Interrupt Controller Driver

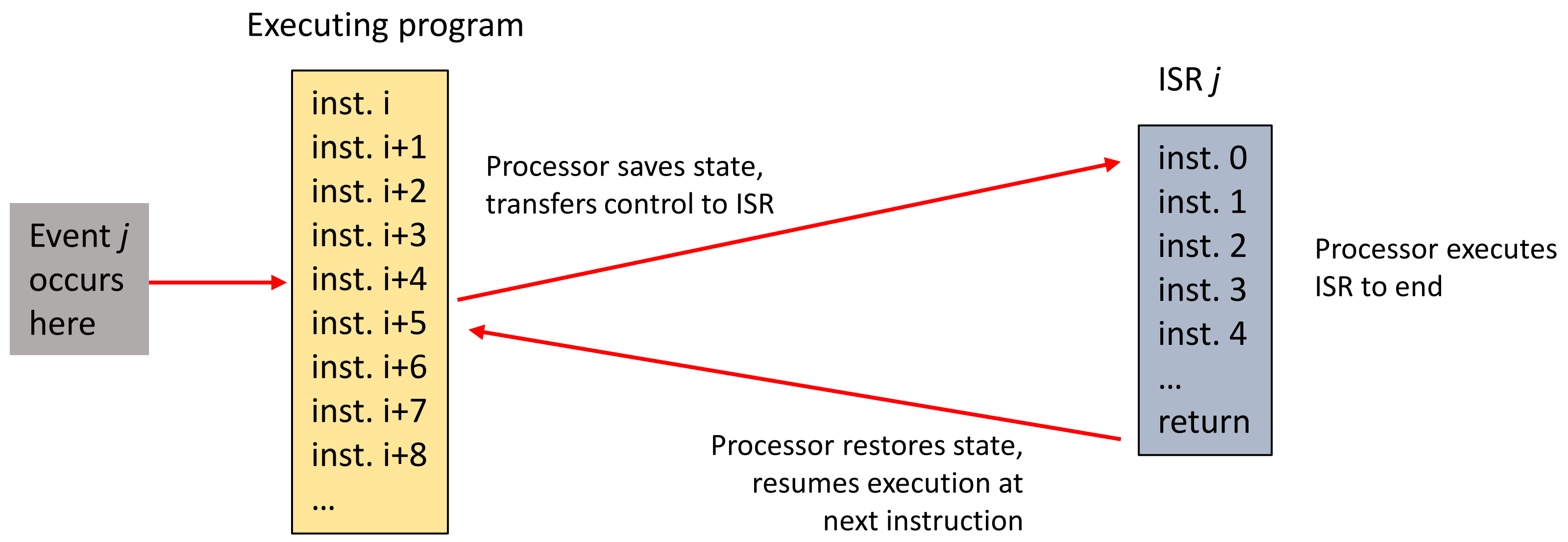
ECEN 330

BYU Electrical & Computer Engineering
IRA A. FULTON COLLEGE OF ENGINEERING

Interrupts

BYU Electrical & Computer Engineering
IRA A. FULTON COLLEGE OF ENGINEERING

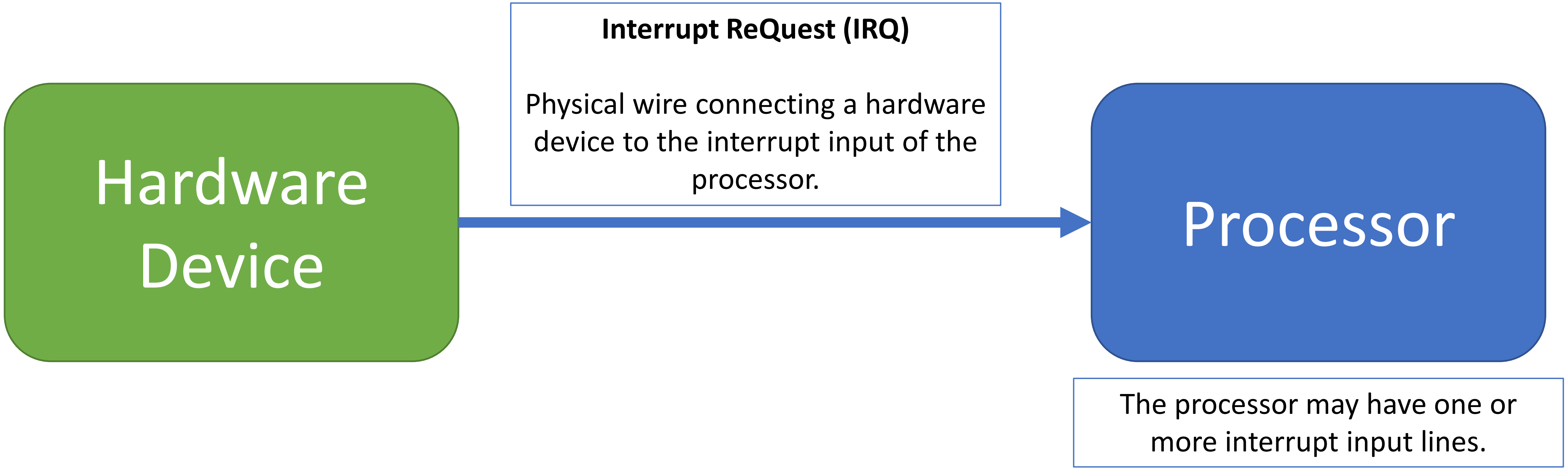
- A signal to the processor that an event occurred requiring a response
- The processor responds by:
  - Saving the state of the code that was running
  - Transferring control to a function written specifically to deal with that event



