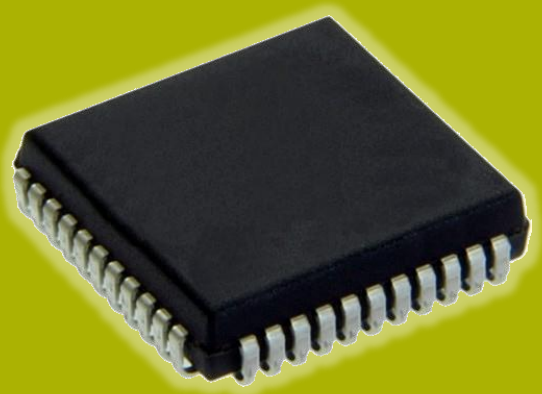
Sensors and Microsystem Electronics: µControllers

Part2: About timers, interrupts, keyboard and display

By Robin Deleener & Stijn Bettens
rdeleene@etro.vub.ac.be (K.5.56)
sbettens@etro.vub.ac.be (PL9.1.63)



Vrije Universiteit Brussel

Last lab session

What did we do?

Questions?

More information?

PointCarré!

- Getting started file + Common mistakes
- 8051 instructionset
- Datasheet T89C51CC01
- Board schematic: MK_DISP + MK_CPU

Extra information:

- *http://www.mikroe.com/chapters/view/68/chapter-5-assembly-language/*
- *8051 tutorial on 8052.com*

Overview

- Timers
- Interrupts
- Simulations
- Timer exercise
- Keyboard readout
- Display data

Timers

- Different timers
- Different modes
- Counting up or down?
- Watchdog timer

Timers

- Timer 0 and timer 1 are identical, except for mode 3
- Mode 0,1,2,3
- Registers: TCON, TMOD, TH0, TL0, TH1, TL1

Timer 0,1: modes

- Mode 0: 13-bit timer (backwards compatibility)
- Mode 1:16-bit timer
  - TL0=lower bits
    TH0=higher bits
- Mode 2: 8-bit timer
  - Auto-reload
  - TH0 unchanged, but can be done manually
- Mode 3: timer0=2 8-bit timers
  - Timer1 can be used, but without TF1-flag

