DEVICE DRIVERS AND INTERRUPTS SERVICE MECHANISM

Lesson-2: Interrupt and Interrupt Service Routine Concept

Interrupt Concept

- Interrupt means event, which invites attention of the processor on occurrence of some action at hardware or software interrupt instruction event.

Action on Interrupt

- In response to the interrupt, the routine or program, which is running presently interrupts and an interrupt service routine (ISR) executes.

Interrupt Service Routine

- ISR is also called device driver in case of the devices and called *exception or signal or trap handler* in case of software interrupts

Interrupt approach for the port or device functions

- Processor executes the program, called interrupt service routine or signal handler or trap handler or exception handler or device driver, related to input or output from the port or device or related to a device function on an interrupt and <u>does not wait</u> and look for the input ready or output completion or device-status ready or set

Hardware interrupt

- Examples— When a device or port is ready, a device or port generates an interrupt, or when it completes the assigned action or when a timer overflows or when a time at the timer equals a preset time in a compare register or on setting a status flag (for example, on timer overflow or compare or capture of time) or on click of mice in a computer
- Hardware interrupt generates call to an ISR

Software Interrupt

Examples

- When software run-time exception condition (for examples, division by 0 or overflow or illegal opcode detected) the processor-hardware generates an interrupt, called *trap, which calls an ISR*
- When software run-time exception condition defined in a program occurs, then a software instruction (SWI) is executed─ called *software interrupt* or *exception or signal, which calls an ISR* .

