Real-time Operating Systems and Systems Programming

IO, Interrupts, Getting to know hardware
Lecture 4

ARVUTITEADUSE INSTITUUT
DEPARTMENT OF COMPUTER SCIENCE

1 year	32 nHz	year number rollover
6 months	64nHz	GMT ↔ BST changeover
8hr	30μHz	AGA coal stove sycle time
10s	0.1Hz	photocopier page printing
1s	1Hz	time-of-day rate
300ms	3Hz	human typing speed
300ms		human reaction time
150ms	7Hz	mechanical switch bounce time
15ms	70Hz	motor car engine speed
	260Hz	middle C
	440Hz	concert pitch A
1ms	1kHz	serial line data rate
125μs	8kHz	digitized speech, telephone quality
64μs	15.6kHz	TV line rate
50μs		Mc68000 interrupt latency
0.5μz	2Mhz	Mc68000 instruction rate
0.075μs	13.5MHz	Video data rate
0.050μs		semiconductor RAM access time
0.01μs	100MHz	Ethernet data rate
10ns	100MHz	memory cycle, PC motherboard
2.5ns	400MHz	logic gate delay
555ps	1.8GHz	cellular telephone transmission
500ps	2GHz	single instruction issue, Pentium IV
0.3ps	3THz	infrared radiation
16fs	600THz	visible light

Operating systems

- Interface between hardware and software
- Provide services for applications
- Provide an abstraction layer for hardware

What services?

- Processes
- Multitasking
- Interrupts
- Memory management
  - Virtual memory
- Protected/supervisor mode
- Disk & Files
- Booting the computer
- Device drivers
- Networking
- Users / authentication
- Graphical UI

What applies for Real-time?

Usually not included in RTOS

- Paged & swappable virtual memory management
- Disk filing system
- Full networking facilities
- Intertask security
- Multi-user support
- GUI

More power (and responsibility)

- Interrupts can be masked
  - Can be used only if max. int. latency (by specification) longer than longest critical section path
- Memory allocation
  - Fixed-size blocks
  - Re-entrant core libraries (allocation on stack)

Other services

- HW initialization
- Real-time clock management
- Critical resource protection
- Intertask communication
- Intertask synchronization
- I/O management
- Multiple interrupt servicing
- Memory allocation and recovery
- Assistance for debugging

POSIX

- POSIX (**P**ortable **O**perating **S**ystem **I**nterface [for Uni**X**])
- Standard for Unix, defines core specifications- command-line, shell, some programs, basic IO.Threading API.

Posix IO

- Program has two inputs:
  - Command line arguments to main()
  - Standard input (keyboard connected to file by default)
- Two outputs
  - Standard output (connected to terminal by default)
  - Standard error (connecterd to terminal by default)
- Standard streams can be redirected

Dealing with standard streams

- Redirection done by piping
  - ./myprogram < inputfile > outputfile
  - ls > outputfile.txt
  - ls | more
- Ending keyboard input:
  - Pressing Ctrl + D on terminal signals EOF (^D)

Hardware

I/O for RT Systems

- Can be complex
  - Desktop computing hides the fact successfully
- Need to understand
  - Port address mappings
  - Register functionalities

