EdSim51 I/O

Pai H. Chou

Concept

- Direct, indirect, index addressing
  - table lookup
- Call-return
- Serial port
  - Polling vs. interrupt
  - sharing of interrupt structure
- Timer interrupt

Assembly Language

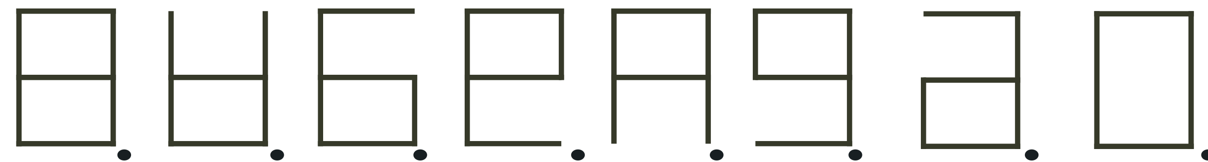
- Directives
  - Commands to the assembler!
    e.g., starting address, allocate memory, ...
- Instructions
  - Correspond to machine instructions
- Labels
  - Symbolic names that mark addresses
- Comments -- started with ;; till end of line

Example of directives: DB vs. EQU

- DATA1: DB "Hello world"
  DATA2: DB 25
  ;; both occupy space in **code memory**,
  ;; because DATA1, DATA2 are labels
  ;; the data is read-only.
- This is somewhat like
  const char DATA1[] = "Hello world"; /* occupies memory*/
  const byte DATA2[1] = {25}; /* occupies memory! */
- COUNT EQU 25 ;; occupies no space
  - MOV R3, #COUNT
    ;; macro expansion into MOV R3,#25
  - This is like #define COUNT 25 /* does not occupy memory */

Back to LED example...

segments to light



Value to write to P1

0x0 0x1 0x2 0x4 0x8 0x10 0x20 0x40

so.. to display digit patterns,

need to write...

0xC0 0xF9 0xA4 0xB0 0x99 0x92 0x82 0xF8 0x80 0x90

Example: define a look-up table for 7-segment LEDs!

```
•           ORG 0H
            CLR    A          ;;  A = 0
Top:        PUSH   ACC        ;; save accumulator
            LCALL  Display
            POP    ACC        ;; restore accumulator
            INC    A          ;;  A++
            JMP    Top
;;  Display is a subroutine
Display:  ;; assume index 0..9 is in A
            MOV    DPTR, #LEDdata
            MOVC A, @A+DPTR  ;; A = LEDdata[A]
            MOV    P1, A         ;; light up LED segments
            RET                  ;; return from subroutine
            ;; data for the table
LEDdata: DB  0C0H, 0F9H, 0A4H, 0B0H, 99H, 92H, 82H, 0F8H, 80H, 90H
            END
```

Run with breakpoint

After "Assm", double-click on the address 0011
(for RET instruction) => set breakpoint and click Run

System Clock (MHz) 12.0 | 1 Update Freq.

SBUF

R/O	W/O	TH0	TL0
0x00	0x00	0x00	0x00

RXD 1, TXD 1, SCON 0x00, TMOD 0x00, TCON 0x00

Register	Value	Register	Value
R7	0x00	B	0x00
R6	0x00	ACC	0x00
R5	0x00	PSW	0x00
R4	0x00	IP	0x00
R3	0x00	IE	0x00
R2	0x00	PCON	0x00
R1	0x00	DPH	0x00
R0	0x00	DPL	0x00
		SP	0x07

pins	bits	Port
0xFF	0xFF	P3
0xFF	0xFF	P2
0xFF	0xFF	P1
0xFF	0xFF	P0

TH1 0x00, TL1 0x00

8051

PC 0x0000 | i | PSW 0 0 0 0 0 0 0 0

Modify Code: Code Memory, addr 0x0000, 0xE4 value

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	E4	C0	E0	12	00	0B	D0	E0	04	80	F6	90	00	12	93	F5
10	90	22	C0	F9	A4	B0	99	92	82	F8	80	90	00	00	00	00
20	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

