80C51 Block Diagram
80C51 Memory

Program Memory
(Read Only)

Data Memory
(Read/Write)

EA = 0
External

EA = 1
Internal

PSEN

OFFFH

0000

00

FFH:

FFFFH:

External

Internal
8051 Memory

- The data width is 8 bits
- Registers are 8 bits
- Addresses are 8 bits
  - i.e. addresses for only 256 bytes!
  - PC is 16 bits (up to 64K program memory)
  - DPTR is 16 bits (for external data - up to 64K)
- C types
  - char - 8 bits  <-- use this if at all possible!
  - short - 16 bits
  - int - 16 bits
  - long - 32 bits
  - float - 32 bits
- C standard signed/unsigned
Accessing External Memory

Figure 4. Executing from External Program Memory

Figure 5. Accessing External Data Memory
Synopsys DW8051
Program Memory

- Program and Data memory are separate
- Can be internal and/or external
  - Small program memory can be implemented on chip
- Read-only
  - Instructions
  - Constant data

```c
char code table[5] = {'1', '2', '3', '4', '5'};
```

- Compiler uses instructions for moving “immediate” data
External Data Memory

- External Data - xdata
  - Resides off-chip
  - Accessed using the DPTR and MOVX instruction
  - We will use xdata if necessary for external memory
  - We will use the SMALL memory model
    - all data is on-chip
    - limited to only ~128 bytes of data!
Internal Data Memory

- Internal data memory contains all the processor state
  - Lower 128 bytes: registers, general data
  - Upper 128 bytes:
    - indirectly addressed: 128 bytes, used for the stack (small!)
    - directly addressed: 128 bytes for “special” functions

Figure 6. Internal Data Memory

Figure 7. Lower 128 Bytes of Internal RAM
Lower 128 bytes

- Register banks, bit addressable data, general data
  - you can address any register!
  - let the C compiler deal with details (for now)

---

**Figure 3.** 128 Bytes of RAM Direct and Indirect Addressable
Data Memory Specifiers

- "data" - first 128 bytes, directly addressed
  - the default
- "idata" - all 256 bytes, indirectly addressed (slower)
- "bdata" - bit-addressable memory
  - 16 bytes from addresses 0x20 to 0x2F
  - 128 bit variables max

```c
bit flag1, flag2;
flag1 = (a == b);
```

- can access as bytes or bits

```c
char bdata flags;
sbit flag0 = flags ^ 0; /* use sbit to "overlay" */
sbit flag7 = flags ^ 7; /* ^ specifies bit */
flags = 0; /* Clear all flags */
flag7 = 1; /* Set one flag */
```
Upper 128 bytes: SFR area

### Table 1. AT89LV55 SFR Map and Reset Values

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0F8H</td>
<td>B</td>
<td>00000000</td>
</tr>
<tr>
<td>0F0H</td>
<td>ACC</td>
<td>00000000</td>
</tr>
<tr>
<td>0E8H</td>
<td>PSW</td>
<td>00000000</td>
</tr>
<tr>
<td>0EOH</td>
<td>T2CON</td>
<td>00000000</td>
</tr>
<tr>
<td>0E8H</td>
<td>T2MOD</td>
<td>XXXXXXXX00</td>
</tr>
<tr>
<td>0E0H</td>
<td>RCAP2L</td>
<td>00000000</td>
</tr>
<tr>
<td>0D8H</td>
<td>RCAP2H</td>
<td>00000000</td>
</tr>
<tr>
<td>0D0H</td>
<td>TL2</td>
<td>00000000</td>
</tr>
<tr>
<td>0C8H</td>
<td>TH2</td>
<td>00000000</td>
</tr>
<tr>
<td>0C0H</td>
<td>IP</td>
<td>XX000000</td>
</tr>
<tr>
<td>0B8H</td>
<td>P3</td>
<td>11111111</td>
</tr>
<tr>
<td>0B0H</td>
<td>IE</td>
<td>0X000000</td>
</tr>
<tr>
<td>0A0H</td>
<td>P2</td>
<td>11111111</td>
</tr>
<tr>
<td>09H</td>
<td>SCON</td>
<td>00000000</td>
</tr>
<tr>
<td>09H</td>
<td>SBUF</td>
<td>XXXXXXXX</td>
</tr>
<tr>
<td>08H</td>
<td>TCON</td>
<td>00000000</td>
</tr>
<tr>
<td>08H</td>
<td>TMOD</td>
<td>00000000</td>
</tr>
<tr>
<td>08H</td>
<td>TL0</td>
<td>00000000</td>
</tr>
<tr>
<td>08H</td>
<td>TL1</td>
<td>00000000</td>
</tr>
<tr>
<td>08H</td>
<td>TH0</td>
<td>00000000</td>
</tr>
<tr>
<td>08H</td>
<td>TH1</td>
<td>00000000</td>
</tr>
<tr>
<td>07H</td>
<td>P0</td>
<td>11111111</td>
</tr>
<tr>
<td>07H</td>
<td>SP</td>
<td>00000111</td>
</tr>
<tr>
<td></td>
<td>DPL</td>
<td>00000000</td>
</tr>
<tr>
<td></td>
<td>DPH</td>
<td>00000000</td>
</tr>
<tr>
<td></td>
<td>PCON</td>
<td>0XX000000</td>
</tr>
</tbody>
</table>

