CpE 213
Digital Systems Design
8051 Instruction Set

Lecture 15

Wednesday 10/12/2005

UMR
UNIVERSITY OF MISSOURI-ROLLA
The Name. The Degree. The Difference.

Overview

- 8051 Arithmetic and Logic Instructions
- Break
- 8051 Branch Instructions
- Note: We will be skipping some example slides today.

  Please read these lecture notes in their entirety.

Arithmetic Instructions

- Arithmetic is an important part of a micro-controller's duties. But we should not expect high-speed, high-accuracy floating point calculations from the 8051. Extensive computation is not the purpose of a microcontroller.
- The 8051 provides the normal add and subtract instructions, with or without carry; multiplication and division; and incrementing and decrementing values.
- The instructions that handle these operations are:
  - ADD, ADDC, SUBB, INC, DEC, MUL, DIV and DA

Arithmetic Operations

- `ADD destination,source`
  - Add source to destination without carry (C)
- `ADDC destination,source`
  - Add source to destination with carry (C)
- `SUBB destination,source`
  - Subtract, with carry, source from destination
- `INC destination`
  - Increment destination by 1
- `DEC destination`
  - Decrement destination by 1
- `MUL AB`
  - Multiply the contents of registers A and B
- `DIV AB`
  - Divide the contents of register A by the contents of register B
- `DA A`
  - Decimal adjust the A register

ADD, ADDC and SUBB

- For ADD, ADDC and SUBB four addressing modes are possible:
  - `ADD  A,7Fh  ;Direct addressing`
  - `ADD  A,@R0  ;Indirect addressing`
  - `ADD  A,R7  ;Register addressing`
  - `ADD  A,#35h  ;Immediate addressing`
- The 8051 has four arithmetic flags: carry (C), auxiliary carry (AC), overflow (OV), and parity (P). They are stored in the PSW (Program Status Word) register.
- In unsigned number arithmetic, we must monitor the status of the CY (carry flag), whereas in signed number arithmetic the OV (Overflow) flag must be monitored.

The SUBB Instruction

- **`SUBB A,source ; A = A - source - CY`**
- The 8051 uses the 2's complement method to perform subtraction.
- The entire process is performed by the internal hardware of the 8051 CPU for every SUBB instruction.
- If after the execution of SUBB the CY = 0, the result is positive, if CY = 1, the result is negative and the destination has the 2's complement of the result.
- Remember to set the carry flag to zero if it is not to be included as part of the subtraction operation.

The INC and DEC Instructions

- Any internal memory location can be incremented or decremented using direct addressing without going through the accumulator.
  - **`INC  7Fh`** increases the content of location 7F by 1.
- No math flags are affected by these instructions.
- The INC instruction can also operate on the 16 bit DPTR register.
- There is no DEC DPTR to match the INC DPTR.

How to Perform the DEC DPTR Instruction?

- Decrementing the DPTR requires a sequence of instructions:

```
        DEC   DPL             ;Decrement low byte
        MOV   R7,DPL          ;Copy to R7
        CJNE  R7,#0FFh,Skip   ;Check if underflow
        DEC   DPH             ;Decrement high byte
Skip:   ----------------
```

Unsigned 8-bit Multiplication & Division

- The 8051 features an auxiliary accumulator, referred to as register B, located at SFR address 0F0h for multiplication and division of unsigned 8-bit numbers.
- The instruction syntax is:

```
MUL  AB   ;Multiply A and B
DIV  AB   ;Divide A by B.
          ;A = A/B  B = remainder
```

- Note there is no comma between A and B in the MUL operands.