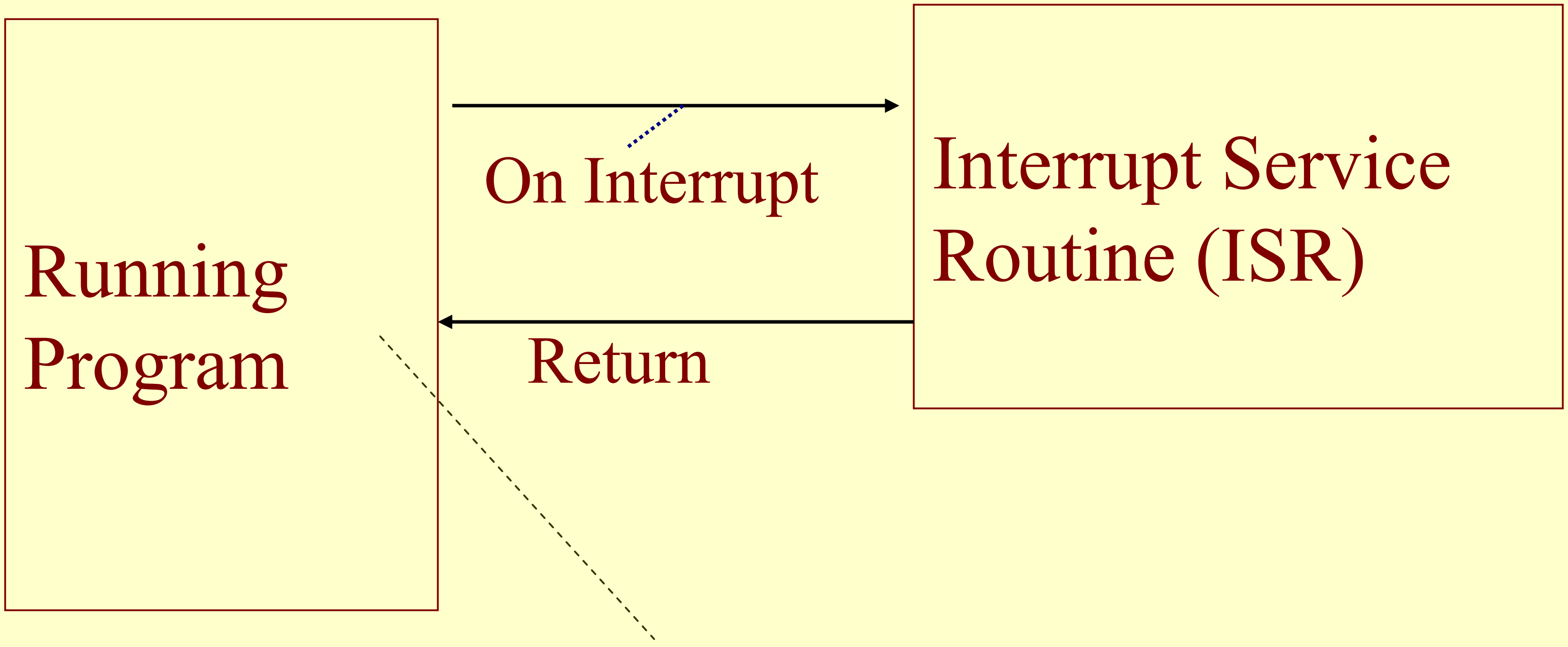
Software Interrupt

- When a device function is to be invoked, for example, *open* (initialize/configure) or *read* or *write* or *close* , then a software instruction (SWI) is executed— called *software interrupt* to execute the required device driver function for open or read or write or close operations.

Interrupt

- Software can execute the software instruction (SWI) or Interrupt *n* (INT *n*) to *signal* execution of ISR (interrupt service routine). The *n* is as per the handler address.
- *Signal* interrupt [The *signal* differs from the function in the sense that execution of signal handler (ISR) can be masked and till mask is reset, the handler will not execute on interrupt. Function on the other hand always executes on the call after a call-instruction.]

Interrupt Driven Actions



No continuous checks of device status

How does call to ISR differ from a function (routine) call?

Routine (function) and ISR call features

- On a function call, the instructions are executed from a new address
- Execute like in as function in the ‘C’ or in a method in Java.
- ISR also the instructions are executed from a new address like in as function in the ‘C’ or in a method in Java

Routine (function) call and ISR call features

- A function call is after executing present instruction in any program and is a planned (user programmed) diversion from the present instruction in sequence of instructions to another sequence of instructions; this sequence of instructions executes till the return from that.
- An ISR call is after executing present instruction in any program and is interrupt related diversion from the current sequence of instructions to another sequence of instructions; this sequence of instructions executes till the return from that or till another interrupt of higher priority

Nesting of function calls and ISRs

- Nesting the function-calls ─ When a function 1 calls another function 2 and that call another function 3, on return from 3, the return is to function 2 and return from 2 is to function. Functions are said to be nested
- The ISR calls in case of multiple interrupt occurrences can be nested or can be as per priority of the interrupts

Using the Interrupt(s) and ISR(s) for each device function (for example, *read*, *write*)

- Use of ISRs (Interrupt Service Routine) by SWIs is the main mechanism used for the device accesses and actions

Interrupt driven IO and device accesses feature

1. There is no continuous monitoring of the status bits or polling for status by the application.
2. Between two interrupt calls the program task(s) continue. There are many device-functions, and each of which executes on a device interrupt.

Example— A 64-kbps UART Input

- When a UART transmits in format of 11-bit per character format, the network transmits at most 64 kbps ÷ 11 = 5818 characters per second, which means every 171.9 µs a character is expected.
- Before 171.9 µs, the receiver port is not checked. Only on interrupt from the port the character is read
- There is no in-between time-gap for polling