CSE 477  8051 Overview  12
<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>DESCRIPTION</th>
<th>DIRECT ADDRESS</th>
<th>BIT ADDRESS, SYMBOL, OR ALTERNATIVE PORT FUNCTION</th>
<th>LSB</th>
<th>RESET VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACC*</td>
<td>Accumulator</td>
<td>E0H</td>
<td>E7 E6 E5 E4 E3 E2 E1 E0</td>
<td></td>
<td>00H</td>
</tr>
<tr>
<td>B*</td>
<td>B register</td>
<td>F0H</td>
<td>F7 F6 F5 F4 F3 F2 F1 F0</td>
<td></td>
<td>00H</td>
</tr>
<tr>
<td>DPTR</td>
<td>Data pointer (2 bytes)</td>
<td></td>
<td>AF AE AD AC AB AA A9 A8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DPH</td>
<td>Data pointer high</td>
<td>83H</td>
<td>EA ES ET1 ET0 EX0</td>
<td></td>
<td>0000000B</td>
</tr>
<tr>
<td>DPL</td>
<td>Data pointer low</td>
<td>82H</td>
<td>BF BE BD BC BB BA B9 B8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IE*</td>
<td>Interrupt enable</td>
<td>A8H</td>
<td>PS PT1 PX1 PT0 PX0</td>
<td></td>
<td>x000000B</td>
</tr>
<tr>
<td>IP*</td>
<td>Interrupt priority</td>
<td>B8H</td>
<td>87 86 85 84 83 82 81 80</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P0*</td>
<td>Port 0</td>
<td>80H</td>
<td>AD7 AD6 AD5 AD4 AD3 AD2 AD1 AD0</td>
<td></td>
<td>FFH</td>
</tr>
<tr>
<td>P1*</td>
<td>Port 1</td>
<td>90H</td>
<td>97 96 95 94 93 92 91 90</td>
<td></td>
<td>FFH</td>
</tr>
<tr>
<td>P2*</td>
<td>Port 2</td>
<td>A0H</td>
<td>A7 A6 A5 A4 A3 A2 A1 A0</td>
<td></td>
<td>FFH</td>
</tr>
<tr>
<td>P3*</td>
<td>Port 3</td>
<td>B0H</td>
<td>B7 B6 B5 B4 B3 B2 B1 B0</td>
<td></td>
<td>FFH</td>
</tr>
<tr>
<td>PCON1</td>
<td>Power control</td>
<td>B7H</td>
<td>SMOD T1 T0 TR T0 TR T0 TR T0</td>
<td></td>
<td>0000000B</td>
</tr>
<tr>
<td>SBUF</td>
<td>Serial data buffer</td>
<td>99H</td>
<td>9F 9E 9D 9C 9B 9A 99 98</td>
<td></td>
<td>xxxxxxxxB</td>
</tr>
<tr>
<td>SCON*</td>
<td>Serial controller</td>
<td>98H</td>
<td>SM0 SM1 SM2 REN TB8 RB8 T1 R1</td>
<td></td>
<td>00H</td>
</tr>
<tr>
<td>SP</td>
<td>Stack pointer</td>
<td>81H</td>
<td>8F 8E 8D 8C 8B 8A 89 88</td>
<td></td>
<td>07H</td>
</tr>
<tr>
<td>TCON*</td>
<td>Timer control</td>
<td>88H</td>
<td>TF1 TR1 TR0 TR0 TR0 TR0 IE1 IT1 IE0 IT0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TH0</td>
<td>Timer high 0</td>
<td>8CH</td>
<td>TF1 TR1 TR0 TR0 TR0 TR0 IE1 IT1 IE0 IT0</td>
<td></td>
<td>00H</td>
</tr>
<tr>
<td>TH1</td>
<td>Timer high 1</td>
<td>93H</td>
<td>9F 9E 9D 9C 9B 9A 99 98</td>
<td></td>
<td>00H</td>
</tr>
<tr>
<td>TL0</td>
<td>Timer low 0</td>
<td>8AH</td>
<td>8F 8E 8D 8C 8B 8A 89 88</td>
<td></td>
<td>00H</td>
</tr>
<tr>
<td>TL1</td>
<td>Timer low 1</td>
<td>8BH</td>
<td>8F 8E 8D 8C 8B 8A 89 88</td>
<td></td>
<td>00H</td>
</tr>
<tr>
<td>TMODE</td>
<td>Timer mode</td>
<td>89H</td>
<td>GATE C/T M1 M0 GATE C/T M1 M0</td>
<td></td>
<td>00H</td>
</tr>
</tbody>
</table>
Accessing SFRs