Interrupt Hardware

BYU Electrical & Computer Engineering
IRA A. FULTON COLLEGE OF ENGINEERING

What does the hardware look like?



Interrupt ReQuest (IRQ)

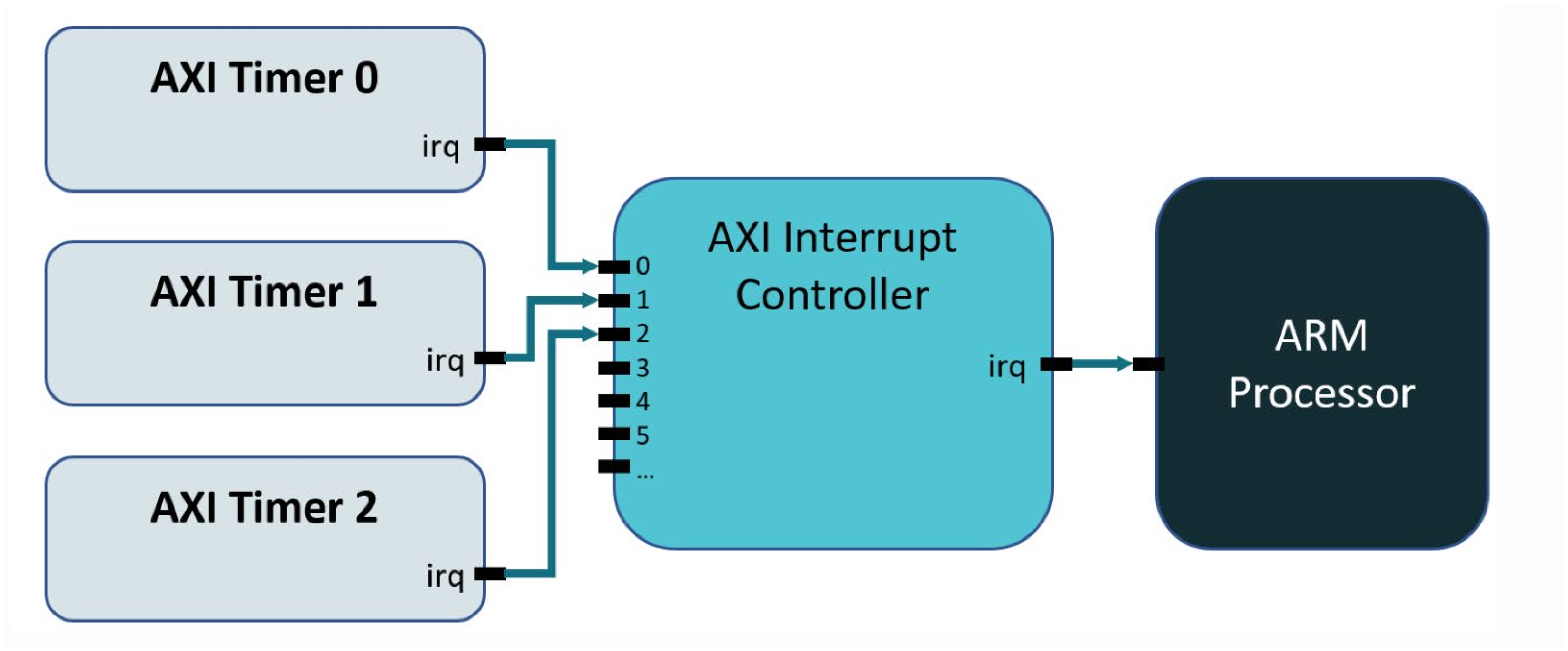
Physical wire connecting a hardware device to the interrupt input of the processor.

