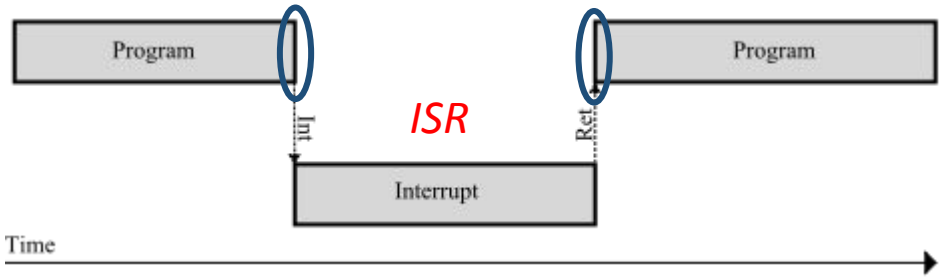
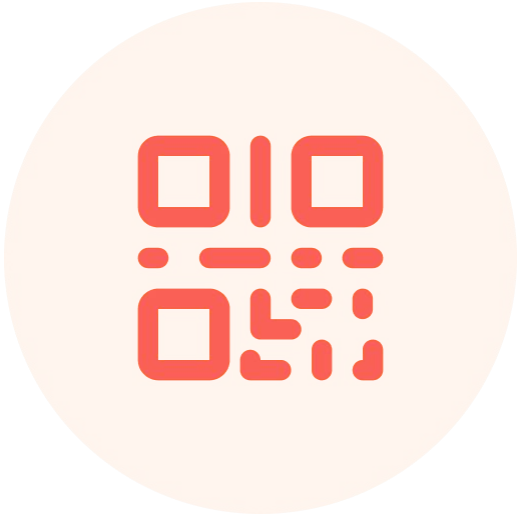
Interrupt

- Normal execution of a program may be preempted if some device requires urgent service.
- The device raises an *interrupt* signal, which is a request for service by the processor. The processor provides the requested service by executing a program called an *interrupt-service routine (ISR)*.
- Its state must be saved in the memory before servicing the interrupt request. The contents of the PC, the contents of the general-purpose registers, and some control information are saved,generally.
- When the ISR is completed, the state of the processor is restored.



slido



Join at slido.com
#736698

ⓘ Start presenting to display the joining instructions on this slide.

Microprocessors

Tuba Ayhan

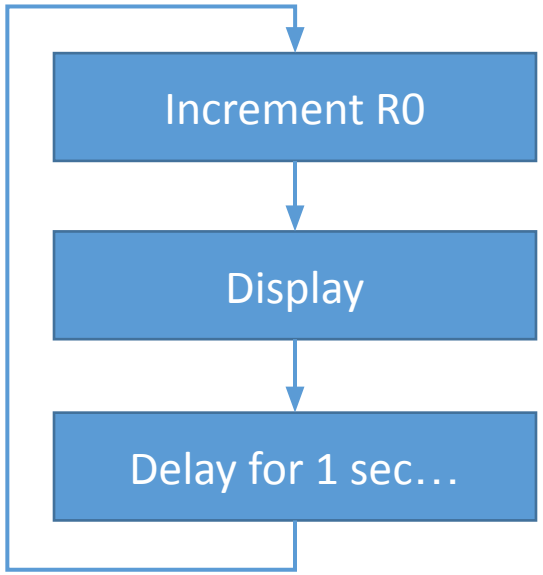
MEF University

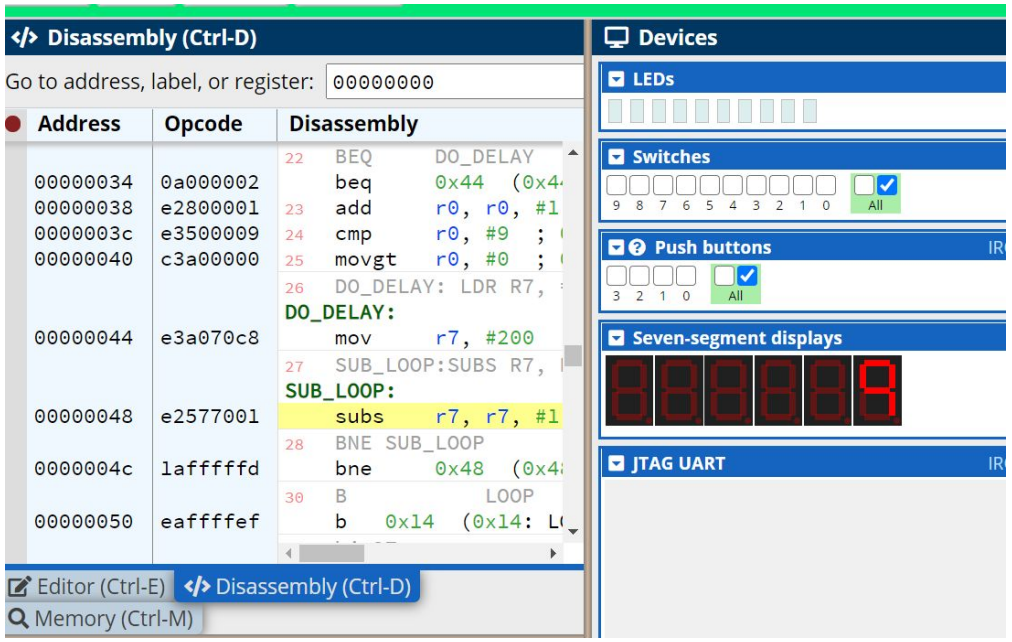
Interrupts

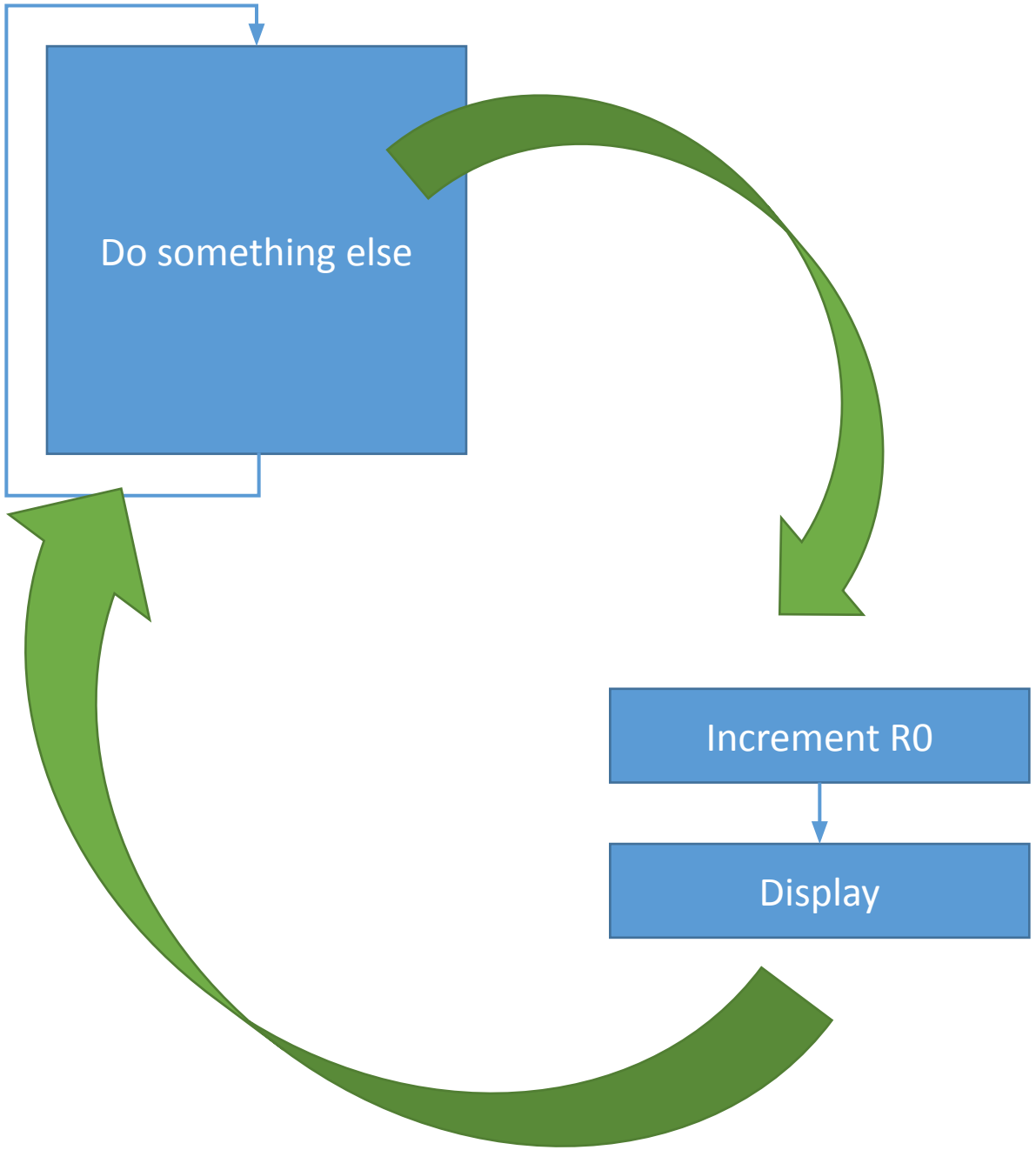
Computer Organization and Embedded Systems, Hamacher et. al

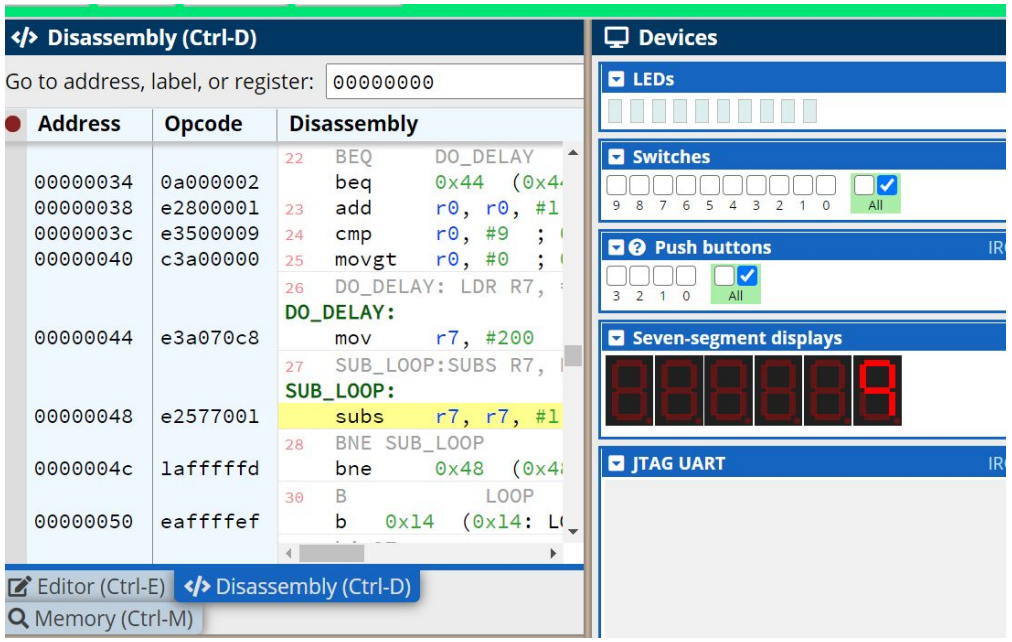
NOTE: Following section contains general interrupt concept.

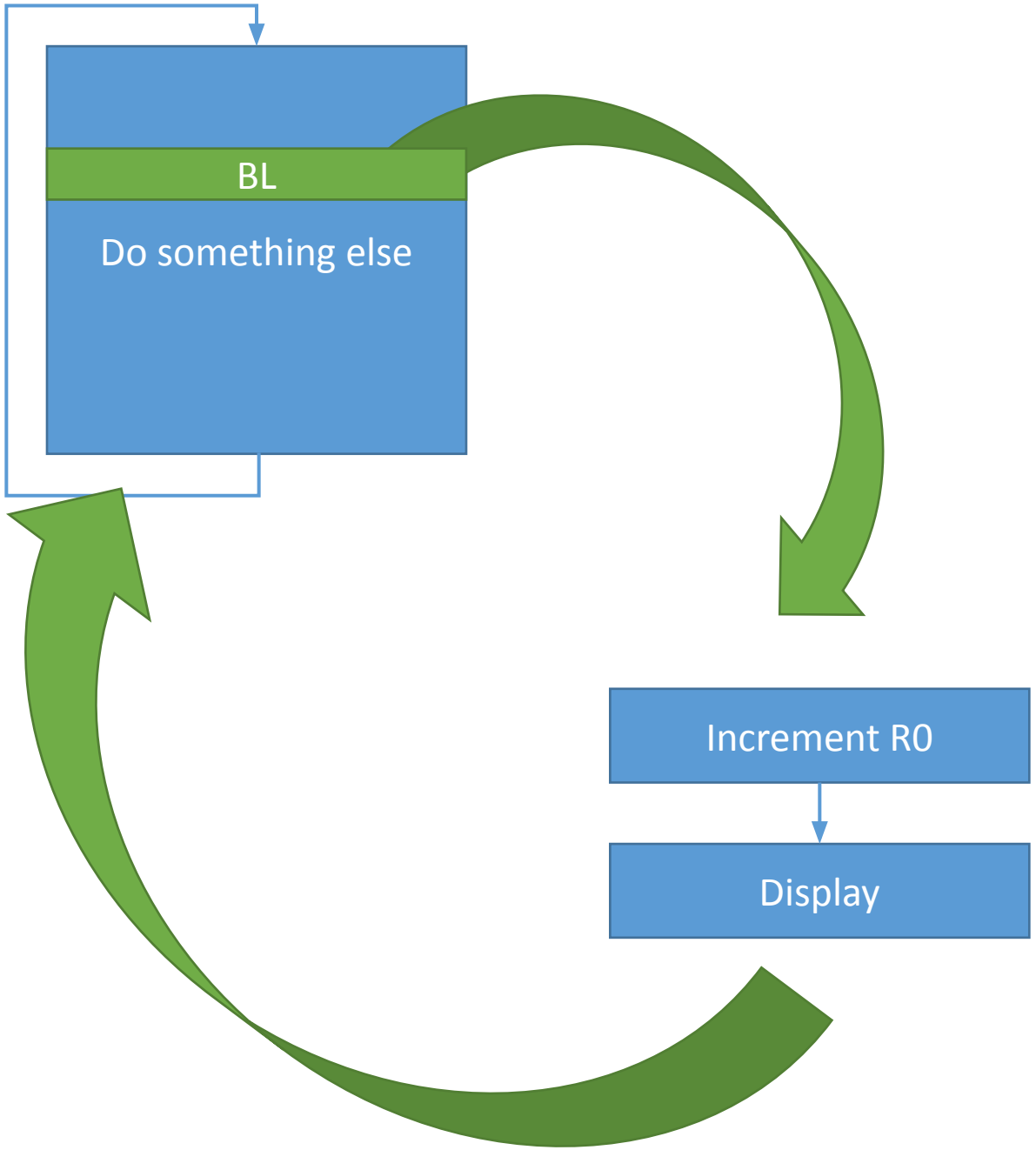
Interrupt handling in ARM will be covered in the next section.

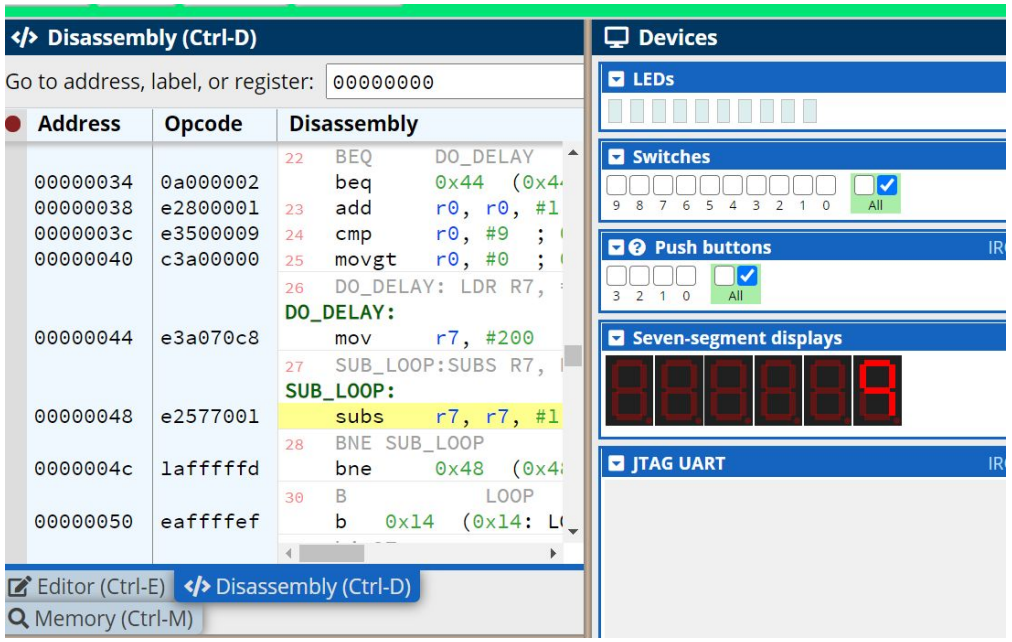

Increment R0
Display
Delay for 1 sec…

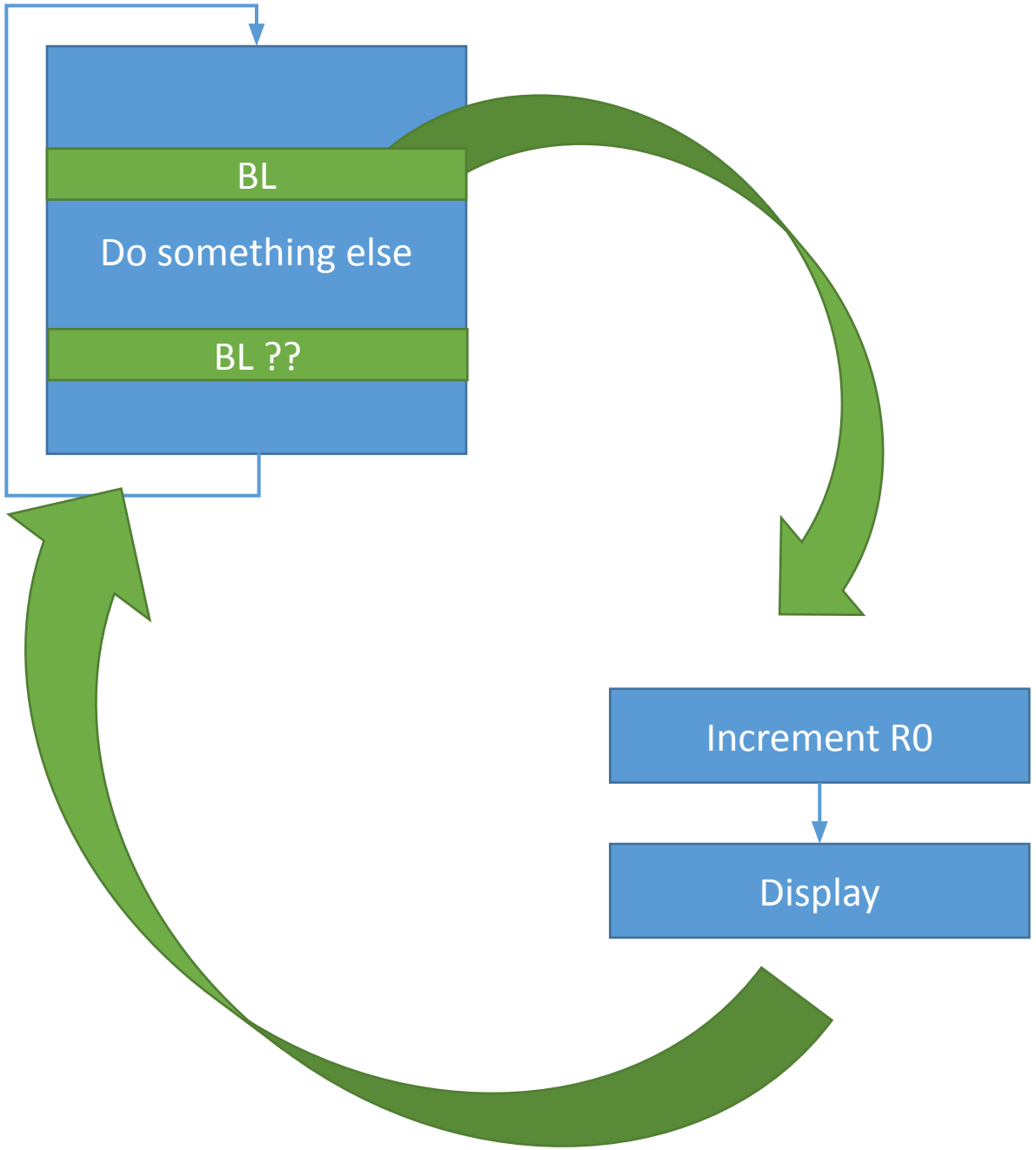

Disassembly (Ctrl-D)
Go to address, label, or register: 00000000
Address
Opcode
Disassembly
00000034 0a000002 beq 0x44 (0x4
00000038 e2800001 23 add r0, r0, #1
0000003c e3500009 24 cmp r0, #9 ;
00000040 c3a00000 25 movgt r0, #0 ;
26 DO_DELAY: LDR R7,
DO_DELAY:
00000044 e3a070c8 mov r7, #200
27 SUB_LOOP:SUBS R7,
SUB_LOOP:
00000048 e2577001 subs r7, r7, #1
28 BNE SUB_LOOP
0000004c 1afffffd bne 0x48 (0x4
30 B LOOP
00000050 eafffffe b 0x14 (0x14: L
Editor (Ctrl-E)
Disassembly (Ctrl-D)
Memory (Ctrl-M)
Devices
LEDs
Switches
9 8 7 6 5 4 3 2 1 0
All
Push buttons
3 2 1 0
All
Seven-segment displays
JTAG UART

