<td>0078</td>
<td>Level 6 interrupt autovector</td>
</tr>
<tr>
<td>31</td>
<td>007C</td>
<td>Level 7 interrupt autovector</td>
</tr>
<tr>
<td>32-47</td>
<td>0080-00BF</td>
<td>TRAP instruction vectors*</td>
</tr>
<tr>
<td>48-63</td>
<td>00C0-00FF</td>
<td>not assigned, reserved by Motorola</td>
</tr>
<tr>
<td>64-255</td>
<td>0100-03FF</td>
<td>user interrupt vectors</td>
</tr>
</tbody>
</table>

**NOTES:**
- * No peripheral devices should be assigned these numbers
- ** TRAP #N uses vector number 32+N
Exception types and priorities

The exceptions in the 68000 are divided into three groups, according to their priority and characteristics

- **Group 0 exceptions**
  - Basically mean that something has gone seriously wrong with the system
  - For this reason, more detailed information is saved on the stack to assist diagnosis
  - You cannot return from a Group 0 exception with the RTE command since the information stored on the stack is different than what RTE expects!

- **Group 1 exceptions**
  - generated by traces, interrupts, illegal op-codes or privilege violations

- **Group 2 exceptions**
  - Occur as part of the normal instruction-execution sequence of a program

<table>
<thead>
<tr>
<th>Group</th>
<th>Exception</th>
<th>Time at which processing begins</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reset, Bus error, Address error</td>
<td>Exception processing begins within two clock cycles</td>
</tr>
<tr>
<td>1</td>
<td>Trace, Interrupt, Illegal op-code, Privilege</td>
<td>Exception processing begins before the next instruction</td>
</tr>
<tr>
<td>2</td>
<td>TRAP, TRAPV, CHK, Divide-by-zero</td>
<td>Exception processing is started by normal instruction execution</td>
</tr>
</tbody>
</table>

Information saved on stack for Group 0

- Memory access type and function code
- Access address (high word)
- Access address (low word)
- Instruction register
- Status register
- Program counter (high word)
- Program counter (low word)

Information saved on stack for Groups 1,2

- Status register
- Program counter (high word)
- Program counter (low word)
Hardware-initiated interrupts

- **RESET**
  - Takes place only under two circumstances
    - Power-up
    - Total and irrecoverable system collapse
  - RESET has the highest priority and will be processed before ANY other exception that is pending or being processed
  - Detected by a low RESET* pin on the 68000
  - On a RESET exception, the 68000 performs the following operations
    - \([SR] \leftarrow \$2700\) (S=1, T=0 and interrupt mask level set to 7)
    - \([SSP] \leftarrow \$0000\) (initialize Supervisor Stack with the first element in the vector table)
    - \([PC] \leftarrow \$0004\) (Initialize PC with the second element in the vector table)
    - Resume execution at the position pointed by PC

- **BUS ERROR**
  - Raised by failure of the system to complete a bus cycle
    - Illegal memory access: the processor tried to access an address not populated by memory
    - Fault to memory access: if error-detecting memory is used
    - Failure to assert VPA*: used for auto-vectored interrupts (details later)
    - Memory privilege violation: when the 68000 uses some form of memory management
  - Detected by a low BERR* pin on the 68000

- **INTERRUPTS… next slides**
Interrupts

- The 68000 provides two interrupt schemes
  - **Vectored**: intended for modern 16-bit peripherals
  - **Auto-vectored**: intended for older 8-bit peripherals
- There are seven levels of interrupts available
- The sequence of operations during an interrupt request is the following
  - A peripheral requires attention by asserting its interrupt request output (IRQ*)
  - The priority encoder produces a 3-bit code with the highest IRQ* line active and passes it to the 68000 on the IPL0*-IPL3* inputs
  - The 68000 compares the level of the interrupt with the interrupt mask flag (I_2I_1I_0) in the SR.
  - If the requested input is greater than (I_2I_1I_0), the interrupt is serviced, otherwise it is ignored
  - If the 68000 decides to service the interrupt:
    - The code 111 is placed on (FC_2FC_1FC_0) to inform the system that an interrupt is about to be serviced
    - The priority of the interrupt is placed on (A_3A_2A_1)
    - (FC_2FC_1FC_0) and (A_3A_2A_1) are passed to an interrupt acknowledge decoder which asserts one of the seven IACK* lines
  - The asserted IACK* line informs the interrupting device that it is about to be serviced
  - The remaining steps are dependent on the type of interrupting device (vectored or auto-vectored) and are detailed in the next two slides
**Vectored interrupts**

- Vectored interrupts are intended for peripherals that can provide an 8-bit vector number (256 values) to the 68000
  - This vector number is stored in register in the peripheral (IVEC)
  - **It is the programmer’s responsibility** to initialize the device with the appropriate vector number!

