Exceptions and Interrupts 1

• Outline
  – Exceptions and Interrupts
  – Exception Handling
  – Interrupts
  – Exception Handling Details
  – Exception Types

• Goal
  – Understand exceptions
  – Understand interrupts

• Reading
  – Microprocessor Systems Design, Clements, Ch. 6
Exceptions and Interrupts

• Exception
  – event that alters normal program execution

• Types of exceptions
  – hardware
    » read error - e.g. memory did not respond
    » page fault - must fetch page from disk
  – software
    » automatic- e.g. divide by zero, illegal instruction
    » user - e.g. CHK2 instruction
  – interrupt

• Interrupt
  – asynchronous exception
  – message from external device seeking CPU attention
  – example
    » disk controller has data ready
68K Family Exceptions

All Exceptions

Hardware
- Reset
  - Vectored
  - Autovectored
- Interrupts
- Bus

Software
- Trace
- Errors
  - Priv
  - Illegal Inst
    - \( \div 0 \)
- Programmer
  - CHK
  - TRAP

...
Exception Handling

• Interrupts and exceptions are unplanned “calls” to supervisor mode
  – operating system software
  – exception handler routines
  – only way for user to enter supervisor mode
    » example: system call to do I/O

• Exception handler
  – code designed to handle specific type or class of exception
  – normally part of operating system
  – mapping from exception type to handler is predefined
    » user does not name handler in subroutine call
    » e.g. TRAPV (trap on overflow) looks up overflow handler starting address in exception vector table
  – RTE - return to user program
  – can have nested exceptions
Event Polling

- **Synchronous** deviations in program flow
  - subroutine calls, conditional branches
  - occur at predetermined points in program
  - example: polling loop for keyboard input

```assembly
KEY_STAT EQU $F00001 input status reg
KEY_VAL  EQU KEY_STAT+2 input data reg
LEA KEY_STAT,A0
LEA KEY_VAL,A1
TEST_LOOP BTST #0,(A0) test LSB of status
           BEQ TEST_LOOP do until 0
           MOVE.B (A1),D1 read data
```

- Problem - CPU busy waiting for input
Interrupts

• **Asynchronous** change in control flow
  – only take action when necessary
  – independent of current program flow
  – peripheral asserts interrupt request line (IRQ)
  – CPU takes interrupt only if priority higher than current task
  – CPU completes current instruction
    » *interrupt latency* should not be too long
  – CPU saves PC and status word on stack
  – CPU jumps to interrupt handler to take care of request
  – when done, PC and status word are restored

• **Interrupt is* transparent* to interrupted program**
  – user does not have to worry about it
<table>
<thead>
<tr>
<th>Types of Interrupts</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Nonmaskable</strong></td>
</tr>
<tr>
<td>- <em>masked</em> interrupt - deferred</td>
</tr>
<tr>
<td>- nonmaskable - cannot defer</td>
</tr>
<tr>
<td>» e.g. loss of power</td>
</tr>
<tr>
<td>- 68000 level 7 request IRQ7* is nonmaskable</td>
</tr>
<tr>
<td><strong>Prioritized</strong></td>
</tr>
<tr>
<td>- if more than one peripheral, need to prioritize interrupt handling</td>
</tr>
<tr>
<td>» e.g. LAN controller &gt; keyboard due to higher speed</td>
</tr>
<tr>
<td>- assign static priority to each peripheral</td>
</tr>
<tr>
<td>- deal with all higher priority interrupts before lower priority</td>
</tr>
<tr>
<td>» can “starve” low-priority devices</td>
</tr>
<tr>
<td>- 68000 has 7 priority levels</td>
</tr>
</tbody>
</table>
Vectored Interrupts

• **Interrupting device identifies itself to CPU automatically**
  – CPU acknowledges interrupt
  – peripheral sends ID number (vector) on data bus
  – CPU uses vector to calculate interrupt handler address
  – 68000 can also handle nonvectored interrupts
    » from older peripherals
    » requires general handler to poll device for its ID, then call specific handler

• **68020-68040**
  – have vector base register to move exception vector table location
  – SSP is called interrupt stack pointer, have another master stack pointer can be used for other system activities
Module Process_exception
{
    [TmpReg] <- [SR] save SR
    S <- 1 supervisor mode on
    T <- 0 trace mode off
    Get VectorNum calc exception type number
    Add <- 4*VectorNum calc handler table index
    Handler <- [M(Add)] get handler add from table
    [SSP] <- [SSP] - 4
    [M([SSP])] <- [PC] push PC on system stack
    [SSP] <- [SSP]-2
    [M([SSP])] <- [TmpReg] push status register
    [PC] <- Handler call handler
}

... ProcessException handler processes exception then does RTE ...
{
    [SR] <- [M([SSP])]
    [SSP] <- [SSP]+2
    [PC] <- [M([SSP])]
    [SSP] <- [SSP]+4
}
End Process_exception
Example: Serial I/O Interrupts

* set up ACIA to interrupt on data receive

SETUP MOVE.B #$03,ACIAC  reset ACIA
MOVE.B #$91,ACIAC  set ACIA to interrupt
RTS  8 bits, no parity

...  

* ACIA interrupt handler  

ACIAH MOVEA.L PTR,A0  ptr to input buffer
MOVE.B ACIAC,D0  read ACIA status reg
BTST.B #$7,D0  tst IRQ bit
BEQ ACIAR  ACIA did not interrupt
BTST.B #0,D0  tst if input ready
BEQ ACIAR  no data ready
MOVE.B ACIAD,(A0)+  read data, put in buf
MOVE.L A0,PTR  save ptr to input buf
ACIAR RTE

...  

ORG $064  Loc of autovector level 1
DC.L ACIAH  put vector to handler in vector table
<table>
<thead>
<tr>
<th>Exception Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Reset</td>
</tr>
<tr>
<td>– RESET* and HALT* low for 10 clocks</td>
</tr>
<tr>
<td>• Bus Error</td>
</tr>
<tr>
<td>– BERR* went low</td>
</tr>
<tr>
<td>• Interrupt</td>
</tr>
<tr>
<td>• Address Error</td>
</tr>
<tr>
<td>– unaligned address in some situations</td>
</tr>
<tr>
<td>• Illegal Instruction</td>
</tr>
<tr>
<td>• Divide by Zero</td>
</tr>
<tr>
<td>• Privilege Violation</td>
</tr>
<tr>
<td>• Trace</td>
</tr>
<tr>
<td>– after each instruction if T-bit set</td>
</tr>
</tbody>
</table>
## Exception Types

- **Line 1010 Emulator**
  - MSBs 1010/1111 are unimplemented and illegal
  - special exception, use to emulate unimplemented instruction

- **Line 1111 Emulator**
  - same as 1010 but different vector
  - use to emulate coprocessor instructions

- **Uninitialized Interrupt Vector**
  - peripheral sends #$0F vector if vector not initialized

- **Spurious Interrupt**
  - acknowledge interrupt, no device responds

- **Double Bus Fault**
  - bus error when handling bus error, CPU halts

- **Format Error**
  - exception stack frame wrong format during RTE

- **CHK, CHK2, TRAP, TRAPV, TRAPcc**
  - generated if condition satisfied