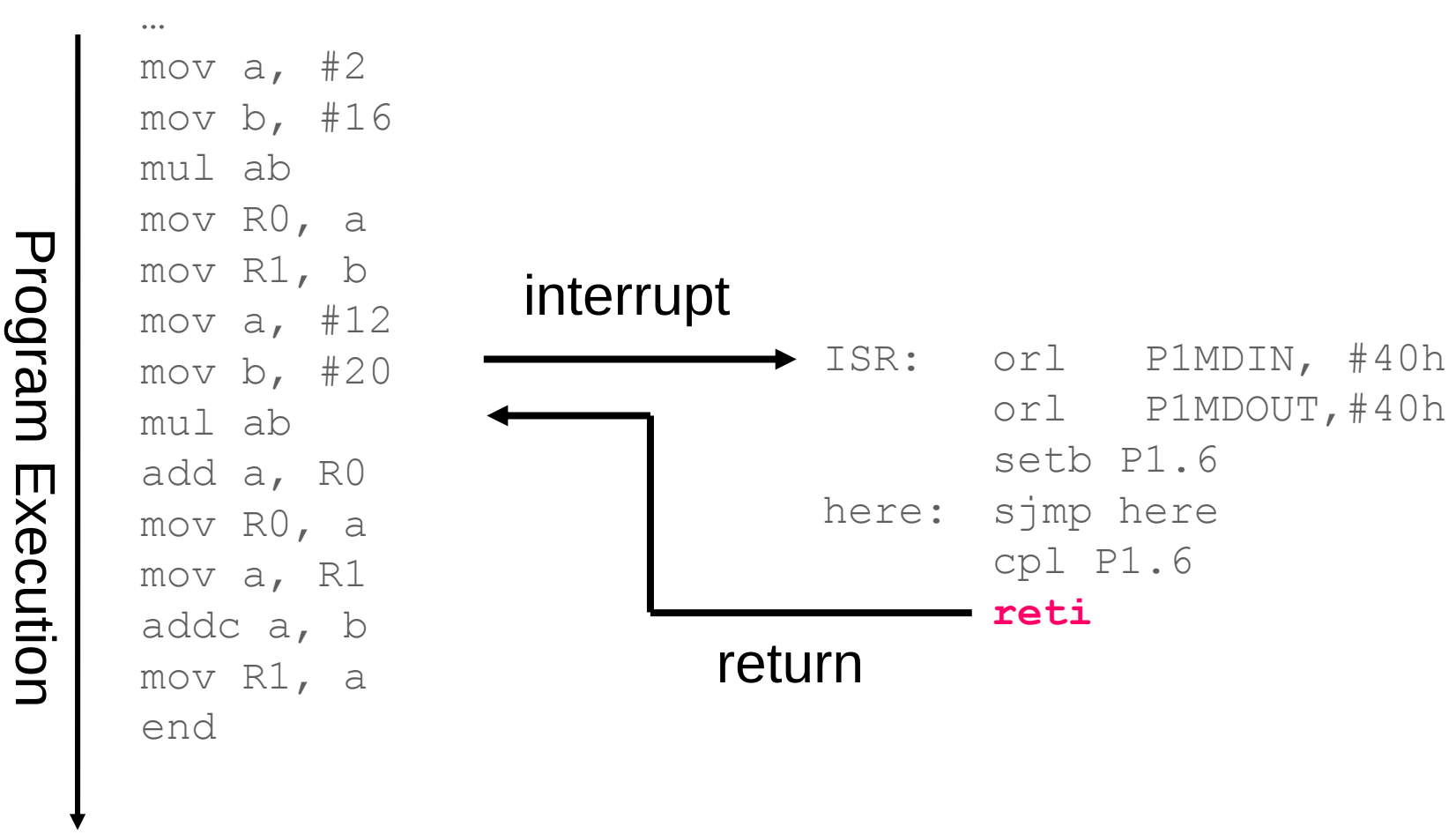
Timer 2

- No standard 8051 architecture
- 16-bit timer/counter, with auto-reload
- Up/down counting

Watchdog Timer

- Resets the µC when it crashes
- Critical applications
- Applications where human interaction is not wanted / impossible
- Once enabled, disable only possible by a reset
- Maximum 2 sec

Interrupts?


Program Execution
…
mov a, #2
mov b, #16
mul ab
mov R0, a
mov R1, b
mov a, #12
mov b, #20
mul ab
add a, R0
mov R0, a
mov a, R1
addc a, b
mov R1, a
end
interrupt
ISR: orl P1MDIN, #40h
orl P1MDOUT,#40h
setb P1.6
here: sjmp here
cpl P1.6
reti
return

Interrupt: what happens?

- Stack ← PC
- SP++
- PC ← Interrupt address vector
- Lower priority interrupts are blocked