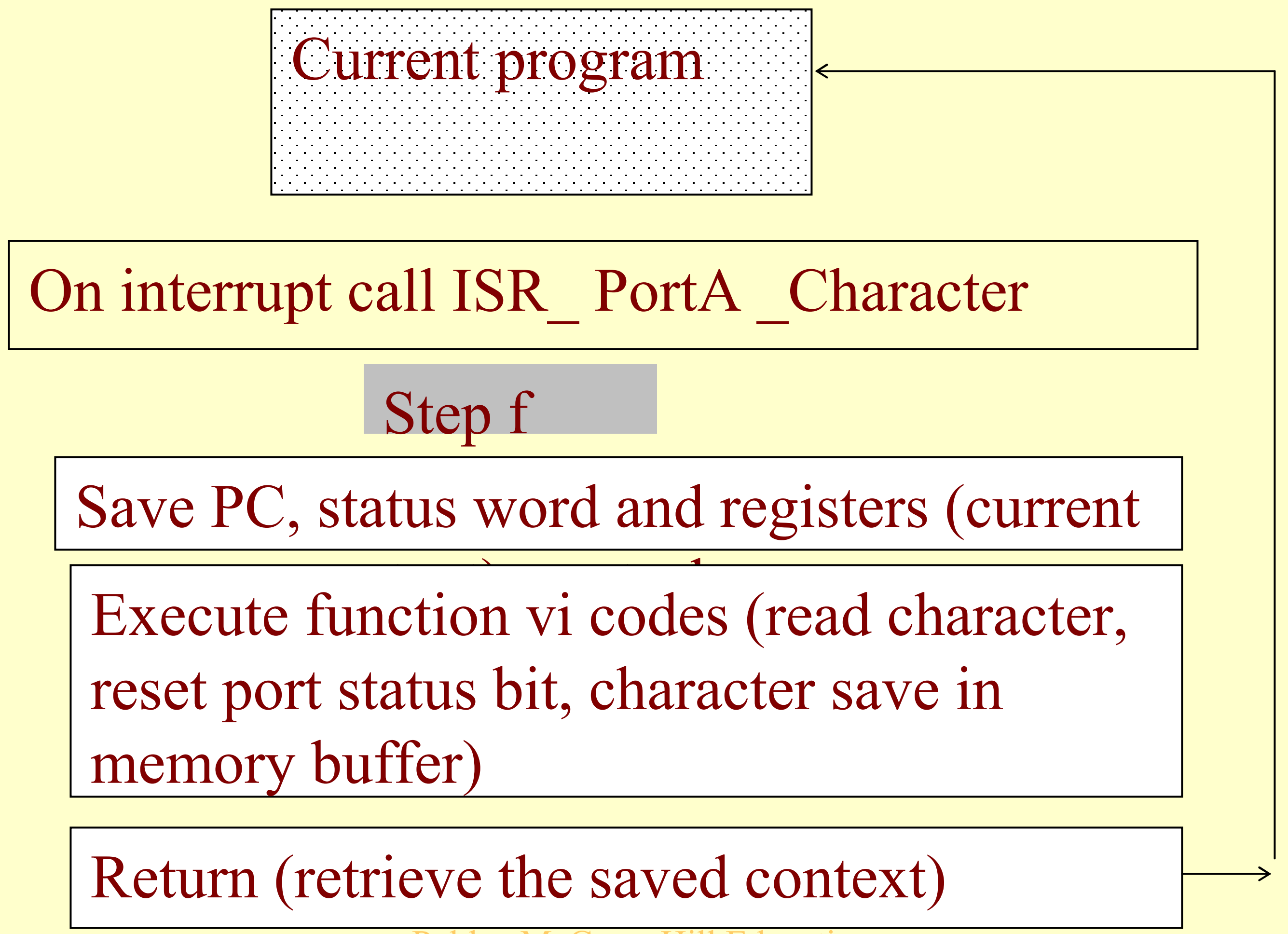
Ports A and B with interrupt generation and interrupt service (handling) mechanism

- Port *A* be at in a PC, and port B be its modem input which puts the characters on the telephone line.
- Let ISR_ PortA_Character be a routine that receives an input character from Port A on interrupt and Out_B routine re-transmits an output character to Port *B*.

ISR_ PortA _Character

- Step *f* function (*vi*): *Read* Port *A* character. *Reset* the status bit so that modem is ready for next character input (generally resetting of status bit is automatic without need of specific instruction for it). *Put* it in memory buffer. Memory buffer is a set of memory addresses where the bytes (characters) are queued for processing later.
- Return from the interrupt service routine.