The processor may have one or more interrupt input lines.

BYU Electrical & Computer Engineering
IRA A. FULTON COLLEGE OF ENGINEERING

What happens if there are more devices than CPU interrupt inputs?

Interrupt Controller

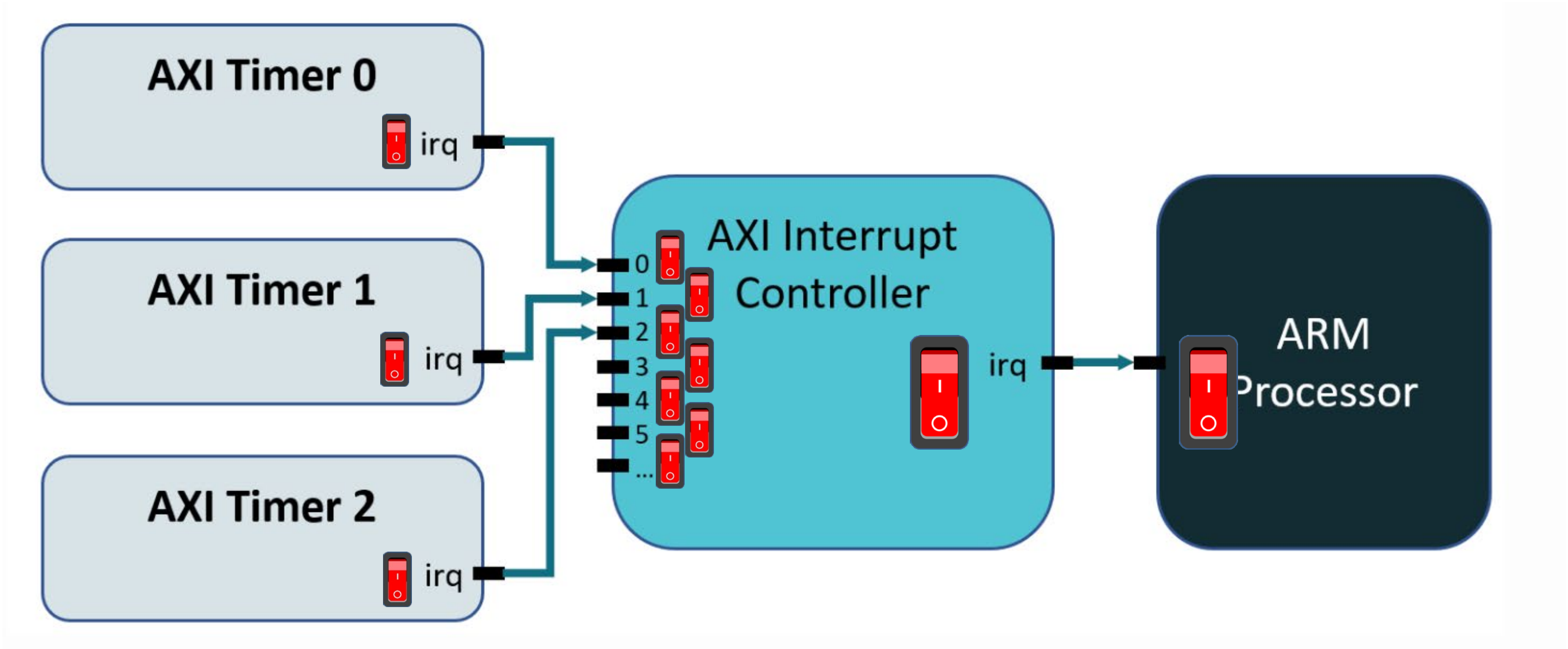


When a hardware device sends an IRQ to the interrupt controller,
the interrupt controller sends an IRQ to the processor.
(assuming appropriate things are enabled)

