4. Interrupt

Hardware interrupt:

If in an external device (for example I/O interface) a predefined event occurs, this device issues an interrupt request to the CPU.

Design Issues:

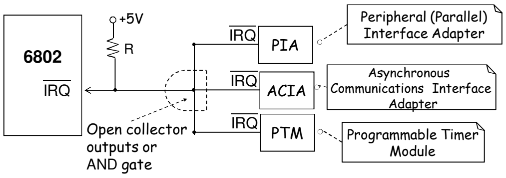
- **Source:** There can be multiple interrupt sources. How does the processor determine which device issued the interrupt request?
- **Priority:** In case of simultaneous, multiple interrupt requests, how does the processor decide which one to process?
- **Start address of the ISR:** If an interrupt request of a device is accepted, how does the processor determine the starting address of the interrupt service routine (ISR) (or interrupt handler) related to the requesting device? Autovectored, vectored.

4.1 Interrupt Systems of the CPUs

a) The CPU provides only an Interrupt Request (IRQ) input.

It does not provide an acknowledgement output.

Example: MC6802

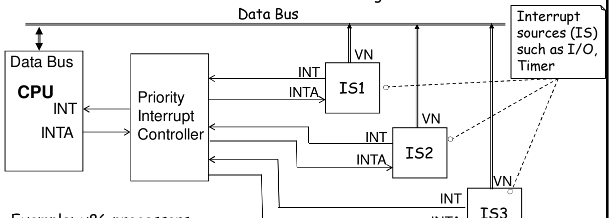


- There is only one interrupt service routine (ISR).
- The source of the interrupt is identified by polling at the beginning of the ISR.
- The order in which the interrupt sources are polled determines the priority.
- To make the source device remove its interrupt request (acknowledgement), a software action is necessary (reading the port of the source , writing to the control register etc.).

---

b) The CPU provides an **Interrupt Request** (INT) input an **Interrupt Acknowledge** (INTA) output. CPU uses **vectored interrupts**.

- Interrupt sources are connected to the CPU over an interrupt priority controller.
- In the case of multiple interrupt requests the controller decides which device will get the acknowledgement signal (INTA).
- The source device that receives the INTA places a word (vector number - VN) on the data bus.
- The CPU uses the vector to determine the starting address of the related ISR.



Example: x86 processors

4.2 Vector Address:

The CPU keeps the information about the interrupt handlers in a **vector table**.

This table is used to associate an interrupt request with a specific ISR.

There are two techniques:

1. The table contains the starting addresses of the ISRs.

The interrupt source gives the CPU an index to table as the vector number.

Using this index the CPU accesses an entry in the table, gets the starting address of the ISR and writes it to the program counter (PC).

Example: MC 68000.

2. The table itself contains executable code, namely the ISR.

In practice, an interrupt handler cannot be stored entirely inside the interrupt vector table.

Therefore, the code at each entry is "**JMP ISR_address**" where ISR_address is the address of the interrupt service routine.

Vectored and Autovectored interrupts:

In the **vectored interrupt** technique, external interrupt source (for example I/O interface) supplies the vector address to the processor.

In systems with **autovectored interrupts** external devices don't supply the vector number in response to the interrupt acknowledge.

Each interrupt input (or level) of the processor has a fixed and predetermined vector number (a specific row in the vector table).

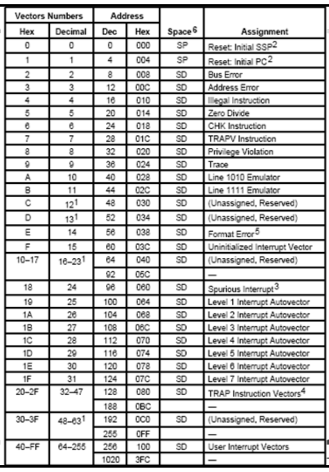
Remember in 6802 NMI, IRQ, SWI have their own rows in the table.

The processor "knows" where to find the address of the ISR if an interrupt is issued to the NMI pin.

