



University of British Columbia
Electrical and Computer Engineering
Digital Systems and Microcomputers
CPEN312

Lecture 17: Interrupts

Dr. Jesús Calviño-Fraga P.Eng.
Department of Electrical and Computer Engineering, UBC
Office: KAIS 3024
E-mail: jesusc@ece.ubc.ca
Phone: (604)-827-5387

March 30, 2017

Copyright © 2009-2017, Jesús Calviño-Fraga. Not to be copied, used, or revised without explicit written permission from the copyright owner.

Objectives

- Understand how interrupts operate.
- Configure interrupt service vectors in the 8051 microcontroller.
- Enable/disable interrupts.
- Set interrupt priorities.
- Save/restore interrupt service routine registers.

Lecture 17: Interrupts

2

Copyright © 2009-2017, Jesús Calviño-Fraga. Not to be copied, used, or revised without explicit written permission from the copyright owner.

Interrupts

- Interrupt uses:
 - Handshake I/O thus preventing CPU from being tied up.
 - Providing a way to handle some errors: illegal opcodes, dividing by 0, power failure, etc.
 - Getting the CPU to perform periodic tasks: generate square waves, keep time of day, measure frequency, etc.

Lecture 17: Interrupts

3

Copyright © 2009-2017, Jesus Calvino-Fraga. Not to be copied, used, or revised without explicit written permission from the copyright owner.

Interrupts

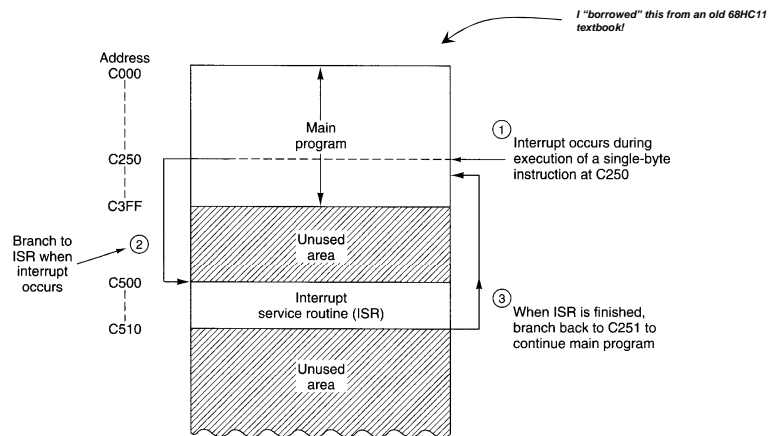


FIGURE 8.11 Example of how an interrupt causes a change in the execution of a program.

Lecture 17: Interrupts

4

Copyright © 2009-2017, Jesus Calvino-Fraga. Not to be copied, used, or revised without explicit written permission from the copyright owner.

Interrupts

- Most processors provide a way of enabling / disabling all maskable interrupts. For the 8051:
 - clr EA** ;Disable interrupts
 - setb EA** ;Enable interrupts
- Some other interrupts are non-maskable and they MUST be serviced. For example, the X86 has the “Non-Maskable Interrupt” NMI.
- Maskable interrupts can be enabled/disabled individually. For the 8051 use register IE:

Lecture 17: Interrupts

5

Copyright © 2009-2017, Jesus Calvino-Fraga. Not to be copied, used, or revised without explicit written permission from the copyright owner.

IE: INTERRUPT ENABLE REGISTER. (Address A8H)

EA	EC	ET2	ES	ET1	EX1	ET0	EX0
----	----	-----	----	-----	-----	-----	-----

Bit	Name	Description
7	EA	Interrupt Enable Bit: EA = 1 interrupt(s) can be serviced, EA = 0 interrupt servicing disabled.
6	BPE	Breakpoint Enable bit. (CV-8052)
5	ET2	Timer 2 Interrupt Enable. (8052)
4	ES	Serial Port Interrupt Enable
3	ET1	Timer 1 Overflow Interrupt Enable.
2	EX1	External Interrupt 1 Enable.
1	ET0	Timer 0 Overflow Interrupt Enable.
0	EX0	External Interrupt 0 Enable.

Lecture 17: Interrupts

6

Copyright © 2009-2017, Jesus Calvino-Fraga. Not to be copied, used, or revised without explicit written permission from the copyright owner.

() Bit addressable registers

- If the location address of an special function register (SFR) is a multiple of 8, then the register is bit addressable and you can use the **setb** and **clr** instructions.
- IE is bit addressable!