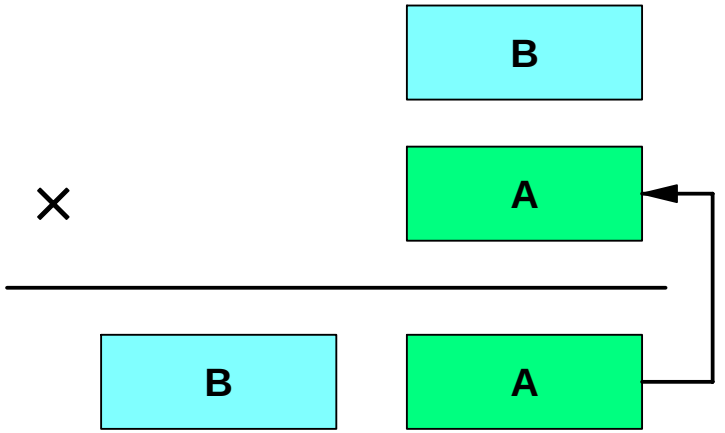
Unsigned Multiplication

- Multiplicand: Register A

  Multiplier: Register B

  B, A = Product

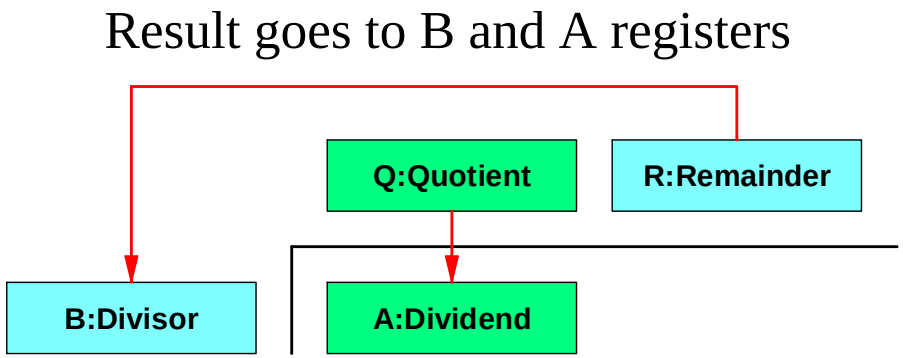
  (High byte in B: Low byte in A)
- The product after a multiplication is always a double-width product
- If the product is greater than 255, the overflow (OV) flag is set; otherwise, it is cleared. The carry flag is always cleared to 0.
- The original contents of A and B are lost.



Result goes to A and B registers

8-Bit Unsigned Division

- Dividend: A
  Divisor: B
  Quotient: A
  Remainder: B
- Both the carry and overflow flags are cleared.
- If B contained 00h, the overflow flag is set to indicate a division by zero.



The BCD System

- The BCD (Binary Coded Decimal) is essentially a cross between the binary and decimal systems, where each decimal digit is replaced by its 4-bit binary equivalent.
- Example: The number 729 in BCD is

7	2	9
0111	0010	1001

- In the 8051, the 8-bit value in the accumulator is adjusted to form two BCD digits of four bits each. This is known as packed BCD format.

Decimal Adjust (DA)

- To obtain a BCD number as the result of adding two BCD numbers, the DA (decimal adjust) instruction is used:

```
DA    A
```

- The decimal adjust instruction is associated with the use of the ADD and ADDC instructions.
- The DA instruction works only on register A.
- **DA A** is executed immediately after the addition.
- Example: BCD addition of 23 and 47 = 70

```
MOV   R0,#23h  ;R0 = 00100011B
MOV   A,#47h   ; A = 01000111B
ADD   A,R0     ; A = 01101010B
               ;(simple binary addition)
DA      A      ; A = 01110000 (70 in BCD)
```

Logical Instructions

- The logical instructions provide the facility to perform bitwise AND, OR, XOR, NOT, rotate and nibble (4 bits) swapping in the accumulator.
- The logic combination comprises all bits of both operands.
- Syntax:

```
ANL destination,source  ; dest. = dest. AND src.
ORL  destination,source ; dest. = dest. OR src.
XRL  destination,source ; dest. = dest. XOR src.
```

- The operations can use all four addressing modes for the source of a data byte (register, direct, indirect and immediate).
- The A register or a direct address in internal RAM is the destination of the logical operation result.

Logical Operation Examples

- These instructions perform Boolean operations on bytes of data, bit by bit.
- No flags are affected unless the direct address is the PSW.
- Only internal RAM or SFRs may be logically manipulated.
- The ANL instruction may be used to reset (clear) certain bits in a byte, without changing the others.
- The ORL instruction is used to set certain bits.