MC68000 can use both techniques: vectored and autovectored interrupts

---

Interrupt Vector Table of the MC 68000:

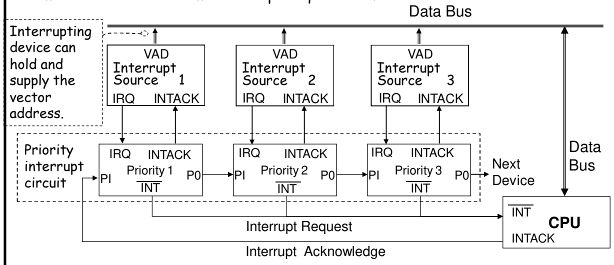


Vectors Numbers		Address			
Hex	Decimal	Dec	Hex	Space[6]	Assignment
0	0	0	000	SP	Reset: Initial SSP[2]
1	1	4	004	SP	Reset: Initial PC[2]
2	2	8	008	SD	Bus Error
3	3	12	00C	SD	Address Error
4	4	16	010	SD	Illegal Instruction
5	5	20	014	SD	Zero Divide
6	6	24	018	SD	CHK Instruction
7	7	28	01C	SD	TRAPV Instruction
8	8	32	020	SD	Privilege Violation
9	9	36	024	SD	Trace
A	10	40	028	SD	Line 1010 Emulator
B	11	44	02C	SD	Line 1111 Emulator
C	12[1]	48	030	SD	(Unassigned, Reserved)
D	13[1]	52	034	SD	(Unassigned, Reserved)
E	14	56	038	SD	Format Error[5]
F	15	60	03C	SD	Uninitialized Interrupt Vector
10–17	16–23[1]	64	040	SD	(Unassigned, Reserved)
		92	05C		—
18	24	96	060	SD	Spurious Interrupt[3]
19	25	100	064	SD	Level 1 Interrupt Autovector
1A	26	104	068	SD	Level 2 Interrupt Autovector
1B	27	108	06C	SD	Level 3 Interrupt Autovector
1C	28	112	070	SD	Level 4 Interrupt Autovector
1D	29	116	074	SD	Level 5 Interrupt Autovector
1E	30	120	078	SD	Level 6 Interrupt Autovector
1F	31	124	07C	SD	Level 7 Interrupt Autovector
20–2F	32–47	128	080	SD	TRAP Instruction Vectors[4]
		188	0BC		—
30–3F	48–63[1]	192	0C0	SD	(Unassigned, Reserved)
		255	0FF		—
40–FF	64–255	256	100	SD	User Interrupt Vectors
		1020	3FC		—

4.3 Priority Interrupt Hardware

a) Serial Priority Interrupt Hardware (Daisy Chain) 1

- Priority interrupt circuit (chain) consists of links with different priority levels.
- Each interrupt source is connected to one of these links (Interrupt Request - IRQ).
- The first link (at head of the chain) has the highest priority (priority 1) level.
- All modules share the same interrupt request line.



The function of the Priority chain:

- The interrupt acknowledge line is daisy chained through the modules.
- When the CPU accepts an interrupt, it sends out an interrupt acknowledge.
- The interrupt acknowledge signal from the CPU arrives to the first link in the chain (Priority in: PI).

a. If the device connected to this link has issued the request,

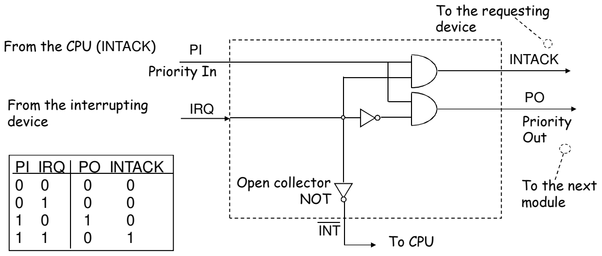
1. The module sends the INTACK signal to its device (interrupt source).
2. The device (interrupt source) puts its vector address (VAD) on the data bus.
3. The module negates (deactivates) its PO (Priority out) so that the next module cannot get the acknowledge signal.

b. If the device connected to this link has not issued the request,

1. The module asserts (activates) its PO (Priority out) so that the next module gets the right to check the request and supply the vector address.

