CpE 213
Digital Systems Design
8051 Addressing Modes
8051 Instruction Set

Lecture 14

Wednesday 9/28/2005

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

Overview

- Programmer's model of the 8051
- Addressing modes for the 8051
- 8051 instruction set
- Note: We will be skipping some example slides today.

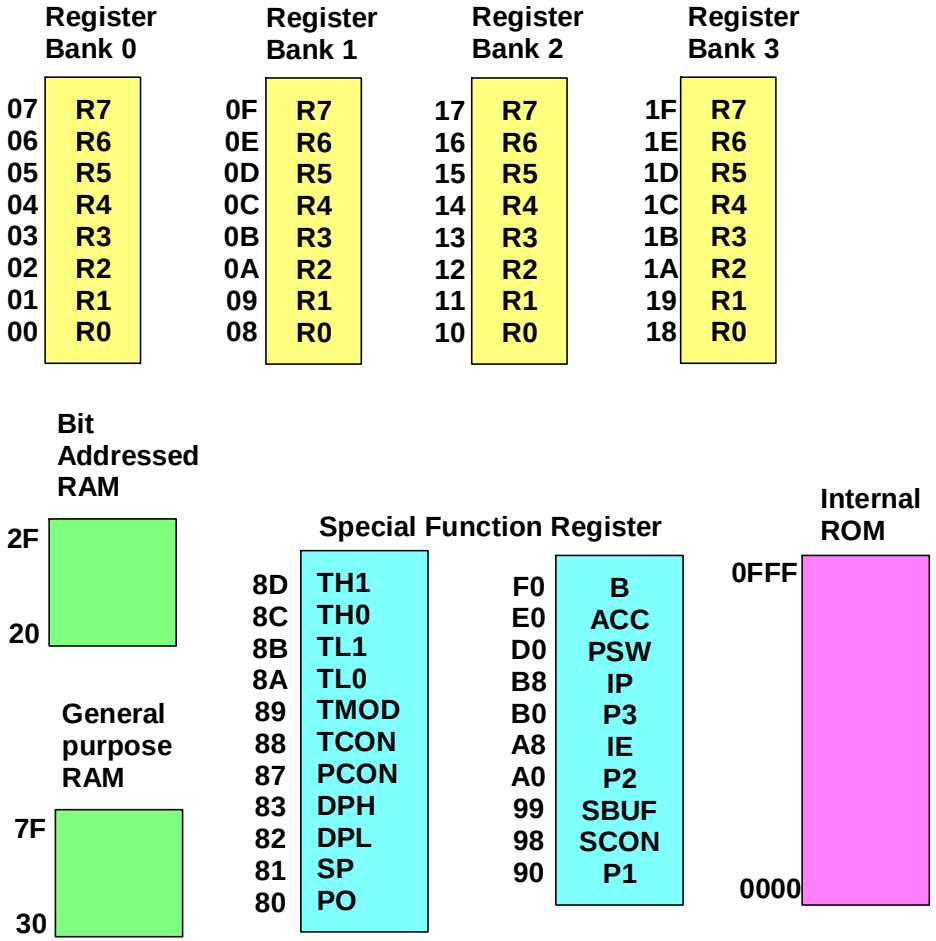
Please read these lecture notes in their entirety.

8051 Addressing Modes

Some slides adapted from Mr. K. T. Ng and Dr. H. J. Pottinger

Programmer's model of 8051

- The programmer's model (sometimes called a software model) of the 8051 includes the registers and accumulators accessible to the programmer.



Addressing Modes

- Most instructions act on data, which may be in internal MCU registers or out in memory.
- It isn't sufficient to simply state CLR — Clear what ?
- There are different ways of specifying the operand location:
  - CLR A ; the target is the Accumulator
  - CLR C ; the target is the Carry Flag.
- The different ways of pointing out an operand's location (source and destination) are the addressing modes.
- The addressing mode used by an instruction is specified by some of the bits in its opcode.