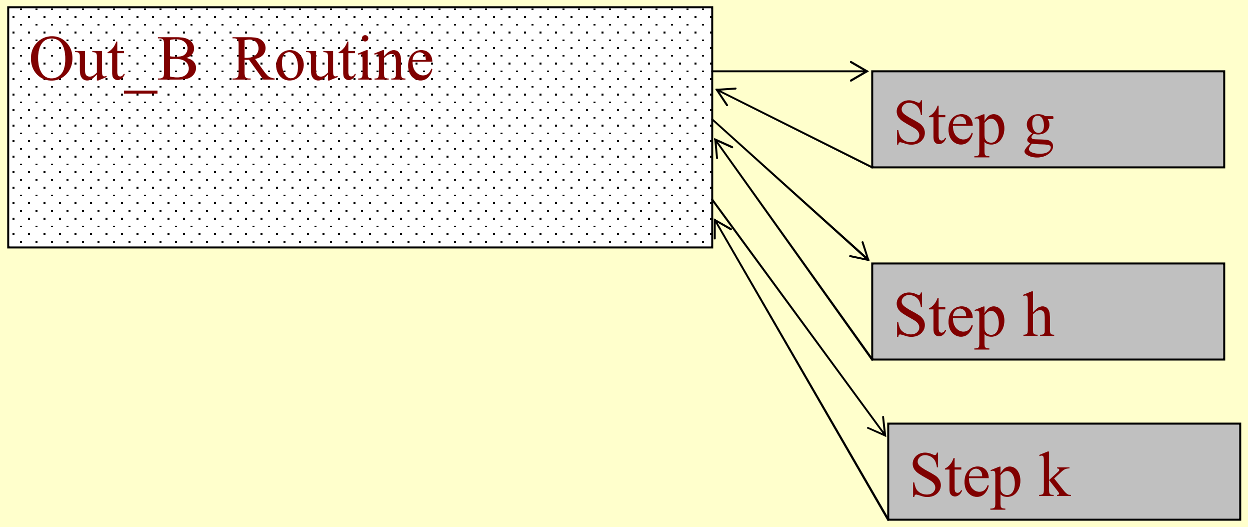
ISR_ PortA _Character


Current program
On interrupt call ISR_ PortA _Character
Step f
Save PC, status word and registers (current
Execute function vi codes (read character, reset port status bit, character save in memory buffer)
Return (retrieve the saved context)

Out_B routine

- Step *g*: Call function (vii) to decrypt the message characters at the memory buffer and return for next instruction step *h*.