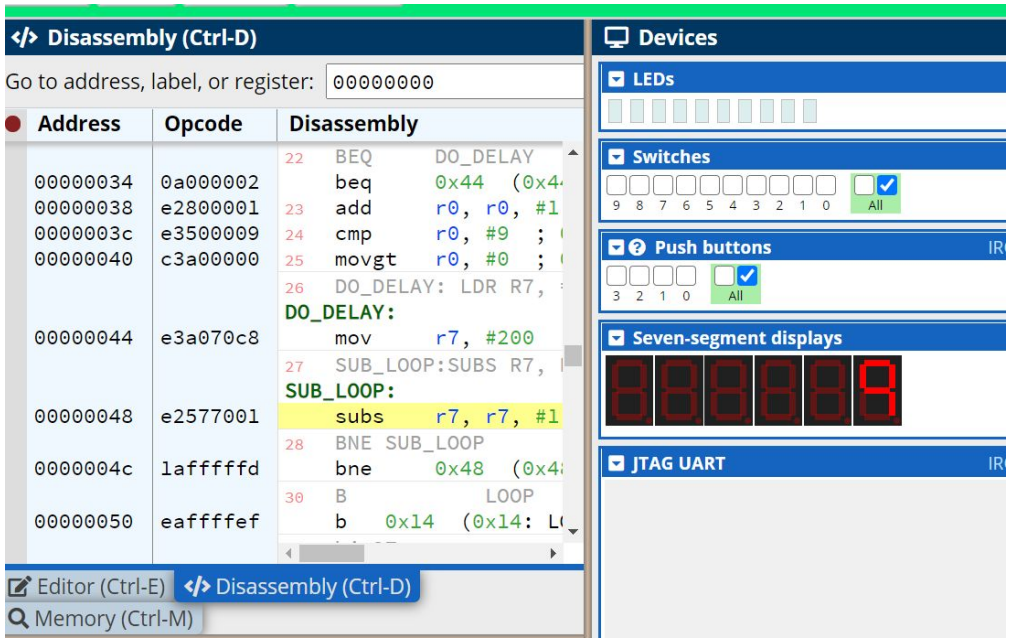

Do something else
Increment R0
Display

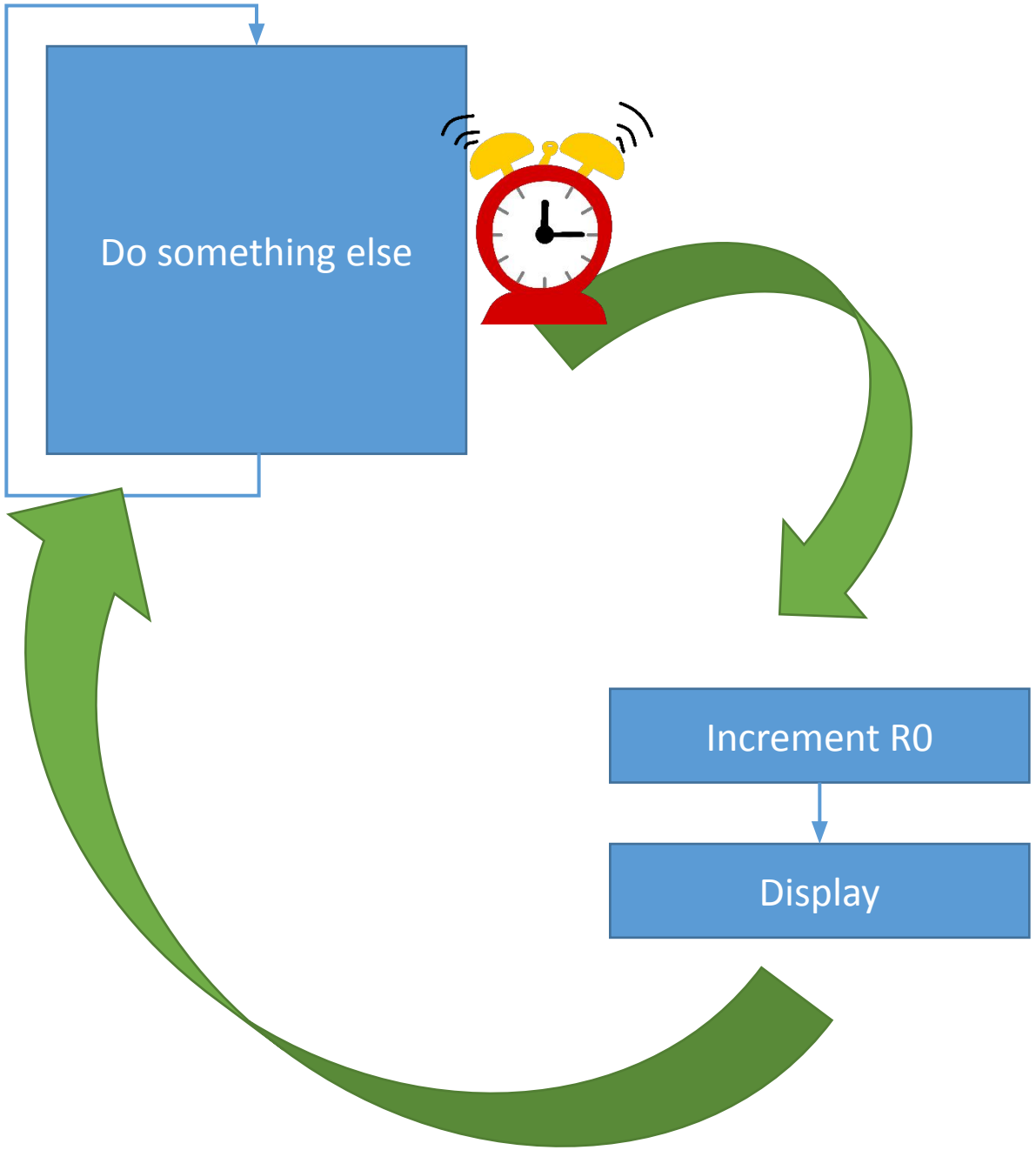

Disassembly (Ctrl-D)
Go to address, label, or register: 00000000
Address
Opcode
Disassembly
00000034 0a000002 beq 0x44 (0x4
00000038 e2800001 add r0, r0, #1
0000003c e3500009 cmp r0, #9 ;
00000040 c3a00000 movgt r0, #0 ;
DO_DELAY: LDR R7,
DO_DELAY:
00000044 e3a070c8 mov r7, #200
SUB_LOOP:SUBS R7,
SUB_LOOP:
00000048 e2577001 subs r7, r7, #1
BNE SUB_LOOP
0000004c 1afffffd bne 0x48 (0x4
B LOOP
00000050 eafffffe b 0x14 (0x14: L
Editor (Ctrl-E)
Disassembly (Ctrl-D)
Memory (Ctrl-M)
Devices
LEDs
Switches
9 8 7 6 5 4 3 2 1 0
All
Push buttons
3 2 1 0
All
Seven-segment displays
JTAG UART

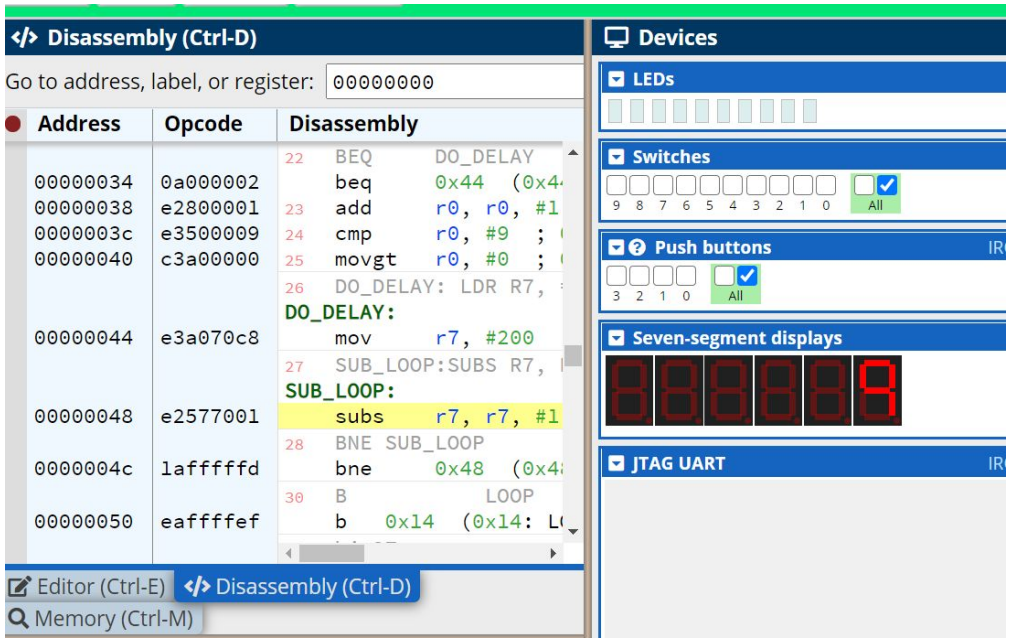

BL
Do something else
Increment R0
Display


Disassembly (Ctrl-D)
Go to address, label, or register: 00000000
Address
Opcode
Disassembly
00000034 0a000002 beq 0x44 (0x4
00000038 e2800001 add r0, r0, #1
0000003c e3500009 cmp r0, #9 ;
00000040 c3a00000 movgt r0, #0 ;
DO_DELAY: LDR R7,
DO_DELAY:
00000044 e3a070c8 mov r7, #200
SUB_LOOP:SUBS R7,
SUB_LOOP:
00000048 e2577001 subs r7, r7, #1
BNE SUB_LOOP
0000004c 1afffffd bne 0x48 (0x4
B LOOP
00000050 eafffffe b 0x14 (0x14: L
Editor (Ctrl-E)
Disassembly (Ctrl-D)
Memory (Ctrl-M)
Devices
LEDs
Switches
9 8 7 6 5 4 3 2 1 0
All
Push buttons
3 2 1 0
All
Seven-segment displays
JTAG UART

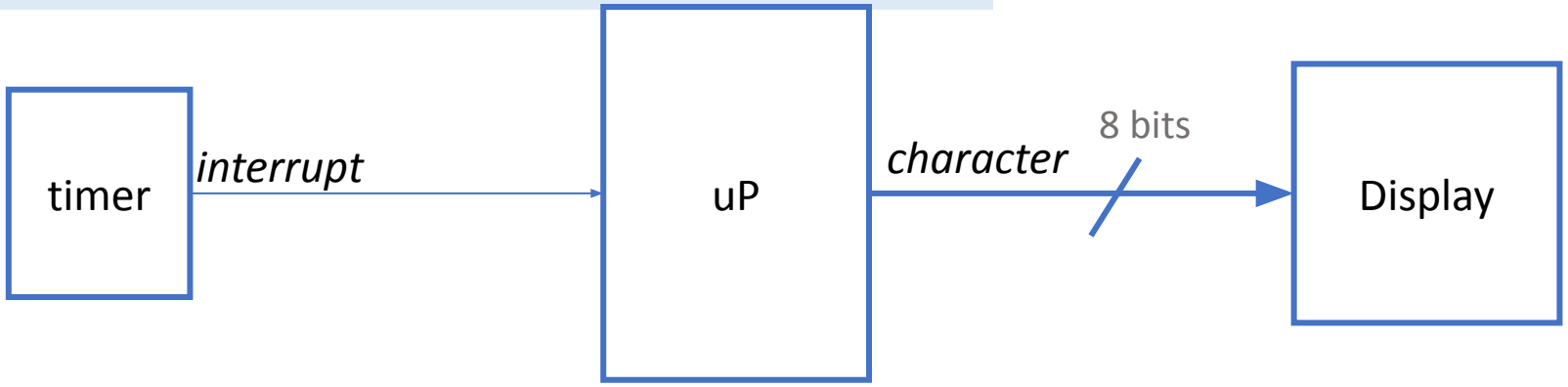

BL
Do something else
BL ??
Increment R0
Display

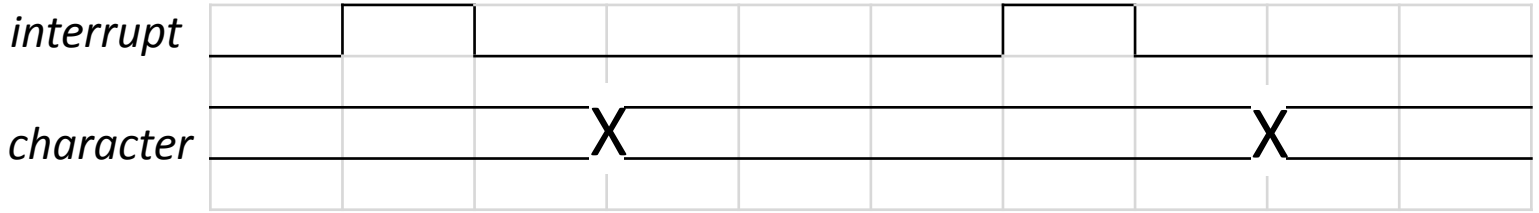

Disassembly (Ctrl-D)
Go to address, label, or register: 00000000
Address
Opcode
Disassembly
00000034 0a000002 beq 0x44 (0x4
00000038 e2800001 add r0, r0, #1
0000003c e3500009 cmp r0, #9 ;
00000040 c3a00000 movgt r0, #0 ;
DO_DELAY: LDR R7,
DO_DELAY:
00000044 e3a070c8 mov r7, #200
SUB_LOOP:SUBS R7,
SUB_LOOP:
00000048 e2577001 subs r7, r7, #1
BNE SUB_LOOP
0000004c 1afffffd bne 0x48 (0x4
B LOOP
00000050 eafffffe b 0x14 (0x14: L
Editor (Ctrl-E)
Disassembly (Ctrl-D)
Memory (Ctrl-M)
Devices
LEDs
Switches
9 8 7 6 5 4 3 2 1 0
All
Push buttons
3 2 1 0
All
Seven-segment displays
JTAG UART


Do something else
Increment R0
Display


Disassembly (Ctrl-D)
Go to address, label, or register: 00000000
Address
Opcode
Disassembly
00000034 0a000002 beq 0x44 (0x4
00000038 e2800001 add r0, r0, #1
0000003c e3500009 cmp r0, #9 ;
00000040 c3a00000 movgt r0, #0 ;
DO_DELAY: LDR R7,
DO_DELAY:
00000044 e3a070c8 mov r7, #200
SUB_LOOP:SUBS R7,
SUB_LOOP:
00000048 e2577001 subs r7, r7, #1
BNE SUB_LOOP
0000004c 1afffffd bne 0x48 (0x4
B LOOP
00000050 eafffffe b 0x14 (0x14: L
Editor (Ctrl-E)
Disassembly (Ctrl-D)
Memory (Ctrl-M)
Devices
LEDs
Switches
9 8 7 6 5 4 3 2 1 0
All
Push buttons
3 2 1 0
All
Seven-segment displays
JTAG UART

Interrupts

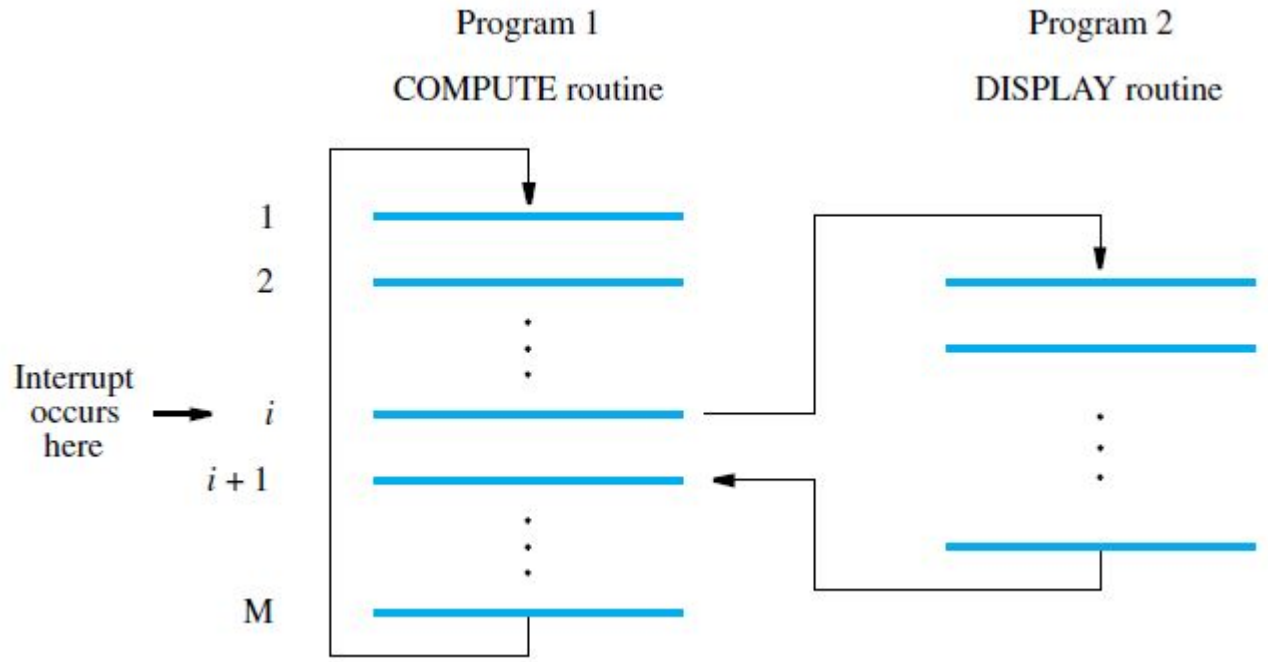




- Consider a task that requires continuous extensive computations to be performed and the results to be displayed on a display device, every ten seconds.
- The ten-second intervals is determined by a timer circuit.
- The timer circuit raise an interrupt request once every ten seconds.
- In response, the processor displays the latest results.

Interrupts

- The processor continuously executes the COMPUTE routine.
- DISPLAY routine sends the latest results to the display device.
- When uP receives an interrupt request from the timer, it suspends the execution of the COMPUTE routine and executes the DISPLAY routine.
- Upon completion of the DISPLAY routine, the processor resumes the execution of the COMPUTE routine.
- A **Return-from-interrupt** instruction at the end of the interrupt-service routine reloads the PC from that temporary storage location. (The return address must be saved.)



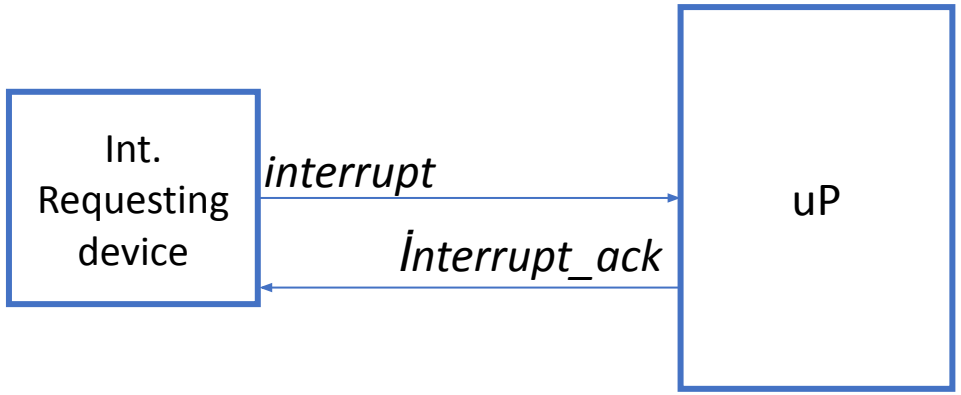
slido

What may be the potential problems with the interrupts?

ⓘ Start presenting to display the poll results on this slide.

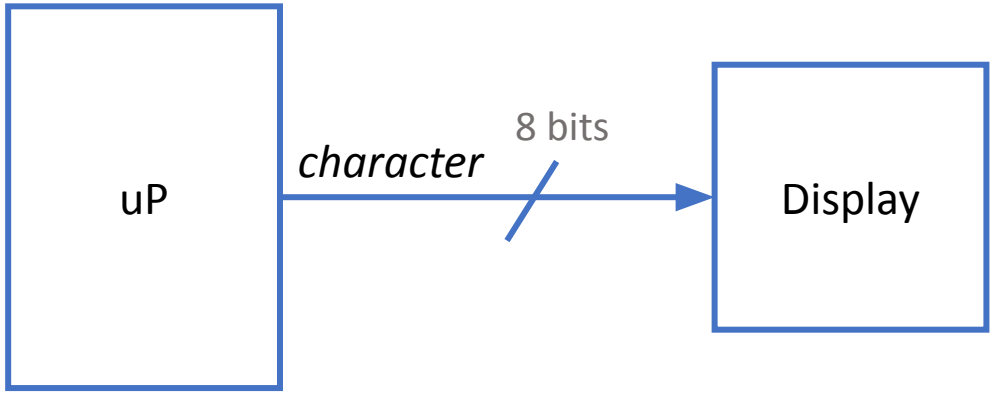
Interrupts

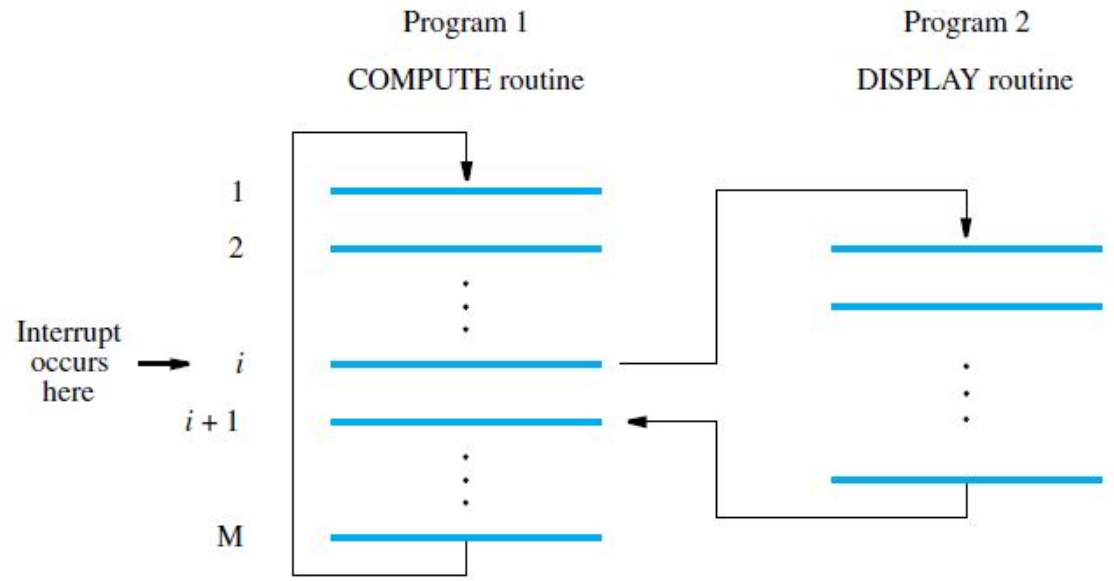
- **interrupt acknowledge signal**:
- The processor must inform the device that its request has been recognized so that it may remove its interrupt-request signal.
- It is sent through the interconnection network.



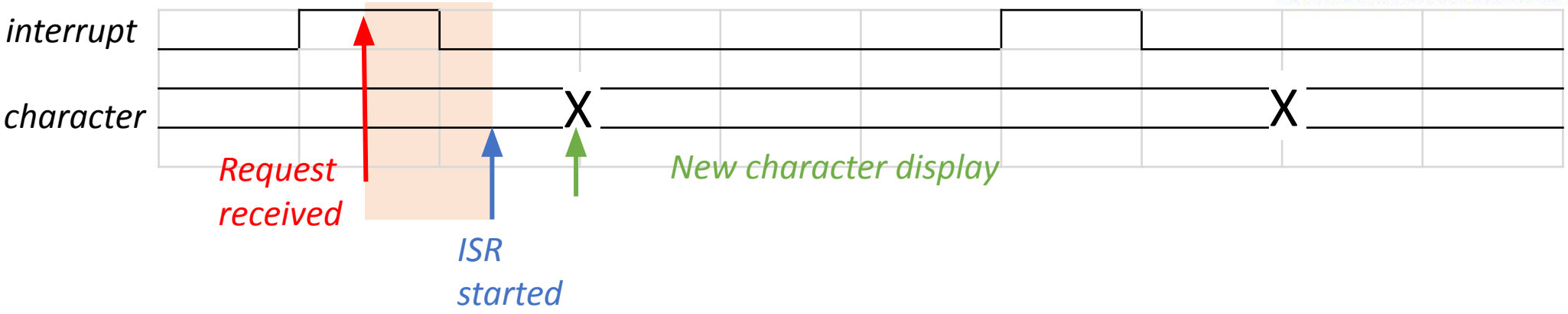
Interrupts

- **interrupt service routine (ISR)**:
- Display routine in this example.
- It resembles a subroutine performing DISPLAY.
- ISR may not have any relation to the portion of the program being executed at the time the interrupt request is received.
- Before starting execution of the ISR status information and contents of processor registers must be saved.





Interrupt latency



- is the delay between the time an interrupt request is received and the start of execution of the ISR.
- Most modern processors **save only the minimum amount of information** needed to maintain the integrity of program execution. Because:
- Saving and restoring registers involves memory transfers the total execution time increases. This represents **execution overhead**.