8051 Addressing Modes

- Each instruction contains a destination, and in most cases a source. The way the destination and the source are accessed is determined by the addressing mode.
- There are eight basic ways of specifying source/destination operand addresses for the 8051:
  - Register
  - Direct
  - Indirect
  - Immediate
  - Relative
  - Absolute
  - Long
  - Indexed

Register Addressing Mode

- An instruction is said to be using the register addressing mode if either source or destination or both operands are located in the CPU registers.
- In the 8051, the programmer can access registers A, DPTR, and R0 to R7.
- The register addressing mode is the most efficient way of specifying source and destination operands, for two reasons:
  - one or both of the operands is/are in the registers and no memory access is required.
  - Instructions using the register mode tend to be shorter, as only 3 bits are needed to identify a register. In contrast, we need at least 8 bits to identify a memory location.

Examples of Register Addressing Mode

Instruction	Operation
`MOV A,#12`	Copy the number 12 to A
`MOV A,R3`	Copy the contents of R3 to A
`CLR A`	Clear A
`MOV R0,#80`	Copy the number 80 to R0
`MOV DPTR,#1234H`	Copy the number 1234H to DPTR

Note: register to register moves using register addressing mode occur between registers A and R0 to R7 ONLY.

Direct Addressing Mode

- In direct addressing , the operand (either source or destination or both) is specified by an 8-bit address field in the instruction.
- In the 8051, all128 bytes of internal RAM and the SFRs may be addressed using the direct addressing mode.
- Most assemblers and compilers provide equates or standard symbol names for the SFRs and I/O ports, so the SFRs names may be used in lieu of their direct addresses.
  - Examples: P0 stands for address 80H
    TMOD stands for address 89H

Direct Addressing Mode

- The characteristic of this address mode is that the location of the operand is fixed and cannot be changed as the program execution progresses.
- Direct addressing is the most simple mode to understand, however its main drawback is its inflexibility for addressing elements in a table of data.

Standard Names for the SFRs

Standard SFR Name	Description	Address (Hex)
A	Accumulator	E0
B	B Register	F0
DPL	Data Pointer Low Byte	82
DPH	Data Pointer High Byte	83
IE	Interrupt Enable Register	A8
IP	Interrupt Priority Register	B8
PO	Port 0	80
P1	Port 1	90
P2	Port 2	A0
P3	Port 3	B0
PCON	Power Control Register	87
PSW	Program Status Word	D0
SBUF	Serial Data Buffer Register	99
SCON	Serial Port Control Register	98
SP	Stack Pointer	81
TCON	Timer/Counter Control Register	88
TMOD	Timer/Counter Mode Register	89
TH0	Timer 0 High Byte Register	8C
TL0	Timer 0 Low Byte Register	8A
TH1	Timer 1 High Byte Register	8D
TL1	Timer 1 Low Byte Register	8B

Examples of Direct Addressing Mode

Instruction	Operation
`MOV 80h, A or` **`MOV P0, A`**	Copy contents of A to the port 0 latch
`MOV A, 80h or` **`MOV A, P0`**	Copy contents of port 0 pins to A
`MOV A, addr`	Copy contents of direct address with label *addr* to A
`MOV R0, 12H`	Copy contents of RAM location 12H to Register 0
`MOV 0A8h, 77H or` **`MOV IE, 77H`**	Copy contents of RAM location 77h to IE register

Warnings for Direct Addressing Mode

- MOV instructions that refer to direct addresses above 7FH should be used carefully.
- Moving data to a port changes the port latch, whereas moving data from a port gets data from port pins (more later).
- Moving data from a direct address to itself is not predictable and could lead to errors.
  - Example: MOV SUM, SUM ⌛