- The interesting SFRs are bit-addressable addresses 0x80, 0x88, 0x90, ..., 0xF8
- SFRs can be addressed by bit, char or int

```c
sbit EA = 0xAF; /* one of the interrupt enables
sfr Port0 = 0x80; /* Port 0 */
sfr16 Timer2 = 0xCC; /* Timer 2 */
sbit LED0 = Port1 ^ 2; /* Define a port bit */

EA = 1;        /* Enable interrupts */
Port0 = 0xff;  /* Set all bits in Port 0 to 1
if (Timer2 > 100) . . .
LED0 = 1;      /* Turn on one bit in Port 2 */
```
Ports

- Port 0 - external memory access
  - low address byte/data
- Port 2 - external memory access
  - high address byte
- Port 1 - general purpose I/O
  - pins 0, 1 for timer/counter 2
- Port 3 - Special features
  - 0 - RxD: serial input
  - 1 - TxD: serial output
  - 2 - INT0: external interrupt
  - 3 - INT1: external interrupt
  - 4 - TO: timer/counter 0 external input
  - 5 - T1: timer/counter 1 external input
  - 6 - WR: external data memory write strobe
  - 7 - RD: external data memory read strobe
Ports

a. Port 0 Bit

b. Port 1 Bit

c. Port 2 Bit

d. Port 3 Bit
Ports

- Port 0 - true bi-directional
- Port 1-3 - have internal pullups that will source current

**Output pins:**
- Just write 0/1 to the bit/byte

**Input pins:**
- Output latch must have a 1 (reset state)
  - Turns off the pulldown
  - Pullup must be pulled down by external driver
- Just read the bit/byte
Program Status Word

- Register set select
- Status bits

Figure 10. PSW (Program Status Word) Register in 80C51 Devices
Instruction Timing

- One “machine cycle” = 6 states (S1 - S6)
- One state = 2 clock cycles
  - One “machine cycle” = 12 clock cycles (DW8051: 4 clocks)
- Instructions take 1 - 4 cycles
  - e.g. 1 cycle instructions: ADD, MOV, SETB, NOP
  - e.g. 2 cycle instructions: JMP, JZ
  - 4 cycle instructions: MUL, DIV
Instruction Timing

a. 1-byte, 1-cycle Instruction, e.g., INC A

b. 2-byte, 1-cycle Instruction, e.g., ADD A,#data
Timers

- Base 8051 has 2 timers
  - we have 3 in the DW8051
- Timer mode
  - Increments every machine cycle (4 or 12 clock cycles)
- Counter mode
  - Increments when T0/T1 go from 1 - 0 (external signal)
- Access timer value directly
- Timer can cause an interrupt
- Timer 1 can be used to provide programmable baud rate for serial communications
- Timer/Counter operation
  - Mode control register (TMOD)
  - Control register (TCON)
Mode Control Register (TMOD)