Enabling and Disabling Interrupts

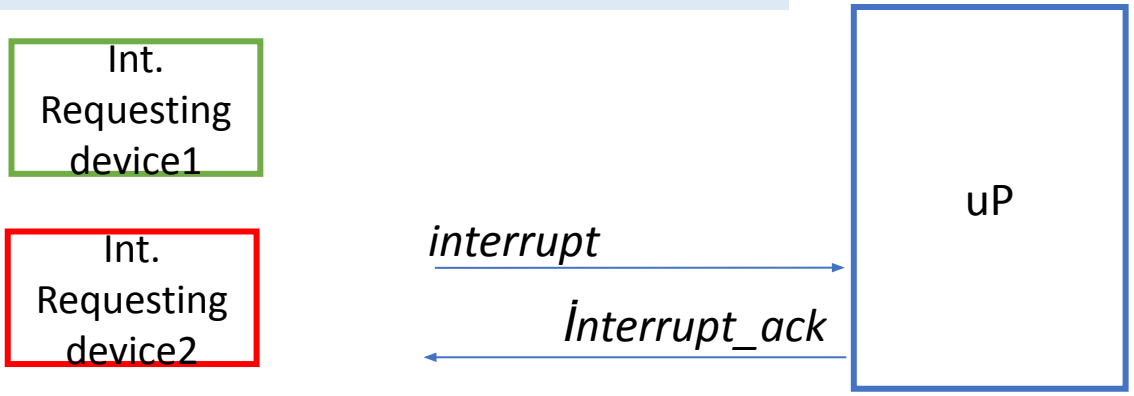
- One bit of the register be assigned for enabling/disabling interrupts.
  - Usually IE or ID
  - The programmer can set or clear IE/ID to cause the desired action.
  - The processor (usually) automatically disable interrupts before starting the execution of the interrupt-service routine

Handling an interrupt request

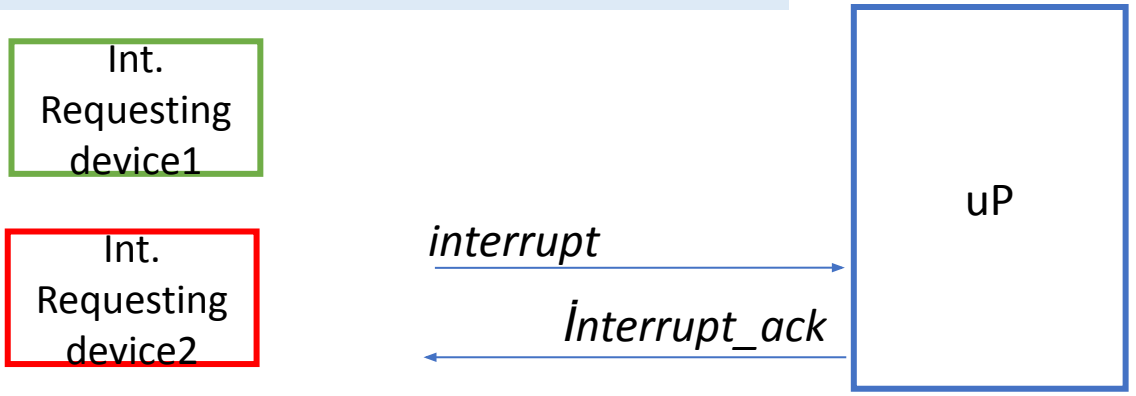
1. The device raises an interrupt request.
2. The processor interrupts the program currently being executed and saves the contents of the PC and PS registers.
3. *Interrupts are disabled by clearing/setting the IE/ID bit in the status register*.
4. The device is informed that its request has been recognized, and in response, it deactivates the interrupt-request signal. The action requested by the interrupt is performed by the ISR.
5. Upon completion of the interrupt-service routine, the saved contents of the PC and status registers are restored.

Interrupts may or may not be enabled.

Handling Multiple Devices



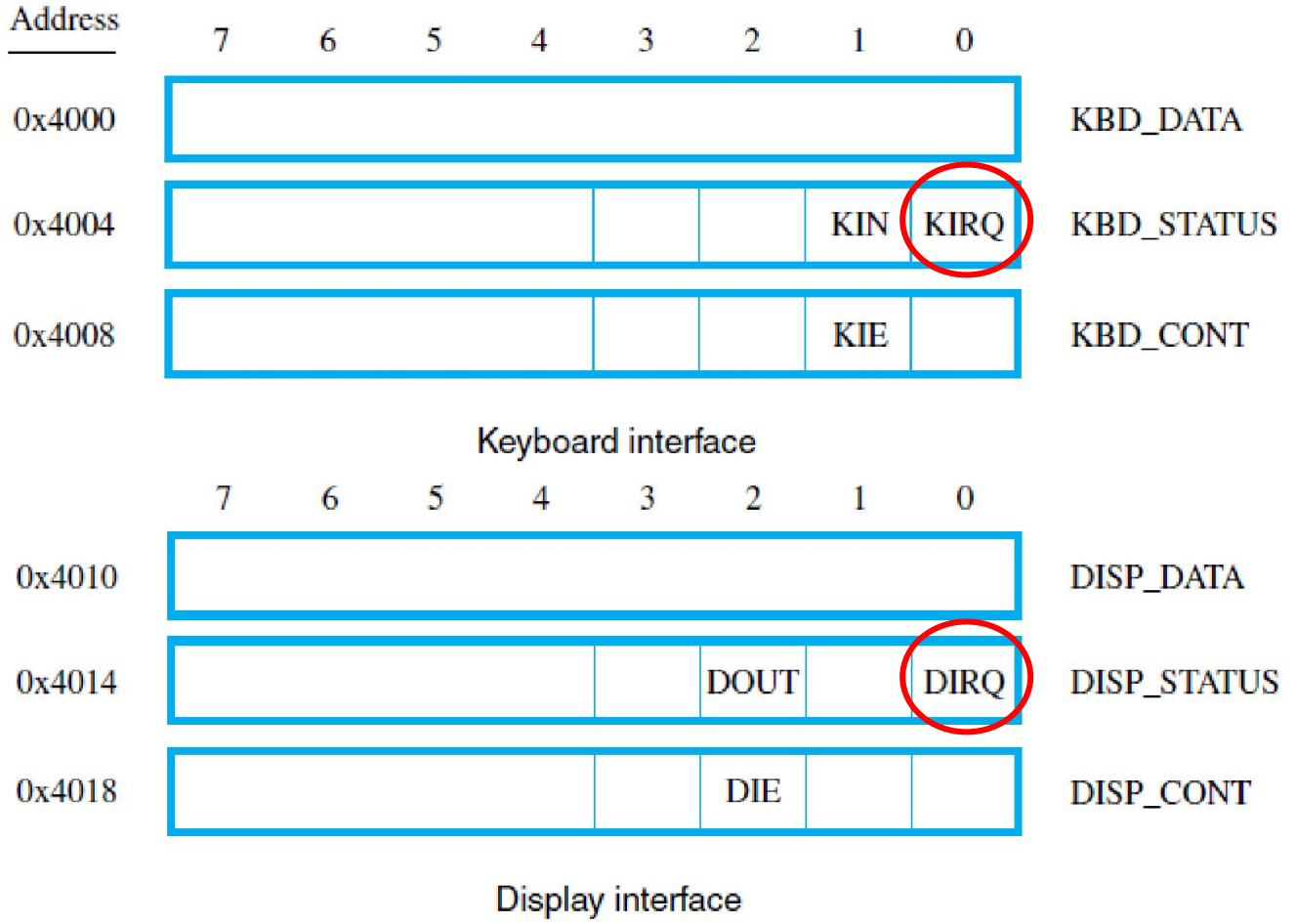
Handling Multiple Devices



1. How can the processor determine which device is requesting an interrupt?
2. If they require different ISRs, how can the processor obtain the correct ISR address?
3. Should a device be allowed to interrupt the processor while another interrupt is being serviced?
4. How should two or more simultaneous interrupt requests be handled?

Handling Multiple Devices

- A device's Interrupt ReQuest (IRQ) is available in its status register.
- Identify the device requesting the interrupt:
- The ISR polls all I/O devices in the system.
  - ☺ The simplest way
  - ☹ Time spent interrogating the IRQ bits of devices that may not be requesting any service

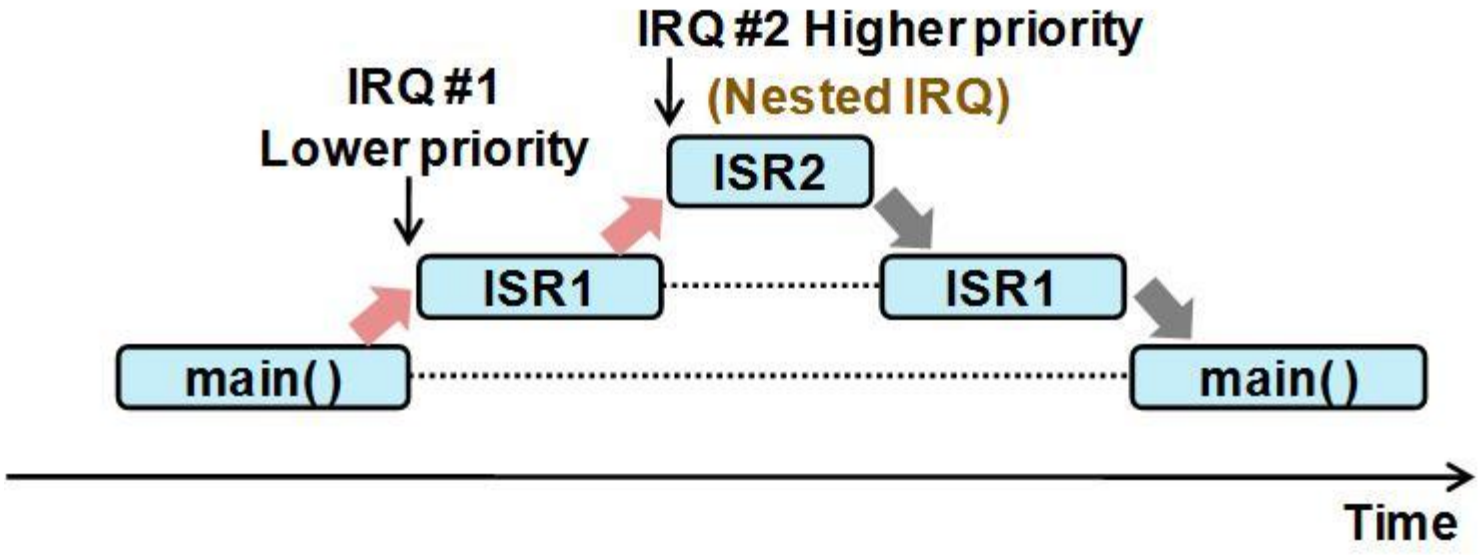


Vectored Interrupts

- A device requesting an interrupt may identify itself directly to the processor:
  - it has its own interrupt-request signal
  - it sends a special code to the processor through the interconnection network
- The processor's circuits determine the memory address of the required interrupt-service routine.
- Typically, **an area in the memory** holds the addresses of interrupt-service routines: **interrupt vectors**.
- The **interrupt vector table** is –usually, in the lowest-address range. The interrupt-service routines may be located anywhere in the memory.

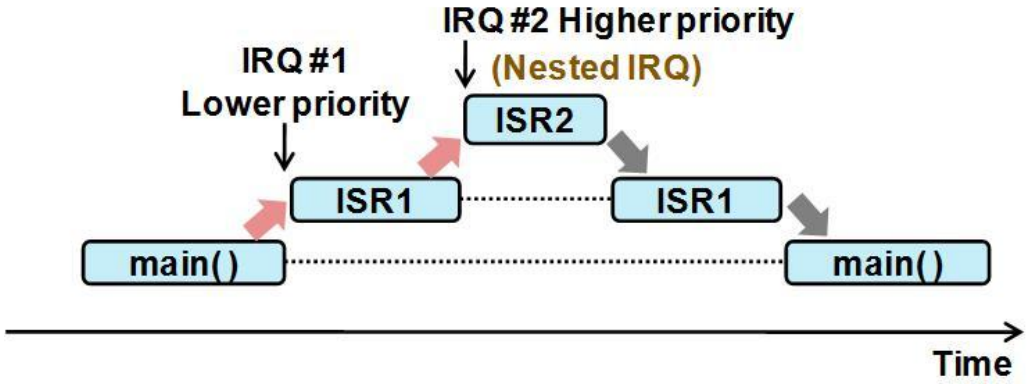
0x00	B ISR1
0x04	B ISR2
0xFA0	ISR1 instructions
0xFF0	ISR1 instructions

Interrupt Nesting



Interrupt Nesting

- I/O devices should be organized in a priority structure: **multiple-level priority organization**
- An interrupt request from a high-priority device should be accepted while the processor is servicing a request from a lower-priority device
- Assign a priority level to the processor that can be changed under program control



- During execution of an ISR,
  - The priority of the processor is raised to that of the device
  - Interrupts from devices that have the same or lower level of priority are disabled
  - Interrupt requests from higher-priority devices will continue to be accepted.

Simultaneous Requests

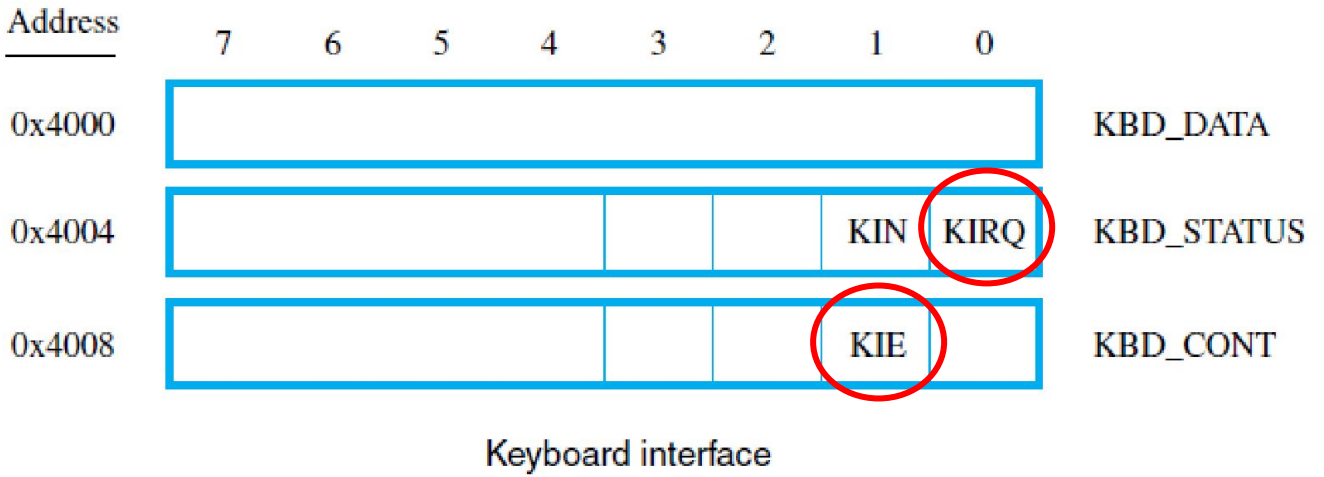
- Problem: Deciding which request to service first
- Solution 1: Polling the status registers of the I/O devices
  - Priority is determined by the order in which the devices are polled
- Solution 2: Use vectored interrupts
  - Ensure that only one device is selected to send its interrupt vector code
  - Use an appropriate interface circuit

The device

MEF UNIVERSITY
Your Freedom in Learning

Interrupt controlled I/O

- Device is allowed to interrupt the processor whenever it is ready for an I/O transfer
- Control register
  - It can be accessed as an addressable location
  - It involves the interrupt-enable bit, IE
  - Simple devices - a keyboard, require little control
  - Complex devices - have a number of possible modes of operation



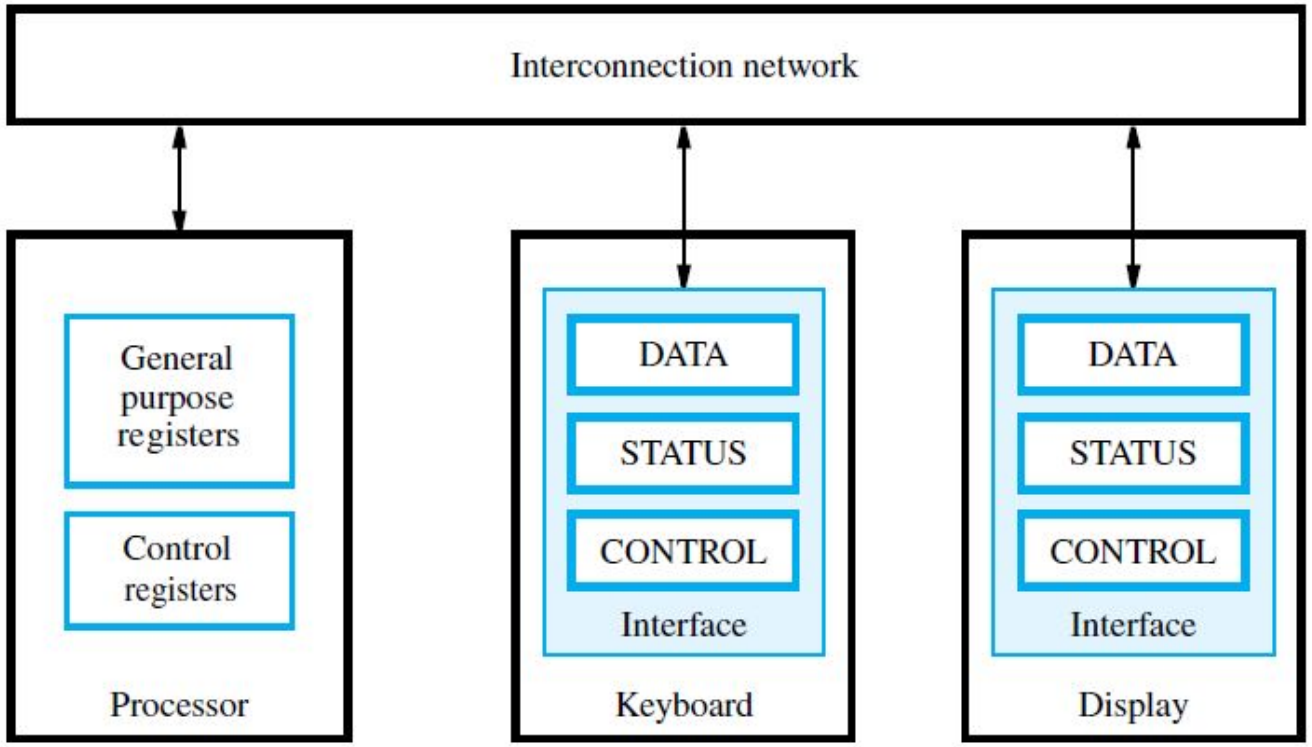
Keyboard interface

- Example: The keyboard status register includes bits KIN and KIRQ. Control register involves KIE: Keyboard interrupt enable.
  - If, KIE = 1
  - KIRQ □1, when an interrupt request has been raised, but not yet serviced.

I/O Device Interface

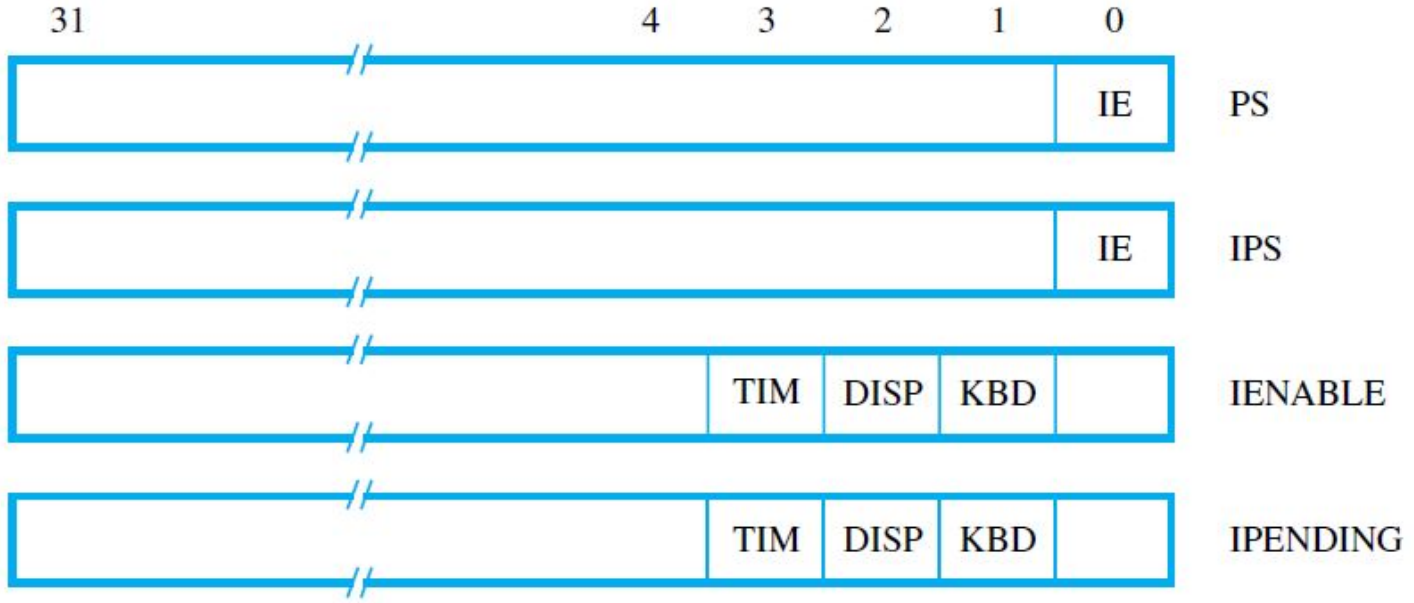
governs the operation of the device.

- Data registers: a buffer for data transfers
- Status registers: hold info about the current status of the device
- Control registers: hold info that controls the operational behavior of the device



Processor Control Registers

- PS is status register.
- IPS register is used to automatically save the contents of PS when an interrupt request is received and accepted



- IENABLE register allows the processor to selectively respond to individual I/O devices.
- IPENDING register indicates the active interrupt requests This is convenient when multiple devices may raise requests at the same time.

The uP

MEF UNIVERSITY
Your Freedom in Learning

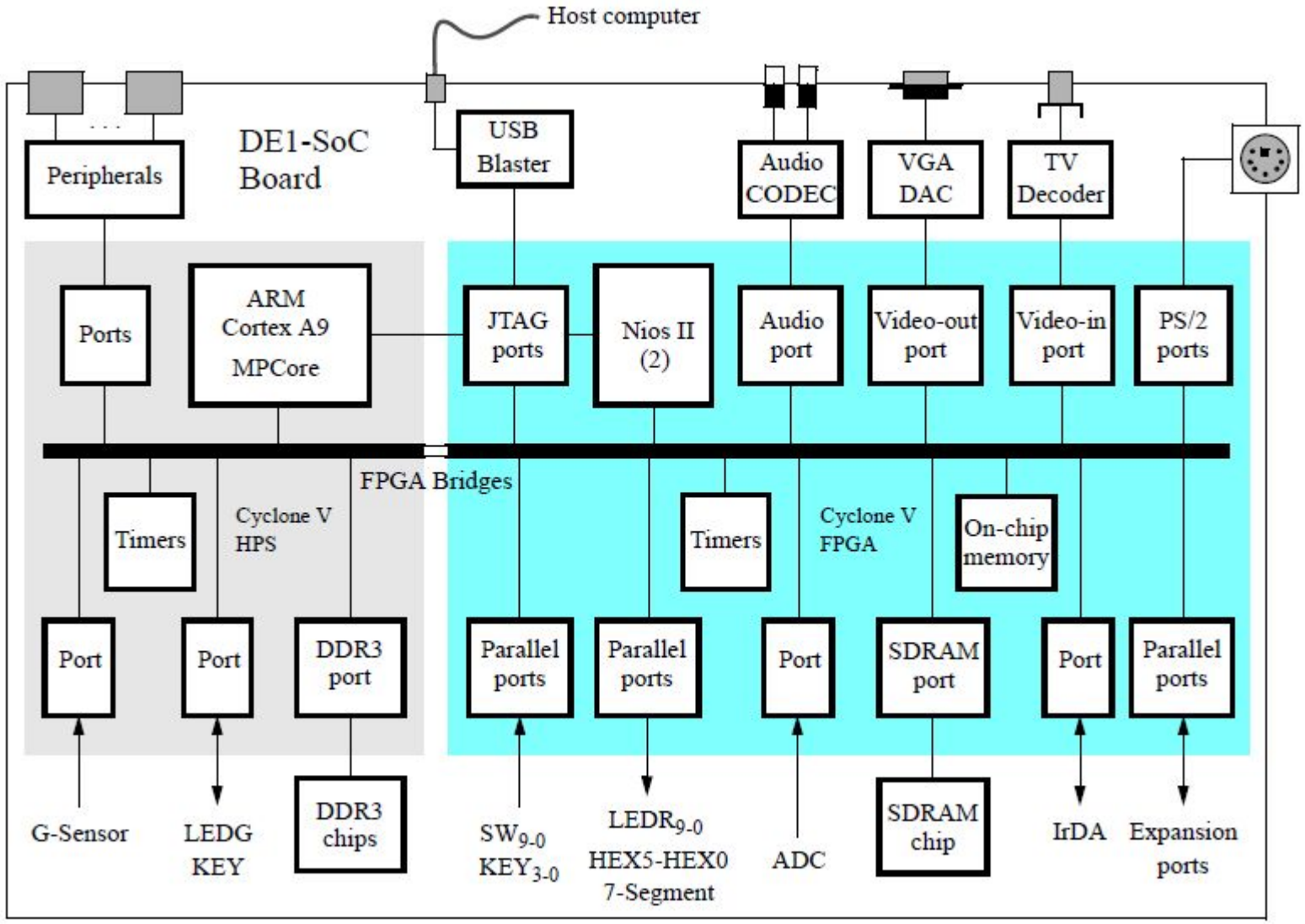
Microprocessors

Tuba Ayhan

MEF University

Interrupt handling – modes of operation

USING THE ARM GENERIC INTERRUPT CONTROLLER



Ref: DE1-SoC Computer System with ARM Cortex-A9

Modes of Operation

	User	the basic mode. Unprivileged: has restricted access to system resources, certain types of processor operations and instructions are prohibited.
Exception Modes	**System**	full access to system resources. It can be entered only from one of the exception modes below.
	Supervisor	is entered when the processor executes a supervisor call instruction, SVC. It is also **entered on reset or power-up**.
	Abort	is entered if the processor attempts to access a non-legitimate memory location: i.e. a word access for an address that is not word-aligned
	Undefined	is entered if the processor attempts to execute an unimplemented instruction
	IRQ	response to an **I**nterrupt **R**e**Q**uest.
	FIQ	response to a **F**ast interrupt request. They are used in some Cortex-A9 systems to provide faster service for more urgent requests. Not in this course.

slido

Which mode is entered on reset?