Indirect Addressing Mode

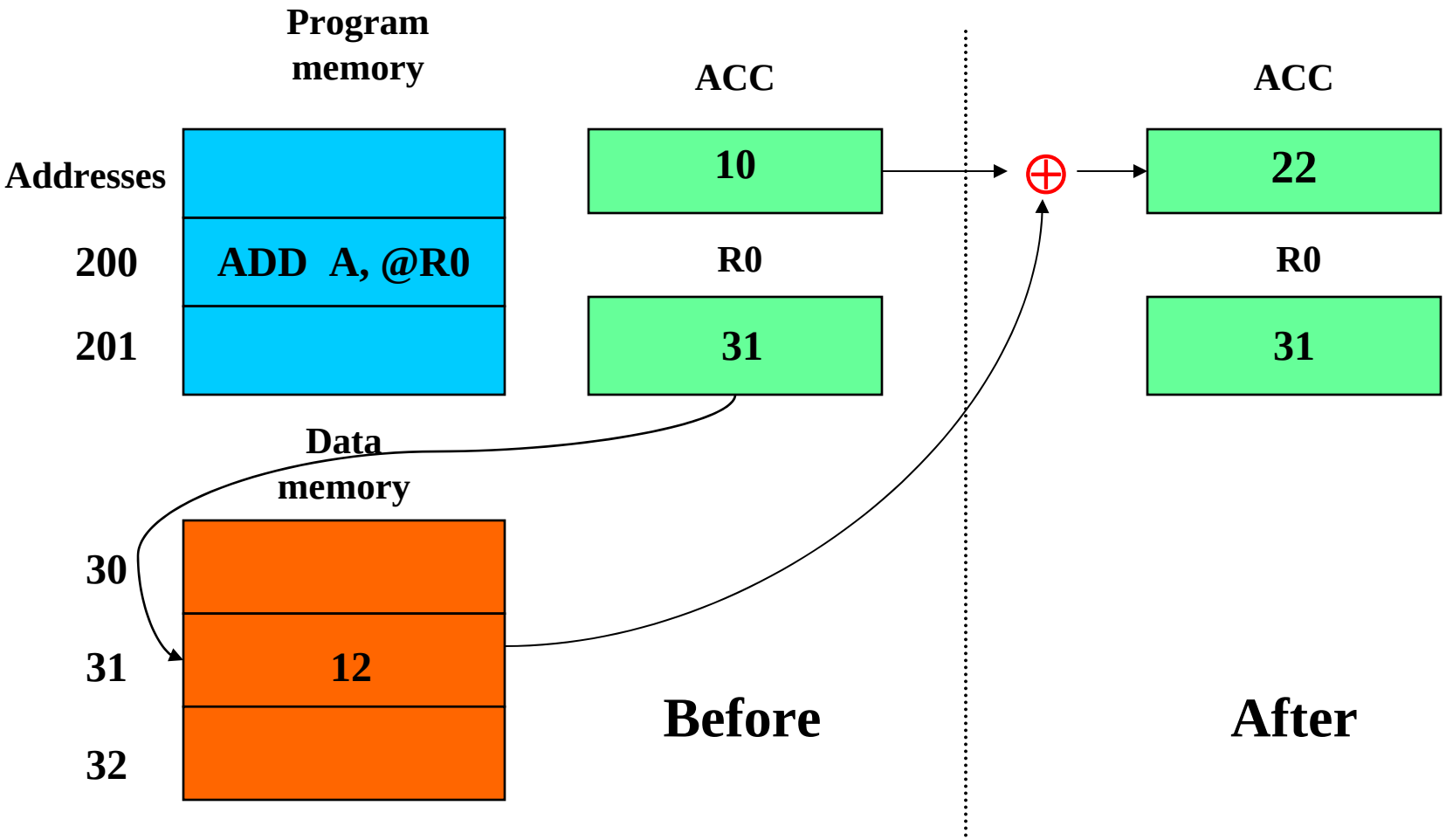
- In indirect addressing, the instruction specifies a register that contains the address of the operand.
- The register itself is not the address, but rather the number in the register is the address.
- In the 8051, the indirect addressing mode uses register R0 or R1 as the data pointer.
- The symbol used for indirect addressing is the “at” sign, which is printed as @ preceding the register R0 or R1.
- The essential advantage of indirect addressing is that the address of the data can be calculated at run time, as you might do when stepping through a table of data.

Examples of Indirect Addressing Mode

Instruction	Operation
MOV @R1,A	Copy the data in A to the address pointed to by the contents of R1
MOV A,@R0	Copy the contents of the address pointed to by register R0 to the A register
MOV @R1,#35H	Copy the number 35H to the address pointed to by register R1
MOV @R0,80H or **MOV @R0,P0**	Copy the contents of the port 0 pins to the address pointed to by register R0.

- The number in register R0 or R1 must be a RAM address in the range 00h to 7Fh (FFh for the 8052).
- Only registers R0 or R1 may be used for indirect addressing.

