Recall... Compilation Flow!
Software

High Level Language

Compiler

Assembly Language

Assembler

Machine Code

Loader

Hardware

Word processor; Video game
Windows 2000
Unix
C++
Java
C- compiler
High level language (hll) and processor dependent
Instructions
Processor dependent
Pentium, Power PC
MC68000
Processor dependent
Binary,
Processor dependent
Pentium, Power PC
MC68000
Processor: Pentium III, Power PC
Memory
IO devices
Application SW: Calculator

High level SW Program

begin
var a, b, c;
int ....;
float ....;
a := 3;
b := 2;
c := a + b;
end

Compiler
Allocate vars to registers
A ↔ R1
b ↔ R2
c ↔ R4

Assembly Program
LOAD #3, R1
LOAD #2, R2
ADD R1, R2, R4

Machine Program

1000 00 0011
1000 01 0010
1010 11 00 00
Recall... Status Register!
A diagram illustrates the components of an MC68000 system. The system includes an I/O Device, MC68000 processor, and Memory. The MC68000 is connected to Memory via an Address bus (24 bits) and a Data bus (16 bits). There is also a Control bus connecting these components.

The I/O Device is connected to the MC68000 via an I/O bus (3 lines). The MC68000 has a Status Register, which includes:

- **Trace bit**: 0: normal, 1: trace
- **Mode bit**: 0: user, 1: super user
- **Mask bits**: 7 levels of interrupts
- **Condition code bits**: X, N, Z, V, C

The diagram visually represents these connections and features.
Recall... Trap Instructions!
move #248, d7
trap #14 ; char in D0.B is displayed

move #247, d7
trap #14 ; char read into D0.B

move #228, d7
trap #14 ; stops program execution
Exceptions and MC68000

- **Internal**: caused by software (instruction execution), e.g., divide by zero, trap
- **External**: caused by hardware (external devices), e.g., reset, interrupt

- By an exception, normal program is suspended and jump to exception handling **subroutine**

- Exceptions can be **recoverable** (e.g. interrupt) or **unrecoverable** (e.g. divide by zero)
Handling Exceptions

• Each exception "mapped" to a unique exception vector number (0....255)

• Vector number index for exception vector table (256 32-bit address = 1KB reserved)

• Vector table address at V*4 points to execution handling routine location in memory

• Exception handling routines differs depending on the exception type
## Exception vector table

<table>
<thead>
<tr>
<th>Vector number</th>
<th>Address</th>
<th>Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$000000</td>
<td>Reset: Initial SSP</td>
</tr>
<tr>
<td>1</td>
<td>$000004</td>
<td>Reset: Initial PC</td>
</tr>
<tr>
<td>2</td>
<td>$000008</td>
<td>Bus error</td>
</tr>
<tr>
<td>3</td>
<td>$00000C</td>
<td>Address error</td>
</tr>
<tr>
<td>4</td>
<td>$000010</td>
<td>Illegal instruction</td>
</tr>
<tr>
<td>5</td>
<td>$000014</td>
<td>Division by zero</td>
</tr>
<tr>
<td>6</td>
<td>$000018</td>
<td>CHK instruction</td>
</tr>
<tr>
<td>7</td>
<td>$00001C</td>
<td>Trapv instruction</td>
</tr>
<tr>
<td>8</td>
<td>$000020</td>
<td>Privilege violation</td>
</tr>
<tr>
<td>9</td>
<td>$000024</td>
<td>Trace</td>
</tr>
<tr>
<td>10</td>
<td>$000028</td>
<td>Line $A$ emulator</td>
</tr>
<tr>
<td>11</td>
<td>$00002C</td>
<td>Line $F$ emulator</td>
</tr>
<tr>
<td>12</td>
<td>$000030</td>
<td>Reserved by Motorola</td>
</tr>
<tr>
<td>13</td>
<td>$000034</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>$000038</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>$00003C</td>
<td>Uninitialized interrupt vector</td>
</tr>
<tr>
<td>16</td>
<td>$000040</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>$000044</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>$000048</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>$00005F</td>
<td>Reserved by Motorola</td>
</tr>
<tr>
<td>24</td>
<td>$000060</td>
<td>Spurious interrupt</td>
</tr>
<tr>
<td>25</td>
<td>$000064</td>
<td>Level 1 Interrupt autovector</td>
</tr>
<tr>
<td>26</td>
<td>$000068</td>
<td>Level 2 Interrupt autovector</td>
</tr>
<tr>
<td>27</td>
<td>$00006C</td>
<td>Level 3 Interrupt autovector</td>
</tr>
<tr>
<td>28</td>
<td>$000070</td>
<td>Level 4 Interrupt autovector</td>
</tr>
<tr>
<td>29</td>
<td>$000074</td>
<td>Level 5 Interrupt autovector</td>
</tr>
<tr>
<td>30</td>
<td>$000078</td>
<td>Level 6 Interrupt autovector</td>
</tr>
<tr>
<td>31</td>
<td>$00007C</td>
<td>Level 7 Interrupt autovector</td>
</tr>
<tr>
<td>32</td>
<td>$000080</td>
<td>TRAP Instruction vectors</td>
</tr>
<tr>
<td>47</td>
<td>$0000BF</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>$0000C0</td>
<td>Reserved by Motorola</td>
</tr>
<tr>
<td>63</td>
<td>$0000FF</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>$000100</td>
<td>User interrupt vectors</td>
</tr>
<tr>
<td>255</td>
<td>$0003FF</td>
<td></td>
</tr>
</tbody>
</table>
Exception Execution (high level view)
Sequence of Events on Internal Exception