- **Modes 0-3**
- **GATE** - allows external pin to enable timer (e.g. external pulse)
  - 0: INT pin not used
  - 1: counter enabled by INT pin (port 3.2, 3.3)
- **C/T** - indicates timer or counter mode
Timer/Counter Control Register (TCON)

- **TR** - enable timer/counter
- **TF** - overflow flag: can cause interrupt
- **IE/IT** - external interrupts and type control
  - not related to the timer/counter
Timer/Counter Mode 0

- clk
- Divide by 12
- Divide by 4
- T0M (or T1M)
- C/T
- TL0 (or TL1)
- Mode 0
- Mode 1
- TH0 (or TH1)
- TF0 (or TF1)
- INT
- To Serial Port (Timer 1 only)
- int0_n (or int1_n)
- TR0 (or TR1)
Timer/Counter Mode 2

- 8-bit counter, auto-reload on overflow
Timer/Counter Mode 3

- Applies to Timer/Counter 0
- Gives an extra timer
Interrupts

- Allow parallel tasking
  - Interrupt routine runs in “background”
- Allow fast, low-overhead interaction with environment
  - Don’t have to poll
  - Immediate reaction
- An automatic function call
  - Easy to program
- 8051 Interrupts
  - Serial port - wake up when data arrives/data has left
  - Timer 0 overflow
  - Timer 1 overflow
  - External interrupt 0
  - External interrupt 1
Interrupt Vector

- For each interrupt, which interrupt function to call
- In low program addresses

0x00 - Reset PC address
0: 0x03 - External interrupt 0
1: 0x0B - Timer 0
2: 0x13 - External interrupt 1
3: 0x1B - Timer 1
4: 0x23 - Serial line interrupt

- Hardware generates an LCALL to address in interrupt vector
- Pushes PC (but nothing else) onto the stack
- RETI instruction to return from interrupt
Writing Interrupts in C

- The C compiler takes care of everything
  - Pushing/popping the right registers (PSW, ACC, etc.)
  - Generating the RTI instruction
  - No arguments/no return values

```c
unsigned int count;
unsigned char second;

void timer0 (void) interrupt 1 using 2 {
    if (++count == 4000) {
        second++;
        count = 0;
    }
}
```

- Timer mode 2
- Reload value = 6
Timer Interrupts

- Wakeup after N clock cycles, i.e. at a specified time
- Wakeup every N clock cycles (auto reload)
  - Allows simple task scheduling
  - Clients queue function calls for time i
  - Interrupt routine calls functions at the right time
- Wakeup after N events have occurred on an input
Design Problem 1 - frequency counter

- Measure the frequency of an external signal
- Display as a number using the 7-segment display
  - e.g. number represents exponent of 2 or 10
Example Timer Setup

What does this setup do?

```
TMOD = 0x62;  // 01100010;
TCON = 0x50;  // 01010000;
TH1 = 246;
TH0 = 6;
IE = 0x8A;  // 10001010;
```
Using the timers

void counterInterrupt ( void ) interrupt 3 using 1 {
    timeLow = TL0;
    TL0 = 0;
    timeHigh = count;
    count = 0;
    if (timeHigh == 0 && timeLow < 10) *ledaddress = 0x6f;
    else if (timeHigh == 0 && timeLow < 100) *ledaddress = 0x6b;
    else if (timeHigh < 4) *ledaddress = 0x02;
    else if (timeHigh < 40) *ledaddress = 0x04;
    else if (timeHigh < 400) *ledaddress = 0x08;
    else if (timeHigh < 4000) *ledaddress = 0x10;
    else if (timeHigh < 40000) *ledaddress = 0x20;
    else *ledaddress = 0xf0;  // default
}

void timerInterrupt ( void ) interrupt 1 using 1 {
    count++;
}
Design Problem 2 - Measure the pulse width

Problem: send several bits of data with one wire
- Serial data
  - precise, but complicated protocol
- Pulse width
  - precise enough for many sensors
  - simple measurement
Design Problem 3 - Accelerometer Interface

- **Accelerometer**
  - Two signals, one for each dimension
  - Acceleration coded as the duty cycle
    - pulse-width/cycle-length
    - cycle time = 1ms - 10ms (controlled by resistor)
      - 1ms gives faster sampling
      - 10ms gives more accurate data
Controlling Interrupts: Enables and Priority