Indirect Addressing Mode Illustration



Immediate Addressing Mode

- Immediate addressing is used when an operand is treated as constant data and not an address.
- With the exception of the DPTR, all immediate data is 8 bits.
- The pound sign (#) is commonly used to indicate a constant number.
  - Note the difference between:
    - ADD A, #30h and
    - ADD A, 30h
- This addressing mode can only be used to specify the source operand.
- Another addressing mode is required to specify the destination operand.

Good Uses for Immediate Data

- Immediate variables are useful when the operand they represent does not have to change while the program is running.
- Some examples:
  - Initializing a total to zero
  - Setting a particular ASCII character (e.g. CR or LF)
  - Using a fixed value (e.g. dividing by 100 to calculate a percentage)

Examples of Immediate Addressing

Operation	Instruction
MOV A, #0AFH	Copy the immediate data AFH to A
ANL 15H, #88H	Logical AND (bit by bit) the contents of the address 15h with the immediate data 88H
MOV DPTR, #0ABCDH	Copy the immediate data ABCDH to the DPTR register
MOV R3, #1CH	Move the immediate data 1CH to R3
MOV R2, #'A'	Move the ASCII character A (with value = 41h) to register R2

Noteworthy for Immediate Addressing

- It is impossible to have immediate data as a destination.
- When the hex number starts with the letters A to F, it must be preceded with the digit 0, otherwise the assembler assumes the number is an address variable.
- Hexadecimal numbers must have a suffix of ‘H’ e.g. 7AH.
- Binary numbers must have a suffix of ‘B’ e.g. 10110111B.
- Numbers with no suffix are assumed to be decimal.
- ASCII characters must be surrounded by single-quote marks e.g. ‘F’.

Relative Addressing Mode

- Relative addressing is used only with certain jump instructions.
- A relative address (or offset) is an 8-bit value, that is added to the program counter to form the effective address of the next instruction to be executed.
- The range of the jump can be from -128 to +127 memory addresses.
- This effectively alters the flow of the program; causing the program to skip to another instruction either in advance or behind the instruction that would normally follow next.

