Overview

- In last 2 lectures we studied interrupts

- Today: Periodic tasks
  - Real time periodic task and the problems of jitter and drift
  - Implementation of real time periodic task in POSIX RT
    - using timer signals (software interrupt)
    - and signal handlers (ISR)

- Books on POSIX RT programming
  - Programming for The Real World, Bill O. Gallmeister, O'Reilly&Associates, Inc
  - POSIX Programmer's Guide, Donald Lewine, O'Reilly&Associates, Inc
  - Chapters 5 & 6 are available in the lab. You group reading assignment is to read them and to discuss.

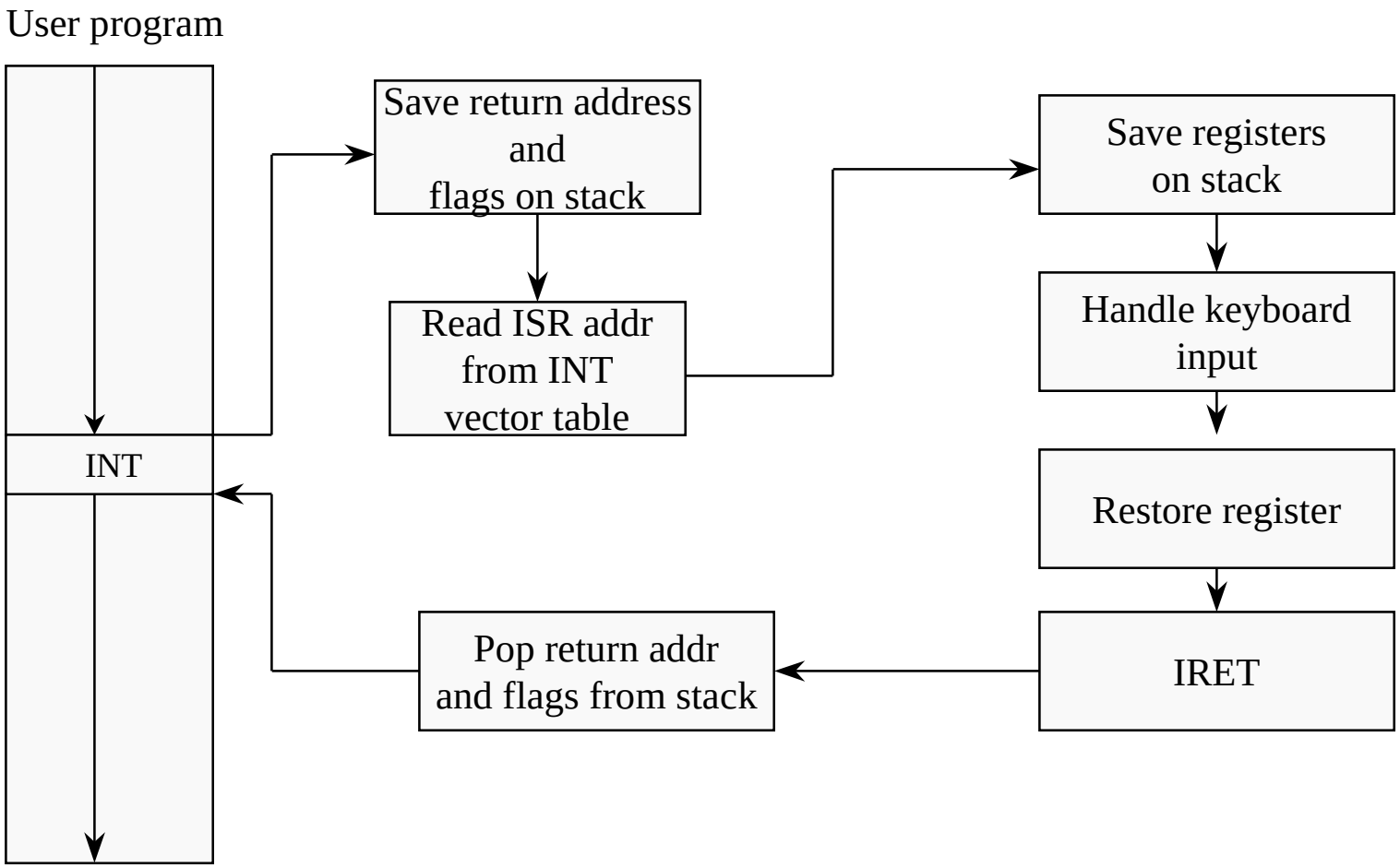
Review: Generating an Interrupt

- Interrupts can be generated by microprocessor, for example, detecting an overflow. The interrupt type number of processor generated interrupts are defined by the processor designers.