- This mechanism is also called **hardware poll**.
- If the request of a device is accepted, this device deactivates its IRQ output.
- The device keeps its request (IRQ) active until it gets the acknowledgement.

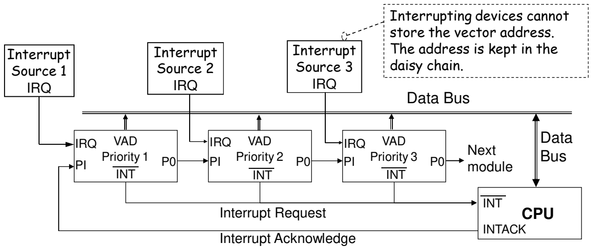
The structure of a stage of the daisy chain:



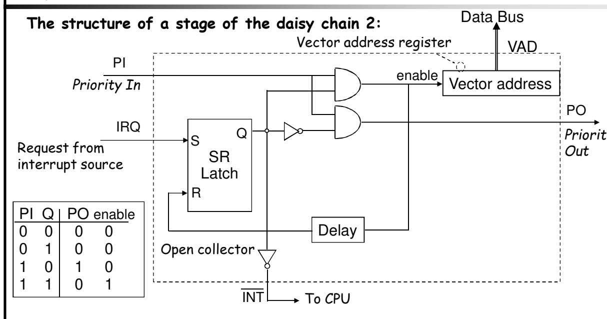
PI	IRQ	PO	INTACK
0	0	0	0
0	1	0	0
1	0	1	0
1	1	0	1

a) Serial Priority Interrupt Hardware (Daisy Chain) 2

- In the previous system (Daisy Chain 1) the interrupt sources can store their vector addresses and place them on data bus after receiving the INTACK signal.
- Some devices don't have this capability. Therefore; the vector address is kept in the daisy chain.
- Moreover, keeping the request active until it is accepted by the CPU can also be the responsibility of the controller if the device does not have this capability.



The structure of a stage of the daisy chain 2:

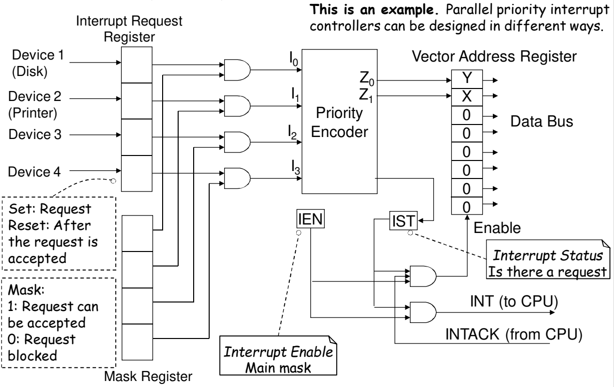


PI	Q	PO	enable
0	0	0	0
0	1	0	0
1	0	1	0
1	1	0	1

- The interrupt request from the device sets the SR latch.
- The interrupt acknowledge signal from the CPU arrives to the first link in the chain (Priority in: PI).
- If the device connected to this link has issued the request the module (not the device) places the vector address (VAD) on the data bus and resets the latch.

b) A Parallel Priority Interrupt Controller

This is an example. Parallel priority interrupt controllers can be designed in different ways.



Vector addresses of the devices in this system:

Device 1: 0000 0000
Device 2: 0000 0001
Device 3: 0000 0010
Device 4: 0000 0011

Truth table of the priority encoder:

Inputs				Outputs		
I_0	I_1	I_2	I_3	X = Z_1	Y = Z_0	IST
1	x	x	x	0	0	1
0	1	x	x	0	1	1
0	0	1	x	1	0	1
0	0	0	1	1	1	1
0	0	0	0	Φ	Φ	0

Logical expressions:

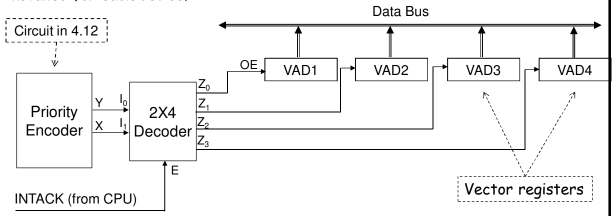
$X = Z_1 = I_0' I_1'$

$Y = Z_0 = I_0' I_1 + I_0' I_2'$

$(IST) = I_0 + I_1 + I_2 + I_3$

In the exemplary system (4.12) the vector addresses of the devices are predetermined and fixed (Device 1 (Disk): 0000 0000).

