NOMBRE DEL ALUMNO:

Diego Moisés González Solís

CARRERA:

Ing. Mecatrónica

MATERIA:

EMBEBIDOS

GRADO Y GRUPO:

8°-B

CUATRIMESTRE:
ENERO-ABRIL

INTERRUPCIONES

En los sistemas embebidos el tiempo es un elemento muy importante.

Un procesador embebido normalmente tiene que
- medir tiempos
- generar eventos basados en tiempo
- responder en tiempo real a eventos que ocurren en tiempos impredecibles

Un procesador embebido normalmente tiene que Un sistema embebido tiene, generalmente, que atender varias tareas, algunas de ellas periódicas, otras disparadas por eventos.

Las interrupciones son cambios en el flujo de control, no ocasionados por el programa que se ejecuta, sino por algún otro suceso que necesita el servicio inmediato de la CPU por lo general relacionado con los dispositivos de E/S. Por ejemplo, un programa puede pedirle al controlador de disco que empiece a transferir información y que genere una interrupción cuando acabe la transferencia.

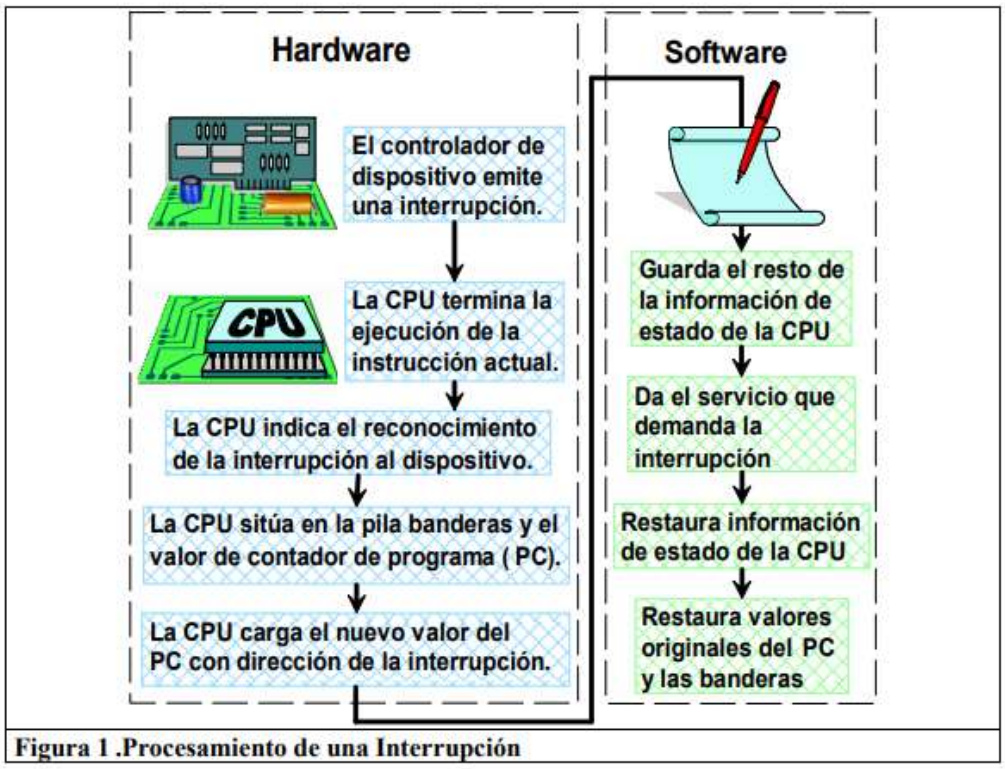
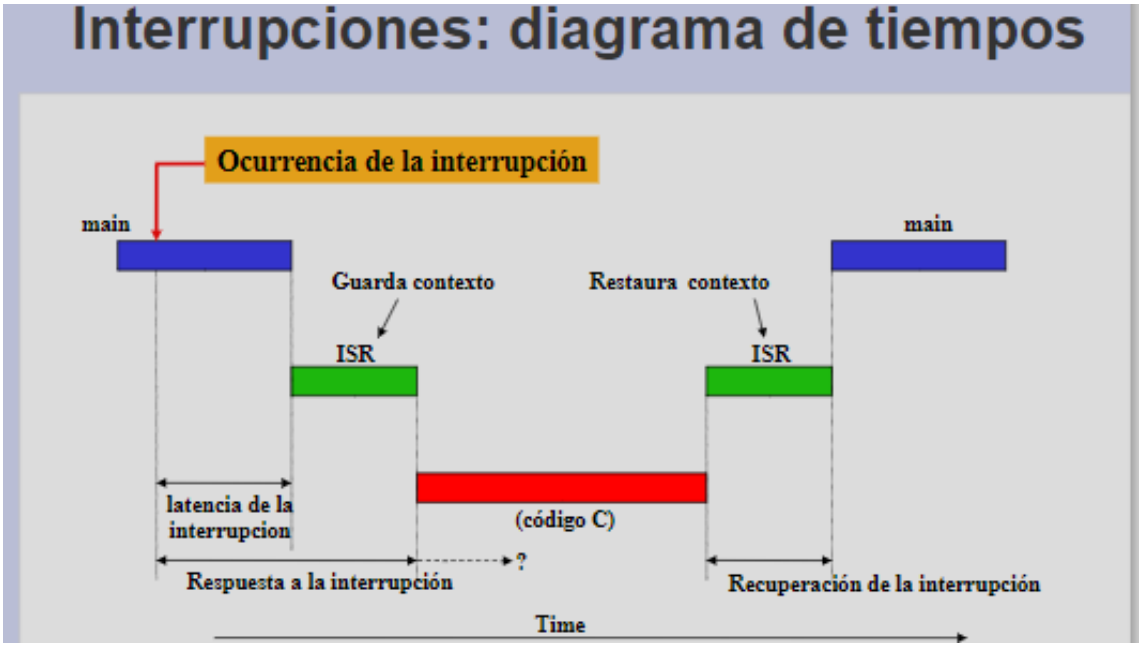