Hardware access

- Done by accessing HW ports & registers
  - Memory mapped
  - I/O mapped

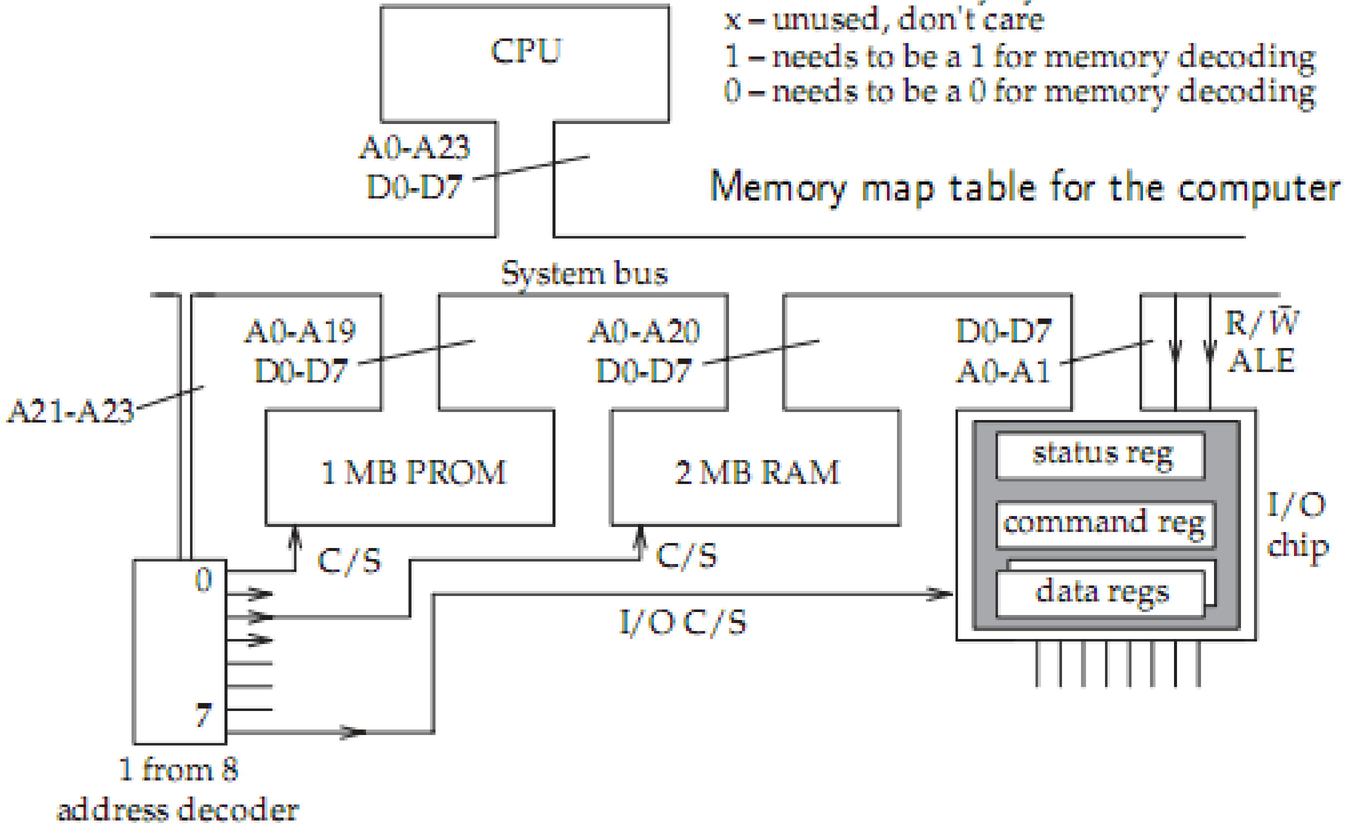
Memory mapped

- I/O registers behave like memory locations

Device	Size	Address pins	24 bit address bus	Address range
PROM1	1 MB	20	000x ++++ ++++ ++++ ++++ ++++	00 0000–0F FFFF
RAM1	2 MB	21	001+ ++++ ++++ ++++ ++++ ++++	20 0000–3F FFFF
RAM2	2 MB	21	010+ ++++ ++++ ++++ ++++ ++++	40 0000–5F FFFF
RAM3	2 MB	21	011+ ++++ ++++ ++++ ++++ ++++	60 0000–7F FFFF
I/O	4 B	2	111x xxxx xxxx xxxx xxxx xx++	E0 0000–E0 0003
			Aliases {	E0 0004–E0 0007
				E0 0008–E0 000B
				E0 000C–E0 000F
				. . .