- Instructions are enclosed by RETI
- RETI? PC ← SP--

Interrupts

- 14 sources
- 4 priority levels
- Interrupt registers: enable interrupts!
- Instructions: normally sequential, but interrupts can change program order

how to configure interrupts?

- Interrupt Enable registers:
  IE0, IE1
- Priority registers:
  IPL0, IPH0, IPL1, IPH1
- Interrupt Address Vectors

OR:

- Bitwise: EA, ET0, … → datasheet

Interrupt address vectors

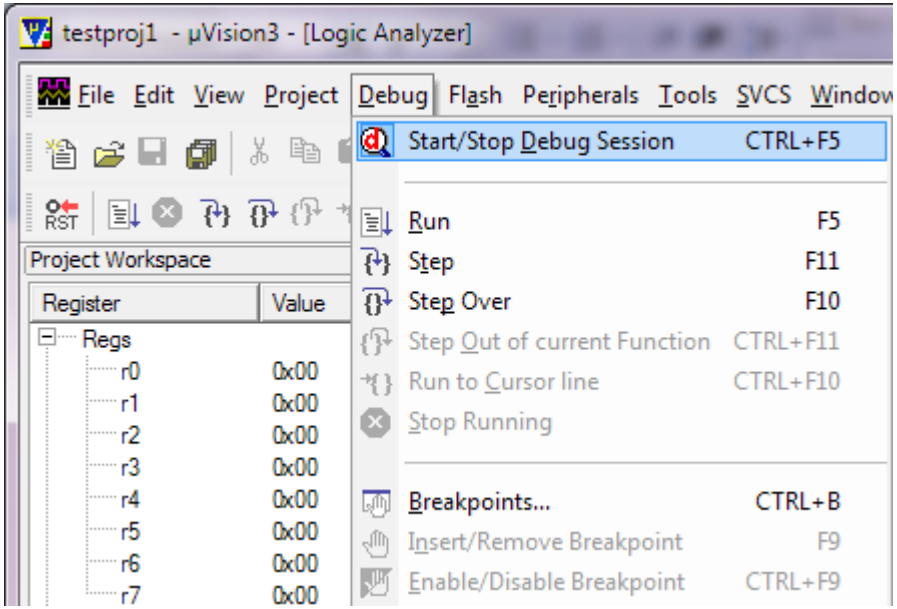
Interrupt Name	Abbrev.	Interrupt Address Vector
External interrupt	INT0	0003h
Timer0	TF0	000Bh
External interrupt	INT1	0013h
Timer1	TF1	001Bh
PCA	CF or CCFn	0033h
UART	RI or TI	0023h
Timer2	TF2	002Bh
CAN	Txok, Rxok, Err or OvrBuf	003Bh
ADC	ADCI	0043h
CAN Timer Overflow	OVRTIM	004Bh