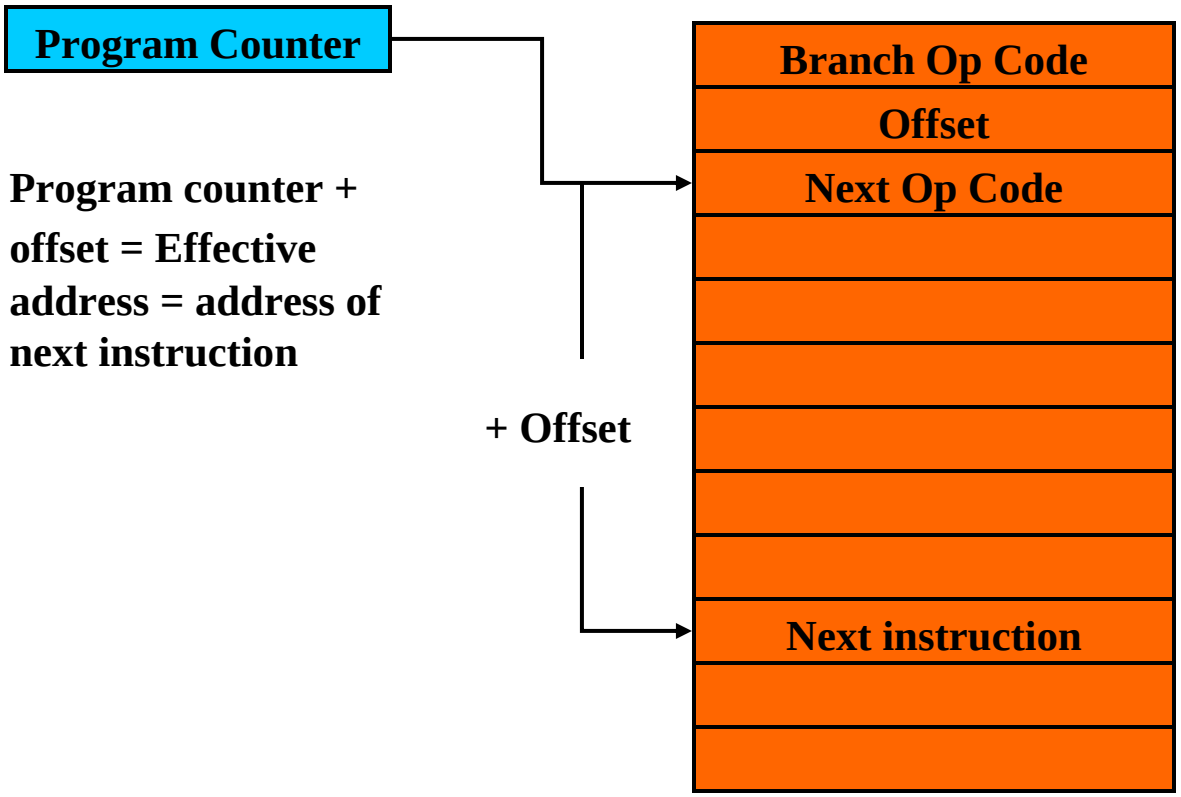
Relative Addressing Mode

- The jump destinations are usually specified as labels; the assembler determines the relative offset accordingly.
  - e.g. **`SJMP  LOOP1`**

Examples of Relative Addressing

Instruction	Operation
SJMP NXT	Jump to the relative address with label 'NXT'; this is an unconditional jump and is always performed.
DJNZ R1, DWN	Decrement register R1 by 1 and jump to the relative address specified by the label 'DWN' if the resulting value of R1 is not zero.

Relative Addressing Mode Illustration

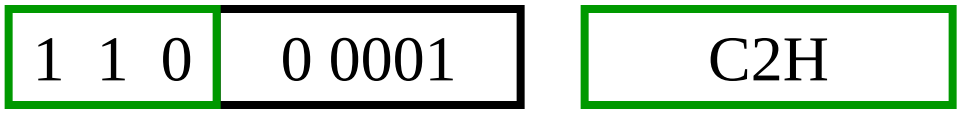


Absolute Addressing Mode

- Absolute addressing is used only with the ACALL and AJMP instructions.
- Its operation is similar to the relative addressing mode, but it provides branching to a destination within a 2K range.
- Eleven address bits are embedded in the two byte instruction.
- Example: **`AJMP ABIG`** Jump to absolute short range address with label ‘ABIG’
- The primary advantage of absolute addressing:
  - increased speed of execution
  - reduced code size: 2 bytes per instruction instead of 3 bytes, as compared to long addressing.

Example

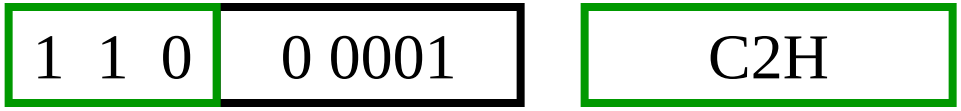
- **`AJMP next`**
- Eleven address bits are embedded in the two byte instruction.
- Encoding:



- What about the other five bits?
  (11 + 5 = 16 bits)
- *Effective address* = PC(15:11),OP(7:5),B2

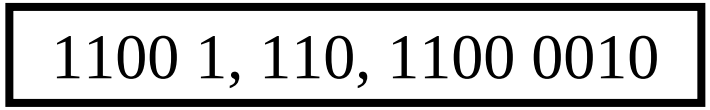
```
C800 C1 C2  AJMP next   ; PC<- ????
 A) pc<- 06C2h  B) something else
```

Example

- **`AJMP next`**
- Eleven address bits are embedded in the two byte instruction.
- Encoding: 