- Interrupts can be generated by software. For example, the interrupt type number 3 H was, by convention, reserved as a software interrupt for break points used by debugger. To invoke it
  - INT 3H

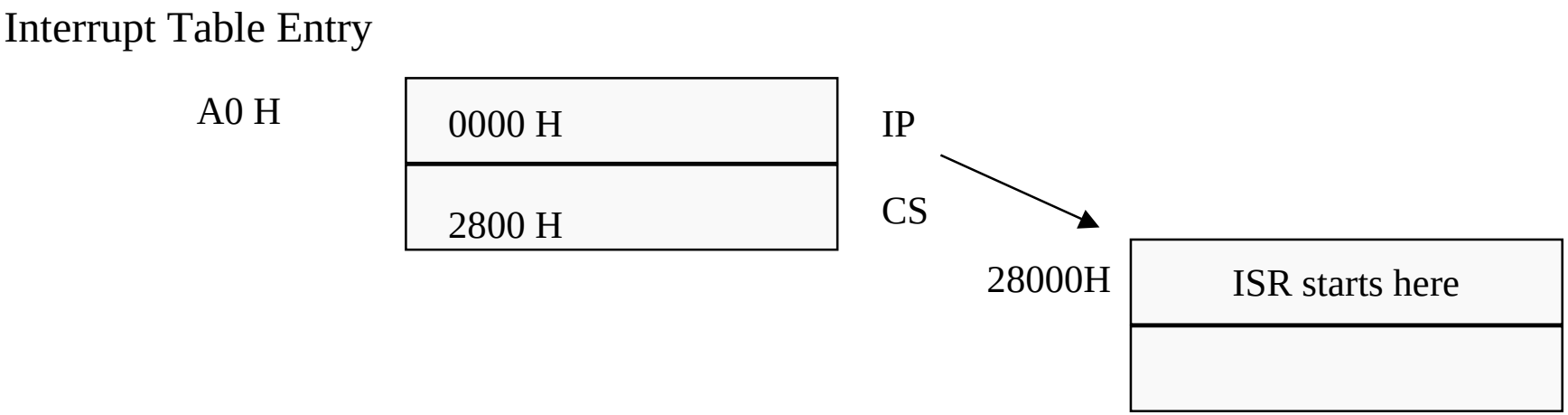
- Interrupt can be generated by external hardware and the hardware is responsible to put the 8 bit interrupt type number on the address bus <AD0…AD7> after it receives the acknowledgement of interrupt from the microprocessor.

Review: Interrupt Model



Big Picture

•Suppose that the ISR starts right at 28000H, the starting address of the code segment.