Setting Up the Interrupt Controller

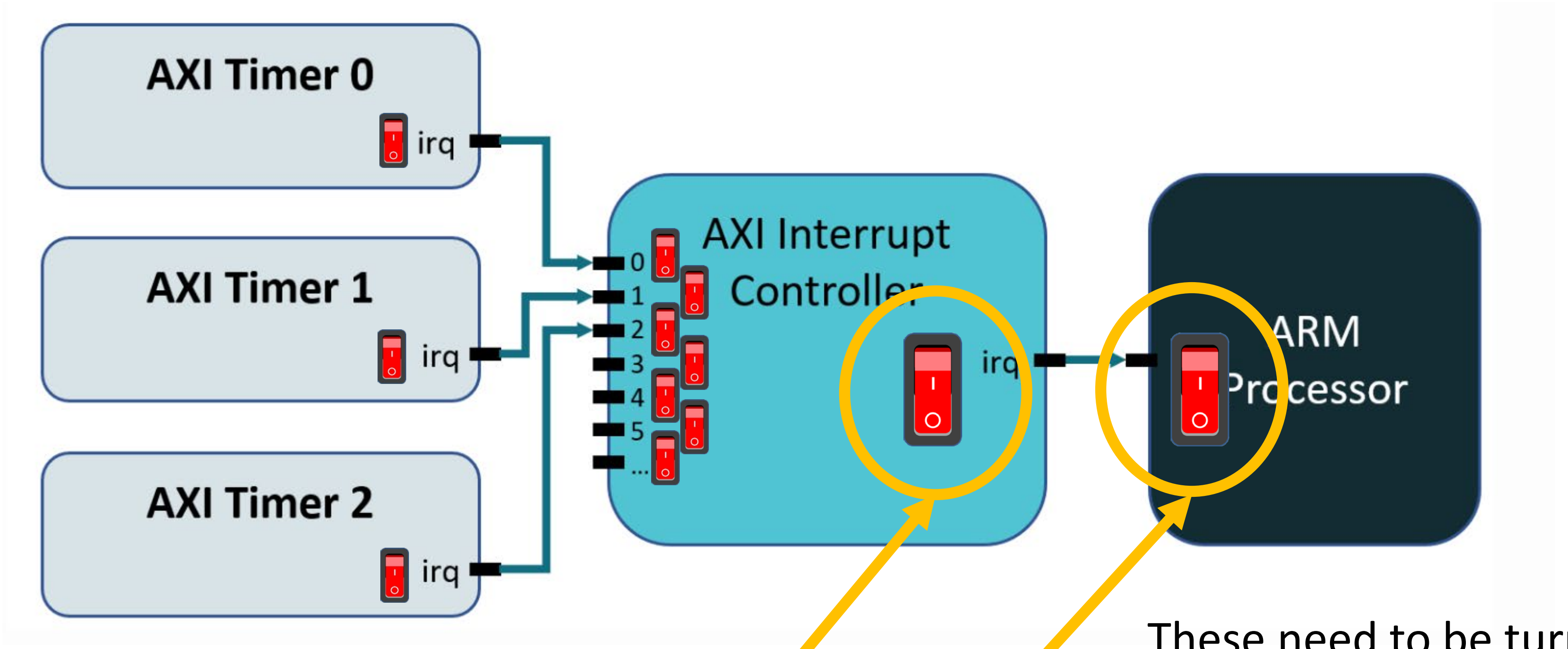
BYU Electrical & Computer Engineering
IRA A. FULTON COLLEGE OF ENGINEERING

For interrupts to function, they have to be enabled:



Setting Up the Interrupt Controller

For interrupts to function, they have to be enabled:



Set both bits in "Master Enable Register"