- Step *h*: Call function (viii) to encode the message character and return for next instruction step *k*
- Step *k*: Call function (ix) to transmit the encoded character to Port B
- Return from the function.

Out_B Routine



Each step has three parts, save context, execute codes, and retrieve the context for return

Application of Interrupt based IOs and device functions

- In case of multiple processes and device functions with no device status polling or wait

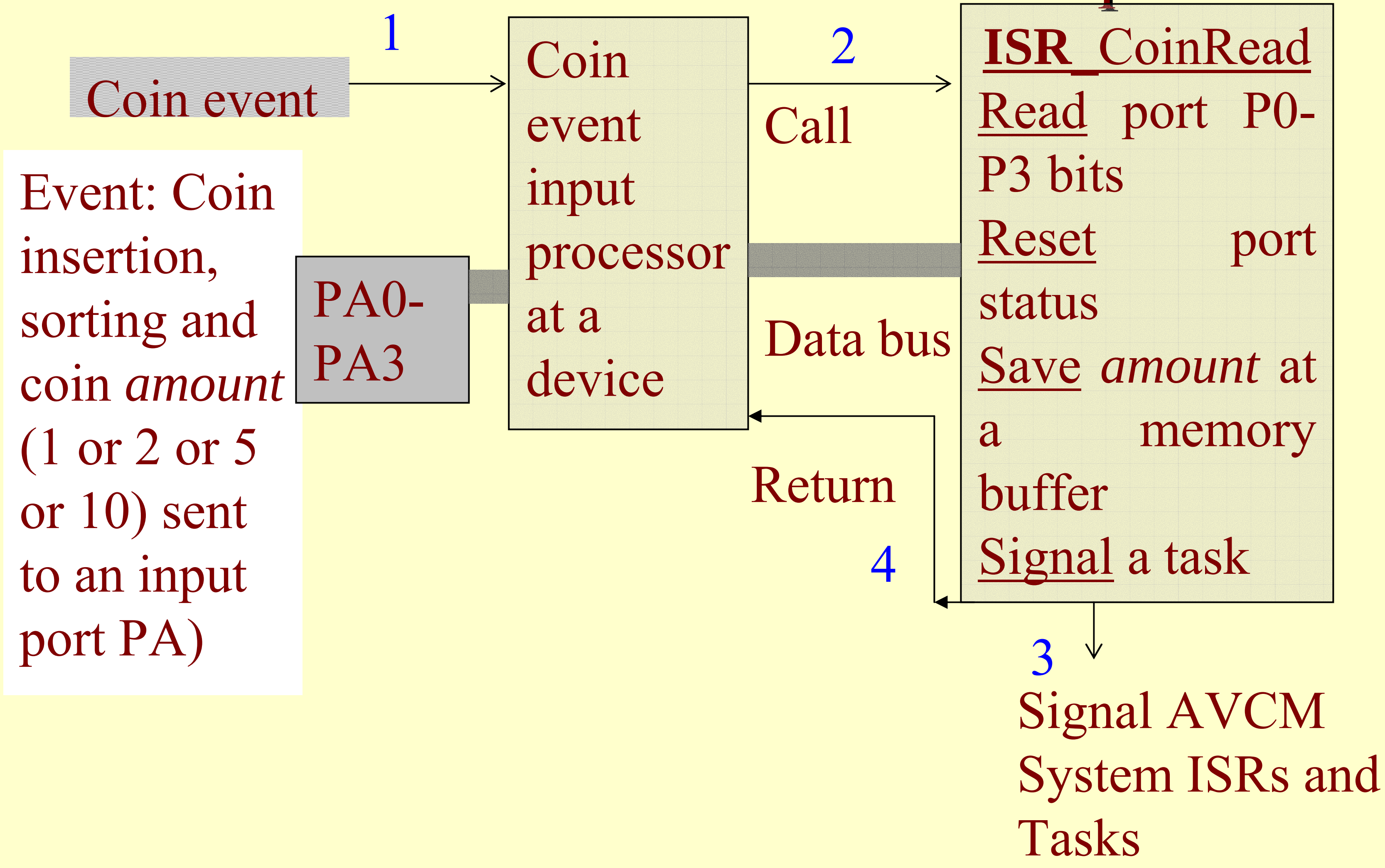
An automatic chocolate vending machine (ACVM)