To make the system more flexible a separate vector address register can be installed for each device.



When the interrupt request of a device is accepted, the OE (Output enable) pin of the related register is activated.

The vector address registers are installed in the system so that the CPU can write to them.

The vector address can be initialized (or modified) by system programs.

4.4 Interrupt Processing: Necessary actions before and after the ISR

Before the ISR:

Remember; the CPU checks the interrupt requests after the execution cycle.

If a request is accepted, the CPU enters the interrupt cycle (Slide 1.18).

In the interrupt cycle the following actions are performed by the CPU.

These actions are internal operations of the CPU; they are **not** performed by a program.

SP ← SP-1	Stack pointer is decremented (depending on address length: 1, 2, 4)
M[SP] ← PC	Return address to stack
INTACK ← 1	Interrupt acknowledge
PC ← VAD	PC ← Vector address or PC ← Table [Vnum.] (from Vector table)
SP ← SP-1	
M[SP] ← SR	SR: Status register to stack
IEN ← 0	Other interrupts are masked (disabled). This bit is in SR (Status reg.)

Start address of the ISR

In the next fetch cycle the CPU continues with the first instruction of the ISR, because the PC includes its starting address (PC ← VAD).

Some CPUs also push internal registers to the stack. Some CPUs leave this job to the programmer.

Returning from the ISR:

Remember the Interrupt service routines are terminated by a special instruction; "return from interrupt" (RTI).

This instruction performs the following necessary operations to return from the ISR to the interrupted previous program.

SR ← M[SP]	Status register from stack (Remember IEN is in SR)
SP ← SP+1	(Stack pointer is incremented (depending on the length of SR: 1, 2)
PC ← M[SP]	(Return address)
SP ← SP+1	(Stack pointer is incremented (depending on address length: 1, 2, 4)

(If internal registers were also pushed to the stack automatically in the interrupt cycle, they pulled by the RTI.)

Note that the CPU enters the interrupt cycle **only before** starting the ISR.

Returning operations are performed by the last instruction (RTI) of the ISR.

Conclusion:

Interrupt processing operations are **time-consuming** (many memory accesses).

Therefore frequent interrupt requests can degrade the performance of a system.

For example, interrupt-driven I/O is not suitable for applications (file transfer) where I/O operations are performed very frequently.

Example: Interrupt-driven I/O

Problem:

Instruction cycle of a CPU has the following 5 states (cycles) with the given durations:

1. Instruction fetch: 60 ns, 2. Instruction Decode: 20 ns, 3. Operand fetch: 60 ns, 4. Execution: 30 ns, 5. Interrupt: 200 ns.

Housekeeping operations in the interrupt cycle (saving return address, taking the vector address, etc.) take 200 ns.

Interrupt-driven I/O technique is used to transfer 10 words from the I/O interface to the memory.

The interrupt service program transfers one word each time and it takes 500 ns (Housekeeping operations in the interrupt cycle are not included).

Assume that we start a clock (Time = 0) when the CPU begins to run the program.

The first interrupt request arrives from the I/O interface as the CPU is in the instruction fetch cycle for the first instruction (Time = 5ns).

a. When (Time =?) can be the first word transferred from the I/O interface to the memory? Why?

b. When (Time =?) will be all the 10 words transferred if the I/O interface is always ready to transfer?

Example: Interrupt-driven I/O (cont'd)

Solution:

Remember; interrupt requests are checked after the execution of the instruction. If there is a request and interrupts are enabled, the CPU enters the interrupt cycle.