+ – used internally by device
x – unused, don't care
1 – needs to be a 1 for memory decoding
0 – needs to be a 0 for memory decoding

Memory map table for the computer system

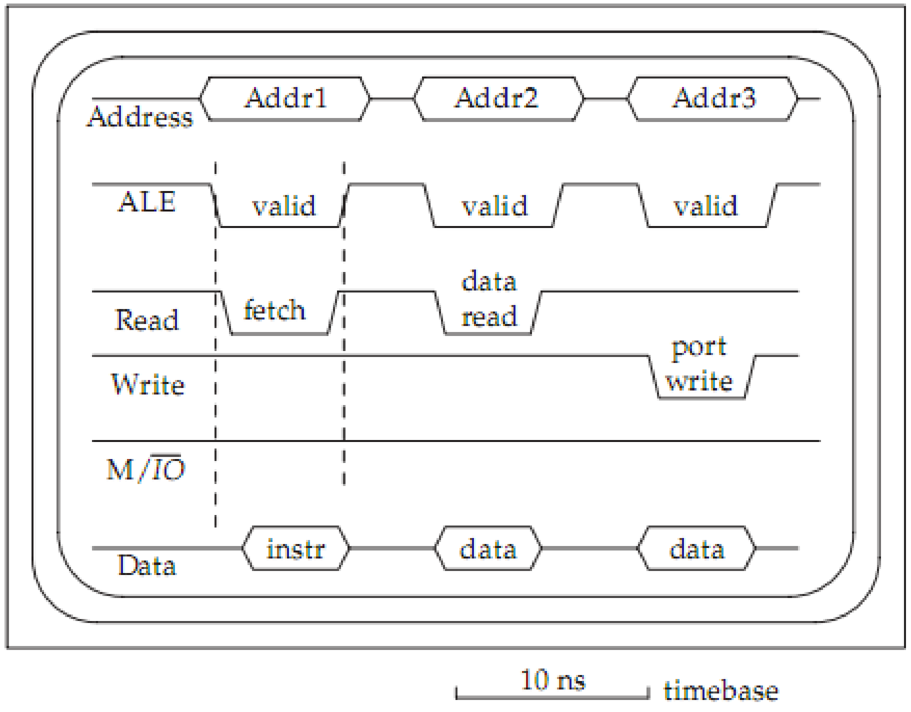


Memory-mapped I/O, showing the address decoding for a 24 bit CPU

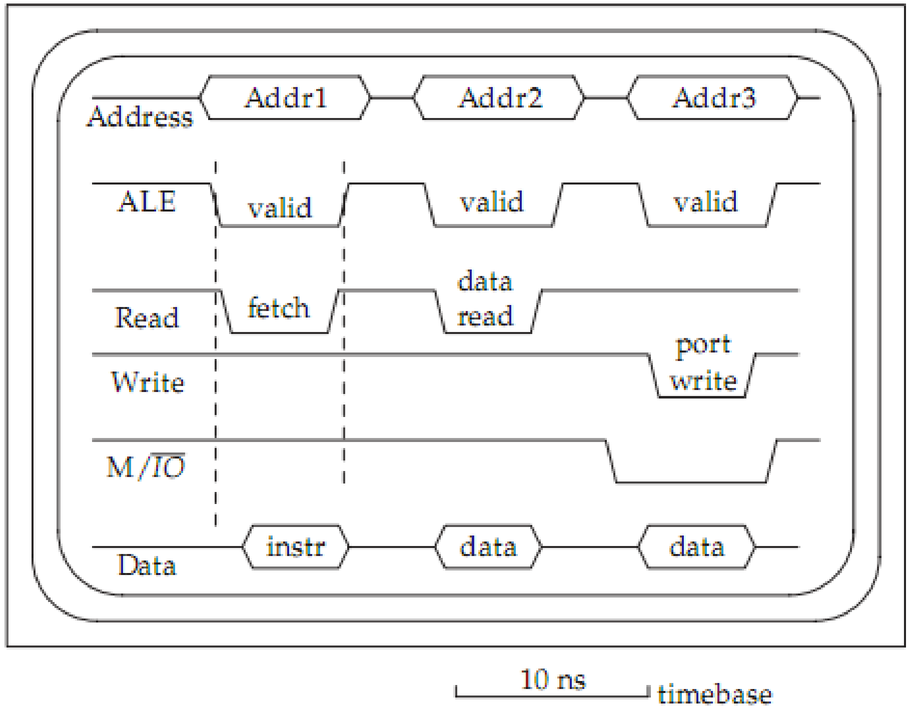
IO Mapped

- More bus control lines, extra instructions
- Independent address space for I/O ports
- Intel (IN & OUT instructions)
- Better caching: we need to read “raw data” for I/O
- C lang extensions: inb() outb() functions.