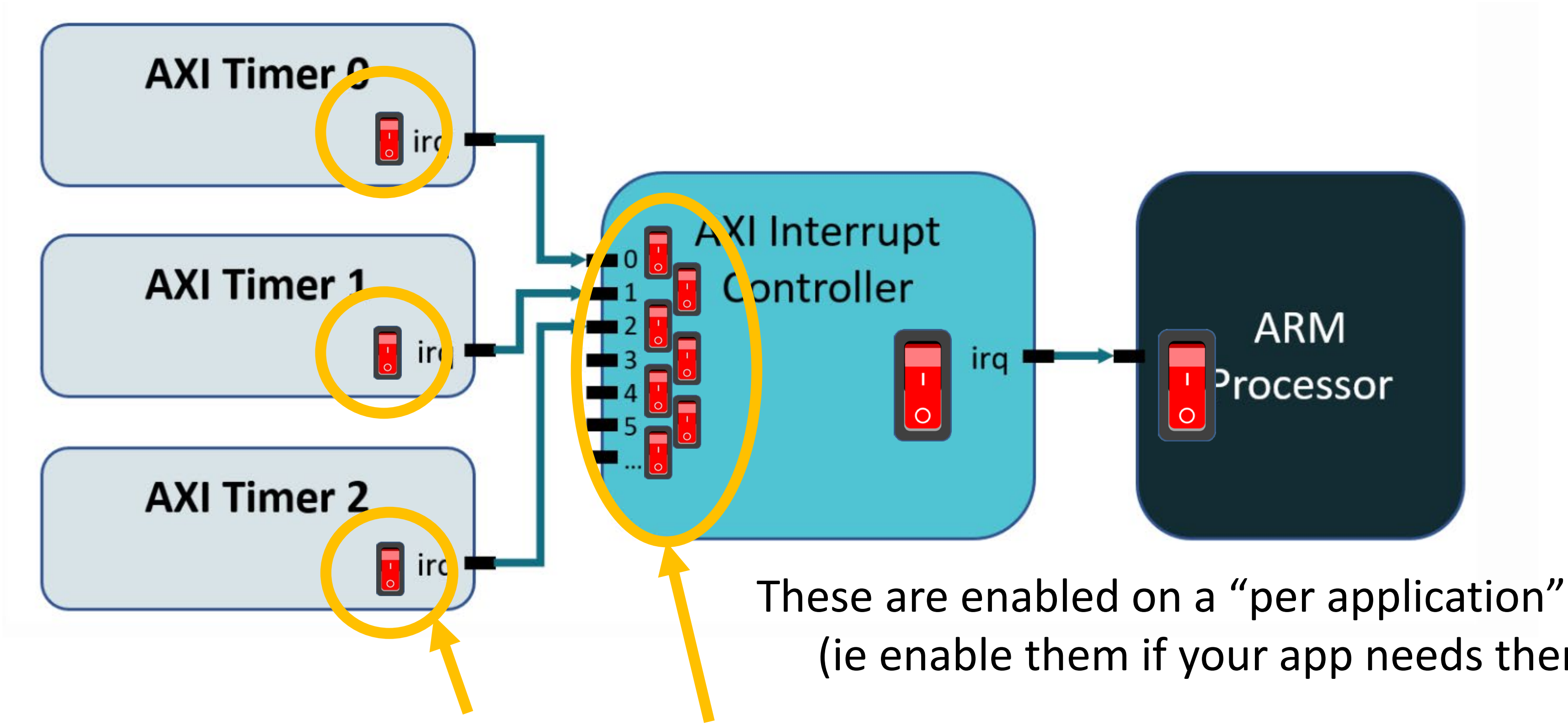
```
armInterrupts_init();
armInterrupts_enable();
```

These need to be turned on whenever you use interrupts, so turn them on in your **`interrupts_init()`** function

Setting Up the Interrupt Controller

BYU Electrical & Computer Engineering
IRA A. FULTON COLLEGE OF ENGINEERING

For interrupts to function, they have to be enabled:



These are enabled on a "per application" basis
(ie enable them if your app needs them)

`intervalTimer_enableInterrupt()`

`interrupts_irq_enable()` – Use IER or SIE register
`interrupts_irq_disable()` – Use IER or CIE register

Inside **`interrupts_init()`**, it's a good idea to disable all of the interrupt inputs.

Enabling the Interrupt Controller

The last setup step:

Specify an interrupt service routine (ISR).

- This is a function in your code that is called when the processor detects an interrupt.

```
static void interrupts_isr() {
    …
    …
}
```

This function will be a helper function in your Interrupt Controller Driver (inside *interrupts.c*)

How do you do this?

- Call the following and provide a function pointer:
  `armInterrupts_setupIntc(interrupts_isr);`

- Do this inside your **`interrupts_init()`** function

Now you are done setting up your interrupt controller!

At this point you should have written these functions:

`interrupts_init()`

`interrupts_irq_enable()`

`interrupts_irq_disable()`

```
static void interrupts_isr() {
    …
    …
}
```

So what should you do in your ISR function?

(Students often struggle getting this right.)