**Figure 17. Interrupt Enable (IE) Register**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Position</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>EA</td>
<td>IE.7</td>
<td>Disables all interrupts. If EA = 0, no interrupt will be acknowledged. If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.</td>
</tr>
<tr>
<td>IE.6</td>
<td>Reserved.</td>
<td></td>
</tr>
<tr>
<td>IE.5</td>
<td>Reserved.</td>
<td></td>
</tr>
<tr>
<td>ES</td>
<td>IE.4</td>
<td>Enables or disables the Serial Port interrupt. If ES = 0, the Serial Port interrupt is disabled.</td>
</tr>
<tr>
<td>ET1</td>
<td>IE.3</td>
<td>Enables or disables the Timer 1 Overflow interrupt. If ET1 = 0, the Timer 1 interrupt is disabled.</td>
</tr>
<tr>
<td>EX1</td>
<td>IE.2</td>
<td>Enables or disables External Interrupt 1. If EX1 = 0, External Interrupt 1 is disabled.</td>
</tr>
<tr>
<td>ET0</td>
<td>IE.1</td>
<td>Enables or disables the Timer 0 Overflow interrupt. If ET0 = 0, the Timer 0 interrupt is disabled.</td>
</tr>
<tr>
<td>EX0</td>
<td>IE.0</td>
<td>Enables or disables External Interrupt 0. If EX0 = 0, External Interrupt 0 is disabled.</td>
</tr>
</tbody>
</table>

**Figure 18. Interrupt Priority (IP) Register**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Position</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP.7</td>
<td>Reserved.</td>
<td></td>
</tr>
<tr>
<td>IP.6</td>
<td>Reserved.</td>
<td></td>
</tr>
<tr>
<td>IP.5</td>
<td>Reserved.</td>
<td></td>
</tr>
<tr>
<td>PS</td>
<td>IP.4</td>
<td>Defines the Serial Port interrupt priority level. PS = 1 programs it to the higher priority level.</td>
</tr>
<tr>
<td>PT1</td>
<td>IP.3</td>
<td>Defines the Timer 1 interrupt priority level. PT1 = 1 programs it to the higher priority level.</td>
</tr>
<tr>
<td>FX1</td>
<td>IP.2</td>
<td>Defines the External Interrupt 1 priority level. PX1 = 1 programs it to the higher priority level.</td>
</tr>
<tr>
<td>PT0</td>
<td>IP.1</td>
<td>Enables or disables the Timer 0 Interrupt priority level. PT0 = 1 programs it to the higher priority level.</td>
</tr>
<tr>
<td>PX0</td>
<td>IP.0</td>
<td>Defines the External Interrupt 0 priority level. PX0 = 1 programs it to the higher priority level.</td>
</tr>
</tbody>
</table>
Interrupt Controls

Figure 19. Interrupt Control System
Interrupt Priorities

- Two levels of priority
  - Set an interrupt priority using the interrupt priority register
  - A high-priority interrupt can interrupt an low-priority interrupt routine
  - In no other case is an interrupt allowed
  - An interrupt routine can always disable interrupts explicitly
    - But you don’t want to do this

- Priority chain within priority levels
  - Choose a winner if two interrupts happen simultaneously
  - Order shown on previous page
Re-entrant Functions

- A function can be called simultaneously by different processes
- Recursive functions must be re-entrant
- Functions called by interrupt code and non-interrupt code must be re-entrant
- Keil C functions by default are *not* re-entrant
  - Does not use the stack for everything
  - Use the reentrant specifier to make a function re-entrant

```c
int calc (char i, int b) reentrant {
    int x;
    x = table[i];
    return (x * b);
}
```
External Interrupts

- Can interrupt using the INTO or INT1 pins (port 3: pin 2,3)
  - Interrupt on level or falling edge of signal (TCON specifies which)
  - Pin is sampled once every 12 clock cycles
    - for interrupt on edge, signal must be high 12 cycles, low 12 cycles
  - Response time takes at least 3 instructions cycles
    - 1 to sample
    - 2 for call to interrupt routine
    - more if a long instruction is in progress (up to 6 more)