1. Instruction executing
2. Exception?
   - T
   - F
3. Identify exception via appropriate vector number (0–255)
4. Saverax in a working register
   Clear trace bit and set supervisor bit
   Push pc (in supervisor stack)
   Push rax (in supervisor stack)
5. Load pc with contents of location (exception vector number) times 4
6. Exception routine is executing
7. Fatal error?
   - T
   - F
8. Pop (supervisor stack) to rax
   Pop (supervisor stack) to pc
9. Execute next instruction
Division by zero
Exception
Return from Exceptions

• After execution of the exception handling routines, return from exception using:
  • rte
    – (1) pops 16-bit top of system stack into status register SR
    – (2) pops next 32-bit of system stack into program counter PC
    – (3) executes instruction at current PC

➤ **Note:** rte vs. rtr : system stack vs. user stack!
## MC68000 Exceptions

<table>
<thead>
<tr>
<th>Group</th>
<th>Exception type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Illegal instruction, unimplemented instruction, privilege violation, trace, trap, trapv, chk, zero divide</td>
</tr>
<tr>
<td>2</td>
<td>Bus error, address error, (external) reset, interrupt</td>
</tr>
</tbody>
</table>
External Exceptions

- **reset (v=0,1)**
  - asserting of external reset pin (reset line)
  - used for booting
  - at least 2 words in ROM!

- **bus error (v=2)**
  - attempt to access out of range memory address
  - memory misalignment
  - conflict in data bus access

- **interrupt (v=25...31)**
  - asserting external interrupt line (3 bits)
  - **7 levels** (001...111) [000 default]
  - allows asynchronous interaction within I/O devices (speed matching and priorities)
MC68000 Memory

Address bus

Data bus

I/O bus

I/O Device

MC68000

Memory

Interrupt

Trace bit:
0: normal
1: trace

Mode bit:
0: user
1: super user

Mask bits
7 levels of interrupts

Status Register

Condition code bits

X
N
Z
V
C
Internal Exceptions

- **division by zero (v=5)**
  - if source of division is zero
  - non-recoverable (displays error message and terminates)

- **trace (v=9)**
  - trace bit set
  - exception generated after each instruction (e.g. to debug code)

- **privileged Instruction (v=8)**
  - attempt to execute privileged instruction while in user mode (e.g. rte)
Exceptions and MC68000