Comparison



Bus activity while accessing memory-mapped ports



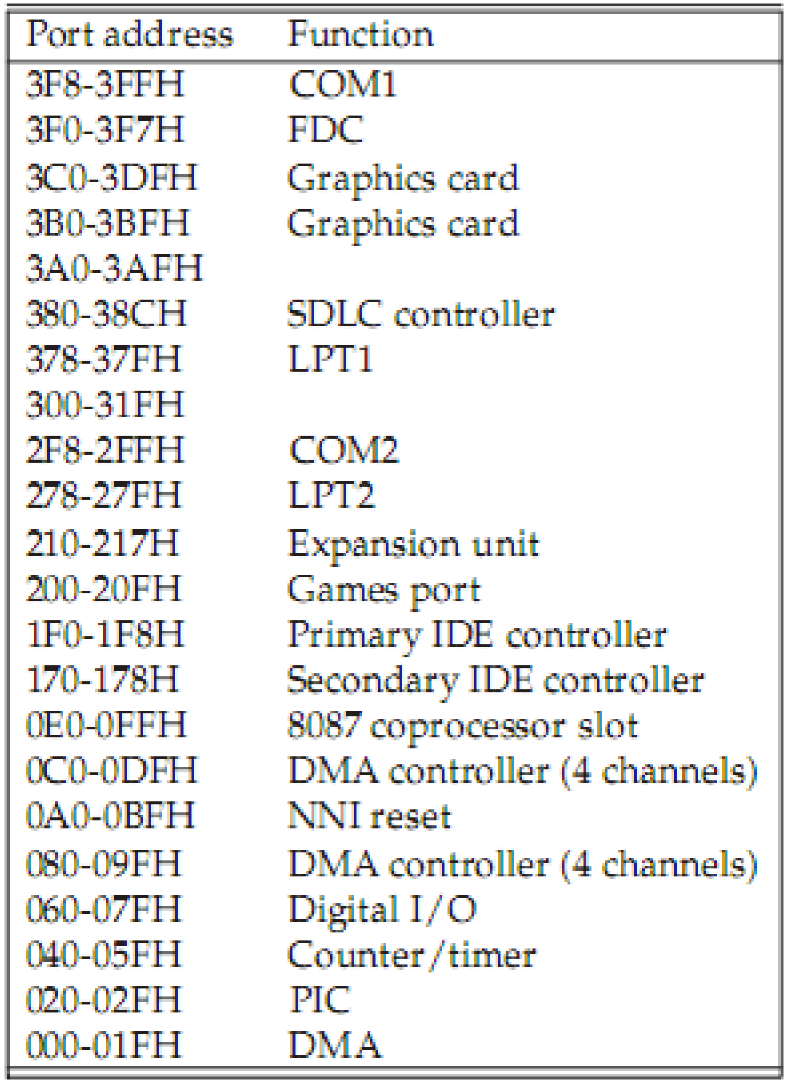
Bus activity while accessing I/O-mapped ports

Programmers view of ports

- For direct I/O:
  - Base address of I/O chip
  - Memory map and function of its registers

Need to Identify:
Command
Status
Data

PC I/O mapped port addresses



Port address	Function
3F8-3FFH	COM1
3F0-3F7H	FDC
3C0-3DFH	Graphics card
3B0-3BFH	Graphics card
3A0-3AFH	
380-38CH	SDLC controller
378-37FH	LPT1
300-31FH	
2F8-2FFH	COM2
278-27FH	LPT2
210-217H	Expansion unit
200-20FH	Games port
1F0-1F8H	Primary IDE controller
170-178H	Secondary IDE controller
0E0-0FFH	8087 coprocessor slot
0C0-0DFH	DMA controller (4 channels)
0A0-0BFH	NNI reset
080-09FH	DMA controller (4 channels)
060-07FH	Digital I/O
040-05FH	Counter/timer
020-02FH	PIC
000-01FH	DMA

A listing of PC I/O-mapped port addresses with standard function

Port polling

- Poll until data arrives
- Problem: CPU fast, devices slow
- Dedicated (spin) vs intermittent (timed) polling

How Unix solves the problem?