Logical Operation Examples

- The setting or resetting of certain bits in a byte is called masking.
- Example: To check if one of bits 0, 1, 2 or 7 in the accumulator is set:

```
        ANL   A,#10000111 ;mask bits 7,2,1,0
        JNZ   Set         ;A will not be 0 if
                          ;one of these bits
                          ;is set
        .
        .
Set:    -------
```

- The XRL instruction is very useful for toggling selected bits.
  - 0 XOR 1 = 1 1 XOR 1 = 0

General Rules For Logical Operations

- ANDing a bit with a 0 CLEARS it.
- ANDing a bit with a 1 leaves it unchanged.
- ORing a bit with a 0 leaves it unchanged.
- ORing a bit with a 1 SETS it.
- XORing a bit with a 0 leaves it unchanged.
- XORing a bit with a 1 toggles it (changes to a 1 if 0, to a 0 if 1).

Logical Operations Without the Accumulator

- Note that the logical operations can be performed on any byte in the internal data memory space without going through the accumulator.
- Example: **`XRL P1,#0FFh`**
  - This instruction inverts (toggles) the port 1 pins through a read-modify-write operation.
- If the direct address destination is one of the port SFRs, the data latched in the SFR, not the pin data, is used.

CLR and CPL Instructions

- The clear and complement instructions are register-specific, operating ONLY on the accumulator.
- The clear instruction **`CLR A`**
  - clears all bits of the accumulator.
  - may be viewed as moving the constant 0 into the accumulator.
- The complement instruction **`CPL A`**
  - complements each bit of the accumulator.
  - Example:

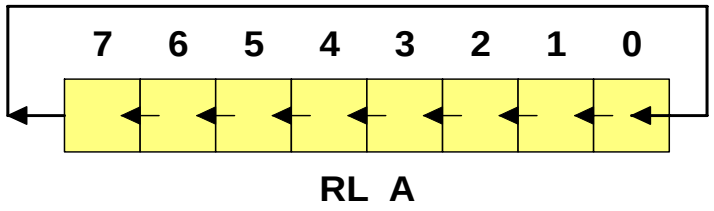
```
MOV   A,#0AAh ;Place AAh in A
CPL     A     ;Complement of AAh
              ;is 55h
CLR    A      ;Clear the Acc.
```

The Rotate Instructions

- The rotate instructions are register-specific, operating only on the accumulator.
- In a rotate instruction, the bit that is shifted out is used as the new bit shifted in (circular rotation).
- There are four rotate instructions available in the 8051:

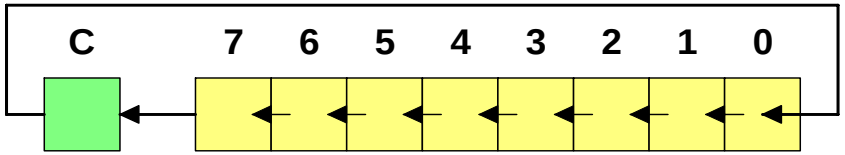
```
RL   A     ;Rotate accumulator left
RLC  A     ;Rotate accumulator left
           ;through carry flag
RR   A     ;Rotate accumulator right
RRC  A     ;rotate accumulator right
           ;through carry
```