Lecture 17: Interrupts

7

Copyright © 2009-2017, Jesus Calvino-Fraga. Not to be copied, used, or revised without explicit written permission from the copyright owner.

Two external Interrupts

- Connected to pins P3.2 (INT0) and P3.3 (INT1) in the standard 8051.
- Can be configured to be edge sensitive or level sensitive. Use bits IT0 and IT1 in SFR TCON to specify falling edge or low level sensitivity.
- There is an application note (somewhere) on how to use the timer inputs T0 and T1 as additional external interrupts.

Lecture 17: Interrupts

8

Copyright © 2009-2017, Jesus Calvino-Fraga. Not to be copied, used, or revised without explicit written permission from the copyright owner.

Interrupts and the stack

- Interrupts in the 8051 make use of the stack.
- The stack is an area of memory where variables can be stacked. It is a LIFO memory: the last variable you put in is the first variable that comes out.
- Register SP (stack pointer) points to the beginning of the stack. SP in the 8051 is incremented **before** is used (**push**), or used and then decremented (**pop**).
- After reset, SP is set to 07H. If you have variables in internal RAM, any usage of the stack is likely to damage them. Therefore, at the beginning of your program set the SP:

mov SP, #7FH ; Set the stack pointer to idata start

Lecture 17: Interrupts

9

Copyright © 2009-2017, Jesus Calvino-Fraga. Not to be copied, used, or revised without explicit written permission from the copyright owner.

Interrupts and the stack

- Additionally, these two instructions can be used to push/pull registers to/from the stack: **push & pop**
- After an interrupt is asserted the CPU:
 - Pushes the address of the next instruction into the stack (two bytes). Some processors also push some or all of the registers into the stack as well (not the 8051 though!).
 - All interrupts of equal or lower priority are disabled.
 - Then the program counter (PC) is set to the Interrupt Service Routine (ISR) vector.
 - The PC will be restored to the interrupted point once the **reti** instruction is executed in the ISR and all interrupts of equal or lower priority are re-enabled.

Lecture 17: Interrupts

10

Copyright © 2009-2017, Jesus Calvino-Fraga. Not to be copied, used, or revised without explicit written permission from the copyright owner.

Interrupt Service Routines (ISR) Vectors

- The 8051 will *lcall* to an specific memory location when an interrupt occurs. The may be different for different 8051 variants. For the CV-8052:

Interrupt source	Address
External 0	0003H
Timer 0	000BH
External 1	0013H
Timer 1	001BH
Serial port	0023H
Timer 2	002BH

Lecture 17: Interrupts

11

Copyright © 2009-2017, Jesus Calvino-Fraga. Not to be copied, used, or revised without explicit written permission from the copyright owner.

Interrupt Service Routines (ISR) Vectors

- Notice that there are only 8 bytes available between vectors. Not enough for a decent ISR, but more than enough for a *ljmp* instruction!
- IF you enable a particular interrupt, there **MUST** be an ISR, or your program **WILL** crash. A fool proof code technique is to setup all the ISR vectors and place a *reti* (return from interrupt) instruction for those that are not used (next example).
- In assembly language you can use the “*org*” directive to set an ISR vector.
- To return from an ISR use the *reti* instruction. To return from a normal routine use the *ret* instruction.

Lecture 17: Interrupts

12

Copyright © 2009-2017, Jesus Calvino-Fraga. Not to be copied, used, or revised without explicit written permission from the copyright owner.

Example 1

```
; Basic interrupt setup

; We need the register definitions for the 8052:
$MOD52

org 0h
ljmp myprogram

; Interrupt Service Routines (see table 5.3 of the textbook)
; Notice that there is not much space to put code between
; service routines, but enough to put a ljmp!

; External interrupt 0
org 3h
reti

; Timer 0 interrupt
org 0bh
reti
```

Dummy ISRs

WARNING: org directives must be sequential!

Lecture 17: Interrupts

13

Copyright © 2009-2017, Jesus Calvino-Fraga. Not to be copied, used, or revised without explicit written permission from the copyright owner.

Example 1 (cont.)

```
; External interrupt 1
org 13h
reti

; Timer 1 interrupt
org 1bh
reti

; Serial port interrupt
org 23h
reti

; Timer 2 interrupt
org 2bh
reti

; Dummy program, just to compile and see...
myprogram:
    mov R1, #00H ; do something
    sjmp myprogram
END
```

Dummy ISRs

Lecture 17: Interrupts

14

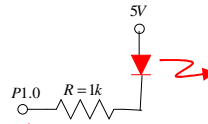
Copyright © 2009-2017, Jesus Calvino-Fraga. Not to be copied, used, or revised without explicit written permission from the copyright owner.