- The analogous problem being “how to read files without the program being blocked”.

Blocking & nonblocking

- IO operations wait until complete: blocking
- Simple: read only when data waiting (kbhit() - DOS/Win)
- Possible to turn off blocking & buffering for keyboard
- fd = open("/dev/ttyS0", O_RDWR | O_NOCTTY | O_NONBLOCK);
- Ioctl() & fcntl()

Blocking

- Device blocking often necessary for fair scheduling
- Threading possible
- select() function for multiple sockets.

I/O access permissions

- Accessing data belonging to another task
- Accessing kernel information
- Both need root permissions
- Solution: Setuid mechanism
  - passwd ps etc

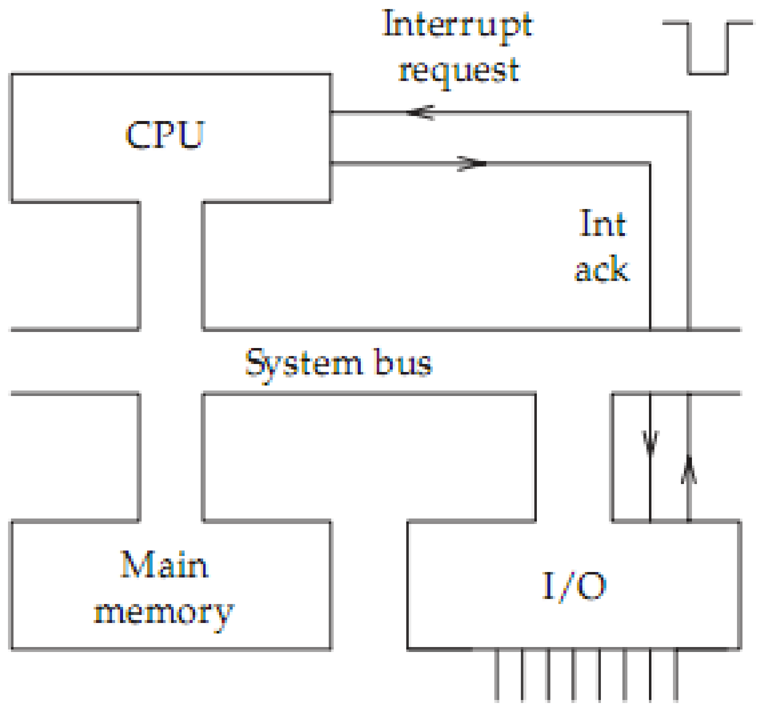
Interrupts

- Interrupt method good for occasional attention
- Requires hardware support, quite common

CPU level

- On every instruction, interrupt line is checked
- On interrupt, selected service routine executed after saving the instruction pointer
- Gets restored afterwards.
- Response in 10μs