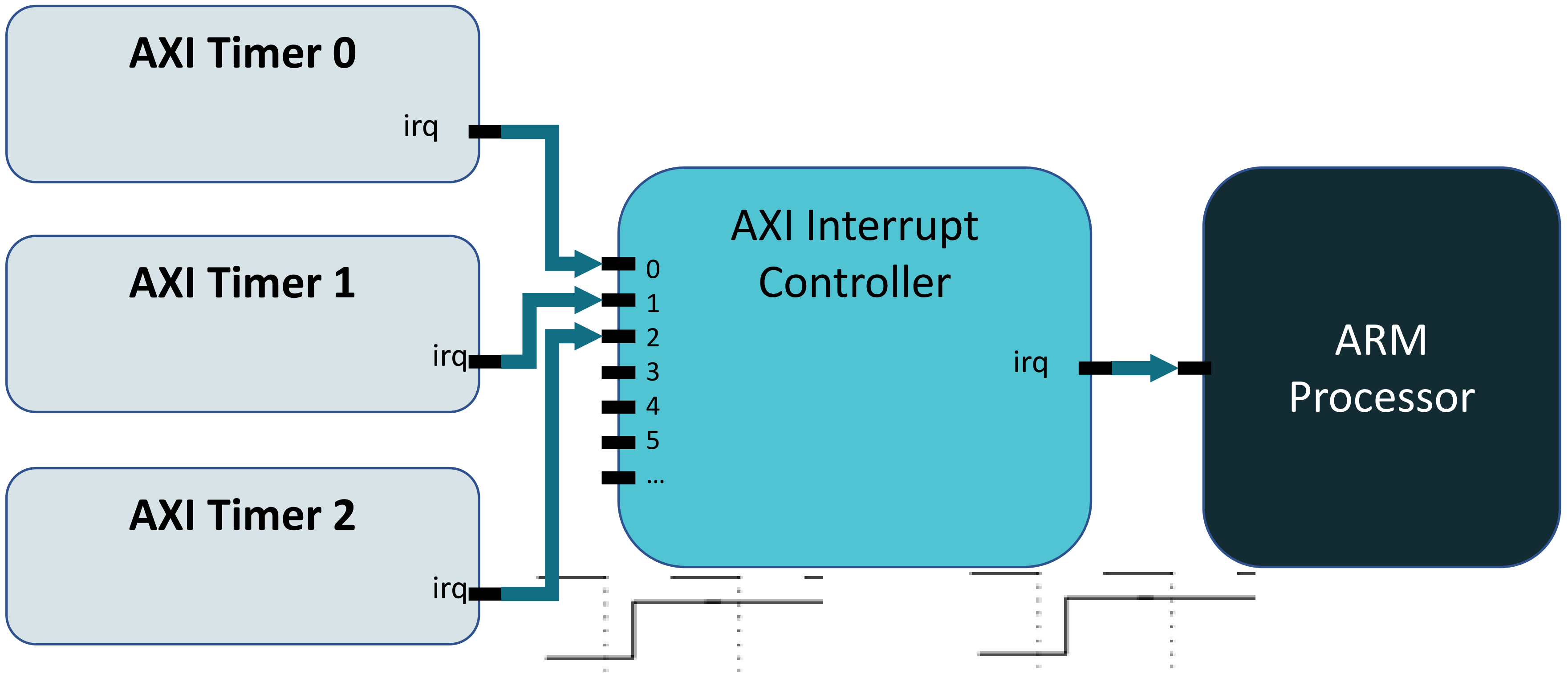
Key Fact:

Hardware devices don't know when it's IRQ has been handled.

- So (typically) they keep sending the IRQ until the software acknowledges/clears it.

Interrupt Controller

BYU Electrical & Computer Engineering
IRA A. FULTON COLLEGE OF ENGINEERING



Q1: If you don’t acknowledge the IRQ on the Interrupt Controller, what will happen when your ISR function completes?