Example 2: Do something (useful) in the main program.

; This program makes an LED connected to P1.0 blink

```
myprogram:
    cpl P1.0
    mov R0, #200
L0:
    djnz R0, L1
    jmp myprogram
L1:
    mov R1, #200
L2:
    djnz R1, L2
    jmp L0
```

This is the Complement bit instruction



In general, MCU pins are better at sinking current than sourcing current

Lecture 17: Interrupts

15

Copyright © 2009-2017, Jesus Calvino-Fraga. Not to be copied, used, or revised without explicit written permission from the copyright owner.

Example 3: Enable timer 0 interrupt and setup an ISR

```
myprogram:
    ; Enable timer 0
    mov a, TMOD
    anl a, #0f0H
    orl a, #0000010B ; 8-bit auto reload timer (this is in binary)
    mov TMOD, a
    mov TH0, #080H ; Set the interrupt rate
    setb TR0 ; Enable timer 0
    setb ET0 ; Enable timer 0 interrupt
    setb EA ; Enable all interrupts!
```

```
Blink:
    cpl P1.0
    mov R0, #200
L0:
    djnz R0, L1
    jmp Blink
L1:
    mov R1, #200
L2:
    djnz R1, L2
    jmp L0
```

Lecture 17: Interrupts

16

Copyright © 2009-2017, Jesus Calvino-Fraga. Not to be copied, used, or revised without explicit written permission from the copyright owner.

Example 3 (cont.) the ISR.

```
; Timer 0 interrupt
org 0bh
cpl P1.1 ; Check this pin with the scope!
reti
```

Lecture 17: Interrupts

17

Copyright © 2009-2017, Jesus Calvino-Fraga. Not to be copied, used, or revised without explicit written permission from the copyright owner.

Example 4: use a *ljmp* to go to the ISR

```
; Timer 0 interrupt
org 0bh
ljmp timer0_ISR

; Other ISR vectors come here! (Not shown to save space)

; Actual ISR for timer 0.
timer0_ISR:
cpl P1.1
reti
```

Lecture 17: Interrupts

18

Copyright © 2009-2017, Jesus Calvino-Fraga. Not to be copied, used, or revised without explicit written permission from the copyright owner.

Saving and Restoring Registers in the Stack

- If your ISR routine uses a register, you must make sure that it will remain **unmodified** before returning to the interrupted program. Example, if ACC was 33 when the ISR was called, it must be set back to 33 before the *reti*.
- As mentioned before you use the instructions *push/pop* to save/restore registers to/from the stack.
- Additionally, you could use one of four available register banks in your ISR.

Lecture 17: Interrupts

19


Copyright © 2009-2017, Jesus Calvino-Fraga. Not to be copied, used, or revised without explicit written permission from the copyright owner.

Example 5: Saving and Restoring Registers in the Stack

```
; Actual ISR for timer 0. There must be a ljmp at address 0BH
timer0_ISR:
    ; The main loop is using both registers R0 and R1,
    ; so if we want to use them in this ISR we should push them
    ; into the stack and restore them before reti.
    push AR0
    push AR1

    cpl P1.1
    mov R1, #55H ;Wreck R1 and R0 so to show that program works!
    inc R0

    ; Restore the register to their original values
    pop AR1
    pop AR0
    reti ; Return from interrupt
```



The 'A' stands for address...

Lecture 17: Interrupts

20

Copyright © 2009-2017, Jesus Calvino-Fraga. Not to be copied, used, or revised without explicit written permission from the copyright owner.

Saving and Restoring Registers in the Stack

- Before using the stack make sure you set the SP register.
- Popular registers to push/pop in ISRs: ACC, DPL, DPH, PSW, R0 to R7. Of course, only if they are used in the ISR.
- Pop registers from the stack in the **REVERSE** order you pushed them! Remember the stack is a LIFO.

Lecture 17: Interrupts

21

Copyright © 2009-2017, Jesus Calvino-Fraga. Not to be copied, used, or revised without explicit written permission from the copyright owner.

Setting the SP register

```
myprogram:
    ; After reset, the stack pointer register is set to 07h
    ; We may need space for variables, so move the SP
    mov SP, #7FH

    ; Enable timer 0
    mov a, TMOD
    anl a, #0F0H
    orl a, #00000010B ; 8-bit auto reload timer
    mov TMOD, a
    mov TH0, #080H ; Set the interrupt rate (see formula in the book)
    setb TR0 ; Enable timer 0
    setb ET0 ; Enable timer 0 interrupt
    setb EA

Blink:
    cpl P1.0
    mov R0, #200

L0:
    djnz R0, L1
    jmp Blink

L1:
    mov R1, #200

L2:
    djnz R1, L2
    jmp L0
```

Lecture 17: Interrupts

22

Copyright © 2009-2017, Jesus Calvino-Fraga. Not to be copied, used, or revised without explicit written permission from the copyright owner.

