8051 Programming

Class 5

EEE4380 Fall 2001

Parivallal Kannan

Center for Integrated Circuits and Systems

University of Texas at Dallas

Topics

- 8051 Addressing Modes
- Jump, Loop and Call instructions
- Subroutines
- Simple delay loops

8051 Addressing Modes

- CPU can access data in various ways
- Specify data directly in the instruction
- Use different Addressing modes for data in code and data memory
- Five modes
  - Immediate
  - Register
  - Direct
  - Register Indirect
  - Indexed

Immediate Addressing Mode

- Operand (data) directly specified in the instruction (opcode)
- Operand is a constant, known during assemble time
- Immediate data has to be preceded by “#” sign
- Eg.

```
mov A, #25H
mov DPTR, #1FFFH
temp EQU 40
mov R1, #temp   ;R1 ← 28H (40 decimal)
```

Register Addressing Mode

- Involves the use of registers to hold data
- Put the operand in a register and manipulate it by referring to the register in the instruction

```
mov A, R0
mov R2, A
ADD A, R1
```

- Source and destination registers must match in size
- There may not be instructions for moving any register to any
  - mov R4, R7   ; invalid
  - Check with the instruction list before using
  - Assembly will fail in these cases

Direct Addressing Mode

- For data stored in RAM and Registers
  - All memory locations accessible by addresses
  - Same with all registers, ports, peripherals (SFRs) in 8051
- Use the address of the operand directly in the instruction
  - mov A, 40H ; copy data in mem[40H] to A
- Register addressing as Direct addressing
  - mov A, 4H ; 4H is the address for R4
  - mov A, R4 ; same as above. Both do the same

    ; but may have different op codes
- All registers and SFRs have addresses
- Stack in 8051 uses only direct addressing modes

Register Indirect Addressing Mode

- A register is used as a pointer
  - Register stores the address of the data
- Only R0, R1 and DPTR can be used for this purpose in 8051
- R0 and R1 can be used for internal memory (256 bytes incl. SFRs) or from 00H to FFH of external memory
  - mov A, @R0 ;copy internal_mem[R0] to A
  - mov @R1, A ;copy A to internal_mem[R1]
  - movx A, @R0 ; copy external_mem[R0] to A
- DPTR can be used for external data memory
  - movx A, @DPTR ;copy ext_data_mem[DPTR] to A
  - movx @DPTR, A ;vice versa

Indexed Addressing Mode

- Use a register for storing the pointer and another register for an offset
- Effective address is the sum base+offset
  - Move code byte relative to DPTR to A. Effective address is DPTR + A
    - movc A, @A+DPTR
  - Move code byte relative to PC to A. Effective address is PC + A
    - movc A, @A+PC
- Widely used for implementing look-up tables, data arrays, character generators etc in code memory (ROM)

Indexed Addressing Mode - Example

- Program to read a value x from P1 and send x^2 to P2

```
            ORG 0
            mov DPTR, #LUT          ; 300H is the LUT address
            mov A, #0FFH
            mov P1, A               ; program the port P1 to input data
back:       mov A, P1               ; read x
            movc A, @A+DPTR         ; get x^2 from LUT
            mov P2, A               ; output x^2 to P2
            sjmp back               ; for (1) loop

            ORG 300H
LUT:        DB 0, 1, 4, 9, 16, 25, 36, 49, 64, 81
```

Program Control Instructions

- Unconditional Branch
  - ajmp addr11 ; absolute jump
  - ljmp addr16 ; long jump
  - sjmp rel ; short jump to relative address
  - jmp @A+DPTR ; jump indirect
- Conditional branch
  - jz, jnz rel ; short conditional jump to rel. addr
  - djnz rel ; decrement and jump if not zero
  - cjne rel ; compare and jump if not equal
- Subroutine Call
  - acall addr11 ; absolute subroutine call
  - lcall addr16 ; long subroutine call
  - ret ; return from subroutine call
  - reti ; return from ISV

Loop using djnz

- Add 3 to A ten times

```
         mov     A, #0         ; clear A
         mov     R2, #10       ; R2 ← 10, can also say 0AH
AGAIN:   add     A, #03        ; add 3 to A
         djnz    R2, AGAIN     ; repeat until R2==0
         mov     R5, A         ; save the result in R5
```

- Loop within loop using djnz

```
        mov     R3, #100
loop1:  mov     R2, #10       ; trying for 1000 loop iterations
loop2:  nop                   ; no operation
        djnz    R2, loop2     ; repeat loop2 until R2==0
        djnz    R3, loop1     ; repeat loop1 until R3==0
```

Conditional Jumps

- jz, jnz : Conditional on A==0
  - Checks to see if A is zero
  - jz jumps if A is zero and jnz jumps is A not zero
  - No arithmetic op need be performed (as opposed to 8086)
- djnz : dec and jump if A not equal to zero
  - djnz Rn, rel
  - djnz direct, rel
- jnc : Conditional on carry CY flag
  - jc rel
  - jnc rel
- Cjne : compare and jump if not equal
  - cjne A, direct, rel
  - cjne ARn, #data, rel
  - cjne @Rn, #data, rel

Unconditional Jumps

- LJMP addr16
  - Long jump. Jump to a 2byte target address
  - 3 byte instruction
- SJMP rel
  - Jump to a relative address from PC+127 to PC-128
  - Jump to PC + 127 (00H – 7FH)
  - Jump to PC – 128 (80H – FFH)
- Target address calculation
  - PC of next instruction + rel address
  - For jump backwards, drop the carry
    - PC = 15H, SJMP 0FEH
    - Address is 15H + FEH = 13H
    - Basically jump to next instruction minus two (current instruction)

Call Instructions

- LCALL addr16
  - Long call. 3 byte instruction.
  - Call any subroutine in entire 64k code space
  - PC is stored on the stack
- ACALL addr11
  - 2 byte instruction
  - Call any subroutine within 2k of code space
  - Other than this, same behavior as LCALL
  - Saves code ROM for devices with less than 64K ROM
- RET
  - Return from a subroutine call
  - Pops PC from stack

Machine Cycle

- Number of clock cycles used to perform one instruction
- Varies with instruction
- Usually the lowest is quoted as the machine cycle
- For 8051, 12 clock cycles are minimum needed per instruction
- Time per machine cycle
  - T_{mc} = Clocks per machine cycle / Clock frequency
  - For 8051 clocked at 11.0592MHz,
    - T_{mc} = 12 / 11.0592M = 1.085 micro seconds
- Time spent executing an instruction
  - T_{instr} = machine cycles for the instruction * T_{mc}
  - For the nop instruction, machine cycles = 1. So
    - T_{instr} = 1 * 1.085 = 1.085 micro seconds

Simple delay loops

- Find the time delay for the subroutine

```
DELAY:      mov R3, #200       ; 1 machine cycle
HERE:       djnz R3, HERE      ; 2 machine cycles
            RET                ; 1 machine cycle
```

- Calculation
  - Total machine cycles = 200*2 + 1 + 1 = 402
  - Time = 402 * 1.085us (assuming 11.0592 MHz clk)

    = 436.17us
- Similarly any delay can be obtained by loop within loop technique
- For much longer delays, use timers