ⓘ Start presenting to display the poll results on this slide.

Privileged modes

- **Privileged**: it allows the use of all processor instructions and operations. i.e. Supervisor (SVC) mode is privileged.
- In practice, with power-on or reset, the processor enters on Supervisor mode. From supervisor mode it is possible to change into User mode.
- In practice, the Supervisor mode is normally used when the processor is executing software such as an operating system. Other software code may run in the User mode, so a level of protection for critical resources is provided.

CPSR and modes

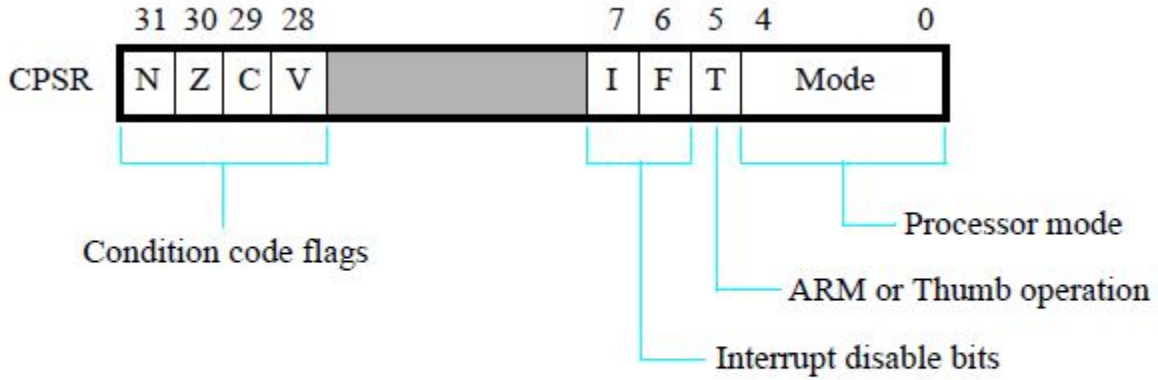


Figure 2. The current processor status register (CPSR).

TABLE 1. Mode Bits

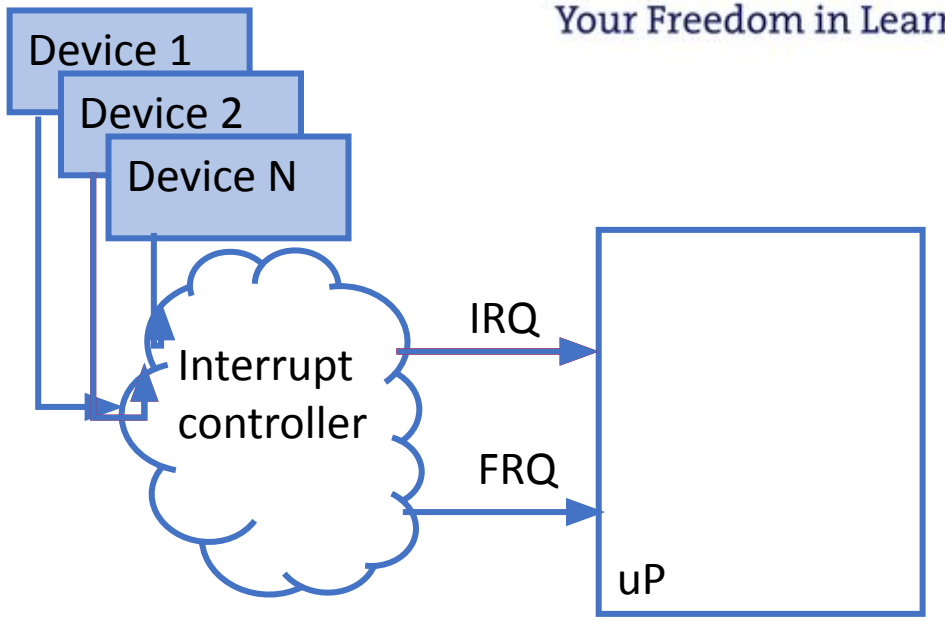
$CPSR_{4-0}$	Operating Mode
10000	User
10001	FIQ
10010	IRQ
10011	Supervisor
10111	Abort
11011	Undefined
11111	System

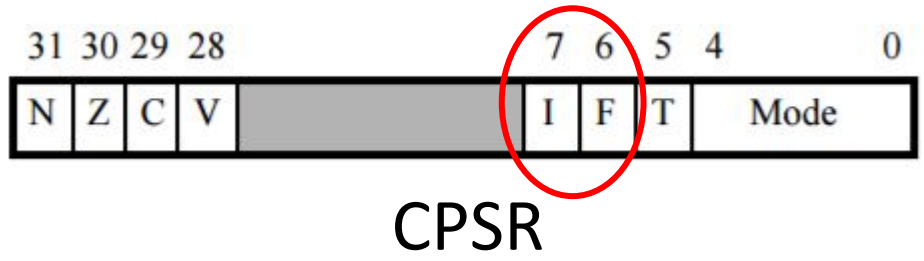
- To manipulate the contents of the CPSR, the processor must be in one of the privileged modes, i.e. SVC mode. Instruction to write into CPSR:

  `MSR (Move Status Register)`

External interrupt requests

- ARM processors have two external interrupt requests: FIQ and IRQ
  - level-sensitive active low inputs
- interrupt controller: accepts interrupt requests from a wide variety of external sources and map them onto FIQ or IRQ
- An interrupt exception can be taken only when the appropriate CPSR disable bit is clear.





IRQ Mode

- Processor enters IRQ mode in response to receiving an IRQ signal from the GIC.
- Before such interrupts can be used, software code has to:

1. **Disable IRQ interrupts** in the A9 processor, by setting the IRQ disable bit in the CPSR to 1.
2. **Configure the GIC**. Interrupts for each I/O peripheral device that is connected to the GIC are identified by a unique interrupt ID.
3. **Configure I/O peripheral device** so that it can send IRQ interrupt requests to the GIC.
4. **Enable IRQ interrupts** in the A9 processor, by setting the IRQ disable bit in the CPSR to 0.

Generic Interrupt Controller (GIC)

- It provides registers for **managing interrupt sources**, **interrupt behavior**, and **interrupt routing to one or more processors**
- It provides support for:
- enabling, disabling, and generating processor interrupts from hardware (peripheral) interrupt sources
- Software-generated Interrupts (SGIs)
- Interrupt masking and prioritization
- Uniprocessor and multiprocessor environments
- Wakeup events in power-management environments
- The ARM architecture Security Extensions, Virtualization Extensions

- Figure 3 shows the general-purpose registers in a Cortex-A9 processor, and illustrates how the registers are related to the processor mode. In User mode, there are 16 registers, *R*0¡*R*15, plus the CPSR. These registers are also available in the System mode, which is not shown in the figure. As indicated in Figure 3, *R*0¡*R*12, as well as the program counter *R*15, are common to all modes except FIQ. But the stack pointer register *R*13 and the link register *R*14 are not common—banked versions of these registers exist for each mode. Thus, the Supervisor mode has a stack pointer and link register that are used only when the processor is in this mode. Similarly, the other modes, such as IRQ mode, have their own stack pointers and link registers. The CPSR register is common for all modes, but when the processor is switched from one mode into another, the current

Microprocessors

Tuba Ayhan

MEF University

Programmer's Interface to the GIC

Using the ARM Generic Interrupt Controller

Generic Interrupt Controller (GIC)

- GIC handles up to 255 s interrupts
- GIC is memory-mapped

Internal:
i.e. Timer

External:
i.e. Ports

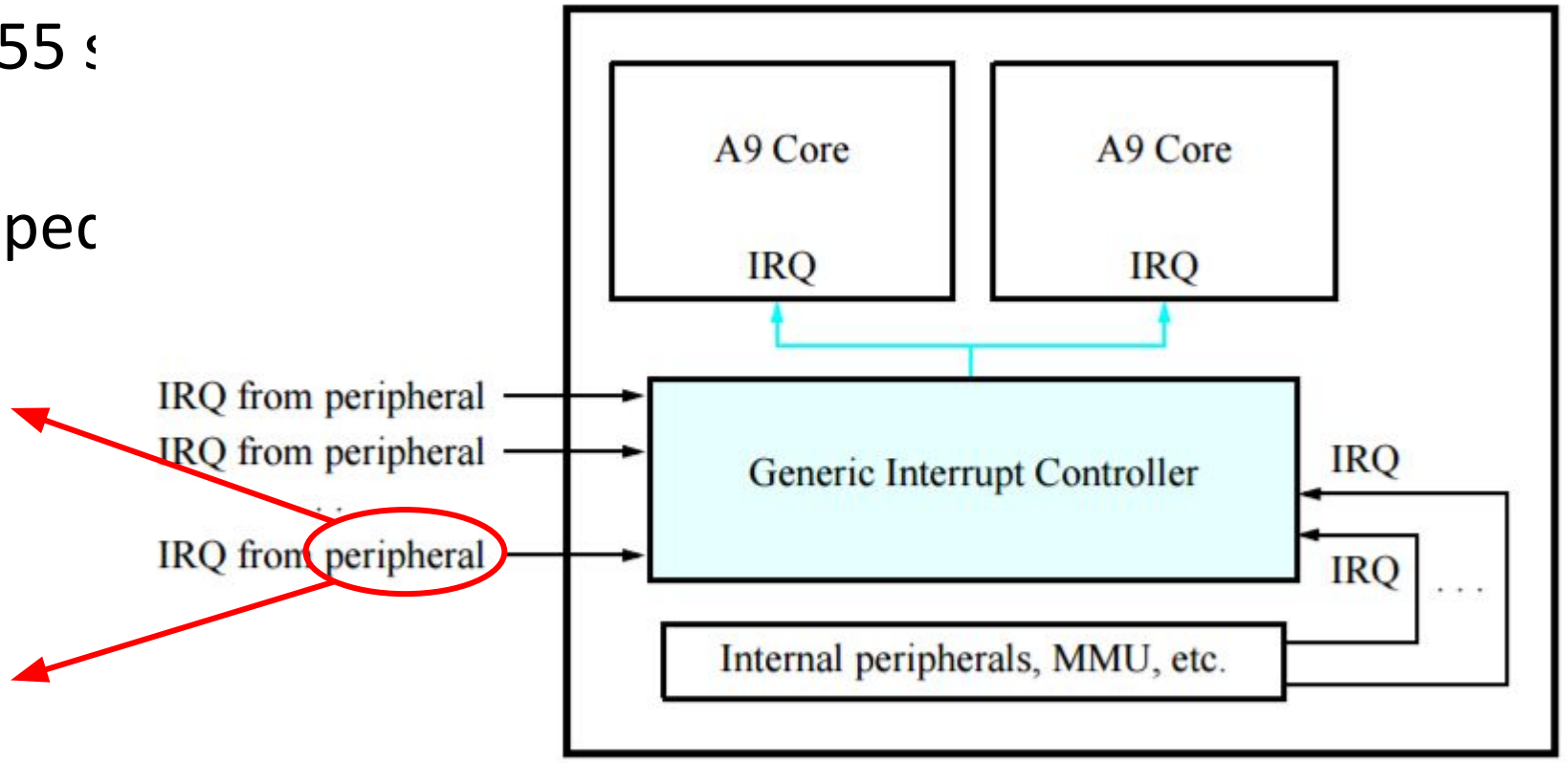


Figure 1. The ARM A9 MPCORE processor.

ARM Generic Interrupt Controller GIC

- GIC is a part of the ARM A9 MPCORE processor.
- GIC is connected to the IRQ interrupt signals of all I/O peripheral devices that are capable of generating interrupts.
- GIC handles up to 255 sources of interrupts. Each device has a unique **interrupt ID**.
- GIC can forward an IRQ signal to one or both of the A9 cores.
- Software code that is running on the A9 core can then query the GIC to determine which peripheral device caused the interrupt, and take appropriate action.

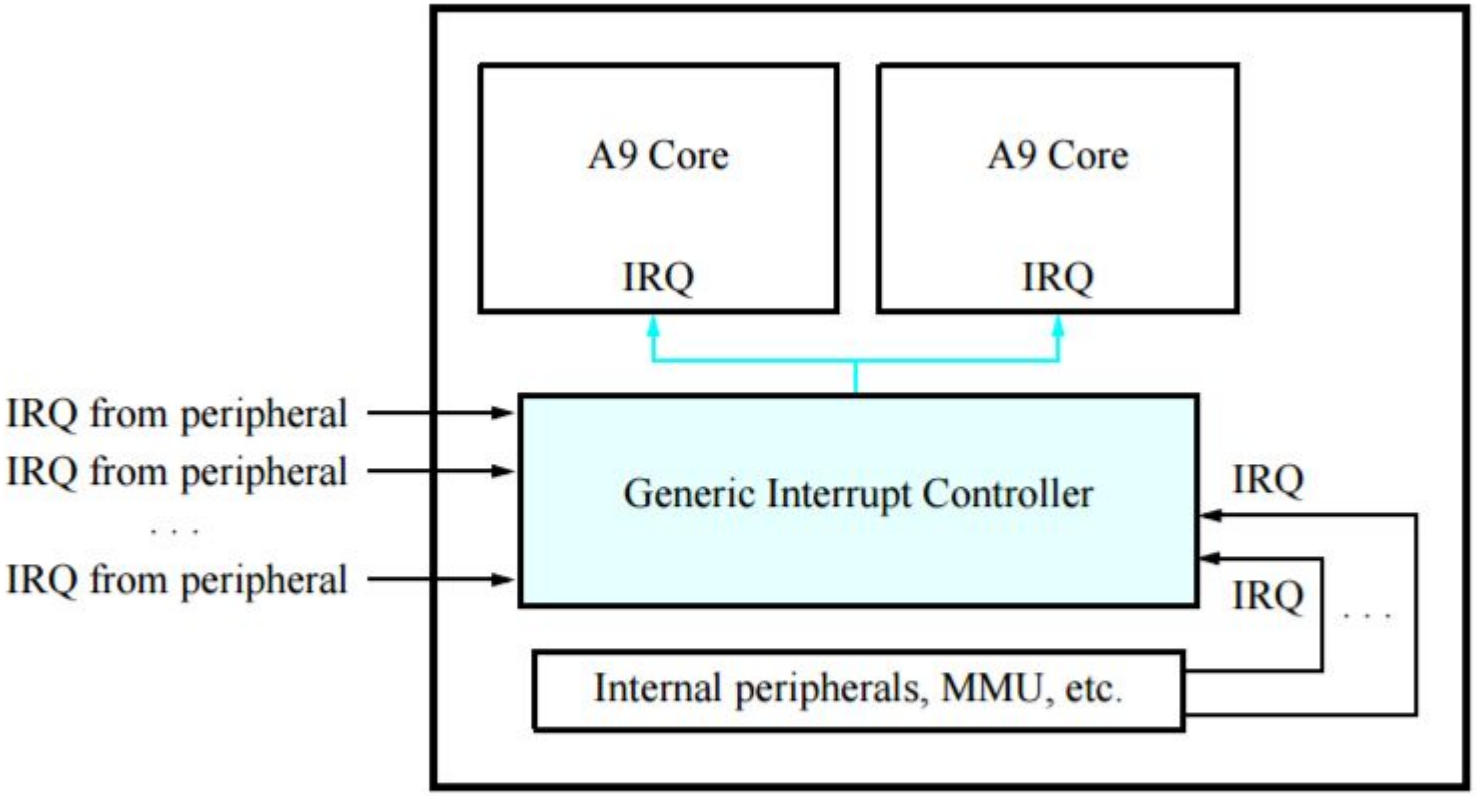


Figure 1. The ARM A9 MPCORE processor.

Programmer's Interface to the GIC

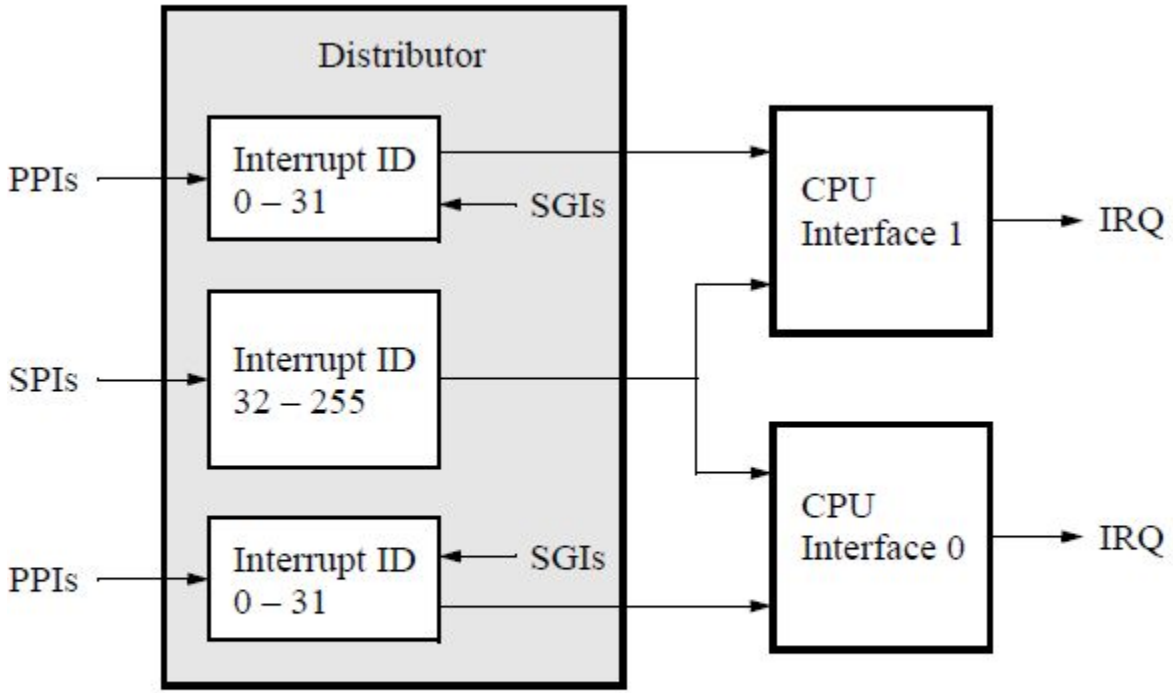
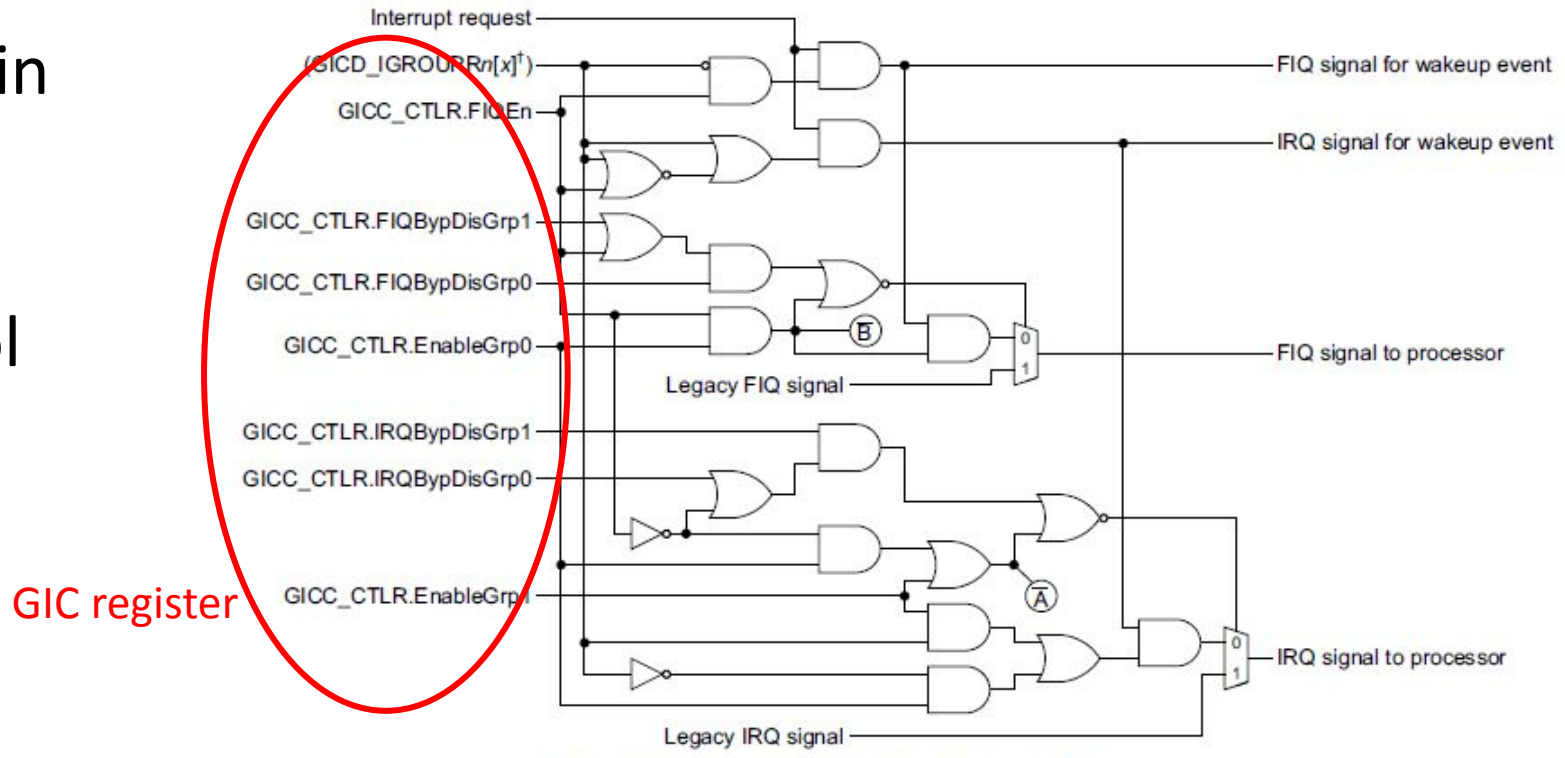


Figure 4. The GIC Architecture.