• **unimplemented instruction (v=10,11)**
  – also called line $A$ or line $F$ emulation exception ($A : V=10, F: V=11$)
  – invalid instruction whose first 4-bit is $A$ or $F$
  – used to implement (emulate) additional instructional instruction, e.g., FP add, FP sub, FP mul, FP div,
    – using dc.w $A000$, $A001$, $A002$, $A003$

• **Illegal instruction (v=4):**
  – invalid instruction whose first 16-bit is $4AFA$, $4AFB$, $4AFC$ or any other except unimplemented instruction
  – can be used to force breakpoints through no-op and exception
Internal Exceptions

• trap (v=32...47):
  – explicit exception via software, allows communication with I/O using 16 different traps, e.g., trap#14
  – rule: V=14+32 = 46
  – O.S. routine for I/O within the exception handling routine

• trapv (v=7)
  – trap on overflow causes exception if overflow bit is set

• chk (v=6):
  – checks if a constant (first op.) is an upper bound of a data register (2nd op) e.g. assume D0=$000000F2

  \[
  \text{chk} \ #5,D0 \quad 5 < $F2 < 405 \\
  \text{chk} \ #405,D0
  \]
Read from external device for example a keyboard:
move #247,D7
trap #14
D0.b <-- Input Buffer

Write in to external device e.g., screen
move #247,D7
trap #14
Output Buffer <-- D0.b
Nested Exceptions
Nested Exceptions

3 levels of priority:

- high 0
- 1 → 7 interrupt sub levels (high 7, ....low 0)
- Low 2
Unimplemented Instruction

```assembly
our_handler

movem.l   d0/a0,-(sp) ; save registers
move.l    usp,a0 ; a0 has usp
move.l    (a0)+,parm1 ; get a
move.l    (a0)+,parm2 ; get b
move.l    a0,usp ; reset usp
move.l    10(sp),a0 ; a0 gets address of illegal instruction
addq      #2,10(sp) ; fix return address
clr.w     d0 ; low order word of
move.b    1(a0),d0 ; d0 has $0000z
asl.w     #2,d0 ; multiply by 4
move.l    #operation,a0 ; a0 has address of the
jsr       (a0) ; table
move.l    0(a0,d0.w),a0 ; a0 has address of
movem.l   (sp)+,d0/a0 ; proper routine
rte       ; execute proper real
parm1     ds.l  1 ; operation
parm2     ds.l  1
operation dc.l  real_add,real_sub,real_mult,real_div ; restore registers
          ; return to “next”
          ; instruction
```
## Illegal Instructions

<table>
<thead>
<tr>
<th>Location</th>
<th>First word</th>
</tr>
</thead>
<tbody>
<tr>
<td>004004</td>
<td>3038</td>
</tr>
<tr>
<td>004016</td>
<td>5540</td>
</tr>
<tr>
<td>004020</td>
<td>6000</td>
</tr>
</tbody>
</table>

(a) Breakpoint table

### Location object code

<table>
<thead>
<tr>
<th>Location</th>
<th>First word</th>
</tr>
</thead>
<tbody>
<tr>
<td>006000</td>
<td>0003</td>
</tr>
<tr>
<td>006002</td>
<td>FD FD</td>
</tr>
<tr>
<td>004000</td>
<td>3E7C 7000</td>
</tr>
<tr>
<td>004004</td>
<td>3038 6002</td>
</tr>
<tr>
<td>004008</td>
<td>6100 000C</td>
</tr>
<tr>
<td>00400C</td>
<td>31C0 6002</td>
</tr>
<tr>
<td>004010</td>
<td>3E3C 00E4</td>
</tr>
<tr>
<td>004014</td>
<td>4E4E</td>
</tr>
<tr>
<td>004016</td>
<td>3F00</td>
</tr>
<tr>
<td>004018</td>
<td>5540</td>
</tr>
<tr>
<td>00401A</td>
<td>6600 0008</td>
</tr>
<tr>
<td>00401E</td>
<td>301F</td>
</tr>
<tr>
<td>004020</td>
<td>6000 0006</td>
</tr>
<tr>
<td>004024</td>
<td>61F0</td>
</tr>
<tr>
<td>004026</td>
<td>C0DF</td>
</tr>
<tr>
<td>004028</td>
<td>4E75</td>
</tr>
</tbody>
</table>