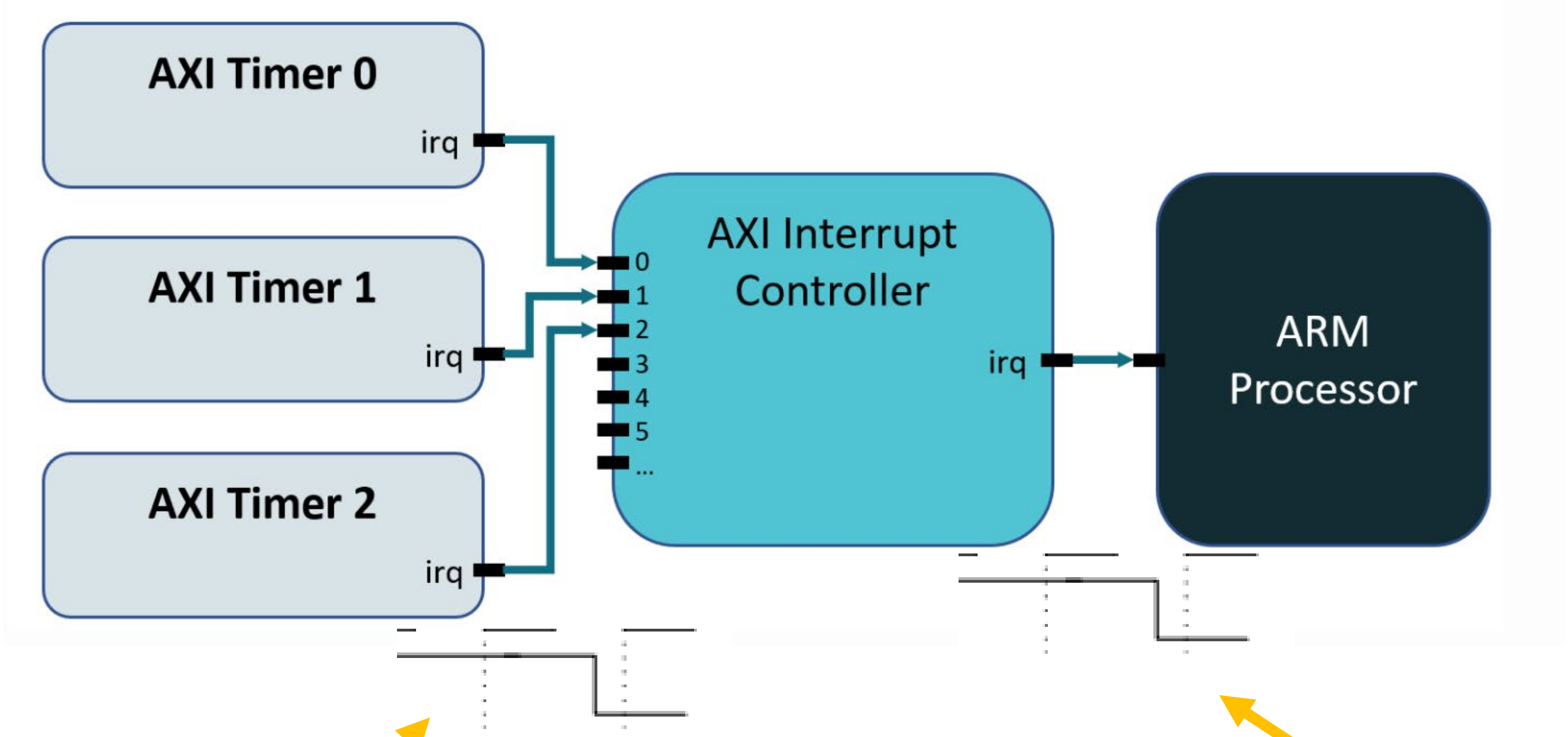
- Your ISR will immediately be called again. Infinite loop! You will never return to your program…

Q2: If you don’t acknowledge the IRQ from the Timer, what will happen?

Q3: Does it matter which you acknowledge first?

Acknowledging Interrupts

BYU Electrical & Computer Engineering
IRA A. FULTON COLLEGE OF ENGINEERING



`intervalTimer_ackInterrupt()`

`interrupts_ack()`

`(Use the IAR register)`

You should now be able to handle interrupts, without your program hanging.