Interrupt: take care

- RETI in stead of RET
- Interrupt can change registers, A, B, DPTR, …
- Interrupt can be paused by a higher priority interrupt
- # PUSH = # POP

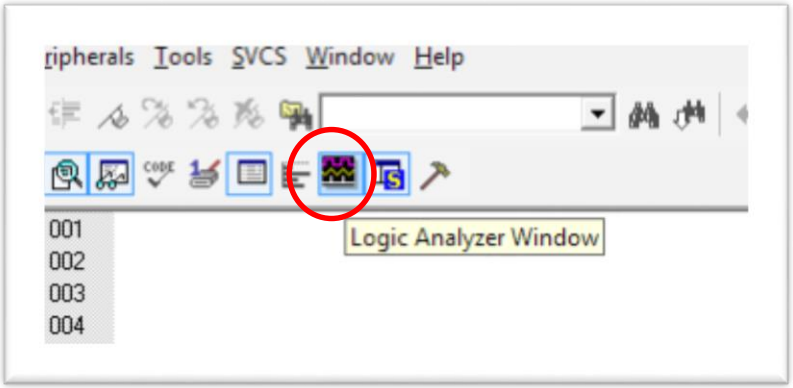
Simulator: opening

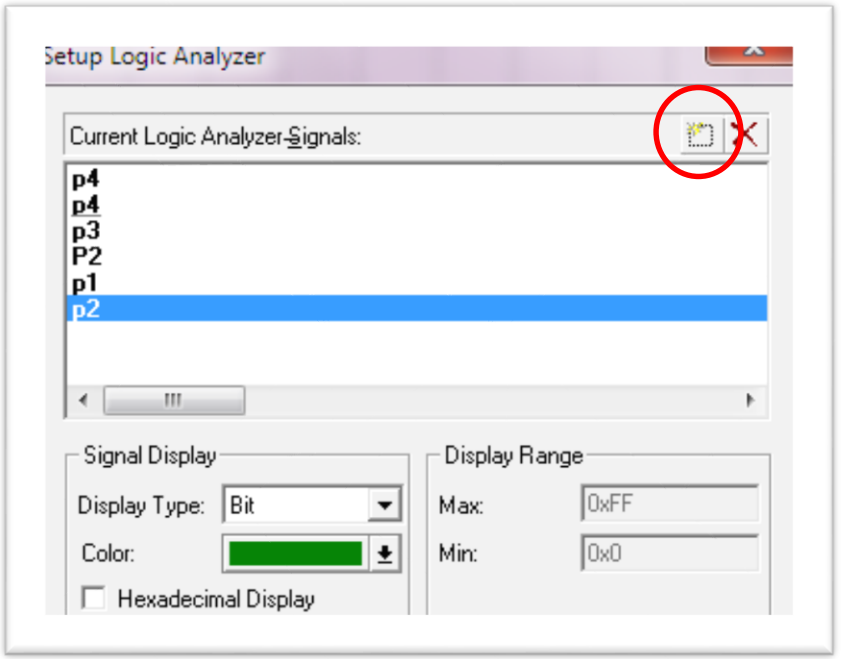
Debug – Start/Stop Debug Session



Simulator: Waveform Graph

View – Logic Analyzer Window
Debug – Setup Logic Analyzer

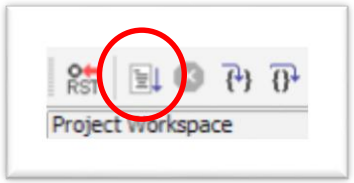




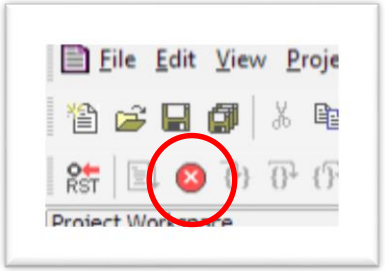
Simulator: start / stop

Adding breakpoints: double click on an empty space in your code

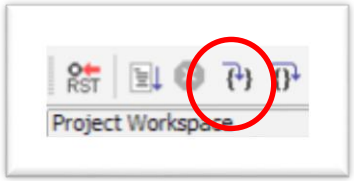
RUN



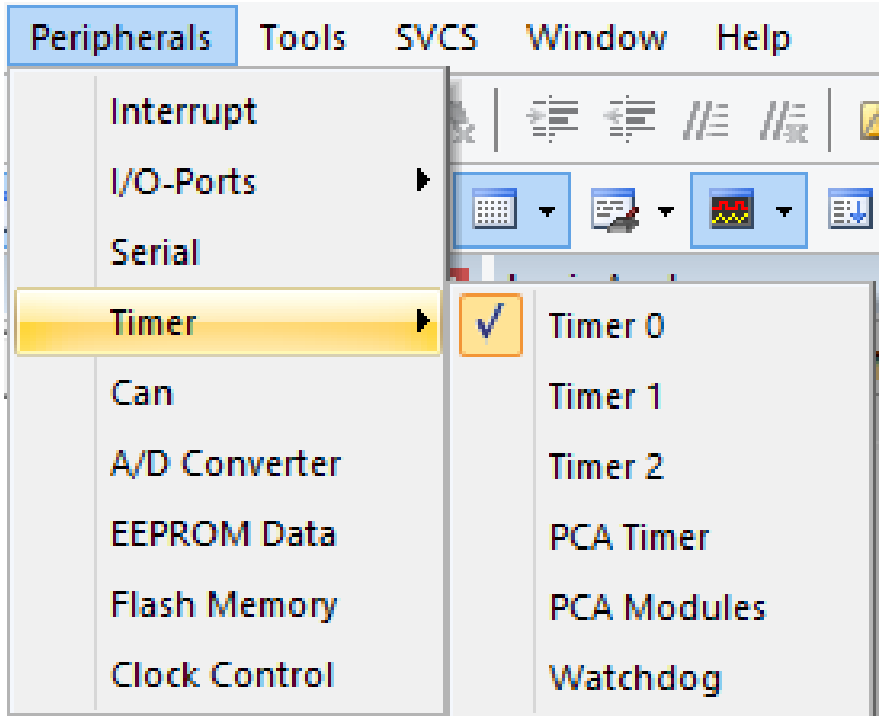
STOP



STEP-by-STEP



Simulator: peripherals



First timer-program

- Make a program that plays a sound (buzzer=P2.2) of 440 Hz when P2.6 is pressed and P2.5 is low. Double the frequency if P2.5 is high. Use timer 0.
- First (simulate) without buttons -> Task 5.
- Then (simulate) with buttons -> Task 6.

- Compare with a tuning fork.

Roadmap

1. Draw the signal that is sent to the buzzer.
2. Define interrupt.
3. What is the length of the time interval between 2 successive interrupts.
4. Select appropriate mode of timer 0.
5. Enable interrupt.
6. Define Interrupt Service Routine (ISR).
7. Add buttons P2.5 and P2.6.

440/880Hz: possible solution

```
ORG 0000h
        LJMP init
ORG 000Bh
        LJMP timer0int

init:   MOV TMOD,#01h
        MOV TH0,#0FBh
        MOV TL0,#08Fh
        SETB ET0
        SETB EA
        SETB TR0
```