Register Banks

- Bits RS1 (bit 4) and RS0 (bit 3) of the Program Status Word (PSW) select one of four available register banks.
- After a power-on reset, register bank 0 is selected.
- Normally you will use a register bank for each different interrupt priority level.

Lecture 17: Interrupts

23

Copyright © 2009-2017, Jesus Calvino-Fraga. Not to be copied, used, or revised without explicit written permission from the copyright owner.

Example 6: Using different register banks in ISRs

```
; Actual ISR for timer 0. Notice that there must be a ljmp at address 0BH
timer0_ISR:
    ; Another possibility is to use a different register bank. First,
    ; Save the two registers we are going to use:
    push ACC
    push PSW
    mov PSW, #00001000B ; Select register bank 1!

    cpl P1.1
    mov R1, #55H ; Wreck R1 and R0 so to demonstrate that program works!
    inc R0
    clr a ; Change also the accumulator...

    ; Restore the registers to their original values
    pop PSW
    pop ACC
    reti ; Return from interrupt
```

Alternatively we can use:

```
clr RS1
setb RS0
```

Question: What register bank will be selected after *reti*? Why?

Lecture 17: Interrupts

24

Copyright © 2009-2017, Jesus Calvino-Fraga. Not to be copied, used, or revised without explicit written permission from the copyright owner.

Interrupt Priority Levels

- The original 8051 has only two interrupt priority levels: high priority and low priority.
- An ISR can only be interrupted by an interrupt of higher priority than its own. If you don't want your ISR interrupted, just disable interrupts (**clr** EA).
- Newer 8051 have four interrupt priority levels. You control the interrupt priority using the IP (also referred as IP0) and IP1.

Lecture 17: Interrupts

25

Copyright © 2009-2017, Jesus Calvino-Fraga. Not to be copied, used, or revised without explicit written permission from the copyright owner.

IP: INTERRUPT PRIORITY REGISTER. (Address B8H)

--	PBP	PT2	PS	PT1	PX1	PT0	PX0
----	-----	-----	----	-----	-----	-----	-----

Bit	Name	Description
7	--	Reserved
6	PBP	Breakpoint priority bit (CV-8052)
5	PT2	Timer 2 Interrupt priority bit.
4	PS	Serial Port Interrupt priority bit
3	PT1	Timer 1 Overflow Interrupt priority bit.
2	PX1	External Interrupt 1 priority bit.
1	PT0	Timer 0 Overflow Interrupt priority bit.
0	PX0	External Interrupt 0 priority bit.

Lecture 17: Interrupts

26

Copyright © 2009-2017, Jesus Calvino-Fraga. Not to be copied, used, or revised without explicit written permission from the copyright owner.

Default Priority for Interrupts

Interrupt source	
External 0	<div>(Highest) ↑ (Lowest)</div>
Timer 0	
External 1	
Timer 1	
Serial port	
Timer 2	

Lecture 17: Interrupts

27

Copyright © 2009-2017, Jesus Calvino-Fraga. Not to be copied, used, or revised without explicit written permission from the copyright owner.

Interrupt Programming with the 8051 in Assembly (summary)

- Set a ***ljmp*** to the ISR into the corresponding memory address for each interrupt source.
- Setup the stack in the main program. (Do this only once!)
- Setup (including priority) and Enable the interrupt to use.
- In the ISR use ***push/pop*** to save restore used registers. You may also use a different register bank.
- Use a ***reti*** instruction to return from the ISR.

Lecture 17: Interrupts

28

Copyright © 2009-2017, Jesus Calvino-Fraga. Not to be copied, used, or revised without explicit written permission from the copyright owner.

Exercises

- Write an Interrupt service routine for timer 0 that generates a 1 kHz square wave in pin P0.0 of the CV-8052 processor.
- Write an interrupt service routine for timer 2 that increments a two digit BCD counter displayed in the 7-segment displays HEX1 and HEX0 of the CV-8052 every second. Make sure that the ISR for this question and the ISR from the previous question can run concurrently in the same processor.

Lecture 17: Interrupts

29

Copyright © 2009-2017, Jesus Calvino-Fraga. Not to be copied, used, or revised without explicit written permission from the copyright owner.