However, your ISR does nothing!
(aside from acknowledging the interrupt)

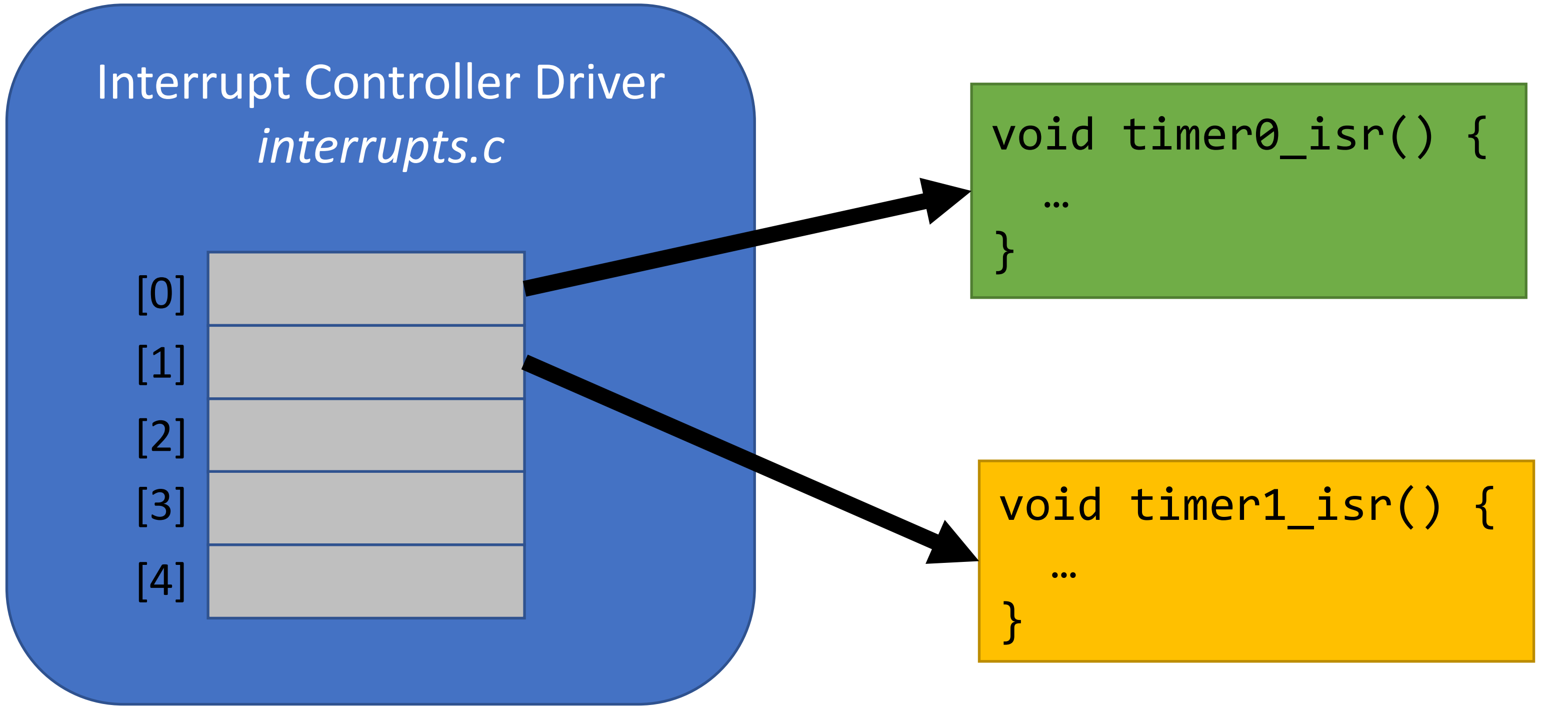
What would you like it to do?

ISR

BYU Electrical & Computer Engineering
IRA A. FULTON COLLEGE OF ENGINEERING

Your interrupt controller driver will allow programs to **register** a **callback function** tied to an IRQ #.

```
void interrupts_register(uint8_t irq, void (*fcn)());
```



```
void timer0_isr() {
  …
}
```

```
void timer1_isr() {
  …
}
```

```
interrupts_register(0, timer0_isr);
```

```
interrupts_register(1, timer1_isr);
```

Array of Function Pointers

BYU Electrical & Computer Engineering
IRA A. FULTON COLLEGE OF ENGINEERING

Declaring function pointer array:

```
static void (*isrFcnPtrs[3])() = {NULL};
```

Storing function pointer in array:

```
isrFcnPtrs[2] = fcn;
```

Calling a function:

```
isrFcnPtrs[2]();
```

ISR

```
static void interrupts_isr() {

        // Loop through each interrupt input
        for (i, to # interrupt inputs - 1) {

                // Check if it has an interrupt pending
                if (input i has pending interrupt) {

                // Check if there is a registered ISR and call it.
                if (isrFcnPtrs[i])
                  isrFcnPtrs[i]();

                }
        }
}
```