Programmer's Interface to the GIC

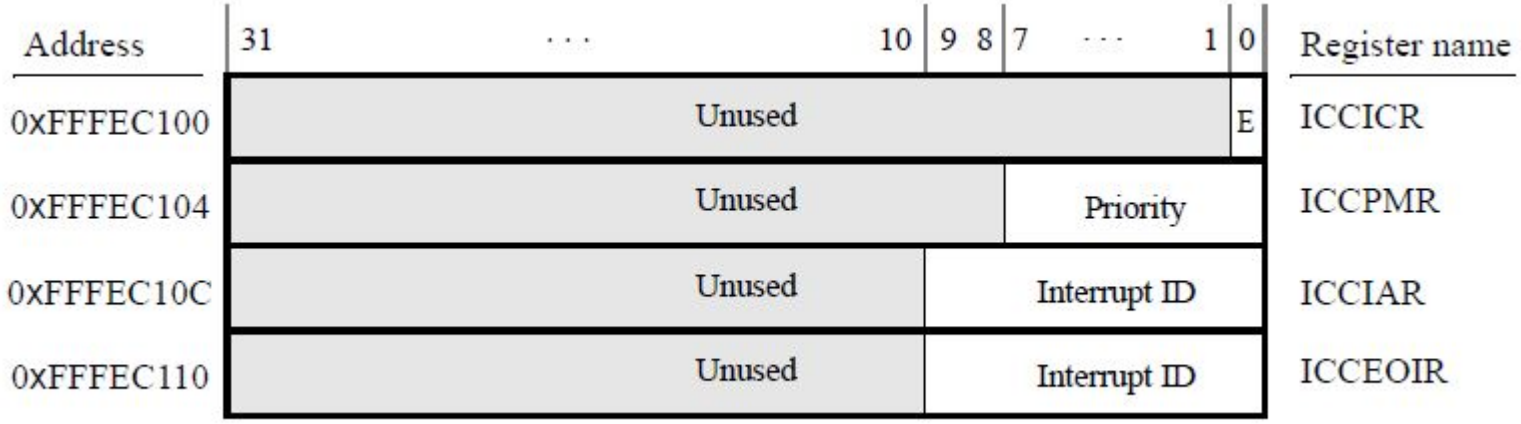
- Write appropriate values in
  - CPU interface registers
  - Distributor registers
- in order to set the control signals.



GIC CPU Interface

- sends IRQ signals to the A9 cores. There is one CPU Interface for each A9 core in the MPCORE.

CPU Interface Registers



CPU Interface Control Register (ICCICR)	is used to enable forwarding of interrupts from the CPU Interface to the corresponding A9 core. Set bit E = 1 to enable and set E = 0 to disable
Interrupt Priority Mask Register (ICCPMR)	is used to set a threshold for the priority-level of interrupts that will be forwarded by a CPU Interface to an A9 core.
Interrupt Acknowledge Register (ICCIAR)	contains the Interrupt ID of the I/O peripheral that has caused an interrupt.
End of Interrupt Register (ICCEOIR)	After writing into the ICCEOIR, the interrupt handler software can then return control to the previously-interrupted main program.

GIC Distributor – 1/3

- can handle 255 sources of interrupts.
- **SPIs: shared peripheral interrupts**, range from 32-255. These interrupts are connected to the IRQ signals of up to 224 I/O peripherals, and these sources of interrupts are common to (shared by) both CPU Interfaces.

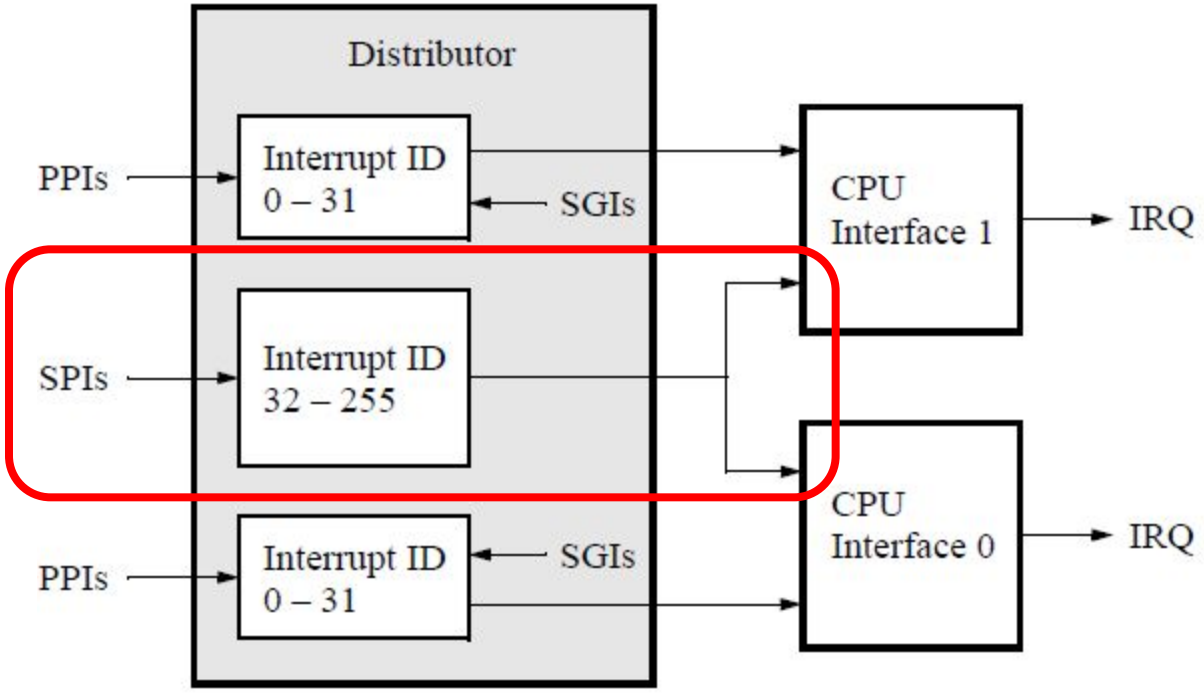


Figure 4. The GIC Architecture.

GIC Distributor – 2/3

- can handle 255 sources of interrupts.
- **PPIs: private peripherals interrupts**, range from 0-31. 32 private interrupts for each of the A9 processors.

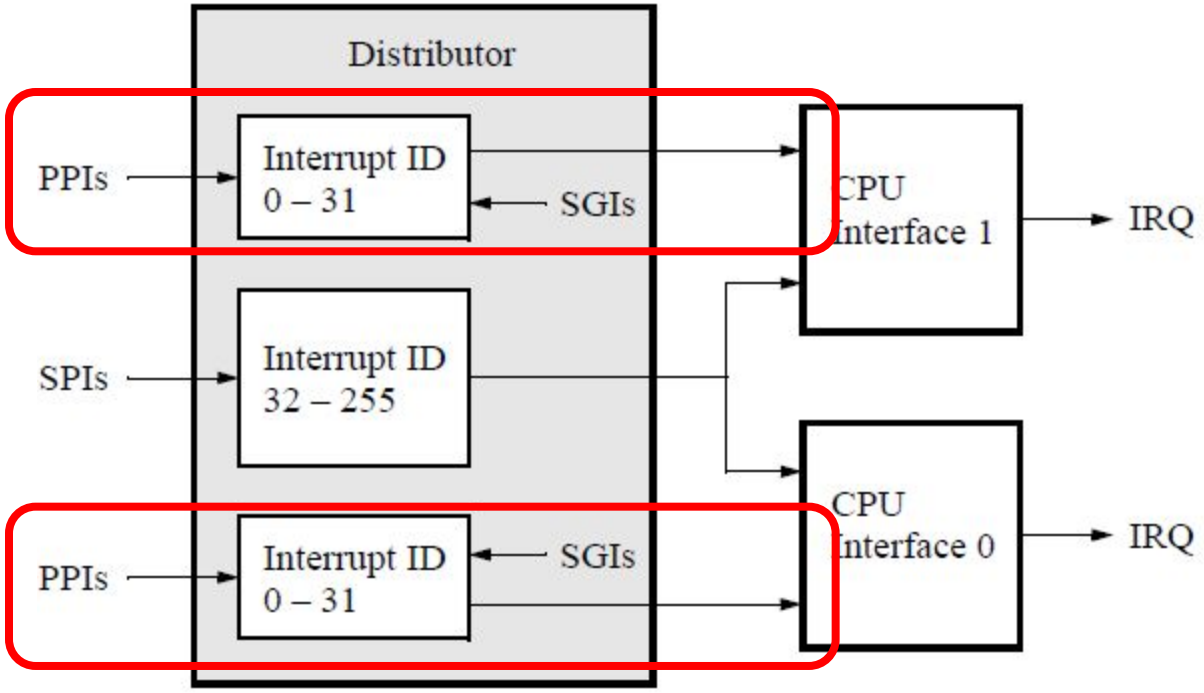


Figure 4. The GIC Architecture.

GIC Distributor – 3/3

- can handle 255 sources of interrupts.
- **SGIs: software generated interrupts**, range from 0-15. SGIs are a special type of private interrupt that are generated by writing to a specific register in the GIC. We do not discuss SGIs further in this course.

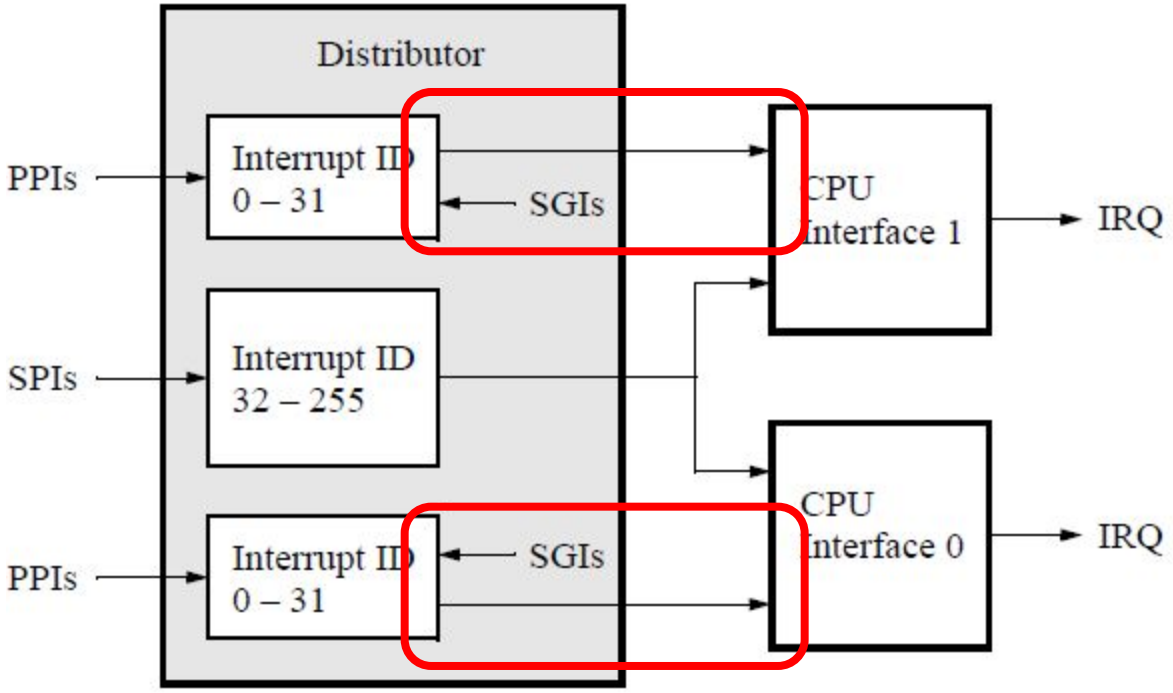
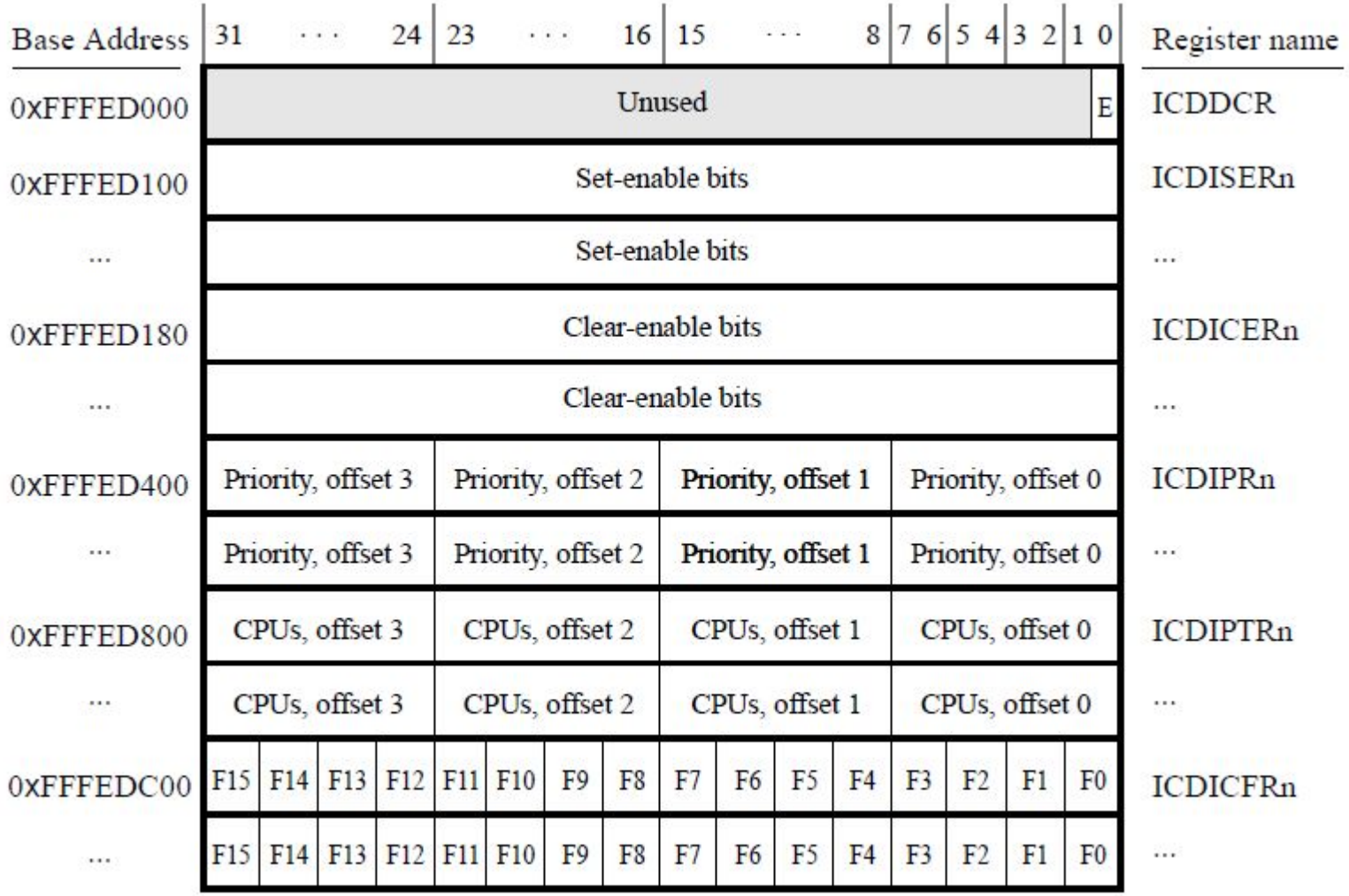


Figure 4. The GIC Architecture.

Distributor registers



Base Address	31 ··· 24	23 ··· 16	15 ··· 8	7 6 5 4 3 2 1 0	Register name
0xFFFED000	Unused			E	ICDDCR
0xFFFED100	Set-enable bits				ICDISERn
...	Set-enable bits				...
0xFFFED180	Clear-enable bits				ICDICERn
...	Clear-enable bits				...
0xFFFED400	Priority, offset 3	Priority, offset 2	Priority, offset 1	Priority, offset 0	ICDIPRn
...	Priority, offset 3	Priority, offset 2	Priority, offset 1	Priority, offset 0	...
0xFFFED800	CPUs, offset 3	CPUs, offset 2	CPUs, offset 1	CPUs, offset 0	ICDIPTRn
...	CPUs, offset 3	CPUs, offset 2	CPUs, offset 1	CPUs, offset 0	...
0xFFFEDC00	F15 F14 F13 F12	F11 F10 F9 F8	F7 F6 F5 F4	F3 F2 F1 F0	ICDICFRn
...	F15 F14 F13 F12	F11 F10 F9 F8	F7 F6 F5 F4	F3 F2 F1 F0	...

Distributor registers

Interrupt Set Enable Registers (ICDISERn)	enable the forwarding of each supported interrupt from the Distributor to the CPU Interface Given a specific Interrupt ID, N, the address of the register that contains its set-enable bit is given by the integer calculation address = 0xFFFED100 + (N / 32) x 4
Interrupt Clear Enable Registers (ICDICERn)	Disable the forwarding of each supported interrupt from the Distributor to the CPU Interface address = 0xFFFED400 + (N / 4) x 4 index = N mod 4
Interrupt Priority Registers (ICDIPRn)	associate a priority level with each individual interrupt address = 0xFFFED400 + (N /4)x4 index = N mod 4
Interrupt Processor Targets Registers (ICDIPTRn)	specify the CPU interfaces
Interrupt Configuration Registers (ICDICFRn)	specify whether each supported interrupt should be handled as level- or edge-sensitive by the GIC Address = 0xFFFEDC00 + (N/16)x4 index = (*N* mod 16)+1
Distributor Control Register (ICDDCR)	enable the Distributor. E = 0 in □ disables, E = 1 □ enables the Distributor.

Example of Assembly Language Code

- Enable Interrupt ID 73 □ parallel port connected to pushbutton KEYs in the DE1-SoC Computer

Address	31 · · · 10	9	8	7 · · · 1	0	Register name
0xFFFEC100	Unused				E	ICCICR
0xFFFEC104	Unused			Priority		ICCPMR
0xFFFEC10C	Unused			Interrupt ID		ICCIAR
0xFFFEC110	Unused			Interrupt ID		ICCEOIR

CPU register

Base Address	31 · · · 24	23 · · · 16	15 · · · 8	7	6	5	4	3	2	1	0	Register name
0xFFFED000	Unused										E	ICDDCR
0xFFFED100	Set-enable bits											ICDISERn
...	Set-enable bits											...
0xFFFED180	Clear-enable bits											ICDICERn
...	Clear-enable bits											...
0xFFFED400	Priority, offset 3	Priority, offset 2	Priority, offset 1	Priority, offset 0								ICDIPRn
...	Priority, offset 3	Priority, offset 2	Priority, offset 1	Priority, offset 0								...
0xFFFED800	CPUs, offset 3	CPUs, offset 2	CPUs, offset 1	CPUs, offset 0								ICDIPTRn
...	CPUs, offset 3	CPUs, offset 2	CPUs, offset 1	CPUs, offset 0								...
0xFFFEDC00	F15	F14	F13	F12	F11	F10	F9	F8	F7	F6	F5	F4	F3	F2	F1	F0	ICDICFRn
...	F15	F14	F13	F12	F11	F10	F9	F8	F7	F6	F5	F4	F3	F2	F1	F0	...

Dist. Reg.

```
/*
 * Configure the Generic Interrupt Controller (GIC)
 */
                .global     CONFIG_GIC
CONFIG_GIC:
                PUSH        {LR}
                /* CONFIG_INTERRUPT (int_ID (R0), CPU_target (R1)); */
                MOV         R0, #73             // KEYs parallel port (Interrupt ID = 73)
                MOV         R1, #1              // this field is a bit-mask; bit 0 targets cpu0
                BL          CONFIG_INTERRUPT

                /* Configure the GIC CPU Interface */
                LDR         R0, =0xFFFEC100     // base address of CPU Interface
                /* Set the Interrupt Priority Mask Register (ICCPMR) */
                LDR         R1, =0xFFFF         // enable interrupts of all priorities levels
                STR         R1, [R0, #0x04]

                /* Set the enable bit in the CPU Interface Control Register (ICCICR) */
                MOV         R1, #1
                STR         R1, [R0]

                /* Set the enable bit in the Distributor Control Register (ICDDCR) */
                LDR         R0, =0xFFFED000
                STR         R1, [R0]
                POP         {PC}
```

```
* Configure Set Enable Registers (ICDISERn) and Interrupt Processor Target Registers (ICDIPTRn).
* The default (reset) values are used for other registers in the GIC.
* Arguments: R0 holds the Interrupt ID (N), and R1 holds the CPU target
*/
CONFIG_INTERRUPT:
        PUSH    {R4-R5, LR}
        /* Configure Interrupt Set-Enable Registers (ICDISERn).
         * reg_offset = (integer_div(N / 32) * 4; value = 1 << (N mod 32) */
        LSR     R4, R0, #3              // calculate reg_offset
        BIC     R4, R4, #3              // R4 = reg_offset
        LDR     R2, =0xFFFED100
        ADD     R4, R2, R4              // R4 = address of ICDISER

        AND     R2, R0, #0x1F           // N mod 32
        MOV     R5, #1                  // enable
        LSL     R2, R5, R2              // R2 = value
        /* using address in R4 and value in R2 set the correct bit in the GIC register */
        LDR     R3, [R4]                // read current register value
        ORR     R3, R3, R2              // set the enable bit
        STR     R3, [R4]                // store the new register value

        /* Configure Interrupt Processor Targets Register (ICDIPTRn).
         * reg_offset = integer_div(N / 4) * 4; index = N mod 4 */
        BIC     R4, R0, #3              // R4 = reg_offset
        LDR     R2, =0xFFFED800
        ADD     R4, R2, R4              // R4 = word address of ICDIPTR
        AND     R2, R0, #0x3            // N mod 4
        ADD     R4, R2, R4              // R4 = byte address in ICDIPTR
        /* using address in R4 and value in R2, write to (only) the appropriate byte */
        STRB    R1, [R4]
        POP     {R4-R5, PC}
```

Microprocessors

Tuba Ayhan

MEF University

Interrupt Handling – with GIC

ARM Generic Interrupt Controller Architecture Specification

Interrupt states

Inactive	An interrupt that is not active or pending
Pending	An interrupt from a source to the GIC that is recognized as asserted in hardware, or generated by software, and is waiting to be serviced by a target processor
Active	An interrupt from a source to the GIC that has been acknowledged by a processor, and is being serviced but has not completed
Active and pending	A processor is servicing the interrupt and the GIC has a pending interrupt from the same source

Interrupt types and sources

- Software Generated Interrupt (SGI):
  - Generated by software writing to the Software Generated Interrupt Register GICD_SGIR.
  - A maximum of 16 SGIs can be generated for each Cortex-A9 processor interface.
  - The system uses SGIs for interprocessor communication.

Peripheral interrupt: This is an interrupt asserted by a signal to the GIC.

- Private Peripheral Interrupt (PPI):
  - An interrupt generated by a peripheral that is specific to a single Cortex-A9 processor.
  - There are 5 PPIs for each Cortex-A9 processor interface.
- Shared Peripheral Interrupt (SPI):
  - An interrupt generated by a peripheral that the Interrupt Controller can route to any, or all, Cortex-A9 processor interfaces.

Peripheral interrupt

- Edge-triggered

This is an interrupt that is asserted on detection of a rising edge of an interrupt signal and then, regardless of the state of the signal, remains asserted until it is cleared by the conditions defined by this specification.

- Level-sensitive

This is an interrupt that is asserted whenever the interrupt signal level is active, and deasserted whenever the level is not active.

Spurious interrupts

- The GIC has signaled to a processor is no longer required.
- If this happens, when the processor acknowledges the interrupt, the GIC returns a special Interrupt ID that identifies the interrupt as a **spurious interrupt**. I.e.:

1. Prior to the processor acknowledging an interrupt:
   - software changes the priority of the interrupt
   - software disables the interrupt
   - software changes the processor that the interrupt targets
2. Another target processor has previously acknowledged that interrupt.