Rotate Operations



RL A

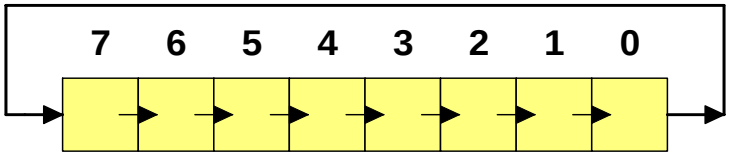
Before: 10011100
After: 00111001



Carry Flag

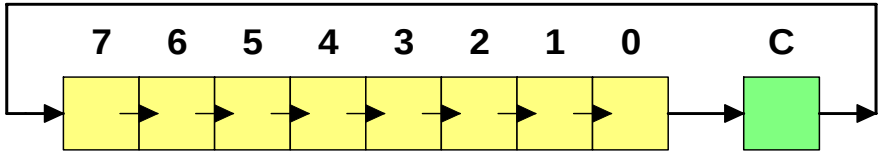
RLC A

Before: 10011100 CY = 0
After: 00111000 CY = 1



RR A

Before: 10011100
After: 01001110



RRC A

Carry Flag

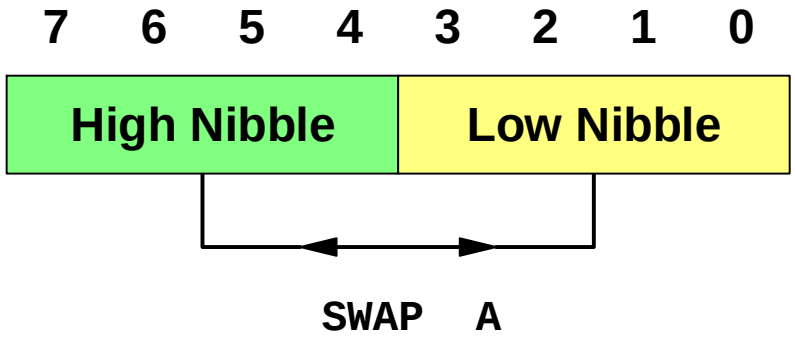
Before: 10011100 CY = 1
After: 11001110 CY = 0

Rotation Operations (Cont.)

- If the C flag is involved, then a total of 9 bits are involved in the rotation, otherwise only the 8 bits of A are involved.
- The carry flag state has to be known before being used in a rotate operation.
- Including the C flag enables programmer to construct rotate operations involving any number of bytes.
- Rotate instructions are useful as in multiplying or dividing by powers of 2.
- Rotating the accumulator left one digit is equivalent to multiplying it by 2, provided that bit 7 is originally 0.
- Similarly, rotating right by one digit is equivalent to division by 2 (catch?).
- Rotate instructions can be used in parallel to serial conversion of a byte (how?).

SWAP Instruction

- The **`SWAP A`** instruction is used to exchange the high and low nibbles of the accumulator.
- This is a useful operation in BCD manipulation.



SWAP Instruction Example

- The following routine quickly converts a decimal number less than 100 into BCD.

```
MOV     B,#10
DIV     AB
SWAP    A
ADD     A,B
```

Boolean Variable Manipulation

- Bit operations are one of the most powerful features of the 8051 microcontroller. Bits may be set or cleared in a single instruction.
- The Boolean logical opcodes operate on any addressable RAM or SFR bit.
- The Boolean variables manipulation instructions include:

```
CLR   bit-operand       ;clear bit
SETB  bit-operand       ;set bit
CPL   bit-operand       ;invert bit
ANL   C,bit-operand     ;C <- (C AND bit)
ORL   C,bit-operand     ;C <- (C OR bit)
MOV   bit-operand,C     ;bit <- C
MOV   C,bit-operand     ;C <- bit
```

Boolean Variable Manipulation

- The bit manipulation is often used to set or reset single bits, for instance, of an output port, without affecting the state of the others.
- No flags, other than the C flag, are affected, unless the flag is an addressed bit.
- For the ANL (AND) and ORL (OR) bit-oriented operations, the source bit may use its complement form
  Example: **`ANL C,/P2.5`**

  Note: The state of P2.5 is not altered.
- The carry flag (C) is the destination for most of the opcodes because the flag can be tested; it acts as a bit 'accumulator'.

Boolean XOR Operation

- Note that there is no Boolean bit-oriented XRL opcode.
- However, it can be implemented by the following sequence of instructions:

  To XOR BIT1 and BIT2:

```
        MOV     C,BIT1
        JNB     BIT2,SKIP
        CPL     C
SKIP:   . . . . . . . . . .
```

Program Branch Instructions

- The flow of a program proceeds sequentially, from instruction to instruction, unless a control transfer instruction is executed.
- The branch instructions allow the microcontroller to go to a different memory location, either unconditionally or under certain test conditions and the microcontroller continues executing machine codes from that new location.
- The branch instructions are classified into three categories:
  - Unconditional/Conditional Jump instructions.
  - Call and Return instructions.
  - Bit jump.