Prioritized Interrupt Basic

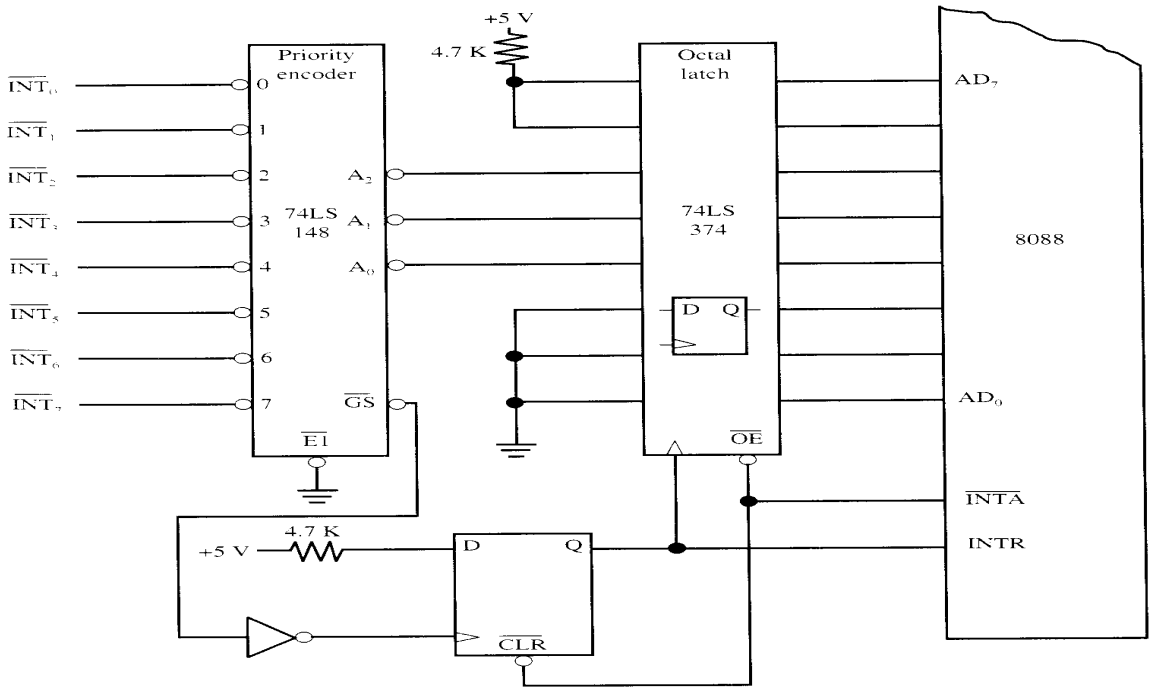


FIGURE 5.9 Prioritized interrupt circuitry

-when several interrupts are pending, the highest one is selected (largest) by the encoder.

- The interrupt type number is 11xxx000.

Nested Priority Interrupt

- when a processor is serving a maskable interrupt, it will be interrupted only when there is a newly arrived higher priority interrupt request.

Priority Interrupt
Controller Chips

•provide a priority mask register to disable interrupt up to any priority level.

•provide a interrupt mask register to disable any particular interrupt request line.

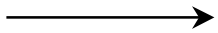
• status registers

Example: Maskable Nested Priority Interrupt

Supposed priority 3 interrupt arrives first, followed immediately priority level 5 (higher interrupt)

<u>interrupt level 3 ISR</u>

- save context AND the old priority mask
- set priority mask level 3 (level 3 or lower disabled)
- enable interrupt
- serving interrupt



<u>interrupt level 5 ISR</u>

- save context AND the old priority mask
- set priority mask level 5 (level 5 or lower disabled)
- enable interrupt
- serving interrupt
- restore context AND the old priority mask
- IRET