Banking

- Interrupt banking
  - In a multiprocessor implementation, for PPIs and SGIs, the GIC can have multiple interrupts with the same interrupt ID. Such an interrupt is called a banked interrupt, and is identified uniquely by the combination of its interrupt ID and its associated CPU interface.
- Register banking
  - Register banking refers to implementing multiple copies of a register at the same address.
  - This occurs in a multiprocessor implementation, to provide separate copies for each processor of registers corresponding to banked interrupts

GIC logic

Distributor

- interrupt prioritization
- distribution to the CPU interface blocks
- Registers: GICD_ prefix

CPU interfaces

- connection to the processors in the system.
- priority masking
- preemption handling for a connected processor in the system.
- Registers: GICC_ prefix.

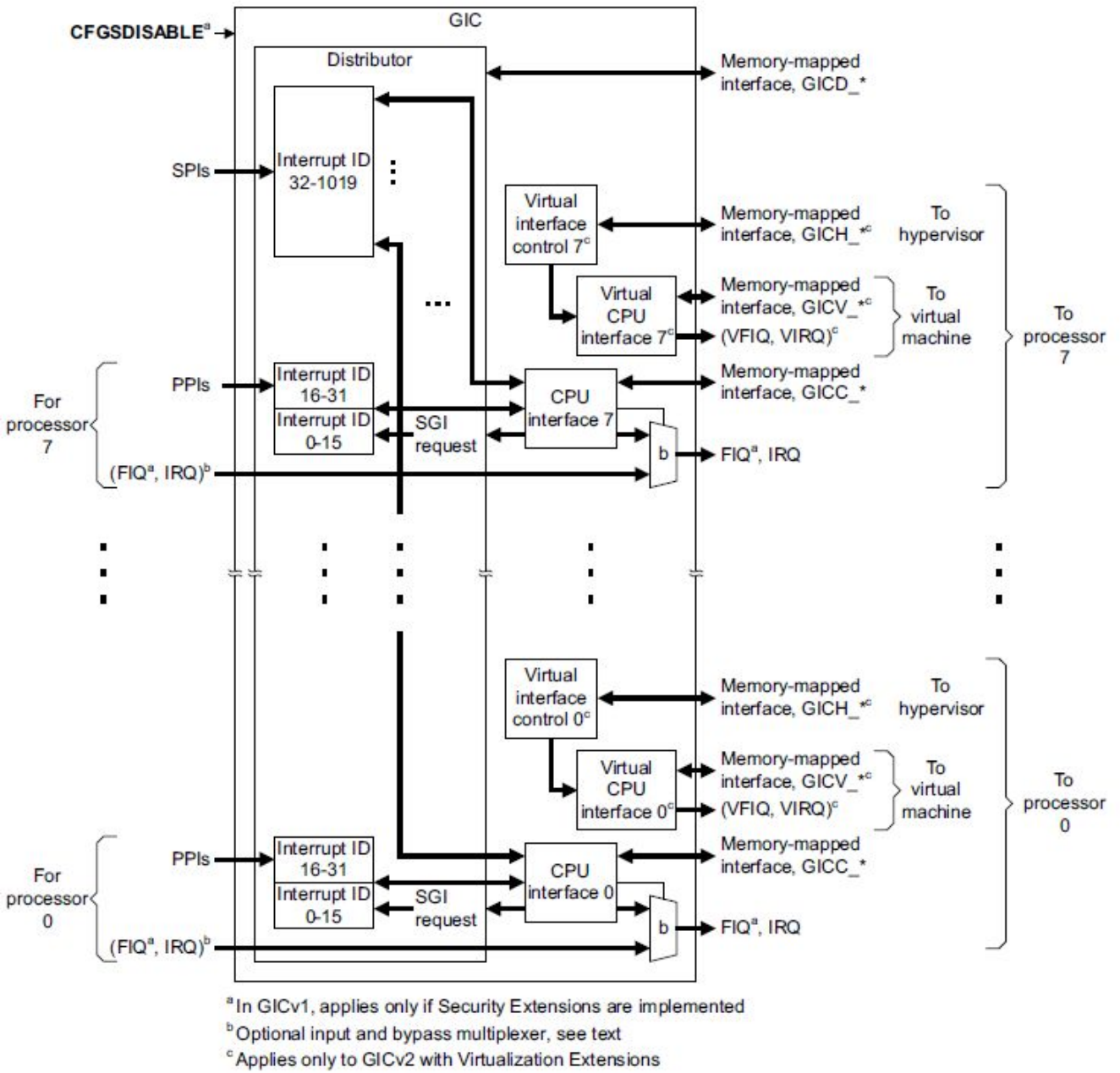


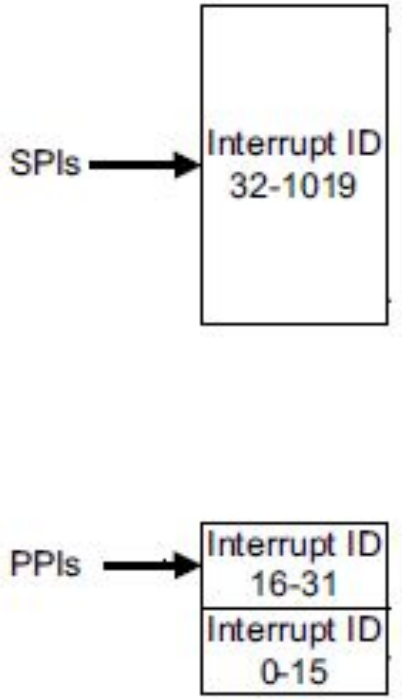
Figure 2-1 GIC logical partitioning

The Distributor

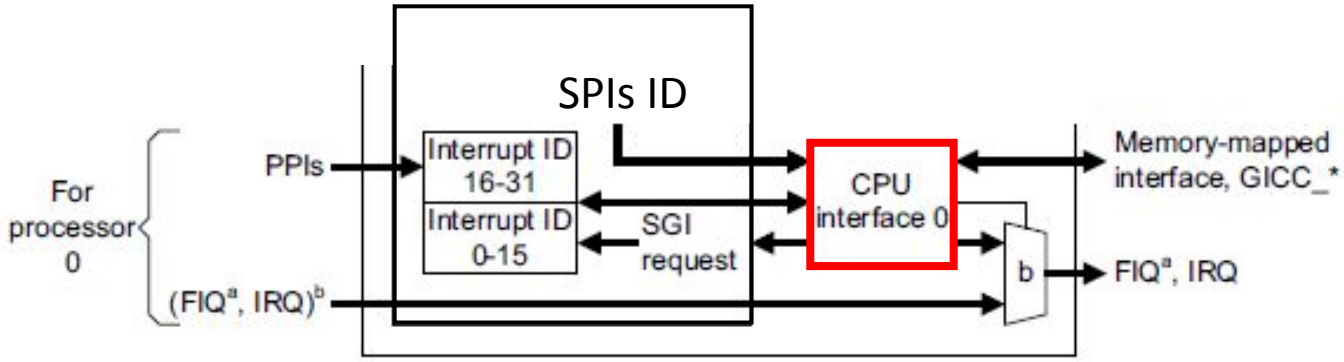
- The Distributor centralizes all interrupt sources, determines the priority of each interrupt, and for each CPU interface forwards the interrupt with the highest priority to the interface, for priority masking and preemption handling.
- The Distributor provides a programming interface for:
  - Globally enabling the forwarding of interrupts to the CPU interfaces.
  - Enabling or disabling each interrupt.
  - Setting the priority level of each interrupt.
  - Setting the target processor list of each interrupt.
  - Setting each peripheral interrupt to be level-sensitive or edge-triggered.
- In addition, the Distributor provides:
  - visibility of the state of each interrupt
  - a mechanism for software to set or clear the pending state of a peripheral interrupt.

Interrupt IDs

- Each CPU interface can see up to 1020 interrupts
- The banking of SPIs and PPIs increases the total number of interrupts supported by the Distributor.



CPU interfaces



- enabling the signaling of interrupt requests to the processor
- acknowledging an interrupt
- indicating completion of the processing of an interrupt
- setting an interrupt priority mask for the processor
- defining the preemption policy for the processor
- determining the highest priority pending interrupt for the processor.

General handling of interrupts

- When the GIC recognizes an interrupt request, it marks its state as pending. Regenerating a pending interrupt does not affect the state of the interrupt.

The GIC interrupt handling sequence is:

1. The GIC determines the interrupts that are enabled.
2. For each pending interrupt, the GIC determines the targeted processor or processors.
3. For each CPU interface, the Distributor forwards the highest priority pending interrupt that targets that interface.
4. Each CPU interface determines whether to signal an interrupt request to its processor, and if required, does so.
5. The processor acknowledges the interrupt, and the GIC returns the interrupt ID and updates the interrupt state.
6. After processing the interrupt, the processor signals End of Interrupt (EOI) to the GIC.

Interrupt handling state machine

- Transition **A**, add pending state:
  - i.e. a peripheral asserts an interrupt request signal
- Transition **B**, remove pending state
  - i.e. the level-sensitive interrupt is pending only because of the assertion of an input signal, and that signal is deasserted
- Transition **C**, pending to active
  - i.e. the interrupt is enabled and has Sufficient priority to be signaled to the processor.
- Transition **D**, pending to active and pending
  - i.e. The interrupt is enabled.
- Transition **E**, remove active state
  - i.e. software deactivates an interrupt

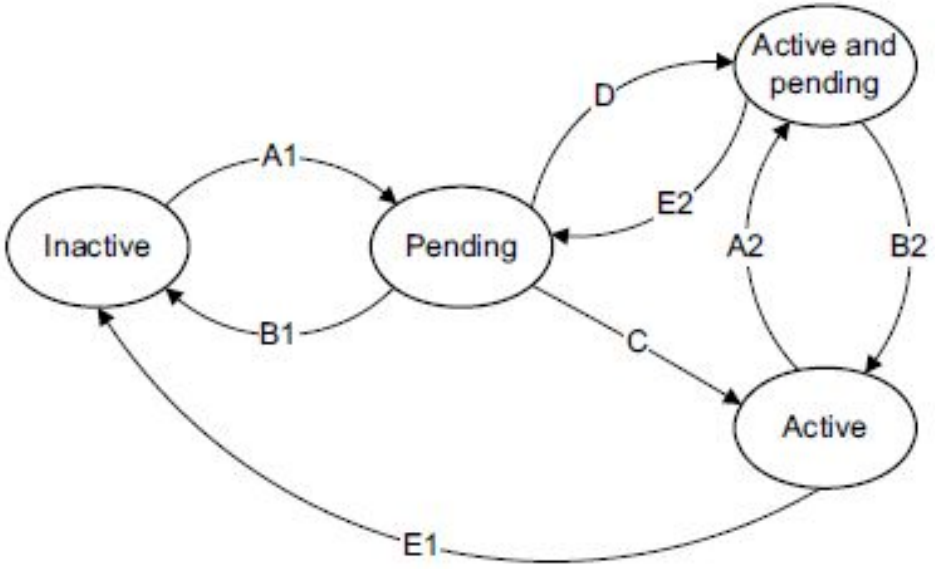


Figure 3-1 Interrupt handling state machine

Interrupt handling

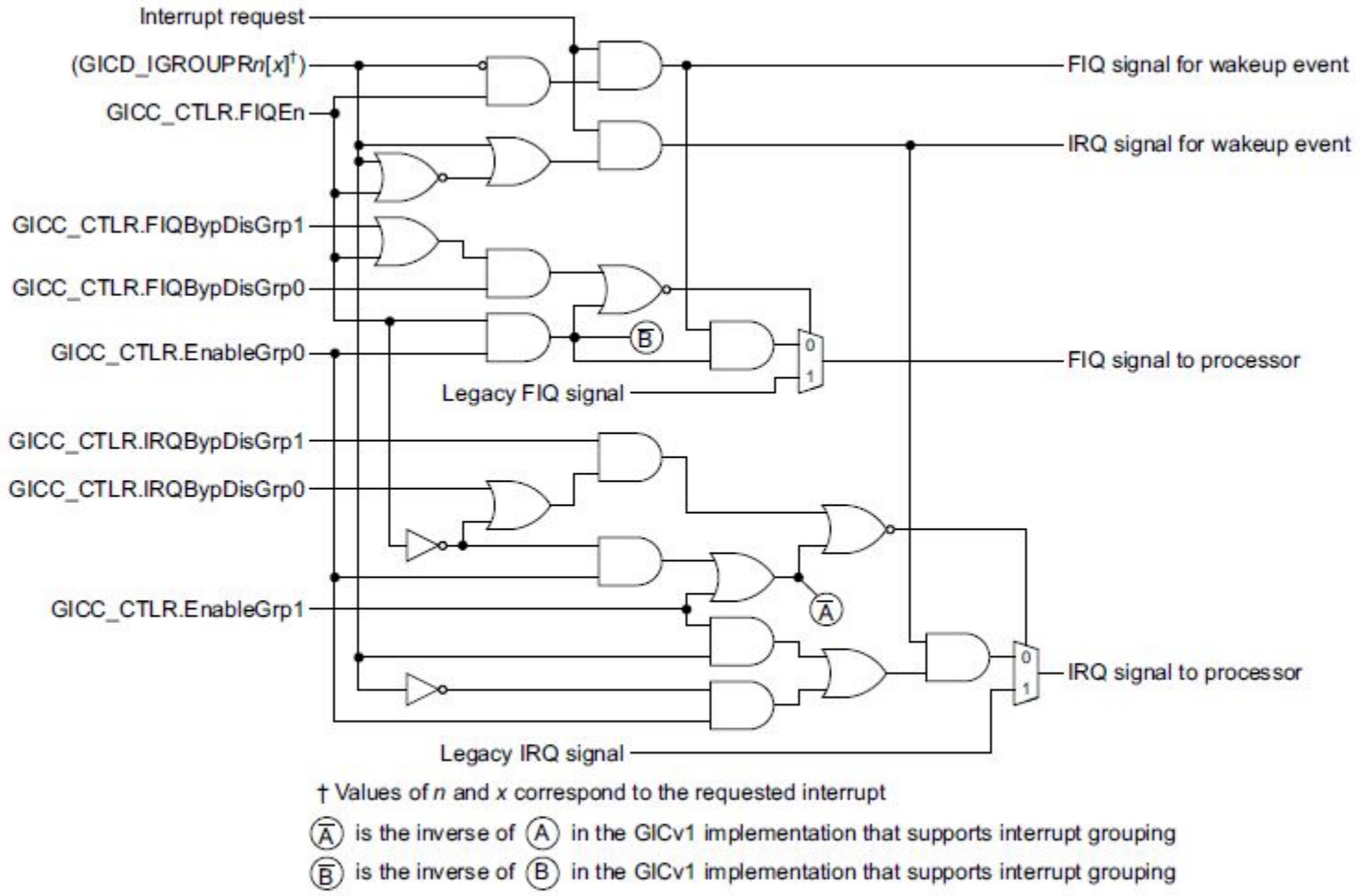


Figure 2-4 GICv2 interrupt bypass logic, with bypass disable

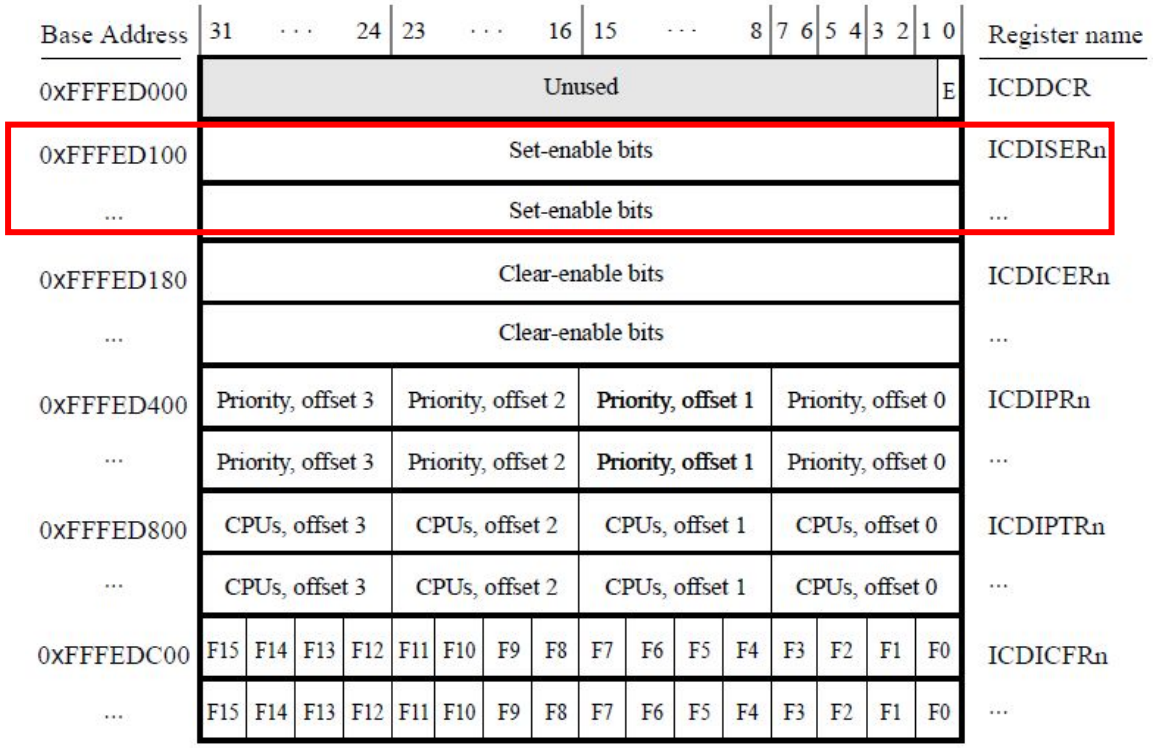
Example of Assembly Language Code

- Enable Interrupt ID **73** □ parallel port connected to pushbutton KEYs in the DE1-SoC Computer

Base Address	31 ··· 24	23 ··· 16	15 ··· 8	7 6	5 4	3 2	1 0	Register name
0xFFFED000	Unused						E	ICDDCR
0xFFFED100	Set-enable bits							ICDISERn
...	Set-enable bits							...
0xFFFED180	Clear-enable bits							ICDICERn
...	Clear-enable bits							...
0xFFFED400	Priority, offset 3	Priority, offset 2	**Priority, offset 1**	Priority, offset 0				ICDIPRn
...	Priority, offset 3	Priority, offset 2	**Priority, offset 1**	Priority, offset 0				...
0xFFFED800	CPUs, offset 3	CPUs, offset 2	CPUs, offset 1	CPUs, offset 0				ICDIPTRn
...	CPUs, offset 3	CPUs, offset 2	CPUs, offset 1	CPUs, offset 0				...
0xFFFEDC00	F15	F14	F13	F12	F11	F10	F9	F8	F7	F6	F5	F4	F3	F2	F1	F0	ICDICFRn
...	F15	F14	F13	F12	F11	F10	F9	F8	F7	F6	F5	F4	F3	F2	F1	F0	...

Dist. Reg.

Interrupt ID **73** **a. ICDISERn**



Base Address	31 ··· 24	23 ··· 16	15 ··· 8	7 6	5 4	3 2	1 0	Register name
0xFFFED000	Unused						E	ICDDCR
0xFFFED100	Set-enable bits							ICDISERn
...	Set-enable bits							...
0xFFFED180	Clear-enable bits							ICDICERn
...	Clear-enable bits							...
0xFFFED400	Priority, offset 3	Priority, offset 2	Priority, offset 1	Priority, offset 0				ICDIPRn
...	Priority, offset 3	Priority, offset 2	Priority, offset 1	Priority, offset 0				...
0xFFFED800	CPUs, offset 3	CPUs, offset 2	CPUs, offset 1	CPUs, offset 0				ICDIPTRn
...	CPUs, offset 3	CPUs, offset 2	CPUs, offset 1	CPUs, offset 0				...
0xFFFEDC00	F15 F14 F13 F12	F11 F10 F9 F8	F7 F6 F5 F4	F3 F2	F1 F0			ICDICFRn
...	F15 F14 F13 F12	F11 F10 F9 F8	F7 F6 F5 F4	F3 F2	F1 F0			...

Dist. Reg.

- The Interrupt Set Enable Registers (ICDISERn) are used to **enable the forwarding of each supported interrupt** from the Distributor to the CPU Interface.
- The **n** postfix in the name ICDISERn means that multiple registers exist.
- The set-enable bits for the first 32 Interrupt IDs are provided in the register at address 0xFFFED100, the next 32 are provided in the register at the following word address, which is 0xFFFED104, and so on.
- Given a specific Interrupt ID, *N*, the address of the register that contains its set-enable bit is given by the integer calculation

$$address = 0xFFFED100 + (N / 32)*4,$$

and the index of the bit inside this register is given by

$$index = N \bmod 32.$$

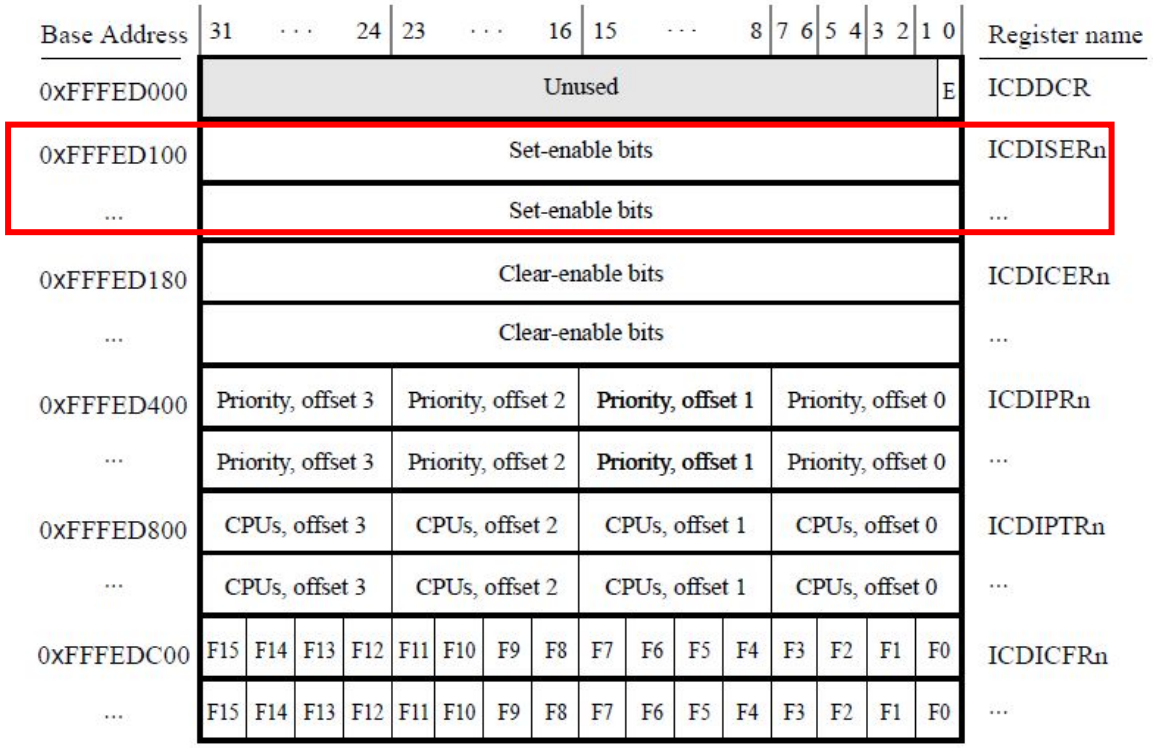
Writing the value 1 into a set-enable bit enables the forwarding of the corresponding IRQ to the CPU Interface.