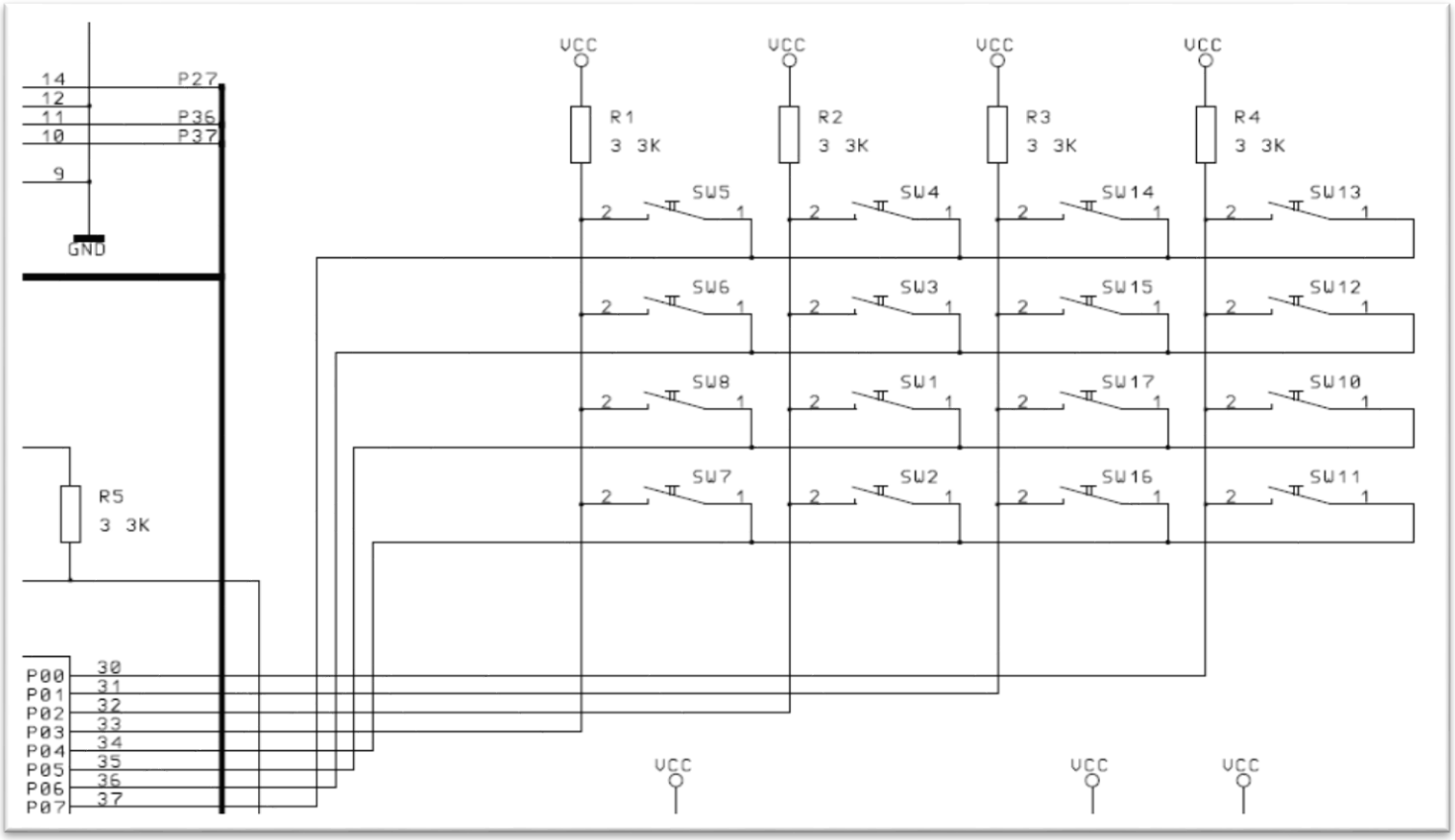
```
main:     LJMP $

timer0int:JB P2.6,nosound
          CPL P2.2
nosound:JB P2.5,Hz880
Hz440:    MOV TH0,#0FBh
          MOV TL0,#08Fh
          LJMP endint
Hz880:    MOV TH0,#0FDh
          MOV TL0,#0C7h
endint:   RETI

end
```

440/880Hz: with an interrupt

```
ORG 0000h
        LJMP init
ORG 000Bh
        LJMP timer0int

init:   MOV TMOD,#01h
        MOV TH0,#0FBh
        MOV TL0,#08Fh
        SETB ET0
        SETB EA

main:   MOV C,P2.6
        CPL C
        MOV TR0, C
        LJMP main
```

```
timer0int:CPL P2.2
        JB P2.5,Hz880
Hz440:  MOV TH0,#0FBh
        MOV TL0,#08Fh
        LJMP endint
Hz880:  MOV TH0,#0FDh
        MOV TL0,#0C7h
endint: RETI

end
```

First difficulty: keyboard



The real work: The display

- P4.1: data
- P4.0: shift (clk)
- P3.2: store → datasheet of the 74HC595

- Bitstream → schematic
- Active low / high? → Schematic
- ! A line **must not** be on the whole time !
- How do I get the brightest display?

Project ideas?

- Feasibility ?
- External peripherals ?
  - Note: only allowed when all 10 tasks are finished after 3th lecture
  - Make them yourself!

GOOD LUCK WITH YOUR PROJECT !