Figura 1 .Procesamiento de una Interrupción

La Figura 1 muestra, en forma muy simplificada, los pasos y los componentes involucrados en el manejo de una interrupción. La señal de petición de interrupción provoca que la CPU detenga el programa en curso, salve su estado (es decir, se guardan todos los contenidos de los registros de la CPU) y transfiera el control a una Rutina de Servicio de Interrupción, o ISR (del inglés “Interrupt Service Routine”) la cual realiza alguna acción apropiada para Luis Eduardo Leyva del Foyo 20 darle servicio a la petición. Al terminar el servicio de la interrupción, se debe continuar el código interrumpido exactamente en él mismo estado en que estaba cuando tuvo lugar la interrupción, lo cual se logra restaurando los registros internos al estado que tenían antes de la interrupción previamente salvado permitiendo continuar el flujo normal de procesamiento. Como puede observarse, un concepto clave relacionado con las interrupciones es la transparencia. Cuando se produce una interrupción, tienen efecto algunas acciones y se ejecutan algunos códigos, pero cuando todo termina, la computadora se debe regresar exactamente al mismo estado en que se encontraba antes de la interrupción.

•Escenario en sistemas de tiempo real
–Procesador ejecutando un programa
–Ocurre algo que necesita ser atendido
–Procesador pone en “espera” el hilo principal y lo atiende
•Secuencia de eventos típica
–Dispositivo que necesita atención se lo indica al
μP a través
de una señal de hardware.
–μP guarda información de la tarea en ejecución (PC) y carga
en PC la dirección de inicio de la ISR.
–Se ejecuta la ISR y al final se retorna con RETI
–Se carga en valor guardado del PC

•Atención:
–ISR comienza a ejecutarse en el “medio” del programa
–CPU está en uso (registros con valores, etc.)
–Al retornar de la ISR debe seguir como si nada hubiera
pasado
•Entonces la ISR tiene que:
–guardar el contexto (estado del procesador: SR, etc.)
–hace lo que tiene que hacer (da respuesta a la interrup.)
–restaura el contexto
–retorna de la subrutina



VENTAJAS DE LAS INTERRUPCIONES

•Retardos uniformes en respuesta a eventos
•Independiente de la complejidad de la tarea que se está ejecutando
•Las “tareas” pueden dormir hasta que ocurra un evento
–Esencial en sistemas multitarea basados en prioridades
•Estas ventajas tienen un costo

Ejemplo de uso de interrupciones

•Descripción
–Control de temperaturas en dos tanques.
–Las temperaturas deben ser iguales.
–Si son diferentes disparar alarma.
•Implementación
–Interrupción periódica para lectura de temperaturas.
–Lectura de temperatura es “instantánea”.
–Verificación de temperaturas se hace en loop principal.

control-obvio.c

```
static int iTemperatures[2];

__interrupt void vReadTemperatures(void)
{
  iTemperatures[0] = !! read in value from hardware;
  iTemperatures[1] = !! read in value from hardware;
}
  int iTemp0;
  int iTemp1;

void main (void)
{

  while(TRUE)
  {
    iTemp0 = iTemperatures[0];
    iTemp1 = iTemperatures[1];
    if (iTemp0 != iTemp1)
      !! Set off howling alarm;
  }
}
```

control_oculto.c

```
static int iTemperatures[2];

__interrupt void vReadTemperatures(void)
{
  iTemperatures[0] = !! read in value from hardware;
  iTemperatures[1] = !! read in value from hardware;
}
  int iTemp0;
  int iTemp1;

void main (void)
{

  while(TRUE)
  {
    if (iTemperatures[0] != iTemperatures[1])
      !! Set off howling alarm;
  }
}
```

control_oculto.c

```
static int iTemperatures[2];

__interrupt void vReadTemperatures(void)
{
  iTemperatures[0] = !! read in value from hardware;
  iTemperatures[1] = !! read in value from hardware;
}
  int iTemp0;
  int iTemp1;

void main (void)
{

  while(TRUE)
  {
    if (iTemperatures[0] != iTemperatures[1])
      !! Set off howling alarm;
  }
}
```

```
          MOV     R1, (iTemperature[0])
          MOV     R2, (iTemperature[1])
          SUB     R1, R2
          JCOND ZERO, TEMP_OK
          ;
          ; Codigo para disparar alarma
          ;
TEMP_OK:
```