The data is transferred in the ISR (Interrupt Service Routine).

RTI and related operations are included in the ISR.

a. First word:

Fetch + Decode + Operand + Execute + Housekeeping + ISR

Time = 60 + 20 + 60 + 30 + 200 + 500 = 870ns

b. 10 words:

One word is transferred in each ISR.

After the ISR, the CPU returns to the main program, runs one instruction, and enters the ISR again.

Time = 10 x 870 = 8700ns (Time-consuming. Overhead is large.)

4.5 Exceptions

Exceptions, are situations that are caused either by programming errors or by anomalous conditions.

In these cases, the processor stops executing the current code, begins running an **exception handling routine**, and then returns to the normal program flow.

Example: Exceptions in MC68000

External:

- Reset
- Bus Error (BERR)
- Interrupts; vectored, autovectored

Internal:

- Trace: If T bit in SR is "1", programs run step by step (for debugging).
- Address error : Word/long access attempt to odd addresses
- Software interrupt (TRAP 0 -15), TRAPV (Trap on overflow), CHK
- Illegal instruction: The op code does not exist.
- Instruction emulation (Instruction starting with $A=1010 and $F=1111)
- Privilege violation: Some instructions are only available in supervisor mode.
- Divide by zero

When the 68000 receives an exception, the following procedure is performed:

- SR → Temp (a copy of the Status Register SR is created.)
- S←1, T←0 (The CPU switches to *supervisor* mode. Trace is disabled.)
- PC (return address) saved to the supervisor stack.
- The copy of SR in Temp (S and T have their original values) saved on the supervisor stack using the SSP (supervisor stack pointer).
- The address of exception handler is obtained from the vector table.
- Data and address registers are not saved on the stack by the 68000.
  The programmer of the service routine must save only the used registers on the stack.

Returning from the exception:

- Service routines must end with the instruction RTE (Return from Exception).
- Status register SR is pulled from the stack.
- Return address is pulled from the stack.

In the case of RESET all these operations are not performed.

In some exceptions (BERR, interrupts) some additional operations are performed.

4.5.1 Privilege Modes

The 68K operates in one of two levels of privilege: the ***supervisor*** mode or the ***user*** mode.

The privilege mode determines which operations (instructions) are legal.

The mode is also indicated by the FCO (Function Codes Output) pins of the processor and optionally used by an external memory management circuit to control the accesses to certain memory locations (or devices) (Slides 3.23-24).

The mode is also used to choose between the supervisor stack pointer (SSP) and the user stack pointer (USP) in instruction references.

Supervisor mode:

The supervisor mode has the higher level of privilege.

The mode of the processor is determined by the S bit of the status register; S=1

All instructions can be executed in the supervisor mode.

User mode:

If the S bit of the status register is clear, the processor is in the user mode.

Most instructions execute identically in either mode. However, some instructions having important system effects are privileged. Example: STOP, RESET.

To ensure that a user program cannot enter the supervisor mode except in a controlled manner, the instructions that modify the entire status register are privileged.

Transition between privilege modes

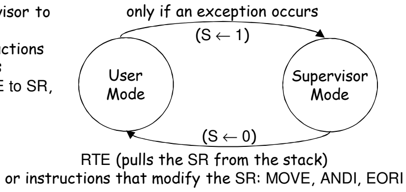
When the processor is in the user mode only exception processing can change the privilege mode.

Remember, during exception processing, the current state of the S bit of the status register is saved, and the S bit is set (S←1), putting the processor in the supervisor mode.

Therefore, exception service programs run in supervisor mode.

If the exception is occurred in the user mode, after the exception handler the processor switches to user mode again, because the RTE instruction pulls the original value of the SR (that includes the S bit) from the stack.

The transition from supervisor to user mode can also be accomplished by the instructions that can modify the status register SR, such as MOVE to SR, ANDI to SR, EORI to SR.



4.5.2 Bus Error (BERR) and Address Error:

A bus error exception occurs when the external logic asserts the BERR' (active low) input of the 68000. See slide 3.19 Avoiding Infinite Waiting.