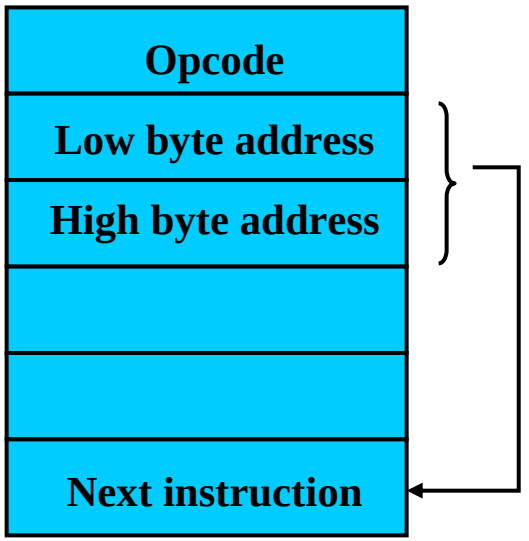
- What about the other five bits?
  (11 + 5 = 16 bits)
- *Effective address* = PC(15:11),OP(7:5),B2

```
C800 C1 C2  AJMP next    ; PC<- 0CEC2h
```



Long Addressing Mode

- The long addressing mode is used only with the LCALL and LJMP instructions.
- Its operation is similar to the absolute addressing mode, but it enables branching to a destination within a 64K range.
- The 3-byte instruction includes a full 16-bit destination address as bytes 2 and 3 of the instruction.



Indexed Addressing Mode

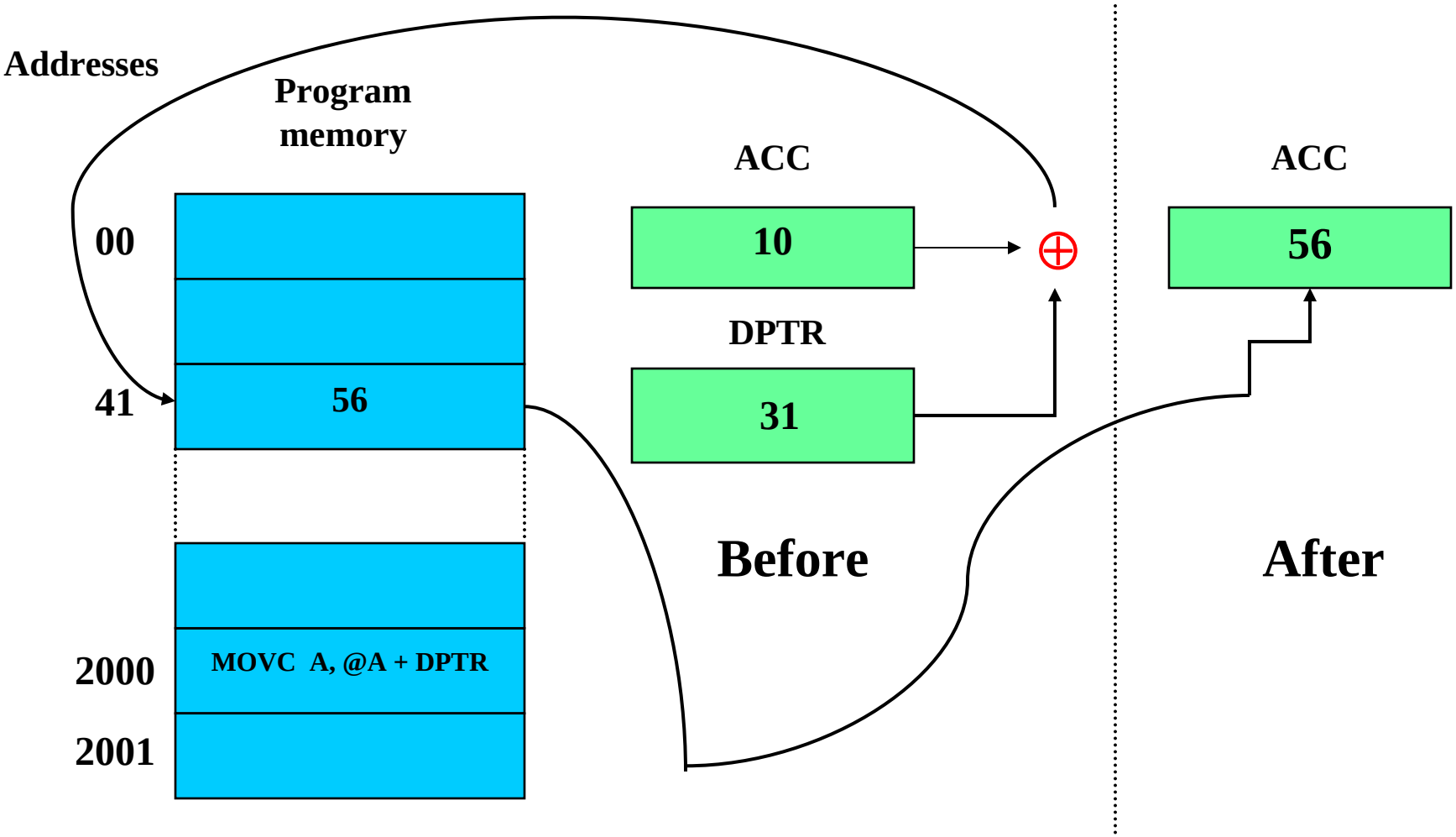
- The indexed addressing mode uses a base register (either the program counter or the data pointer register) and an offset (the accumulator) in forming the effective address for a JMP or MOV instruction.
- Indexed addressing finds a memory location based on an index. It is often used to access elements of an array or look-up table.
- DPTR contains the starting address of the array or a look up table, and the accumulator, A, contains the offset to the element being addressed.
- To step through the array or table, the accumulator is incremented or decremented by a program instruction.