System diagram

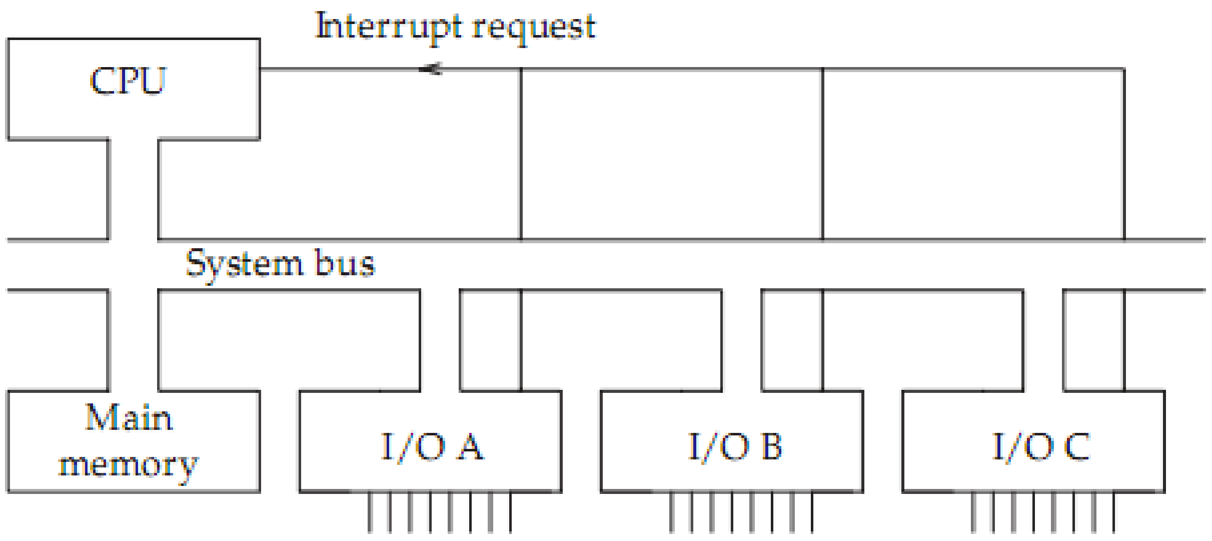


Extension: Exception processing

- Exception: Interrupt may be generated internally
  - CPU error condition
  - Memory access violation
- TRAP instructions from software

Source detection

- Often only one interrupt line
- How to find the source?



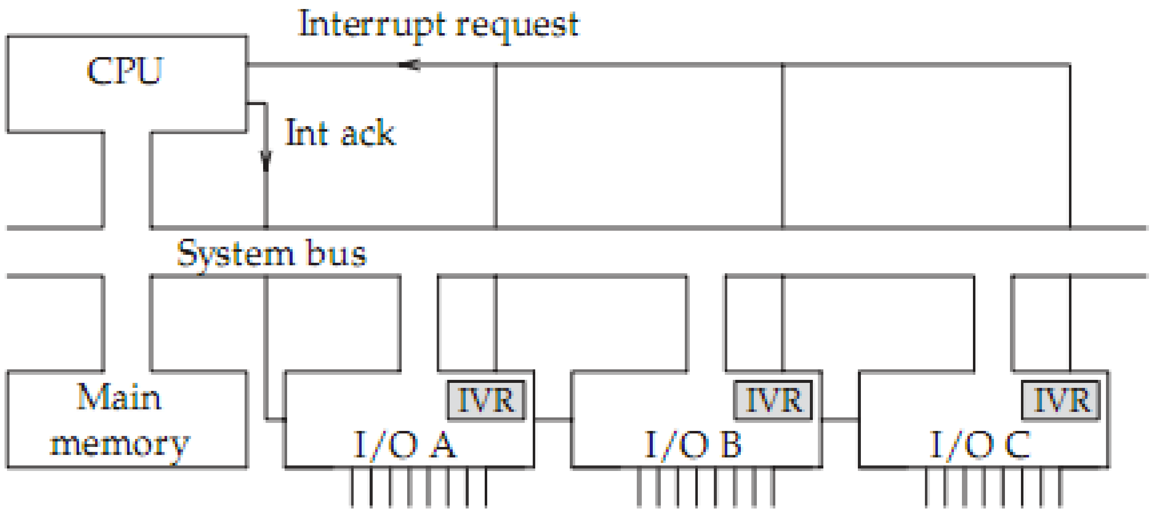
Connection of many interrupting devices to the CPU

Polling

- Slow since all devices must be polled individually
- Does not require extra hardware
- Adequate for small number of devices