An address error exception occurs when the processor attempts to access a word (16-bit) or long word (32-bit) operand or an instruction at an odd address.

An address error is similar to an internally generated bus error.

Unlike the interrupts the **current bus cycle is aborted**.

The current instruction is not finished (even not the bus cycle is not completed).

The current processor activity, whether instruction or exception processing, is terminated, and the processor immediately begins exception processing.
Exception processing for a bus error/address error follows the usual sequence of steps.
However additional information is saved on the supervisor stack.

If a bus error occurs during the exception processing for a bus error, an address error, or a reset, the processor halts, removes itself from the system bus (high impedance).

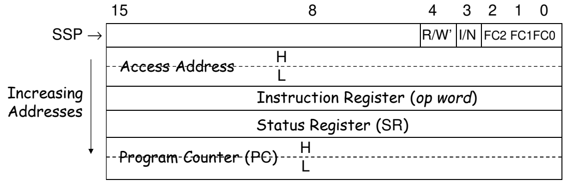
This halt simplifies the detection of a system failure and protects memory contents from erroneous accesses.

Only an external RESET operation can restart a halted processor.

Supervisor Stack Order for Bus or Address Error Exception

Because the processor is fetching the instruction or an operand when the error occurs, the context of the processor is more detailed.

To save more of this context, additional information is saved on the supervisor stack.



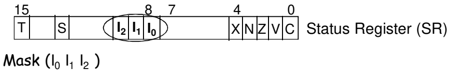
4.5.3 Interrupts in MC68000:

68000 has three interrupt request input-lines (IPL2, IPL1, IPL0).

The 3-bit value at these inputs indicates the level (1-7) of the interrupt request.

All lines negated indicates no interrupt request.

If the priority level of the pending interrupt is greater than the current processor priority (interrupt masks), the request is accepted, and the exception processing sequence is started.

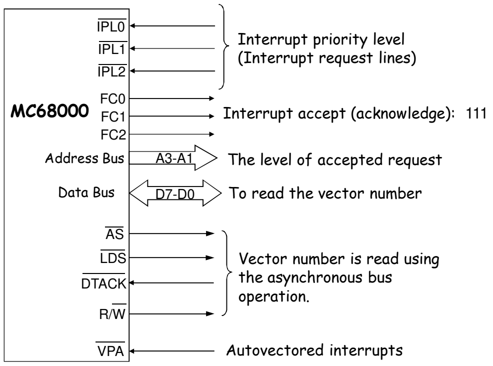


IPL2, IPL1, IPL0 > I2, I1, I0 interrupt is accepted.

IPL2, IPL1, IPL0 ≤ I2, I1, I0 interrupt is not accepted, execution continues with the next instruction.

Priority level 7 is a special case. Level 7 interrupts cannot be inhibited by the interrupt priority mask.
Level 7 provides "**nonmaskable interrupt**" capability.

Interrupt-related signals of the MC68000:



Interrupt priority level (Interrupt request lines)

Interrupt accept (acknowledge): 111

The level of accepted request

To read the vector number

Vector number is read using the asynchronous bus operation.

Autovectored interrupts

When the 68000 accepts an interrupt request, the following actions are performed:

- SR → Temp (a copy of SR is created.)
- S←1, T←0
- PC (return address) saved to the supervisor stack.
- The copy of SR in Temp (S and T have their original values) saved on the stack.
- I2, I1, I0 ← IPL2, IPL1, IPL0 (Mask ← Interrupt Level) The level of the interrupt being acknowledged is written to the masks. Hence, interrupt requests with lower or equal levels are disabled.
- FC2, FC1, FC0 ← 111 (Interrupt Acknowledge)
- A3, A2, A1 ← The level of the interrupt being acknowledged.

a) Vectored interrupts:

- The interrupting device places a vector number on the data bus and asserts DTACK' to acknowledge the cycle.
- 68000 reads the 8-bit vector number from data bus over D7-D0.
- The vector number gives the number of the row of the vector table where the starting address of the interrupt service routine is placed.
- As each row of the table is 4 bytes long, to calculate the address of the row the vector number is multiplied by 4. (See the table in 4.6)