RST | Step | Run | New | Load | Save | Copy | Paste | X

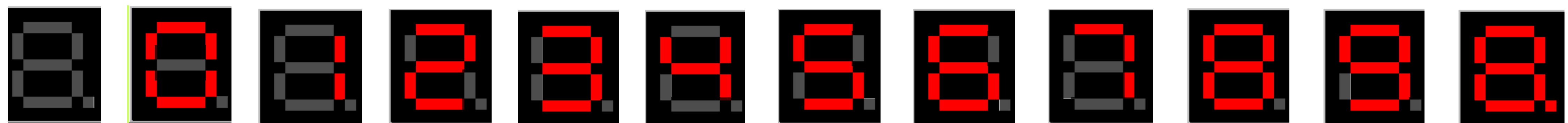
No errors

```
                    ORG  0H
0000|               CLR  A
0001| Top:            PUSH ACC
0003|               LCALL Display
0006|               POP  ACC
0008|               INC  A
0009|               JMP  Top
000B| Display:      MOV  DPTR, #LEDdata
000E|               MOVC  A, @A+DPTR
....|               MOV  P1, A
0011*               RET
              ; lookup table
      LEDdata:      DB   0C0H, 0F9H, 0A4H, 0
                    END
```

break point

LEDdata array in code memory

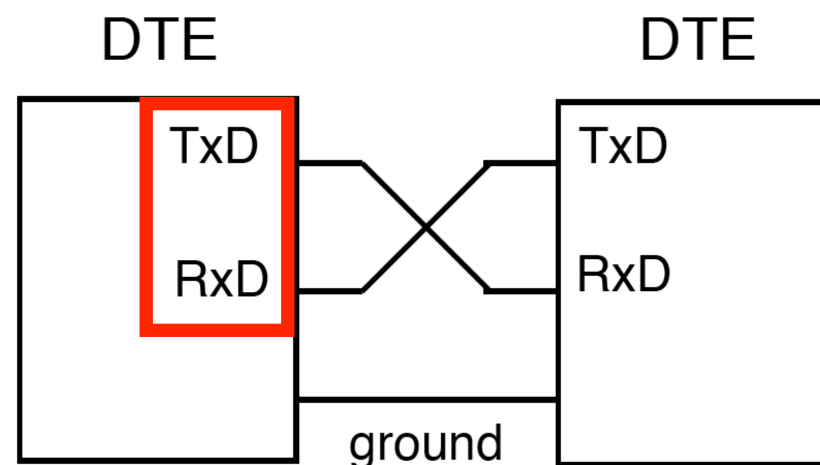
output sequence



Serial Port

- Universal Asynchronous Receiver Transmitter
  - Serial: data shifted in/out serially
  - Asynchronous: no clock; embedded in data
  - Full duplex: Rx (receive) & Tx (transmit) are independent controllers
  - Both sides must run at the same baud rate

e.g., terminal



e.g., mainframe

Accessing UART on MCU

- Configuration
  - Set up a timer with auto-reload to generate timing
  - Enable Rx or Tx (or both)
- Access
  - Reading/Writing register SBUF
  - Test RI or TI flag before reading or writing SBUF!
  - could be polling or interrupt driven

Serial port programming on the 8051

- Easy part: send/receive
  - ```
    MOV  SBUF, data    ;; to send
    MOV  dest, SBUF    ;; to receive
    ```
- Tricky part: initialize the baud rate
  - (just copy the following code for now to run)
  - ```
    MOV  TMOD, #20H    ;; to send
    MOV  TH1, #-6      ;; 4800 baud
    MOV  SCON, #50H    ;; 8-bit 1 stop REN
    SETB  TR1          ;; start timer 1
    ```
  - Run EdSim51 @11.0592MHz for 4800 baud

Polling before accessing SBUF

- Test RI flag before reading from SBUF
  - if RI is false => no valid data has been received!
- Solution: polling RI flag
  - Repeatedly checking RI until it is true
  - after exiting loop, read SBUF and clear RI flag.
  - ```
    PollHere:   JNB  RI,  PollHere ;; while (!RI) ;
                MOV A, SBUF        ;; read it into A
                CLR  RI
    ```
    ```

Code for reading digits from Serial Port and display on LED