- After the appropriate IACK* line is asserted by the 68000, the following operations are performed
  - The peripheral whose interrupt level matches the asserted IACK* will “know” that it is going to be serviced
  - The peripheral then writes the IVEC vector onto the data bus (D7-D0) and asserts the DTACK* line (DTACK stands for Data Transfer Acknowledge)
  - the active DTACK* terminates the IACK cycle and the 68000 will execute the interrupt handler pointed by the vector fetched from (D7-D0)

- **There are two variations to this procedure**
  - If DTACK* is not asserted, BERR* (Bus Error) must be asserted by the external hardware to force a **spurious interrupt exception**
  - If the peripheral has not been initialized with an appropriate vector, it should place $0F$ on the data bus to force an **uninitialized interrupt vector exception**
Auto-vectored interrupts

- Auto-vectored interrupts are intended for earlier peripherals designed for 8-bit processors that cannot provide a vector during an IACK cycle.

- After the appropriate IACK* line is asserted by the 68000, the following operations are performed:
  - The interrupting device will assert the VPA* line (Valid Peripheral Address).
  - Upon receiving an asserted VPA* line, the 68000 assumes the peripheral is a 6800-series and then:
    - Ignores the contents of (D0-D7).
    - Internally generates an interrupt vector from the priority level of the IRQ* line that was asserted.
    - The 68000 reserves vector numbers $19$-$1F$ for auto-vectored interrupts on IRQ1*-IRQ7*:

<table>
<thead>
<tr>
<th>Vector Number</th>
<th>Vector Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0064</td>
<td>Level 1 Interrupt autovector</td>
<td></td>
</tr>
<tr>
<td>0065</td>
<td>Level 2 Interrupt autovector</td>
<td></td>
</tr>
<tr>
<td>006C</td>
<td>Level 3 Interrupt autovector</td>
<td></td>
</tr>
<tr>
<td>0070</td>
<td>Level 4 Interrupt autovector</td>
<td></td>
</tr>
<tr>
<td>0074</td>
<td>Level 5 Interrupt autovector</td>
<td></td>
</tr>
<tr>
<td>0078</td>
<td>Level 6 Interrupt autovector</td>
<td></td>
</tr>
<tr>
<td>007C</td>
<td>Level 7 Interrupt autovector</td>
<td></td>
</tr>
</tbody>
</table>

- When several peripherals are assigned to the same IRQ* level, then the 68000 cannot distinguish between them and the appropriate auto-vectored handler routine MUST poll each of the possible requesters and read their interrupt status registers.

![Diagram of interrupt system](image)
Software-initiated interrupts

- **TRACE**
  - Forced when the T-bit of the SR is set to 1
  - When T=1 a trace exception is generated after each execution of an instruction
  - This enables the programmer to step through the execution of a program

- **Address error:**
  - Occurs when the 68000 attempts to access a 16- or 32-bit longword at an odd address

- **Privilege violation**
  - Occurs when the processor is in user mode and attempts to execute an instruction reserved for the supervisor state

- **Illegal instruction**
  - Occurs when the CPU fetches an op-code from memory that corresponds to an unimplemented instruction
    - This typically occurs when the effective address of a branch instruction is computed wrong

- **Divide-by-zero**
  - Occurs when a number is divided by zero
Software-initiated interrupts

- **CHK**
  - Used to check against bounds
    - For example, checking of array indexes against the boundaries of the array
  - Generated by the assembly instruction `CHK <ea>, Dn`
  - For example, the instruction `CHK D1, D0` will generate a CHK exception if
    - `[D0 (0:15)] < 0`
    - `[D0 (0:15)] > [D1 (0:15)]`

- **TRAPV**
  - Generated by the assembly instruction TRAPV, which forces the exception if the V-bit of the CCR is set to 1

- **Emulators**
  - Op-codes whose four MSBs (bits 12 to 15) are %1010 ($A) or %1111 ($F) are unimplemented in the 68000, but they are not treated as illegal instructions!
  - Line A instructions
    - Used to emulate instructions missing in the 68000
  - Line F instructions
    - Similar to Line A, but is associated with a different vector number
    - Typically used in 68020 to emulate a co-processor when one is not present in hardware
Software-initiated interrupts: TRAPS

- TRAPS are the most useful software user-initiated exceptions available
- TRAPS are not different from line A or line F exceptions except for the vector number associated with them
- Generated with the instruction \texttt{TRAP \#0 to TRAP \#15}, which are associated with vector numbers 32 to 47 decimal
- TRAP is normally used to provide portable code between 68000-based systems with different peripherals
- At first it may seem that the 68000 is limited to 16 TRAP operations
  - This is not the case since it is possible to pass a integer to the trap handler in a data register
  - Within the trap handler, the integer is used as an index into a jump table that points to the desired routine
Exception processing flowchart