Unconditional JUMP Instructions

- For reasons of efficiency, the 8051 has 3 types of jumps. They are coded as:
  - LJMP Label-reference
  - SJMP Label-reference
  - AJMP Label-reference
- Label-reference refers to another instruction in the program for which a has been defined.
- Example:

```
        LJMP  A10
        ADD   A,R1
A10:    MOV   A,#0Ah
```

- After the machine executes the LJMP statement, the next statement to be executed will be the MOV statement. In this example, the ADD statement will never be executed.

LJMP, SJMP and AJMP

- LJMP
  - Long jump, 3 byte instruction, able to jump full 64K.
  - Not efficient if you want to jump a short distance.
- SJMP
  - Short jump, 2 byte opcode, only ≈ ± 128 bytes.
  - Short jumps are relative to the current PC .
  - The PC is incremented twice before the jump is carried out.
    - Example: `SJMP $ ;Branch-to-self`

LJMP, SJMP and AJMP

- If you try to assemble a program with an SJMP that attempts to jump too far, then the assembler will complain and generate an error.
- AJMP
  - Absolute jump, 2 byte opcode
  - Can jump to instruction within same 2K block.
  - Have covered it in Lecture 13 (absolute addressing mode).

NOP

- The NOP (No OPeration) instruction has no addressing mode. It does nothing; goes to the next instruction.
- NOP is often used to kill time in a software timing loop or to leave room in a program for later additions.
- With a crystal of 12 MHz one NOP takes 1 μS. Why?
- No flags are affected by this instruction.

Conditional Program Control Instruction

- These instructions permit alteration of the program flow depending upon the state of certain processor flags or bits.
- All conditional control flow instructions use relative addressing.
- Conditional branching can be thought of as the “IF . . . THEN” structure of the 8051 assembly language.
- The jump distance is limited to -128 to +127 bytes.

Conditional Jumps

- Syntax:

Format	Operation
CJNE destination,source, address	Compare destination and source, jump to address if not equal
DJNZ destination,address	Decrement destination by one, jump to address if the result is not zero
JZ relative address	Jump to relative address if A = 00h
JNZ relative address	Jump to relative address if A > 00h

Conditional Jumps

- The CJNE compare instruction compares two operands, and jumps if they are not equal.
- In addition, it changes the CY flag to indicate if the destination operand is larger or smaller than the source.

Compare	Carry Flag
destination > source	CY = 0
destination < source	CY = 1

- Important: the operands (source and destination) remain unchanged.

CJNE Summary

	#data	direct	Src
A	✓	✓	
@Ri	✓	✗	
Rn	✓	✗	
Dst			

Conditional Jumps

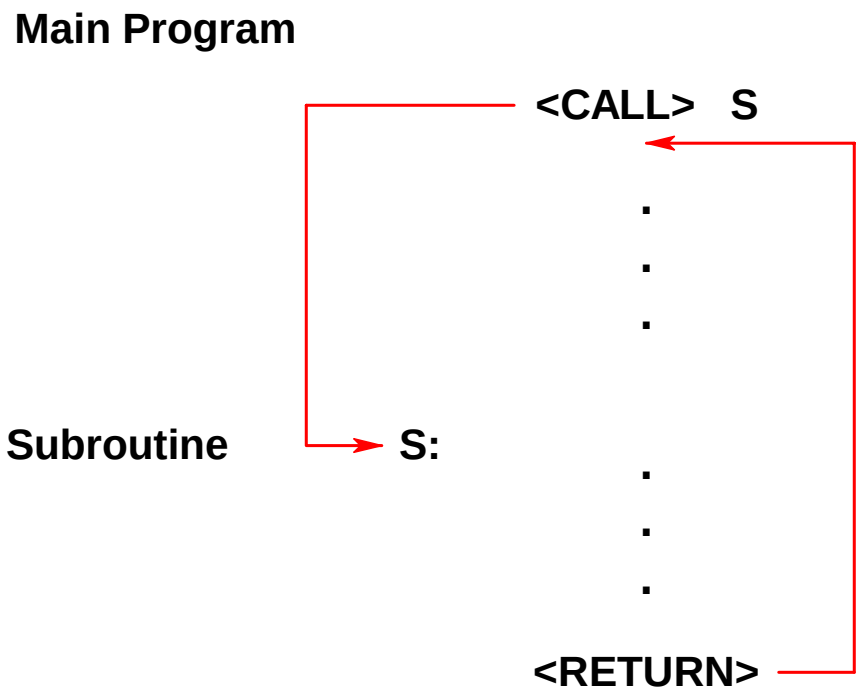
- The JZ and JNZ instructions test the accumulator.
- Note that there is no zero bit in the PSW (unlike WIMP51).
- Both the DJNZ (decrement and jump if not zero) and the CJNE (compare and jump if not equal) instructions are used for loop control.
- Example: Add value 3 to the ACC ten times.

```
          MOV    A,#0         ;Clear ACC
          MOV    R2,#10       ;Load counter R2 = 1
Again:    ADD    A,#03        ;Add 3 to ACC
          DJNZ   R2,Again     ;Repeat until R2 = 0
          MOV    R5,A         ;Save result in R5
```