Examples of Indexed Addressing Mode

Instruction	Operation
MOVC A, @A+DPTR	Copy the code byte found at the ROM address formed by adding register A and the DPTR register to A
MOVC A, @A+PC	Copy the code byte found at the ROM address formed by adding A and the PC to A
JMP @A+DPTR	Jump to the address formed by adding A to the DPTR. This is an unconditional jump and will always be performed.

Note: The PC is incremented by one (to point to the next instruction) before it is added to A to form the final address of the code byte.

Indexed Addressing Illustration



Why So Many Modes?

- The essential motivation for providing several addressing modes comes from the need to efficiently support high level language constructs.
- Example: Suppose we wish to clear an area of memory between 30h and 7Fh, say to hold an array of 80 byte-long elements Array[0] to Array[79].
  - The obvious way to do this is to use a MOV instruction in direct addressing mode for each byte.
  - This program would need 80 3-byte instructions, for a total of 240 bytes of program memory for storage.
  - This is approximately 6 % of the internal 4K byte code memory.

Why So Many Modes?

- A better way is to use a pointer into the array, and increment the pointer each time we do a MOV instruction, this implies the use of indirect addressing.

```
MOV  30h, #00h  ; Clear Array [0]
MOV 31h, #00h  ; and Array [1]
        .              ; Keep on going
        .
        .
MOV 7Eh, #00h  ; Clear Array [78]
MOV 7Fh, #00h  ; Clear Array [79]
```

Direct addressing uses 240 bytes.

```
                MOV  R0, #30h  ;Set up pointer to start of array
clear_arr:  MOV  @R0, #00 ;Clear target byte pointed to by R0
                INC    R0            ; Advance pointer by 1
                CJNE  R0, #80, clear_arr ; Has pointer reached 80 ?
NEXT:       . . . . . .      ; if not over the top THEN again ELSE
                                  ; next instruction
```

Indirect addressing uses ONLY 8 bytes, with a saving of 232 bytes!

8051 Instruction Set

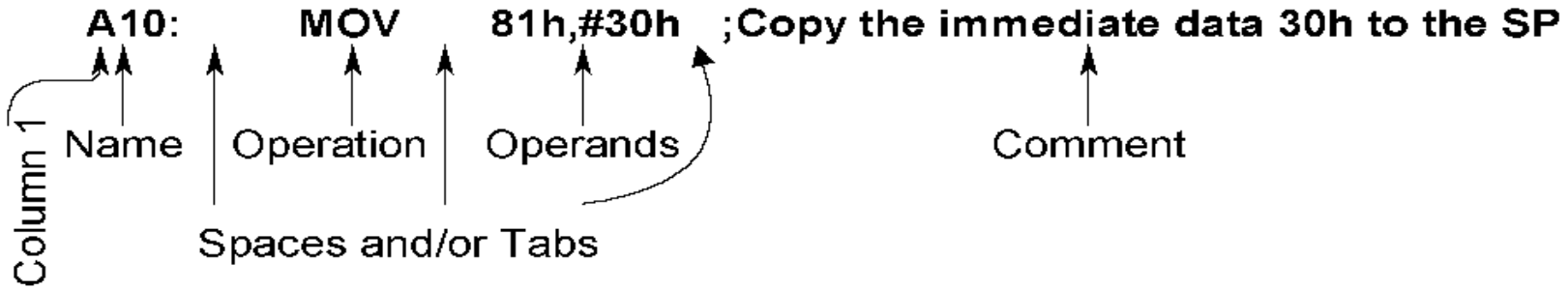
Some slides adapted from Mr. K. T. Ng and Dr. H. J. Pottinger