```
            ORG 0H
            ;; initialize serial port
            MOV  TMOD, #20H    ;; to send
            MOV  TH1, #-6      ;; 4800 baud
            MOV  SCON, #50H    ;; 8-bit 1 stop REN
            SETB  TR1          ;; start timer 1
PollHere:   JNB   RI, PollHere ;; polling
            MOV  A, SBUF       ;; read serial port
            CLR   RI           ;; clear out receive flag
            ADD   A, #-48      ;; convert ASCII to binary
            LCALL  Display
            JMP     PollHere
Display:    MOV    DPTR, #LEDdata
            MOVC A, @A+DPTR    ;; A = LEDdata[A]
            MOV    P1, A       ;; light up LED seg
            RET                ;; return from subroutine
LEDdata:    DB  0C0H, 0F9H, 0A4H, 0B0H, 99H, 92H, 82H, 0F8H, 80H, 90H
            END
```

Setting EdSim51 with proper baud rate

11.0592 MHz   100 Update Freq. may be good

System Clock (MHz) 11.0592
SBUF

100 Update Freq.

RST Step Run New Load Save Copy Paste

No errors

```
                 ORG   0H
                 ;; initialize serial port
0000|            MOV   TMOD, #20H     ;; to
0003|            MOV   TH1, #-6        ;;
0006|            MOV   SCON, #50H    ;; 8-bit
0009|            SETB  TR1

           PollHere:
000B|            JNB     RI, PollHere     ;;
000E|            MOV   A, SBUF             ;;
0010|            CLR     RI
0012|            ADD     A, #-48     ;; conver
0014|            LCALL   Display
0017|            JMP     PollHere