Subroutine Calls and Returns

- A subroutine is a sequence of instructions stored in memory at a specified address.
- The subroutine can be called from various places in the program.
- When a subroutine is called, the address of the next instruction is saved and program control passes to the called subroutine, which then executes its instructions.
- The subroutine terminates with a return instruction directing program control back to the instruction following the one that called the subroutine (the saved address).

Subroutine Processing



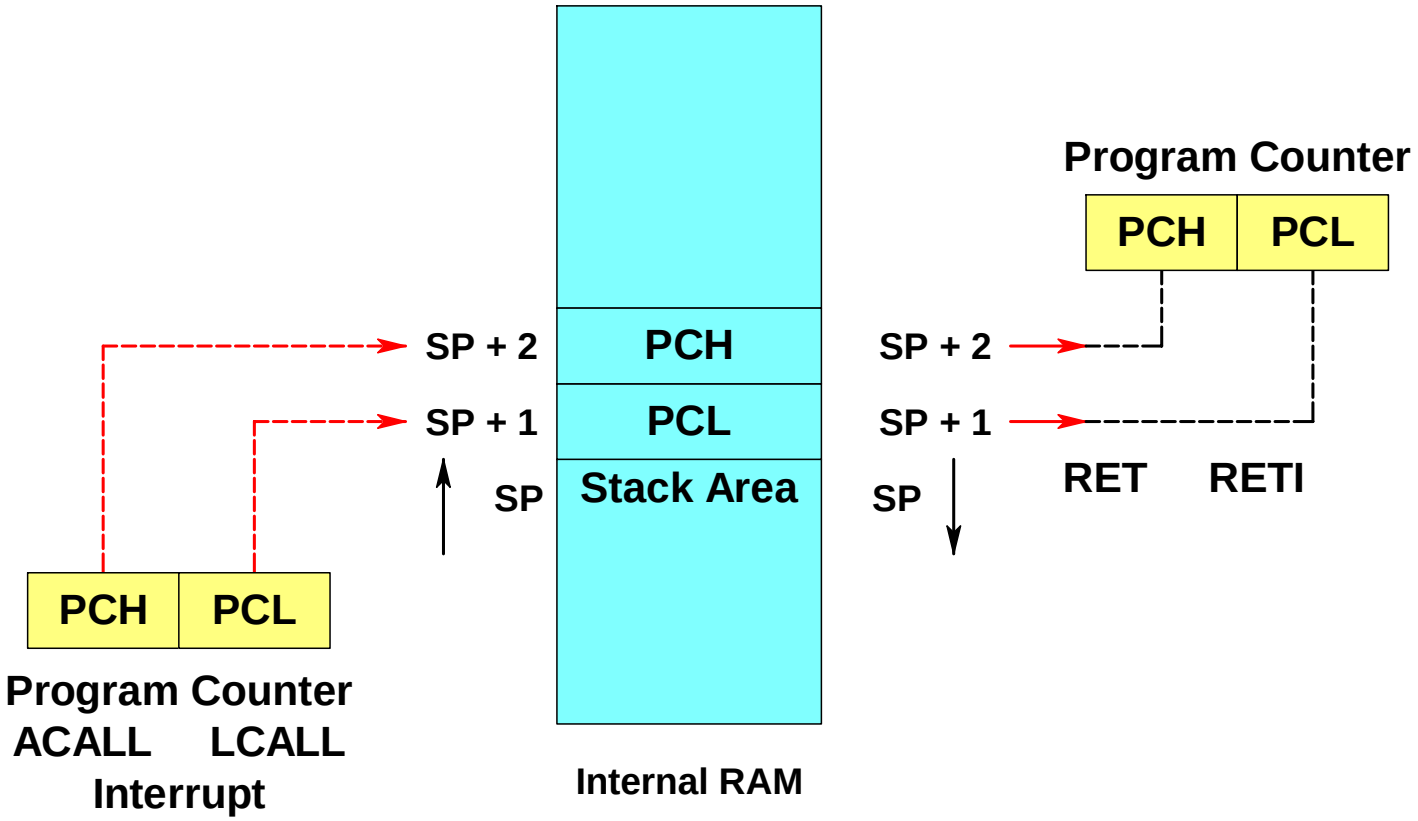
Subroutine Call and Return

- In 8051 assembly, subroutines can be called in two ways:
  - `ACALL addr11`
    - The Absolute Call instruction is used to access subroutines within a 2K region relative to the PC.
  - `LCALL addr16`
    - The Long Call instruction allosw access to a subroutine anywhere in the 64K program memory address space.
- Subroutines must end with a RET instruction, which returns the program to the instruction following the CALL.

CALL and RET instructions

- When calling a subroutine, the PC register (which holds the address of the instruction after the CALL) is pushed onto the stack, and the stack pointer (SP) is incremented by 2.
- The PC is then loaded with the new address and control is transferred to the subroutine.
- The RET instruction pops the high/low bytes of the saved PC from the stack, then decrements the stack pointer by 2.
- Use the LCALL instruction if your subroutines are normally placed at the end of your program.

Storing and Retrieving the Return Address



The RETI Instruction

- When an interrupt (a hardware-generated call) occurs, the microcontroller executes an LCALL instruction to the appropriate service routine for that interrupt.