When the 68000 accepts an interrupt request, the following actions are performed:

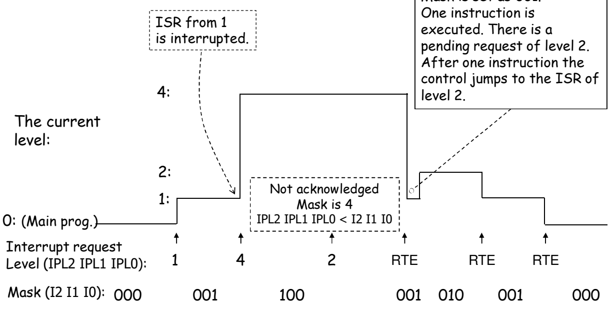
b) Autovectored interrupts:

- If the interrupting device asserts VPA' instead of DTACK', it means that this device will (can) not supply a vector number.
- In this case, 68000 obtains the starting address of the interrupt service routine from predetermined and fixed rows of the vector table.
- Each autovectored interrupt level has its own entry in the table: rows 25 -31 (See 4.6).

Responsibilities of the programmer:

- Registers that are used in the interrupt service routine must be saved on the stack.
- Before returning, same registers must be pulled from stack.
- For these operation (MOVEM - move multiple) instruction can be used.
- If it is necessary the programmer of the ISR can change the value of the interrupt masks. (Remember ISR runs in supervisor mode.)
- Hence the programmer can enable interrupts with lower levels or disable interrupts with higher levels except level 7.
- Returning from the ISR must be done with RTE (Return From Exception).

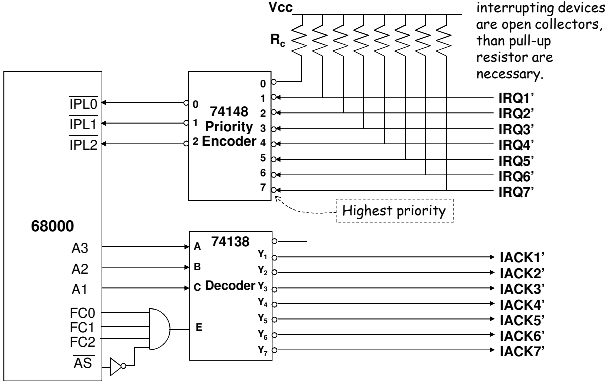
Nested Interrupts:

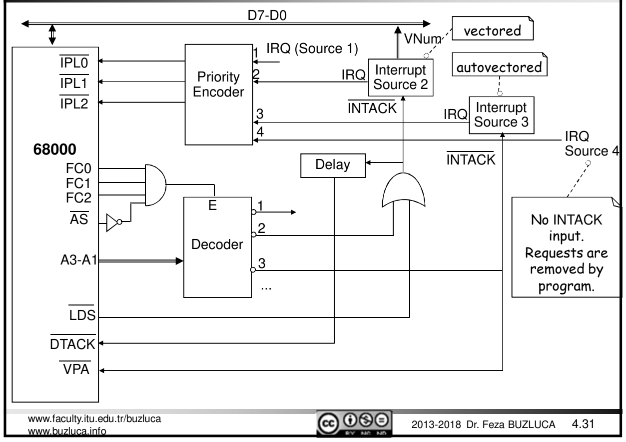


After the ISR of the 4. level the control returns to the ISR of the 1. level. Mask is set as 001. One instruction is executed. There is a pending request of level 2. After one instruction the control jumps to the ISR of level 2.

Generating the Interrupt Acknowledge signal:

If the output of interrupting devices are open collectors, than pull-up resistor are necessary.





4.5.4 Software Interrupts

In MC68000 there are 16 software interrupts which are called trap:

TRAP #0 – TRAP #15

These instructions generate internal interrupts and start exception processing.

Each TRAP has its own entry in the vector table.

The system programmers write necessary system programs (for example for using system resources such as the PIA), and install them as exception service routines in the system.