           Display:  ;; assume index 0..9
0019|            MOV     DPTR,   #LEDdata
001C|            MOVC A, @A+DPTR  ;; A = LE
001D|            MOV     P1, A
001F|            RET
                                ;; data for th
           LEDdata:
                 DB 0C0H, 0F9H, 0A4H, 0B0H
                 END
```

R/O	W/O	TH0	TL0				
0x00	0x00	0x00	0x00	R7	0x00	B	0x00
				R6	0x00	ACC	0x00
RXD	TXD			R5	0x00	PSW	0x00
1	1	TMOD	0x00	R4	0x00	IP	0x00
SCON	0x00	TCON	0x00	R3	0x00	IE	0x00
				R2	0x00	PCON	0x00
pins	bits	TH1	TL1	R1	0x00	DPH	0x00
0xFF	0xFF P3	0x00	0x00	R0	0x00	DPL	0x00
0xFF	0xFF P2		8051			SP	0x07
0xFF	0xFF P1	PC	0x0000	ACC	0 0 0 0 0 0 0 0		
0xFF	0xFF P0						

Code Memory   Modify Code addr 0x0000 0x75 value

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	75	89	20	75	8D	FA	75	98	50	D2	8E	30	98	FD	E5	99
10	C2	98	24	D0	12	00	19	80	F2	90	00	20	93	F5	90	22
20	C0	F9	A4	B0	99	92	82	F8	80	90	00	00	00	00	00	00
30	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
40	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
50	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
60	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Copyright ©2005-2013 James Rogers

Remove All Breakpoints

DI i LD

7 6 5 4 3 2 1 0

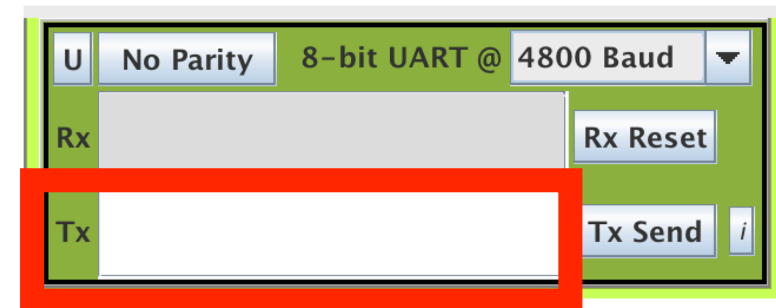
1 2 3 4 5 6 7 8 9 * 0 #

AND Gate Disabled
Key Bounce Disabled
Standard

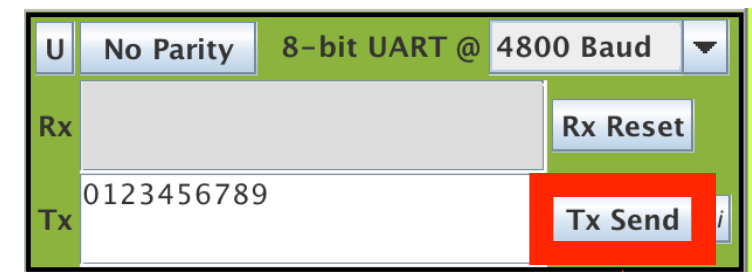
U No Parity 8-bit UART @ 4800 Baud
Rx Rx Reset
Tx Tx Send

Testing Serial Port

- Provide your test data in Tx box
  - e.g., 0123456789 as ASCII
  - data you type here are staged to be received by the 8051's Rx.
- The MCU won't receive anything until you click Tx Send to start sending
- sent characters will be "consumed" and removed from the Tx field



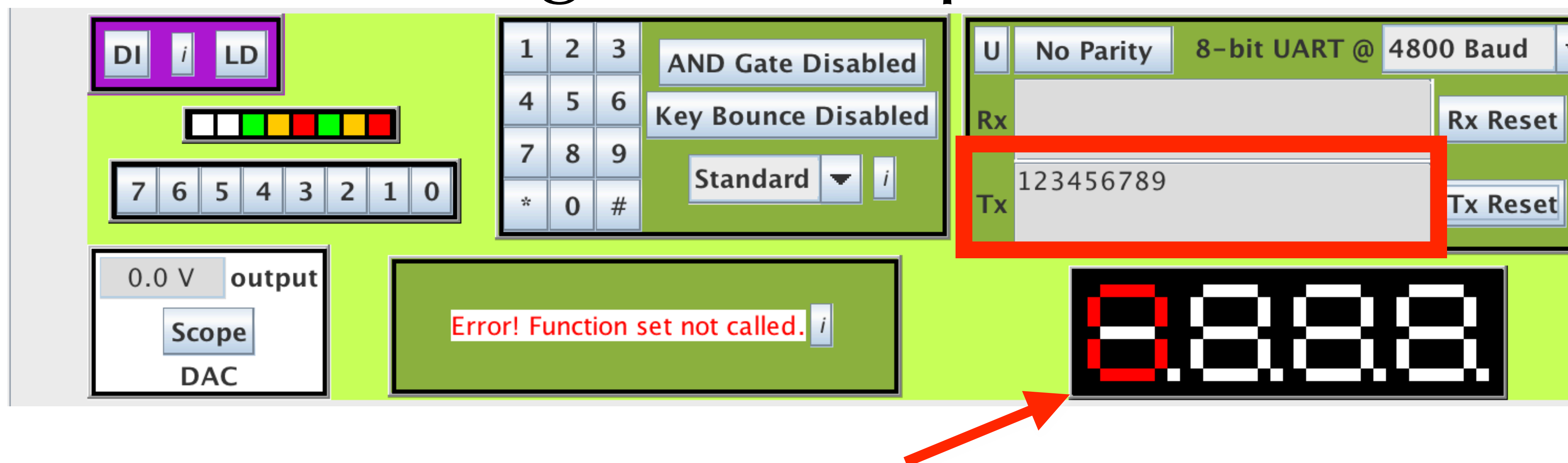
type in here!!



click to start sending

Running serial port code

- Set breakpoint at RET (address 001F)
- Click Run. PC stuck at 0x000B polling
- Click Tx Send to start sending
- On first breakpoint, shows '0' on LED
  - Tx window gobbled up character 0



Disadvantages with Polling

- Polling: e.g., **while** (TF0==0) { }
  - use loop, keep testing a flag until it is set
  - Problem: Wasteful=> not useful work
- Could try polling less often
  - e.g., **while** (TF0==0) { *do some work* }
  - Problem: potentially slow response / long latency

Solution: Interrupts

- Let hardware test flag instead of software
- When the flag is set,
  automatically call a subroutine (handler)
  - This means "interrupting" (suspend) the normal software execution in handler
- Handler returns to normal software
  - Software might not "know" it happened!

Polling vs. Interrupt

polling loop

```
setup (e.g., timer);
while (TF0 == 0) {
}      // wasted cycles!
```

"handling"