- The RETI instruction is used at the end of an interrupt service routine.
- The top two bytes of the stack are popped into the PC and program execution continues at this new address. After popping the stack pointer is decremented by 2.
- While the RET instruction is used at the end of a subroutine associated with the ACALL and LCALL instruction, RETI must be used for interrupt service routines.
- Using RETI at the end of a software-invoked subroutine may enable the interrupt logic erroneously.

Bit Jump Instructions

- Bit jumps all operate according to the status of the carry flag in the PSW or the status of any bit-addressable location.
- All bit jumps are relative to the program counter.
- Jump instructions that test for bit conditions are:

Format	Operation
JC addr	Jump to addr if carry flag is set
JNC addr	Jump to addr if carry flag is reset to 0
JB bit,addr	Jump to addr if bit at bit address is 1
JNB bit,addr	Jump to addr if bit at bit address is 0
JBC bit,addr	Jump to addr if bit at bit address is 1 while at the same time the bit is cleared to 0

Bit Jump Instructions

- If the addressable bit is a flag bit and JBC is used, the flag bit will be cleared.
- The JNB and JB instructions are widely used to monitor a bit and make a decision depending on whether it is 0 or 1.

Group Exercise

- Monitor bit P2.3. If it is high, send a high to low pulse to port bit P1.5.

Instruction Timing

- Most instructions are one machine cycle (12 clock periods).
- You can do a fair timing estimate by assuming one machine cycle per instruction.
- A few instructions are two machine cycles (24 clock cycles).
- Question: Are there any three cycle instructions?

For Friday

- Finish assignment 6.
- Review today's lecture notes.
- Review Chapters 3, 6, and 7.
- Read Chapter 4, Section 8.3, and Appendix C.