- Example: N = 73 □

address = 0xFFFED100 + (3*4) = **0xFFFED10C**

Index = 73 mod 32 = **9**

Interrupt ID **73** **a. ICDISERn**



Base Address	31 ··· 24	23 ··· 16	15 ··· 8	7 6	5 4	3 2	1 0	Register name
0xFFFED000	Unused						E	ICDDCR
0xFFFED100	Set-enable bits							ICDISERn
...	Set-enable bits							...
0xFFFED180	Clear-enable bits							ICDICERn
...	Clear-enable bits							...
0xFFFED400	Priority, offset 3	Priority, offset 2	Priority, offset 1	Priority, offset 0				ICDIPRn
...	Priority, offset 3	Priority, offset 2	Priority, offset 1	Priority, offset 0				...
0xFFFED800	CPUs, offset 3	CPUs, offset 2	CPUs, offset 1	CPUs, offset 0				ICDIPTRn
...	CPUs, offset 3	CPUs, offset 2	CPUs, offset 1	CPUs, offset 0				...
0xFFFEDC00	F15 F14 F13 F12	F11 F10 F9 F8	F7 F6 F5 F4	F3 F2	F1	F0		ICDICFRn
...	F15 F14 F13 F12	F11 F10 F9 F8	F7 F6 F5 F4	F3 F2	F1	F0		...

Dist. Reg.

- The Interrupt Set Enable Registers (ICDISERn) are used to **enable the forwarding of each supported interrupt** from the Distributor to the CPU Interface.
- The **n** postfix in the name ICDISERn means that multiple registers exist.
- The set-enable bits for the first 32 Interrupt IDs are provided in the register at address 0xFFFED100, the next 32 are provided in the register at the following word address, which is 0xFFFED104, and so on.
- Given a specific Interrupt ID, *N*, the address of the register that contains its set-enable bit is given by the integer calculation

$$address = 0xFFFED100 + (N / 32)*4,$$

and the index of the bit inside this register is given by

$$index = N \bmod 32.$$

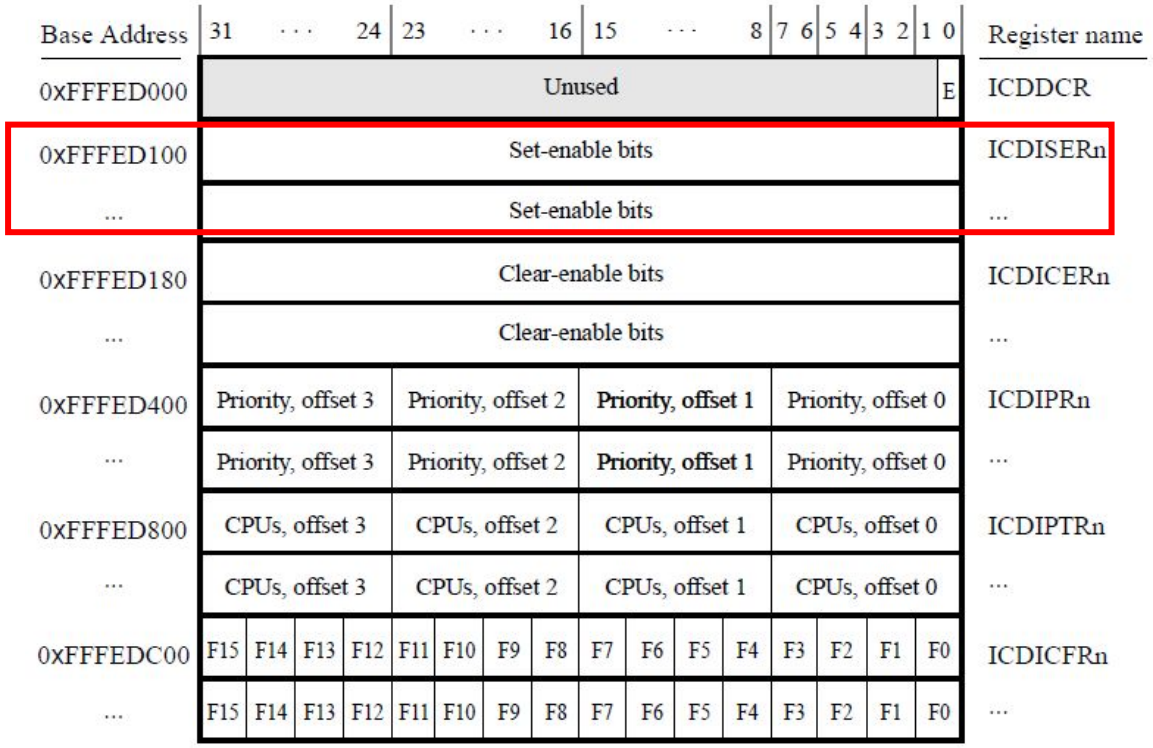
Writing the value 1 into a set-enable bit enables the forwarding of the corresponding IRQ to the CPU Interface.

- Example: N = 73 □

address =

Index =

Interrupt ID **73** **a. ICDISERn**



Base Address	31 ··· 24	23 ··· 16	15 ··· 8	7 6	5 4	3 2	1 0	Register name
0xFFFED000	Unused						E	ICDDCR
0xFFFED100	Set-enable bits							ICDISERn
...	Set-enable bits							...
0xFFFED180	Clear-enable bits							ICDICERn
...	Clear-enable bits							...
0xFFFED400	Priority, offset 3	Priority, offset 2	Priority, offset 1	Priority, offset 0				ICDIPRn
...	Priority, offset 3	Priority, offset 2	Priority, offset 1	Priority, offset 0				...
0xFFFED800	CPUs, offset 3	CPUs, offset 2	CPUs, offset 1	CPUs, offset 0				ICDIPTRn
...	CPUs, offset 3	CPUs, offset 2	CPUs, offset 1	CPUs, offset 0				...
0xFFFEDC00	F15 F14 F13 F12	F11 F10 F9 F8	F7 F6 F5 F4	F3 F2	F1 F0			ICDICFRn
...	F15 F14 F13 F12	F11 F10 F9 F8	F7 F6 F5 F4	F3 F2	F1 F0			...

Dist. Reg.

- The Interrupt Set Enable Registers (ICDISERn) are used to **enable the forwarding of each supported interrupt** from the Distributor to the CPU Interface.
- The **n** postfix in the name ICDISERn means that multiple registers exist.
- The set-enable bits for the first 32 Interrupt IDs are provided in the register at address 0xFFFED100, the next 32 are provided in the register at the following word address, which is 0xFFFED104, and so on.
- Given a specific Interrupt ID, *N*, the address of the register that contains its set-enable bit is given by the integer calculation

$$address = 0xFFFED100 + (N / 32)*4,$$

and the index of the bit inside this register is given by

$$index = N \bmod 32.$$

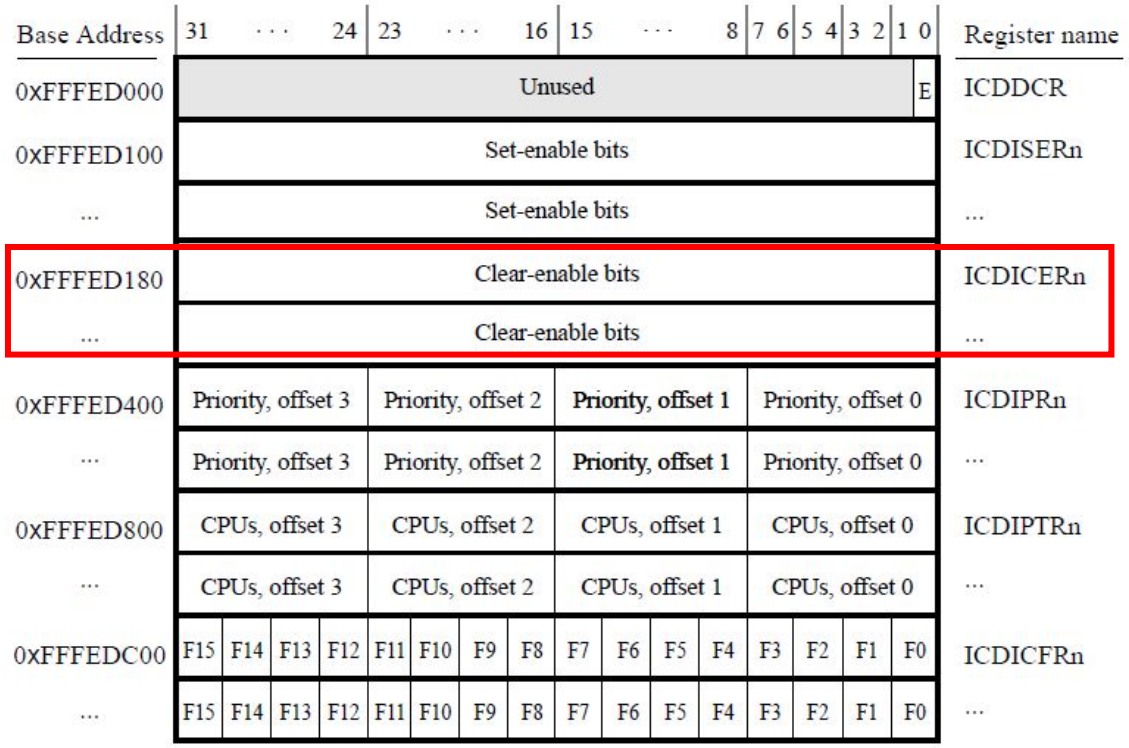
Writing the value 1 into a set-enable bit enables the forwarding of the corresponding IRQ to the CPU Interface.

- Example: N = 73 □

address = 0xFFFED100 + (3*4) = **0xFFFED10C**

Index = 73 mod 32 = **9**

Interrupt ID **73** **b. ICDICERn**

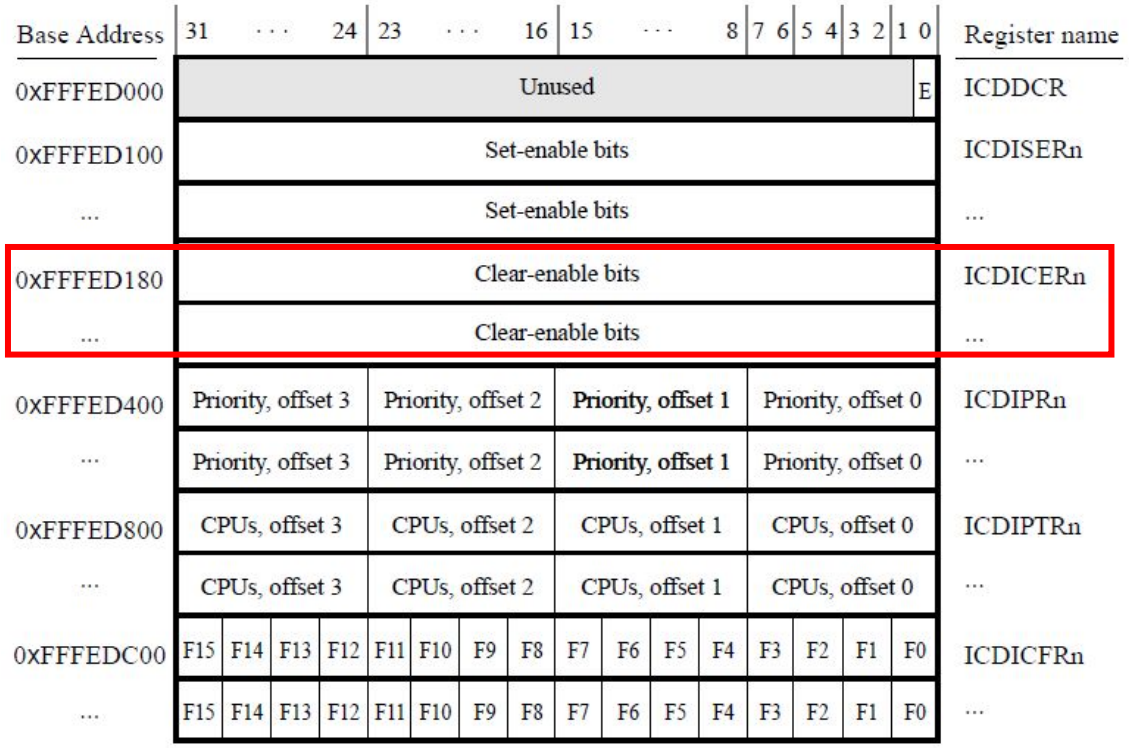


Base Address	31 ··· 24	23 ··· 16	15 ··· 8	7 6	5 4	3 2	1 0	Register name
0xFFFED000	Unused						E	ICDDCR
0xFFFED100	Set-enable bits							ICDISERn
...	Set-enable bits							...
0xFFFED180	Clear-enable bits							ICDICERn
...	Clear-enable bits							...
0xFFFED400	Priority, offset 3	Priority, offset 2	Priority, offset 1	Priority, offset 0				ICDIPRn
...	Priority, offset 3	Priority, offset 2	Priority, offset 1	Priority, offset 0				...
0xFFFED800	CPUs, offset 3	CPUs, offset 2	CPUs, offset 1	CPUs, offset 0				ICDIPTRn
...	CPUs, offset 3	CPUs, offset 2	CPUs, offset 1	CPUs, offset 0				...
0xFFFEDC00	F15 F14 F13 F12	F11 F10 F9 F8	F7 F6 F5 F4	F3 F2 F1 F0				ICDICFRn
...	F15 F14 F13 F12	F11 F10 F9 F8	F7 F6 F5 F4	F3 F2 F1 F0				...

Dist. Reg.

- In the same way that each supported interrupt can be enabled by using ICDISERn, each interrupt can be disabled by using the Interrupt Clear Enable Registers (ICDICERn). The method for calculating the address and index for ICDICERn is the same as that for ICDISERn, except that the base address is 0xFFFED180.

Interrupt ID **73** **b. ICDICERn**



Base Address	31 ··· 24	23 ··· 16	15 ··· 8	7 6 5 4 3 2 1 0	Register name
0xFFFED000	Unused			E	ICDDCR
0xFFFED100	Set-enable bits				ICDISERn
...	Set-enable bits				...
0xFFFED180	Clear-enable bits				ICDICERn
...	Clear-enable bits				...
0xFFFED400	Priority, offset 3	Priority, offset 2	Priority, offset 1	Priority, offset 0	ICDIPRn
...	Priority, offset 3	Priority, offset 2	Priority, offset 1	Priority, offset 0	...
0xFFFED800	CPUs, offset 3	CPUs, offset 2	CPUs, offset 1	CPUs, offset 0	ICDIPTRn
...	CPUs, offset 3	CPUs, offset 2	CPUs, offset 1	CPUs, offset 0	...
0xFFFEDC00	F15 F14 F13 F12	F11 F10 F9 F8	F7 F6 F5 F4	F3 F2 F1 F0	ICDICFRn
...	F15 F14 F13 F12	F11 F10 F9 F8	F7 F6 F5 F4	F3 F2 F1 F0	...

Dist. Reg.

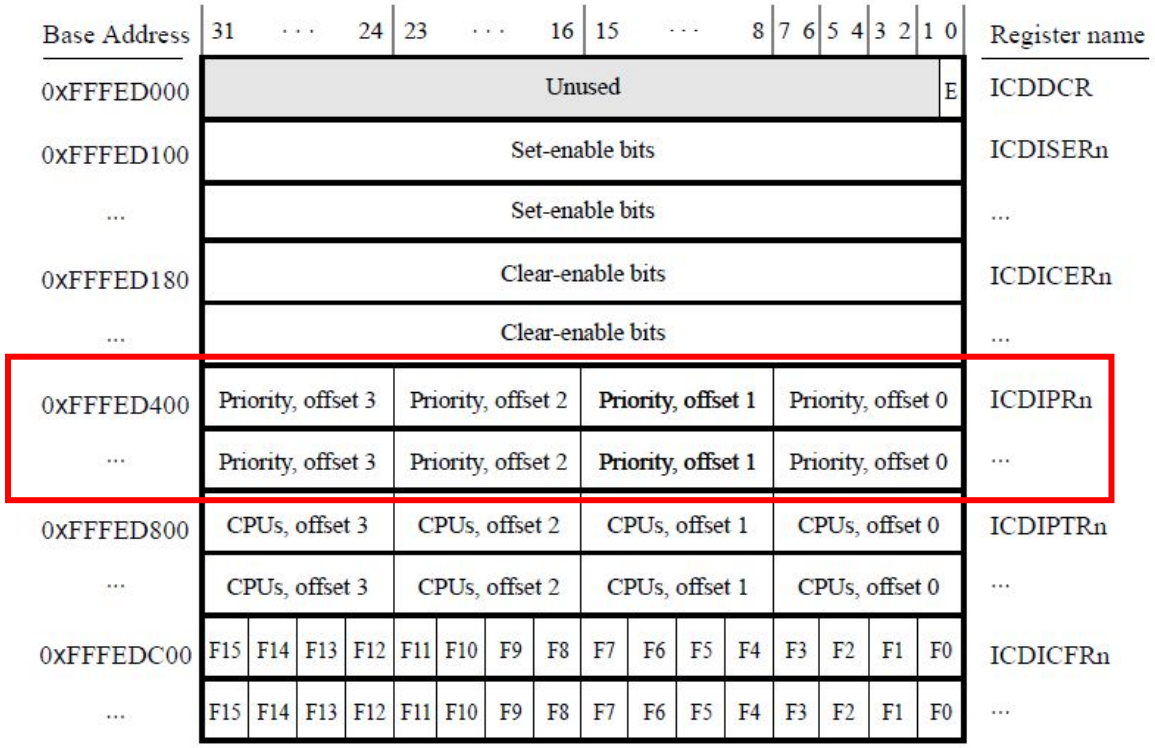
- In the same way that each supported interrupt can be enabled by using ICDISERn, each interrupt can be disabled by using the Interrupt Clear Enable Registers (ICDICERn). The method for calculating the address and index for ICDICERn is the same as that for ICDISERn, except that the base address is 0xFFFED180.

- Example: N = 73 □

address =

Index =

Interrupt ID **73** **c. ICDIPRn**



Base Address	31 ··· 24	23 ··· 16	15 ··· 8	7 6	5 4	3 2	1 0	Register name
0xFFFED000	Unused						E	ICDDCR
0xFFFED100	Set-enable bits							ICDISERn
...	Set-enable bits							...
0xFFFED180	Clear-enable bits							ICDICERn
...	Clear-enable bits							...
0xFFFED400	Priority, offset 3	Priority, offset 2	Priority, offset 1	Priority, offset 0				ICDIPRn
...	Priority, offset 3	Priority, offset 2	Priority, offset 1	Priority, offset 0				...
0xFFFED800	CPUs, offset 3	CPUs, offset 2	CPUs, offset 1	CPUs, offset 0				ICDIPTRn
...	CPUs, offset 3	CPUs, offset 2	CPUs, offset 1	CPUs, offset 0				...
0xFFFEDC00	F15 F14 F13 F12	F11 F10 F9 F8	F7 F6 F5 F4	F3 F2 F1 F0				ICDICFRn
...	F15 F14 F13 F12	F11 F10 F9 F8	F7 F6 F5 F4	F3 F2 F1 F0				...

Dist. Reg.

- The Interrupt Priority Registers (ICDIPRn) are used to associate a priority level with each individual interrupt. On reset, these registers are set to 0x00000000, which represents the highest priority. Each Interrupt ID's priority field is one byte in size, which means that the register at the base address holds the priority levels for Interrupt IDs from 0 to 3. The priority levels for the next four Interrupt IDs use the register at address 0xFFFED404, and so on.
- Setting the priority field for an Interrupt ID to a larger number results in lower priority for the corresponding interrupt.

$$address = 0xFFFED400 + (N / 4)*4$$

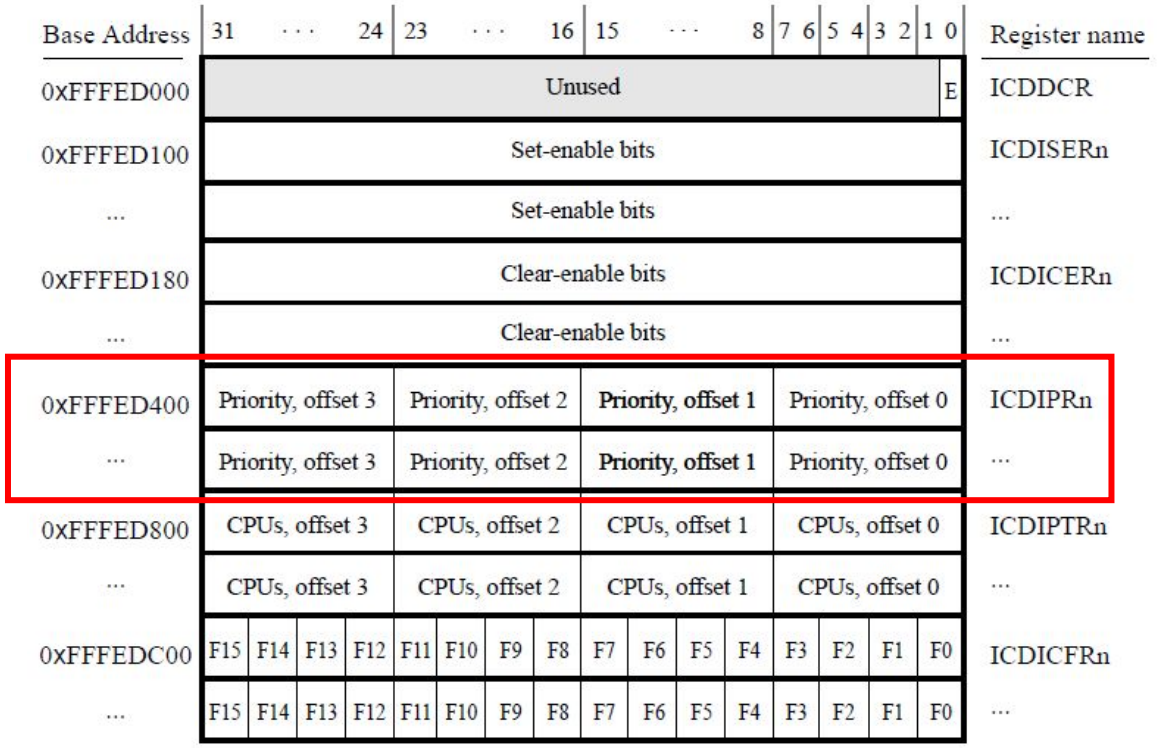
$$offset = N \bmod 4.$$

- Example: N = 73 □

address =

Offset =

Interrupt ID **73** **c. ICDIPRn**



Base Address	31 ··· 24	23 ··· 16	15 ··· 8	7 6	5 4	3 2	1 0	Register name
0xFFFED000	Unused						E	ICDDCR
0xFFFED100	Set-enable bits							ICDISERn
...	Set-enable bits							...
0xFFFED180	Clear-enable bits							ICDICERn
...	Clear-enable bits							...
0xFFFED400	Priority, offset 3	Priority, offset 2	Priority, offset 1	Priority, offset 0				ICDIPRn
...	Priority, offset 3	Priority, offset 2	Priority, offset 1	Priority, offset 0				...
0xFFFED800	CPUs, offset 3	CPUs, offset 2	CPUs, offset 1	CPUs, offset 0				ICDIPTRn
...	CPUs, offset 3	CPUs, offset 2	CPUs, offset 1	CPUs, offset 0				...
0xFFFEDC00	F15 F14 F13 F12	F11 F10 F9 F8	F7 F6 F5 F4	F3 F2	F1	F0		ICDICFRn
...	F15 F14 F13 F12	F11 F10 F9 F8	F7 F6 F5 F4	F3 F2	F1	F0		...

Dist. Reg.