(b) Input program

<table>
<thead>
<tr>
<th>Location</th>
<th>First word</th>
</tr>
</thead>
<tbody>
<tr>
<td>006000</td>
<td>0003</td>
</tr>
<tr>
<td>006002</td>
<td>FD FD</td>
</tr>
<tr>
<td>004000</td>
<td>3E7C 7000</td>
</tr>
<tr>
<td>004004</td>
<td>3038 6002</td>
</tr>
<tr>
<td>004008</td>
<td>6100 000C</td>
</tr>
<tr>
<td>00400C</td>
<td>31C0 6002</td>
</tr>
<tr>
<td>004010</td>
<td>3E3C 00E4</td>
</tr>
<tr>
<td>004014</td>
<td>4E4E</td>
</tr>
<tr>
<td>004016</td>
<td>3F00</td>
</tr>
<tr>
<td>004018</td>
<td>4AFC</td>
</tr>
<tr>
<td>00401A</td>
<td>6600 0008</td>
</tr>
<tr>
<td>00401E</td>
<td>301F</td>
</tr>
<tr>
<td>004020</td>
<td>4AFC 0006</td>
</tr>
<tr>
<td>004024</td>
<td>61F0</td>
</tr>
<tr>
<td>004026</td>
<td>C0DF</td>
</tr>
<tr>
<td>004028</td>
<td>4E75</td>
</tr>
</tbody>
</table>

(c) Corrupted program
Breakpoint handler

```assembly
breakh
    movem.l d0/a0-a1,reg_area  ; save registers
move.l 2(sp),a0             ; a0 gets address of illegal
                     ; instruction

move.w (a0),d0             ; d0.w gets first word of
                     ; illegal instruction

cmp.w #4af0,d0             ; if not #4af0 give a message
bne message1               ; and quit

jsr search_breakpoint_tab  ; if address of breakpoint
                     ; is found in breakpoint
                     ; table return the address
                     ; of where it was found in al
                     ; set cc
                     ; if zero has been returned
                     ; give a message and quit

move.l a1,d0               ; restore first word of
beq message2               ; illegal instruction
                     ;
move.w 4(a1),(a0)          ; save monitor's trace
                     ; handler's address
move.l #traceh,36         ; store trap handler's address
                     ; set trace bit of "old" SR
or.b #80,(sp)             ; save address of illegal
move.l a0,save            ; instruction
movem.l reg_area,d0/a0-a1  ; restore registers
rte                      ; return and execute
                     ; "corrected" instruction

traceh
    jsr trace_handler
    and.b #78,(sp)          ; dumps, etc.
move.w #4af0,save         ; reset trace bit
move.l temp,36            ; make the instruction
rte                      ; illegal again!
                     ; restore trap vector

reg_area ds.l 3
save   ds.l 1
temp   ds.l 1
```
; This code segment demonstrates installing a bus error handler to
; print a simple error message and terminate a program.

Stop equ 228
Output equ 243
org $2000

BusHndlrl equ *

move.w (sp)+,FnCode ; bus error handler
move.l (sp)+,BadAddr ; save function
move.w (sp)+,Instruction ; code info
add #6,sp ; save the address
lea ErrorStr,a5 ; offending instruction
movea #StrEnd,a6 ; delete stacked sr
move.l #Output,d7 ; and pc
trap #14 ; prepare to print
move.l #Stop,d7 ; message
trap #14 ; go to tutor exit

FnCode dc.w 0
BadAddr dc.l 0
Instruction dc.w 0
ErrorStr dc.b 'Bus Error detected'
StrEnd dc.b 0

; User program starts here

org $3000

Init equ *

move.l #BusHndlrl,$0008 ; set up bus
move.w $8FFFF,D0 ; error handler
move.w $0000,D0 ; instrn will

end