- Without interrupt mechanism, one way is the 'programmed I/O- transfer in which that the device waits for the coin continuously, activates on sensing the coin and runs the service routine
- In the event driven way is that the device should also awakens and activates on each *interrupt* after sensing each coin-inserting event.

Interrupt-service routine (ISR) or *device driver function*

- The system awakens and activates on an interrupt through a hardware or software signal. The system on port *interrupt* collects the coin by running a service routine. This routine is called interrupt handler routine for the coin-port read.

Use of ISR in the ACVM example


Coin event
1
Coin event input processor at a device
2
Call
ISR_CoinRead
Read port P0-P3 bits
Reset port status
Save *amount* at a memory buffer
Signal a task
Event: Coin insertion, sorting and coin *amount* (1 or 2 or 5 or 10) sent to an input port PA)
PA0-PA3
Data bus
Return
4
3
Signal AVCM System ISRs and Tasks

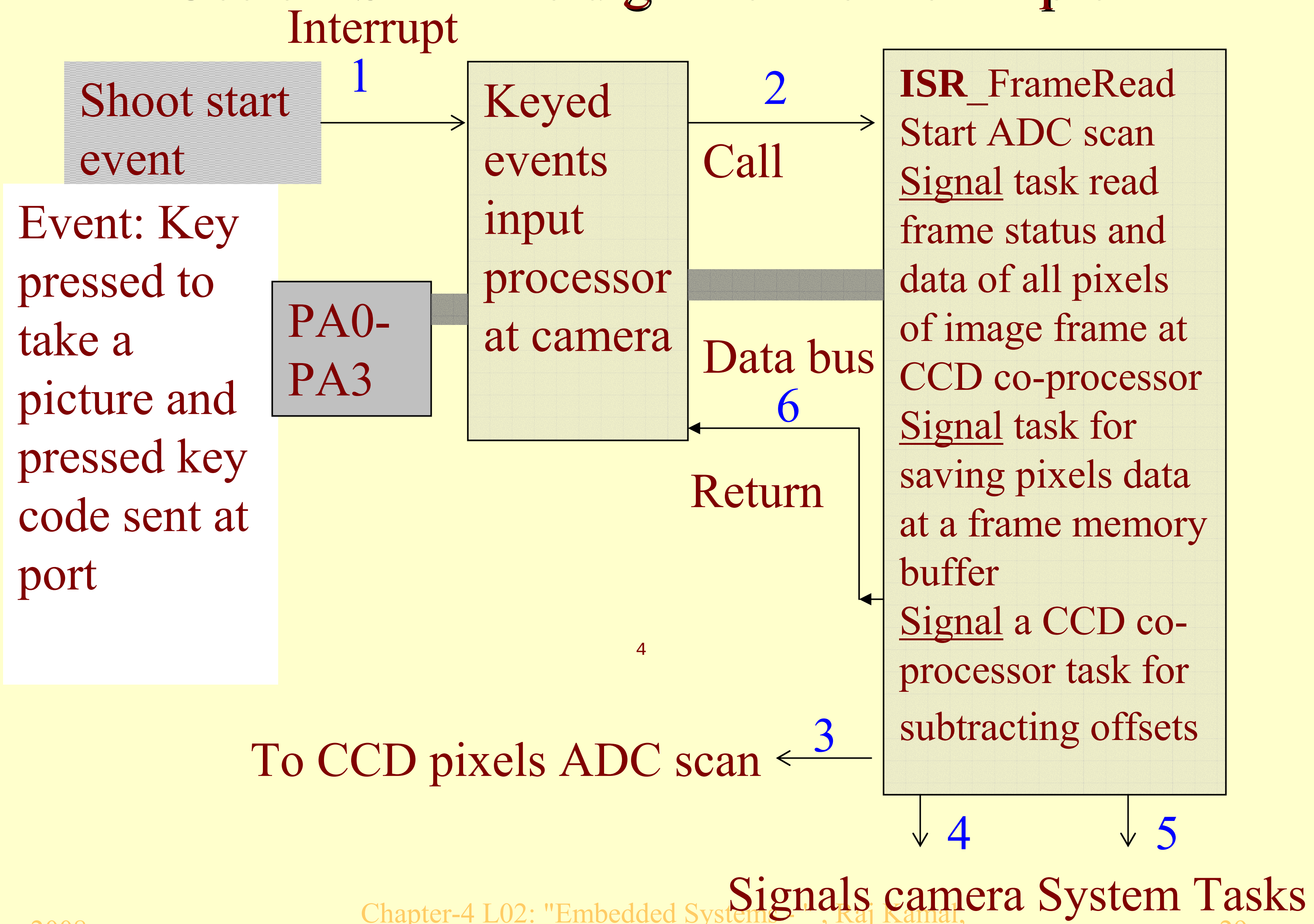
Digital camera system

- Has an image input device.
- The system awakens and activates on interrupt from a switch.
- When the system wakes up and activates, the device should grab an image frame data.
- The interrupt is through a hardware signal from the device switch. On the interrupt, an ISR (can also be considered as camera's imaging device-driver function) for the read starts execution, it passes a message (signal) to a function or program thread or task

Image sense and device frame-buffer

- The thread senses the image and then the function reads the device frame-buffer (called frame grabbing),
- then the function passes a signal to another function or program thread or task to process and compress the image data and save the image frame data compressed file in a flash memory

Use of ISR in the digital camera example


Interrupt
1
Shoot start event
Event: Key pressed to take a picture and pressed key code sent at port
PA0-PA3
Keyed events input processor at camera
2
Call
Data bus
6
Return
ISR_FrameRead
Start ADC scan
Signal task read frame status and data of all pixels of image frame at CCD co-processor
Signal task for saving pixels data at a frame memory buffer
Signal a CCD co-processor task for subtracting offsets
4
3
To CCD pixels ADC scan
4
5
Signals camera System Tasks

Interrupt through a hardware signal from a print-switch at device

- Camera system again awakens and activates on.
- System on *interrupt* then runs another ISR.
- The ISR_ Routine is the device-driver write-function for the outputs to printer through the USB bus connected to the printer.