The authors of the user programs can call these routines to access the system resources by executing the TRAP instructions.

Since the exception service programs run in supervisor mode, using these routines the user can access some system resources in a controlled manner.

Normally, a user may not write directly to a register of the PIA (if it is protected in user mode), but using a service routine the user can send data over the PIA.

4.5.5 Instruction Emulation (Unimplemented instructions)

In the instruction set of the MC68000 there is not any instruction (in machine language) starting with $A (1010) and $F (1111).

Opcodes starting with bit patterns equaling 1010 (Line A) and 1111(Line F) are distinguished as **unimplemented instructions**, and separate exception vectors are assigned to these patterns to permit efficient emulation.

System designers (system programmers) can create their own instructions starting with these opcodes and place them in a program with other instructions.

After such an instruction is fetched and decoding is attempted, the processor determines that execution of an unimplemented instruction is being attempted and starts exception processing.

The exception service routine related to the instruction is written by the system programmer. This routine performs the required operation.

The address in PC that is saved on the stack as a return address before starting the exception's service routine is the address of the unimplemented instruction.

Example:

In a MC68000-based system using the instruction emulation capability the following instructions will be implemented.

- ADD.B address1, address2, address3 (address3)←(address1)+(address2)

This instruction adds two **8**-bit integers in address1 and address2, then writes the result to address3. Addresses are 32 bits.

- ADD.W address1, address2, address3 (address3)←(address1)+(address2)

This instruction performs the same operation on **16**-bit integers.

Solution:

First, the structure (format) of the instructions must be defined.

An exemplary structure:

ADD.B address1,address2,address3 $F000 address1,address2,address3

ADD.W address1,address2,address3 $F001 address1,address2,address3

The last bit of the operation word (*Op word*) is used to indicate the size. 0 : B , 1 : W

ADD.B : 1111 0000 0000 0000 = $F000

ADD.W: 1111 0000 0000 0001 = $F001

ADD.B :

We must write an exception service routine that performs operations related to the instructions.

Before we write service routine, we can implement a main program that can be used to test the service routine:

```
main  lea     stack,a7                          // Stack pointer initial address
      adda.l  #40,a7                            // Stack grows to decreasing addresses
      move.l  #service,($2C) <                  // Service routine starting address to table
      dc.w    $f000,0,$1000,0,$1100,0,$1200          //ADD.B $1000,$1100,$1200
      dc.w    $f001,0,$2000,0,$2100,0,$2200          //ADD.W $2000,$2100,$2200
      ....
      org     $500
stack ds.b    40                                     // Memory allocation for stack
```

Vector address of the Line F exception is ($2C) in the vector table (11th row).

The starting address of the service routine must be written to this entry.

Remember at the beginning of the service routine the used registers must be saved on the stack.

Which register must be saved can be determined after the service program is completely written.

```
service  movem.l d0/a0-a3,-(a7)       D0, A0, A1, A2, A3 to stack
```

The processor saves the SR and PC on the stack.

At the beginning of the service routine 5 registers are saved by the program.

Now the picture of the stack is as shown on the right:

```
service movem.l  d0/a0-a3,-(a7)
        movea.l  22(a7),a0     PC → a0 points to the instruction
        move.w   (a0)+,d0      First 16 bits of the instr OpCode ->d0
        movea.l  (a0)+,a1      Address1 → a1         Instruction
        movea.l  (a0)+,a2      Address2 → a2         Fetch
        movea.l  (a0)+,a3      Address3 → a3
        tst.b    d0            B/W?
        bne      word                     Instruction decoding
        move.b   (a1),d0       Byte operations
        add.b    (a2),d0                         Operand Fetch
        move.b   d0,(a3)                         and
        bra      ret                             Instruction
word    move.w   (a1),d0        Word operations  Execution
        add.w    (a2),d0
        move.w   d0,(a3)
ret     move.l   a0,22(a7)       PC in the stack is updated. <
        movem.l  (a7)+,d0/a0-a3
        rte
```

PC in the stack points to the next instruction.