- serving interrupt
- restore context AND the old priority mask
- IRET

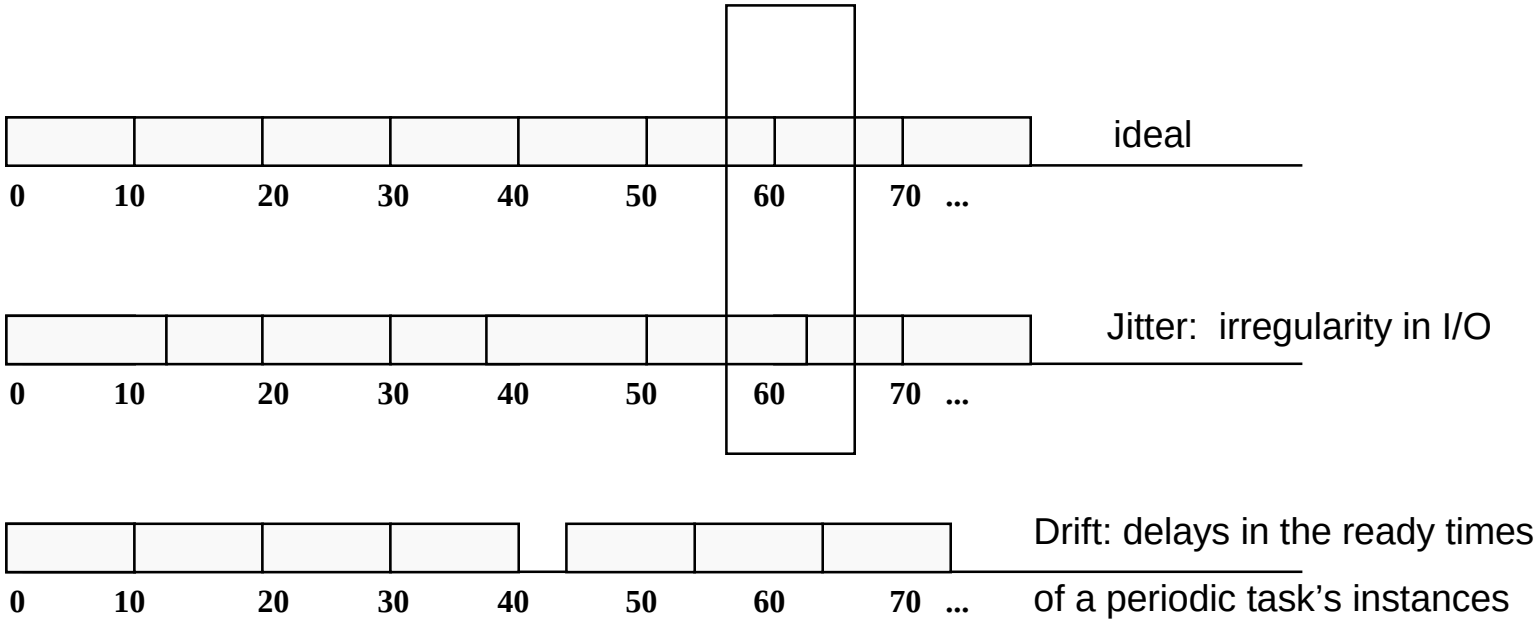
Periodic Tasks

- Periodic tasks are commonly used in embedded real time systems, e.g., a 10Hz task updates the display and a 20 Hz task for sampling the temperature.

- Rate monotonic scheduling (RMS) algorithm are commonly used in practice. RMS assigns high priorities to tasks with higher rates, e.g., 20 Hz task should be given higher priority than 10 Hz tasks.
  - If every instance of a periodic task has the same priority, it is called a fixed priority scheduling method
  - commercial RTOS supports only fixed priority scheduling
  - RMS is an optimal fixed priority scheduling method
  - The timing behavior of a real time system scheduled by RMS, can be fully analyzed and predicted by Rate Monotonic Analysis (RMA). We will study this subject after the mid-term
  - we will study the design and implementation of periodic tasks today.

Periodic Tasks

- A periodic task should repeat regular according to a given period. For example, a periodic tasks with period 10 starting on t = 0.



Evaluation: Where in the Code are Jitters and Drifts

- What is preemption?
- Suppose that we want to control a device using a 20 msec task starting at START_TIME:

```
1.  current_time = read_clock()
2.  if  (START_TIME – current_time < 10 msec) {    //report too late and exit    }
3.  sleep(START_TIME – current_time)
loop
        4.  current_time = read_clock()
        5.  wake_up_time = current_time + 20 msec
        6.  //  read sensor data from the device, it takes << 20 msec
        7.  //  do work, It takes  << 20 msec
        8.  current_time = read_clock()
        9.  //  send control data to the device
        10. sleep(wake_up_time – current_time)
end loop
```

- Can you identify operations that are drift and jitter prone?
- Hint: think of paging, multi-tasking and preemption.

Potential Timing Problems

- E1. It could be swapped out of memory (drift and jitter). Pin it down in memory! In LynxOS, a priority greater than 16 (default) will not be swapped out.

```
1. current_time = read_clock()                          //E2 if preempted, drift
2. If  (START_TIME - current_time < 10 msec) {//report too late and exit}
3. sleep(START_TIME - current_time)                     //E3: if preempted, drift
loop
        4. current_time = read_clock()                  //  same problem as line 1
        5. wake_up_time = current_time + 20 msec        //E4: drift if current time is delayed at line 4
        6.  Read sensor data                            //E5: if preempted, input jitter
        7.    //do work
        8. current_time = read_clock()                  // same problem as line 1
        9.    //send control data to the device         //E6: output jitter (caused by preemption
                                                                or variable execution time)
        10. sleep(wake_up_time - current_time)          //  same as line 3
 end_loop
```

Summary: every line which manipulates time or does I/O has problems.