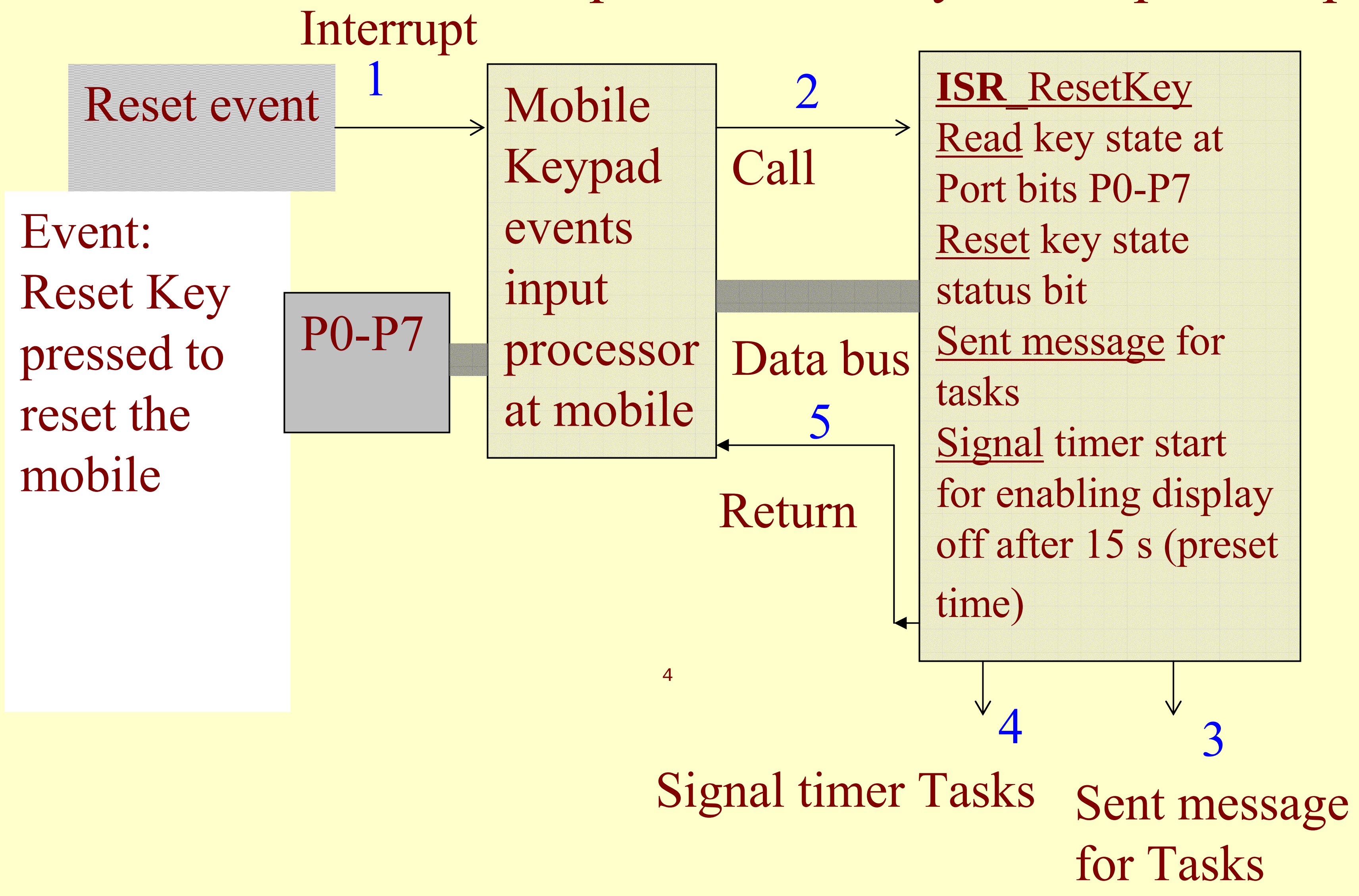
Mobile phone system

- Has a system reset key, which when pressed resets the system to an initial state.
- When the reset key is pressed the system awakens and activates a **reset** **interrupt** through a hardware signal from the reset key.
- On the interrupt, an ISR (can also be considered as reset-key device-driver function) suspends all activity of the system, sends signal to display function or program thread or task for displaying the initial reset state menu and graphics on the LCD screen, and also activates LCD display-off timer device for 15 s timeout (for example).

After the timeout

- The system again awakens and activates on *interrupt* through internal hardware signal from the timer device and runs another ISR to send a control bit to the LCD device.
- The ISR_ Routine is the device-driver LCD off-function for the LCD device. The devices switches off by reset of a control bit in it.

Use of ISR in the mobile phone reset key interrupt example


Interrupt
Reset event
1
Mobile Keypad events input processor at mobile
2
Call
ISR_ResetKey
Read key state at Port bits P0-P7
Reset key state status bit
Sent message for tasks
Signal timer start for enabling display off after 15 s (preset time)
Event: Reset Key pressed to reset the mobile
P0-P7
Data bus
5
Return
4
4
3
Signal timer Tasks
Sent message for Tasks

Summary

We learnt

- Interrupt means event, which invites attention of the processor for some action on hardware or software interrupt instruction (SWI) event.
- When software run-time exception condition is detected, either processor hardware or an SWI generates an interrupt— called *software interrupt* or *trap* or *exception*.
- An embedded system executes many actions or functions or tasks on the interrupts from hardware or software.

We learnt

- An ISR call due to interrupt executes an event. Event can occur at any moment and event occurrences are asynchronous.
- Event can be a device or port event or software computational exceptional condition detected by hardware or detected by a program, which throws the exception. An event can be signaled by soft interrupt instruction in routine.
- An interrupt service mechanism exists in a system to call the ISRs from multiple sources.

We learnt

- Interrupts and interrupt-service routines (device-drivers) play the major role in using the embedded system hardware and devices.
- Interrupt examples of UART IO, ACVM, Camera and mobile phone

End of Lesson 2 of Chapter 4