Vector interrupts

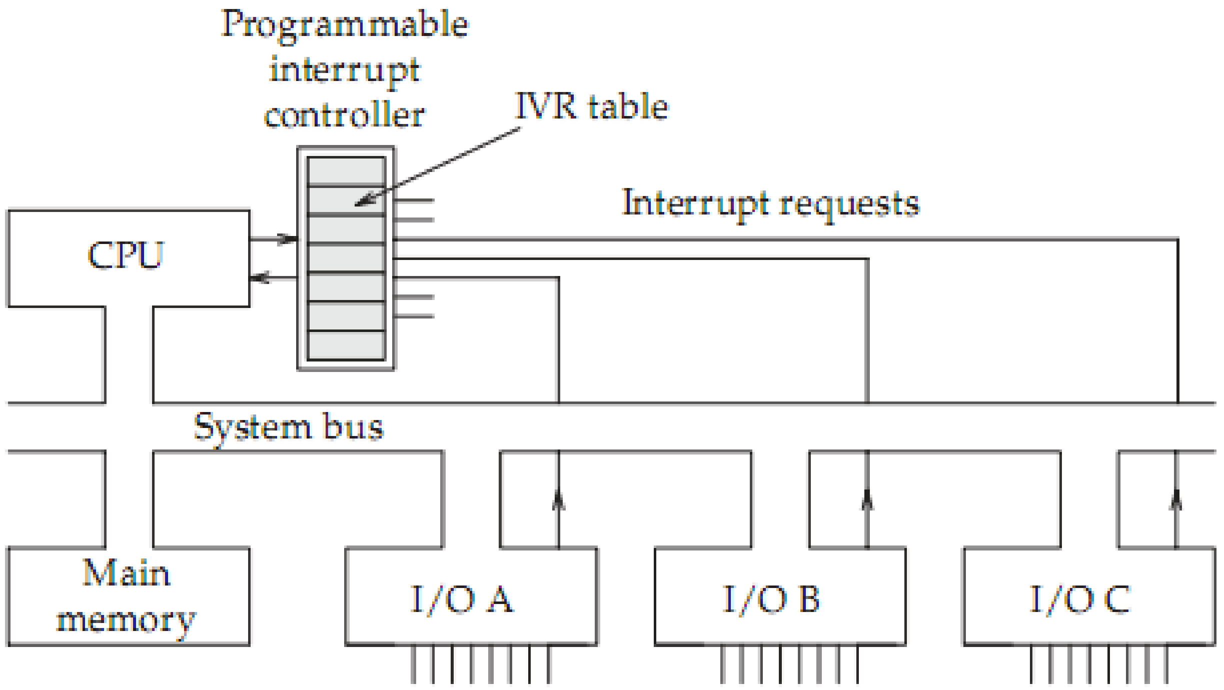
- Interrupt Vector Registers (IVR) in devices
- Motorola Mc68000 family



Vectored interrupts using a daisy-chain prioritization line

PIC interrupts

- Needs Programmable Interrupt Controller (PIC)
- PC method. Centralized prioritizing encoder.



Vectored interrupts using PIC hardware

Actions

- Interrupt
- CPU saves program counter (PC) & CPU status register to stack
- Entry address for Inter. Service Routine (ISR) from Interrupt Vector Table (IVT), written to PC
- ISR starts

ISR

- Store register contents to stack
- Verify source (test device flag for example)
- Remove cause to prevent further interruption
- Reinitialize device?
- …
- POP saved registers from stack, RTE instruction to restore Instruction Pointer & status

PC interrupt structure

- PIC lets through a single IRQ from most urgent device
- Interrupts can be disabled with STI & CLI instr.
- Source: PIC sends 8-bit vector to IVT which stores ISR-s
- IRQ 0 highest, IRQ 15 lowest

PC interrupts

- IRQ0 – system timer for ticks
- IRQ1 – keyboard
- IRQ2 – cascaded second PIC for IRQ8-15
- IRQ3 – COM2 port, often for modems
- IRQ4 – COM1/mouse
- IRQ5 – LPT2, often soundcard
- IRQ6 – floppy
- IRQ7 – LPT1

Interrupt priorities

- How to handle simultaneous interrupts
- Priorities handled in hardware
- Alternative: Deferred interrupt processing
- ISRs split in two. Small immediate service code and larger deferred portion.
- Queues for later processing
- Used on Windows