- The Interrupt Priority Registers (ICDIPRn) are used to associate a priority level with each individual interrupt. On reset, these registers are set to 0x00000000, which represents the highest priority. Each Interrupt ID's priority field is one byte in size, which means that the register at the base address holds the priority levels for Interrupt IDs from 0 to 3. The priority levels for the next four Interrupt IDs use the register at address 0xFFFED404, and so on.
- Setting the priority field for an Interrupt ID to a larger number results in lower priority for the corresponding interrupt.

$$address = 0xFFFED400 + (N / 4)*4$$

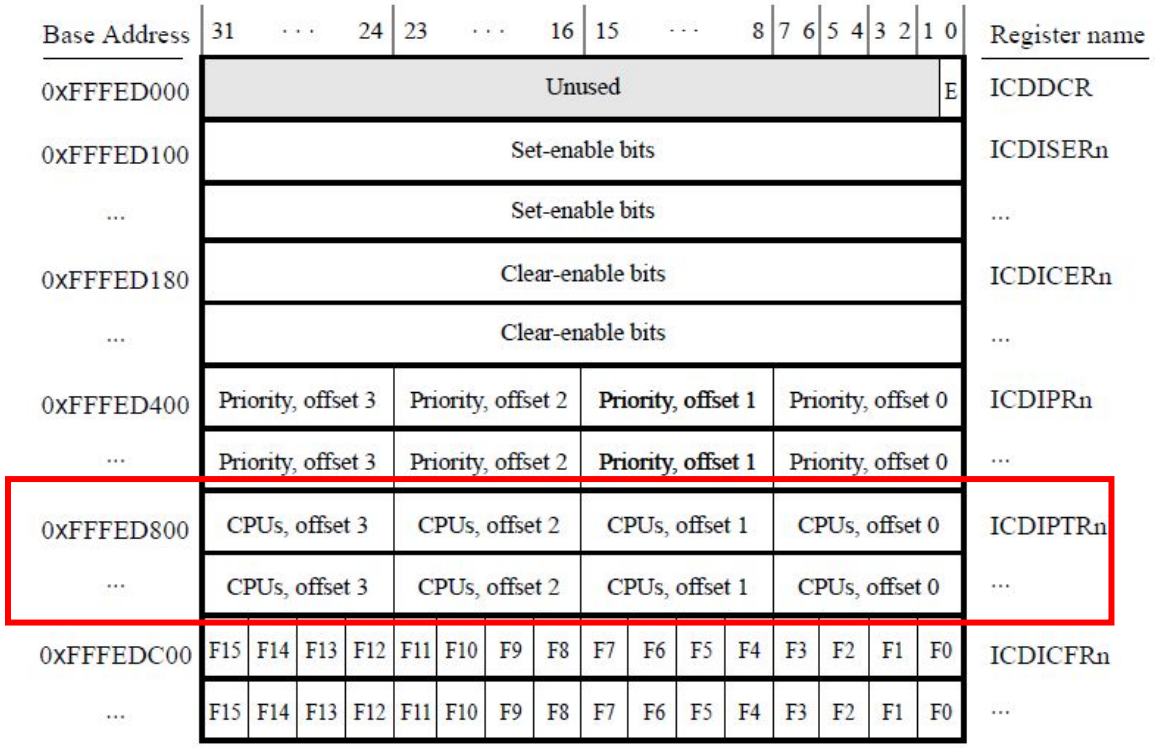
$$offset = N \bmod 4.$$

- Example: N = 73 □

address = 0xFFFED400 + (18*4) = **0xFFFED448**

Offset = 73 mod 4 = **1**

Interrupt ID **73** **d. ICDIPTRn**



Base Address	31 ··· 24	23 ··· 16	15 ··· 8	7 6	5 4	3 2	1 0	Register name
0xFFFED000	Unused						E	ICDDCR
0xFFFED100	Set-enable bits							ICDISERn
...	Set-enable bits							...
0xFFFED180	Clear-enable bits							ICDICERn
...	Clear-enable bits							...
0xFFFED400	Priority, offset 3	Priority, offset 2	Priority, offset 1	Priority, offset 0				ICDIPRn
...	Priority, offset 3	Priority, offset 2	Priority, offset 1	Priority, offset 0				...
0xFFFED800	CPUs, offset 3	CPUs, offset 2	CPUs, offset 1	CPUs, offset 0				ICDIPTRn
...	CPUs, offset 3	CPUs, offset 2	CPUs, offset 1	CPUs, offset 0				...
0xFFFEDC00	F15 F14 F13 F12	F11 F10 F9 F8	F7 F6 F5 F4	F3 F2	F1 F0			ICDICFRn
...	F15 F14 F13 F12	F11 F10 F9 F8	F7 F6 F5 F4	F3 F2	F1 F0			...

Dist. Reg.

- The Interrupt Processor Targets Registers (ICDIPTRn) are used to specify the CPU interfaces to which each interrupt. the CPUs field for each Interrupt ID is one byte in size. This size is used because some versions of the ARM A9 MPCORE have up to eight A9 cores. A target CPU is selected by setting its corresponding bit field to 1. Thus, setting the byte at address 0xFFFED800 to the value 0x01 would target Interrupt ID 0 to CPU 0, setting this same byte to 0x02 would target CPU 1, and setting the byte to the value 0x03 would target both CPU 0 and CPU 1. The scheme for calculating the address of the ICDIPTRn register for a specific Interrupt ID, and also its byte index, is the same as the one shown above for ICDIPRn.

$$address = 0xFFFED800 + (N / 4)*4$$

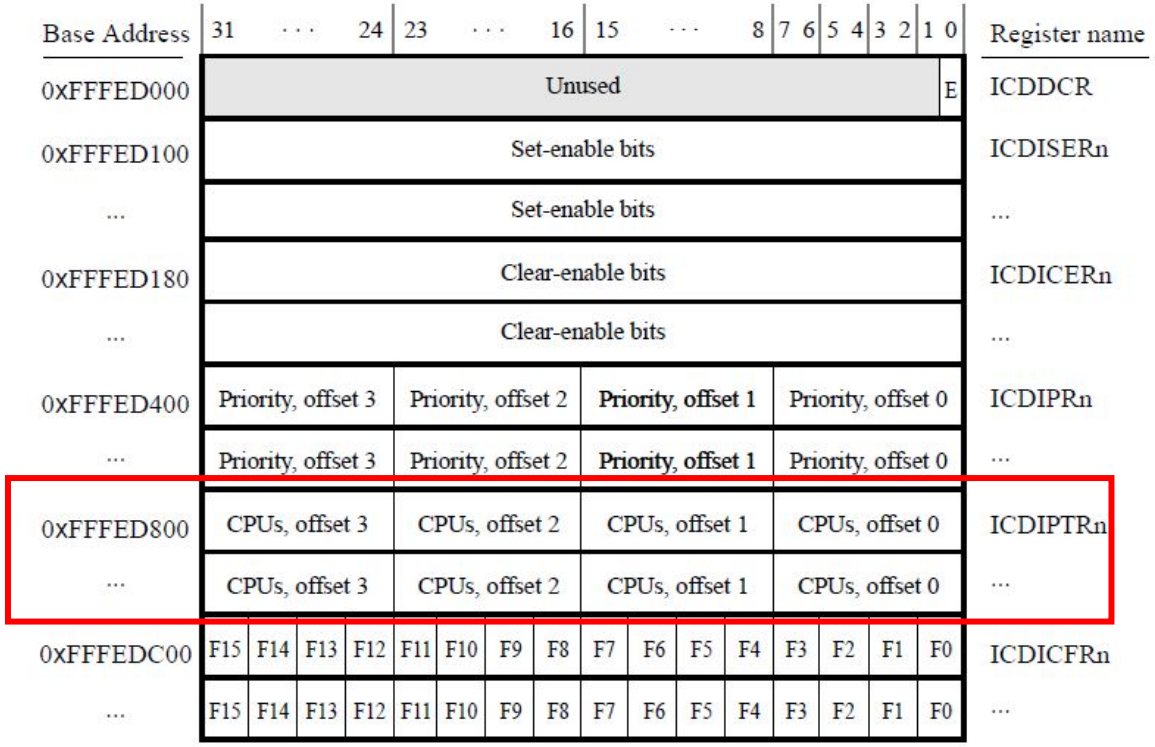
$$offset = N \bmod 4.$$

- Example: N = 73 □

address =

Offset =

Interrupt ID **73** **d. ICDIPTRn**



Base Address	31 ··· 24	23 ··· 16	15 ··· 8	7 6	5 4	3 2	1 0	Register name
0xFFFED000	Unused						E	ICDDCR
0xFFFED100	Set-enable bits							ICDISERn
...	Set-enable bits							...
0xFFFED180	Clear-enable bits							ICDICERn
...	Clear-enable bits							...
0xFFFED400	Priority, offset 3	Priority, offset 2	Priority, offset 1	Priority, offset 0				ICDIPRn
...	Priority, offset 3	Priority, offset 2	Priority, offset 1	Priority, offset 0				...
0xFFFED800	CPUs, offset 3	CPUs, offset 2	CPUs, offset 1	CPUs, offset 0				ICDIPTRn
...	CPUs, offset 3	CPUs, offset 2	CPUs, offset 1	CPUs, offset 0				...
0xFFFEDC00	F15 F14	F13 F12	F11 F10	F9 F8	F7 F6	F5 F4	F3 F2 F1 F0	ICDICFRn
...	F15 F14	F13 F12	F11 F10	F9 F8	F7 F6	F5 F4	F3 F2 F1 F0	...

Dist. Reg.

- The Interrupt Processor Targets Registers (ICDIPTRn) are used to specify the CPU interfaces to which each interrupt. the CPUs field for each Interrupt ID is one byte in size. This size is used because some versions of the ARM A9 MPCORE have up to eight A9 cores. A target CPU is selected by setting its corresponding bit field to 1. Thus, setting the byte at address 0xFFFED800 to the value 0x01 would target Interrupt ID 0 to CPU 0, setting this same byte to 0x02 would target CPU 1, and setting the byte to the value 0x03 would target both CPU 0 and CPU 1. The scheme for calculating the address of the ICDIPTRn register for a specific Interrupt ID, and also its byte index, is the same as the one shown above for ICDIPRn.

$$address = 0xFFFED800 + (N / 4)*4$$

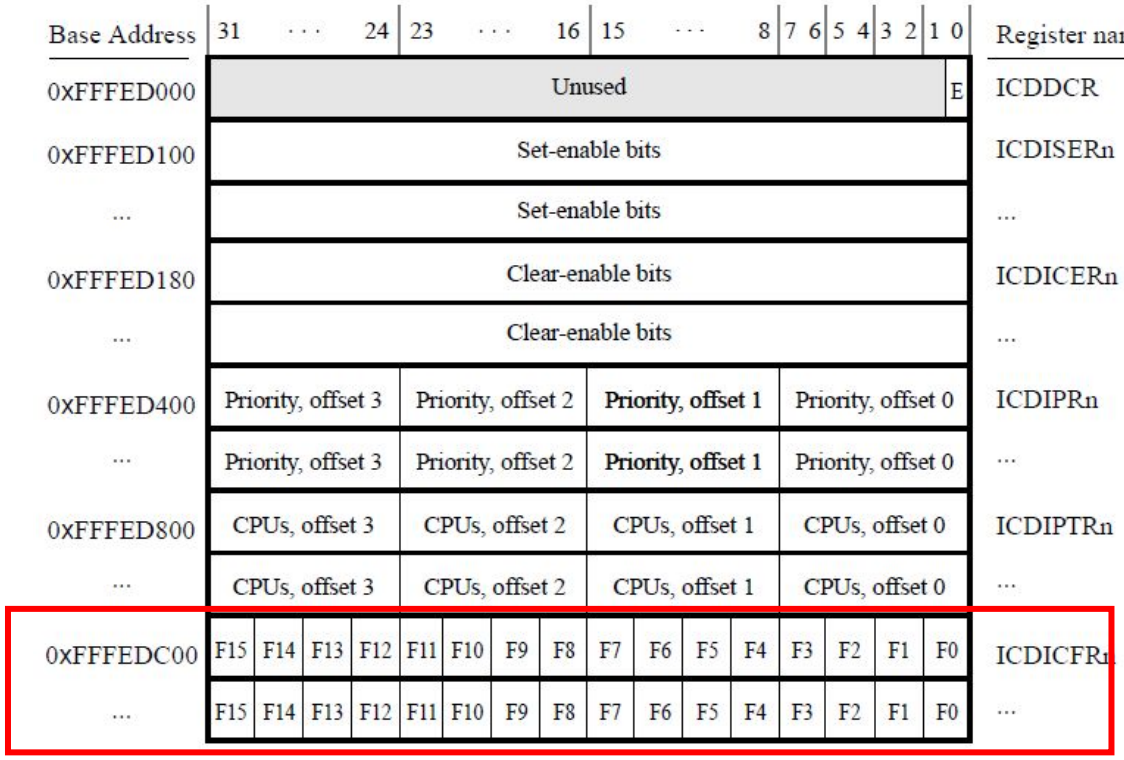
$$offset = N \bmod 4.$$

- Example: N = 73 □

address = 0xFFFED800 + (18*4) = **0xFFFED848**

Offset = 73 mod 4 = **1**

Interrupt ID **73** **e. ICDICFRn**



Base Address	31 ··· 24	23 ··· 16	15 ··· 8	7 6 5 4 3 2 1 0	Register name
0xFFFED000	Unused			E	ICDDCR
0xFFFED100	Set-enable bits				ICDISERn
...	Set-enable bits				...
0xFFFED180	Clear-enable bits				ICDICERn
...	Clear-enable bits				...
0xFFFED400	Priority, offset 3	Priority, offset 2	Priority, offset 1	Priority, offset 0	ICDIPRn
...	Priority, offset 3	Priority, offset 2	Priority, offset 1	Priority, offset 0	...
0xFFFED800	CPUs, offset 3	CPUs, offset 2	CPUs, offset 1	CPUs, offset 0	ICDIPTRn
...	CPUs, offset 3	CPUs, offset 2	CPUs, offset 1	CPUs, offset 0	...
0xFFFEDC00	F15 F14 F13 F12 F11 F10 F9 F8	F7 F6 F5 F4 F3 F2 F1 F0			ICDICFRn
...	F15 F14 F13 F12 F11 F10 F9 F8	F7 F6 F5 F4 F3 F2 F1 F0			...

Dist. Reg.

- The Interrupt Configuration Registers (ICDICFRn) are used to specify whether each supported interrupt should be handled as level- or edge-sensitive by the GIC. There is a **two-bit** field associated with each Interrupt ID. The least-significant bit in this field is not used. Setting the most-significant bit of this field to 1 makes the corresponding interrupt signal edge-sensitive, and setting this field to 0 makes it level-sensitive.
- The first 16 Interrupt IDs use the ICDICFRn register at address 0xFFFEDC00, the next 16 at address 0xFFFEDC04, and so on. Given a specific Interrupt ID, *N*, the address of the ICDICFRn

$$address = 0xFFFEDC00 + (N / 16)*4$$

and the index of the bit inside this register is given by

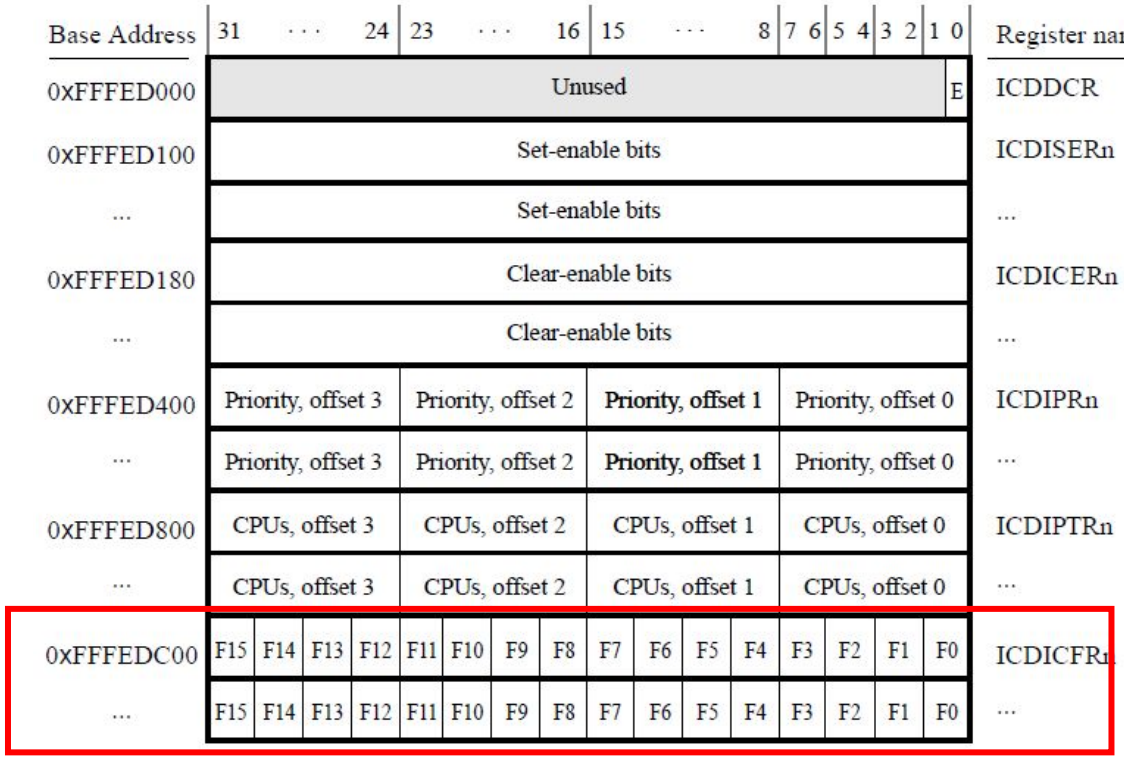
$$index = 2*(N \bmod 16) + 1.$$

- Example: N = 73 □

address =

Index =

Interrupt ID **73** **e. ICDICFRn**



Base Address	31 ··· 24	23 ··· 16	15 ··· 8	7 6 5 4 3 2 1 0	Register name
0xFFFED000	Unused			E	ICDDCR
0xFFFED100	Set-enable bits				ICDISERn
...	Set-enable bits				...
0xFFFED180	Clear-enable bits				ICDICERn
...	Clear-enable bits				...
0xFFFED400	Priority, offset 3	Priority, offset 2	Priority, offset 1	Priority, offset 0	ICDIPRn
...	Priority, offset 3	Priority, offset 2	Priority, offset 1	Priority, offset 0	...
0xFFFED800	CPUs, offset 3	CPUs, offset 2	CPUs, offset 1	CPUs, offset 0	ICDIPTRn
...	CPUs, offset 3	CPUs, offset 2	CPUs, offset 1	CPUs, offset 0	...
0xFFFEDC00	F15 F14 F13 F12 F11 F10 F9 F8	F7 F6 F5 F4 F3 F2 F1 F0			ICDICFRn
...	F15 F14 F13 F12 F11 F10 F9 F8	F7 F6 F5 F4 F3 F2 F1 F0			...

Dist. Reg.

- The Interrupt Configuration Registers (ICDICFRn) are used to specify whether each supported interrupt should be handled as level- or edge-sensitive by the GIC. There is a **two-bit** field associated with each Interrupt ID. The least-significant bit in this field is not used. Setting the most-significant bit of this field to 1 makes the corresponding interrupt signal edge-sensitive, and setting this field to 0 makes it level-sensitive.
- The first 16 Interrupt IDs use the ICDICFRn register at address 0xFFFEDC00, the next 16 at address 0xFFFEDC04, and so on. Given a specific Interrupt ID, *N*, the address of the ICDICFRn

$$address = 0xFFFEDC00 + (N / 16)*4$$

and the index of the bit inside this register is given by

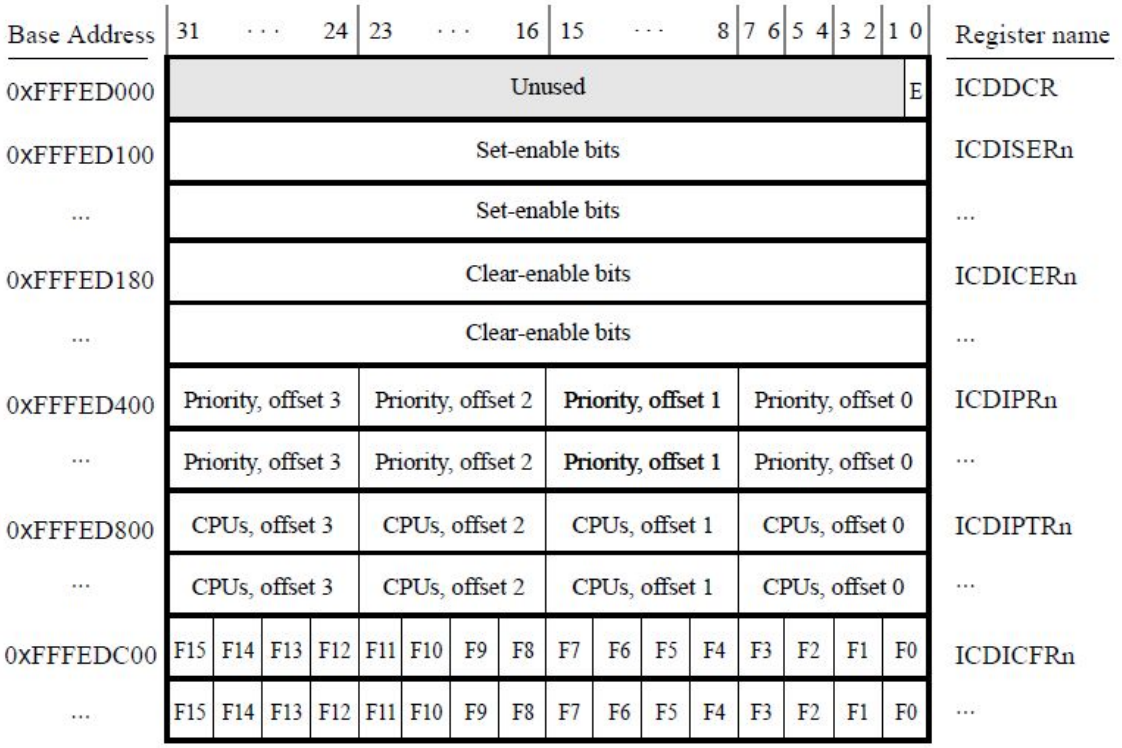
$$index = 2*(N \bmod 16) + 1.$$

- Example: N = 73 □

address = 0xFFFEDC00 + (4*4) = **0xFFFEDC10**

Index = 73 mod 16 +1 = **19**

Interrupt ID **73** **summary**



Base Address	31 ··· 24	23 ··· 16	15 ··· 8	7 6 5 4 3 2 1 0	Register name
0xFFFED000	Unused			E	ICDDCR
0xFFFED100	Set-enable bits				ICDISERn
...	Set-enable bits				...
0xFFFED180	Clear-enable bits				ICDICERn
...	Clear-enable bits				...
0xFFFED400	Priority, offset 3	Priority, offset 2	Priority, offset 1	Priority, offset 0	ICDIPRn
...	Priority, offset 3	Priority, offset 2	Priority, offset 1	Priority, offset 0	...
0xFFFED800	CPUs, offset 3	CPUs, offset 2	CPUs, offset 1	CPUs, offset 0	ICDIPTRn
...	CPUs, offset 3	CPUs, offset 2	CPUs, offset 1	CPUs, offset 0	...
0xFFFEDC00	F15 F14 F13 F12	F11 F10 F9 F8	F7 F6 F5 F4	F3 F2 F1 F0	ICDICFRn
...	F15 F14 F13 F12	F11 F10 F9 F8	F7 F6 F5 F4	F3 F2 F1 F0	...

Dist. Reg.

- N = 73 Note that following operation clears all other interrupts.

ICDISER: address = **0xFFFED10C,** Index = **9**

LDR R0,=**0xFFFED10C**

MOV R1,#1

LSL R1, #9

STR R1,[R0]

ICDIPTR: address = **0xFFFED848,** Offset = **1**

LDR R0,=**0xFFFED848**

MOV R1,#0x04 // assign a CPU

LSL R1,#8

STR R1,[R0]

- N = 73 Note that following operation clears all other interrupts.

IICDIPR: address = **0xFFFED448,** Offset = **1**

LDR R0,=**0xFFFED448**

MOV R1,#0xFF // priority is 1 byte.

LSL R1, #8 // shift by one byte.

STR R1,[R0]

ICDICFR: address = **0xFFFEDC10,** Index = **19**

LDR R0,=**0xFFFED10C**

MOV R1,#1

LSL R1, #19

STR R1,[R0]

Or, Leave defaults.

slido

Are you ready for tomorrow?

ⓘ Start presenting to display the poll results on this slide.