```
TF0 = 0;
other code to "handle" timer
```

vs

```
setup (enable interrupt)
regular program code
```

```
ISR(for timer, UART, etc) {
        TF0 = 0;
        other code to handle..
}
```

ISR is called automatically when the interrupt condition is detected by hardware

Terminology

- Interrupt vector:
  - address of an interrupt service routine
- Interrupt vector table:
  - data structure of interrupt vectors
- Interrupt service routine (ISR)
  - also known as *interrupt handler*
  - called by a processor to handle an interrupt

Steps in an Interrupt

- CPU finishes current instruction
- CPU pushes next PC on stack,
  save other interrupt status in internal reg
- CPU Jumps to the interrupt vector (address of ISR)
- CPU runs until RETI (return from **interrupt**)
  => don't use RET -(for regular subroutines)
- CPU restores interrupt status, pops stack into PC

Interrupt types in 8051

- **Reset** - a special kind of interrupt
  - Jump to 0000H, "reset handler"
    (or: handler is at 0H), but no RETI
- Timer 0 and 1 (jump to 000BH, 001BH)
- INT0, INT1 pins (jump to 0003H, 0013H)
- Serial (both Rx and Tx): jump to 0023H

8051 Interrupt vector table

Interrrupt	address	pin	Flag clear
Reset	0000H	9	Auto
INT0	0003H	P3.2 (12)	Auto
TF0	0013BH		Auto
TF1	0001BH	P3.3 (13)	Auto
UART	0023H		manual

- 0000H: a jump (2 or 3 bytes) to _main
  - (if you want to use interrupts)
- 0003H, 000BH, 0013H, ... (8 byte spaces)
  - Handler code (if fit in 8 bytes), or
    jump to handler routine if too long

Serial port: review

- SBUF register
  - write SBUF => transmit;
    read SBUF => receive
- Flags
  - TI == 1 when ready for next byte
  - RI == 1 when a byte has been received
- Flag could be polled or used as interrupt

8051: same vector for both Tx and Rx

- one ISR for both Tx and Rx
- User must check whether TI or RI is on
  - TI on => ready to send next char
  - RI on => read char from SBUF
- User is responsible for clearing the flag!
  - Both could be set, but might handle just either Rx or Tx at a time

Serial port interrupts -- revisited

- Same ISR shared between RI and TI
  - Both RI and TI could have triggered!
  - ISR checks which of RI, TI needs servicing
- Issues
  - Use of software interrupt with TI
  - Shared data structure

Code memory layout

```
        ORG 0H
        JMP   Main ;; on startup, jump to main()
        ORG  23H  ;; this is the location for the ISR for serial port
        JMP   Serial_ISR
        ;; initialize serial port
Main:       LCALL  InitUart
            SETB   ES ;; enable interrupt for serial port
            SETB   EA ;; enable all interrupts
LoopHere: JMP   LoopHere       ;; infinite loop, could do useful work
Serial_ISR:                ;; make sure it's RI
            JNB   TI, Check_RI
            CLR   TI
Check_RI: JNB  RI, Serial_Done
            MOV  A, SBUF          ;; read serial port
            CLR   RI                  ;; clear out receive flag
            ADD   A, #-48          ;; convert ASCII to binary
            LCALL  Display       ;; update the display
Serial_Done:   RETI                      ;; return from ISR
```

user code

interrupt service routine

Code for Init UART and Display

- (Code continues from previous page)
-
```
InitUart:  MOV  TMOD, #20H    ;; to send
           MOV  TH1, #-6      ;; 4800 baud
           MOV  SCON, #50H    ;; 8-bit 1 stop REN
           SETB TR1           ;; start timer 1
```

library code for UART

```
Display:   MOV    DPTR, #LEDdata
           MOVC A, @A+DPTR   ;; A = LEDdata[A]
           MOV    P1, A       ;; light up LED seg
           RET                ;; return from subroutine
LEDdata: DB  0C0H, 0F9H, 0A4H, 0B0H, 99H, 92H, 82H, 0F8H, 80H, 90H
           END
```

library code for LED display

- Assemble code and run with numeric characters in Tx (e.g., 246813570)
- Run and click [Tx Send] button
- PC spins at address 0x002c, which is `LoopHere: JMP LoopHere`
- Interrupt causes Serial_ISR to be invoked (interrupting the user loop!)

Run the interrupt version of serial-to-LED code

- As usual, set clock rate to 11.0529 MHz
- serial port at 4800 baud
- Type in digits into Tx field, click Tx Send
- Run code spin at address 0x002c
- Watch UART trigger interrupts (by invoking the ISR, which invokes Display.
- The LED should display the consumed digit

Other interrupts

- External interrupts (pins INT0, INT1)
  - triggered when those pins gets pulled low
  - Interrupt enable by EX0, EX1; flags INT0, INT1
- Timer interrupts (two timers)
  - triggered when counter rolls over to 0000H
  - interrupt enabled by ET0, ET1, flags are TF0, TF1
- Reset (power on or reset pin)
  - jumps to code address 0000H