Review: Assembly Language Statement Syntax

- A typical assembly language statement consists of 4 fields:

  [Name:] Operation [operands] [;comment]

- The fields in the square brackets are optional in some statements.
- The label, if given, must start in column 1.
- If no label is given, the opcode or operand portions must start past column 1.
- Each portion is separated from the others by any combination of spaces and /or tabs.



Data Transfer Instructions

- The 8051 microcontroller has a group of data transfer instructions to move data around without alteration in between registers and memory.
- These instructions include “MOV”; “MOVC”; “MOVX” “PUSH”; “POP”; “XCH”; and “XCHD.”
- They are the most used and flexible of the instruction categories.

The MOV Instruction

- The most basic operation: copying data from one place to another.
- Write as: `MOV destination,source`
- destination and source are operands.
- Source
  - immediate value (e.g. #35h)
  - register (R1, R2, . . . R7, and A)
  - Memory locations (internal memory address:00 - 7Fh and SFRs)
- Destination
  - Any of the above except immediate value - Why?

MOV Instruction Examples

- **`MOV A,addr`**
  - Copy data from direct address *addr* to Acc
- **`MOV 3Ah,#3Ah`**
  - Copy immediate data 3Ah to RAM location 3Ah.
  - Note the difference between immediate data and address.
- **`MOV addr1, addr2`**
  - Copy data from direct address *addr2* to direct address *addr1*
  - Note this allows data to be transferred between any two internal RAM or SFR locations without going through the Accumulator.

More MOV Examples

- **`MOV  @R1, #35h`**
  - Copy the number 35h to the address in R1.
- **`MOV  add_1, @R0`**
  - Copy the contents of the address in R0 to add_1
- **`MOV  @R0,80h`**
  - Copy the contents of the port 0 pins to the address in R0
- It should be noted that ONLY registers R0 and R1 can be used for indirect addressing.
- One of four addressing modes can be used:
  - Immediate, register, direct and indirect addressing modes.

Data Movement Summary

Dest \ Source	A	#data	direct	@Ri	Rn
A	✗	✓	✓	✓	✓
direct	✓	✓	✓	✓	✓
@Ri	✓	✓	✓	✗	✗
Rn	✓	✓	✓	✗	✗

The MOVX Instruction

- This instruction is used to transfer data between external RAM and internal register A.
- The letter X is added to the MOV to serve as a reminder that the data move is external to the 8051.
- It uses the accumulator as either the source or destination operand.
- Syntax:

```
MOVX    A,@Ri
MOVX    A,@DPTR
MOVX    @Ri,A
MOVX    @DPTR,A
```

- MOVX is normally used with external RAM or I/O addresses.

The MOVX Instruction

- MOVX can only use indirect addressing. The indirect address is specified by using:
  - a 1-byte address (@Ri, where Ri is either R0 or R1)
    - Ri can address ONLY 256 bytes! Useful when?
  - a 2-byte address (@DPTR, this precludes the use of Port 2)
    - DPTR can address 64K bytes.
- The read and write strobe to external RAM (/RD & /WR) are activated only during the execution of a MOVX instruction (more on this in the next lecture).

The MOVC Instruction

- The letter C is added to MOV to highlight the use of the opcode for moving data from a source address in Code ROM to the A register in the 8051.
- Syntax: **`MOVC A,@A + DPTR`**

  `MOVC A,@A + PC`
- For the instruction: **`MOVC A,@A + PC`**
  - The PC is incremented by one (to point to the next instruction) before it is added to A to form the final address of the code byte.
  - All data is moved from code memory to the A register.

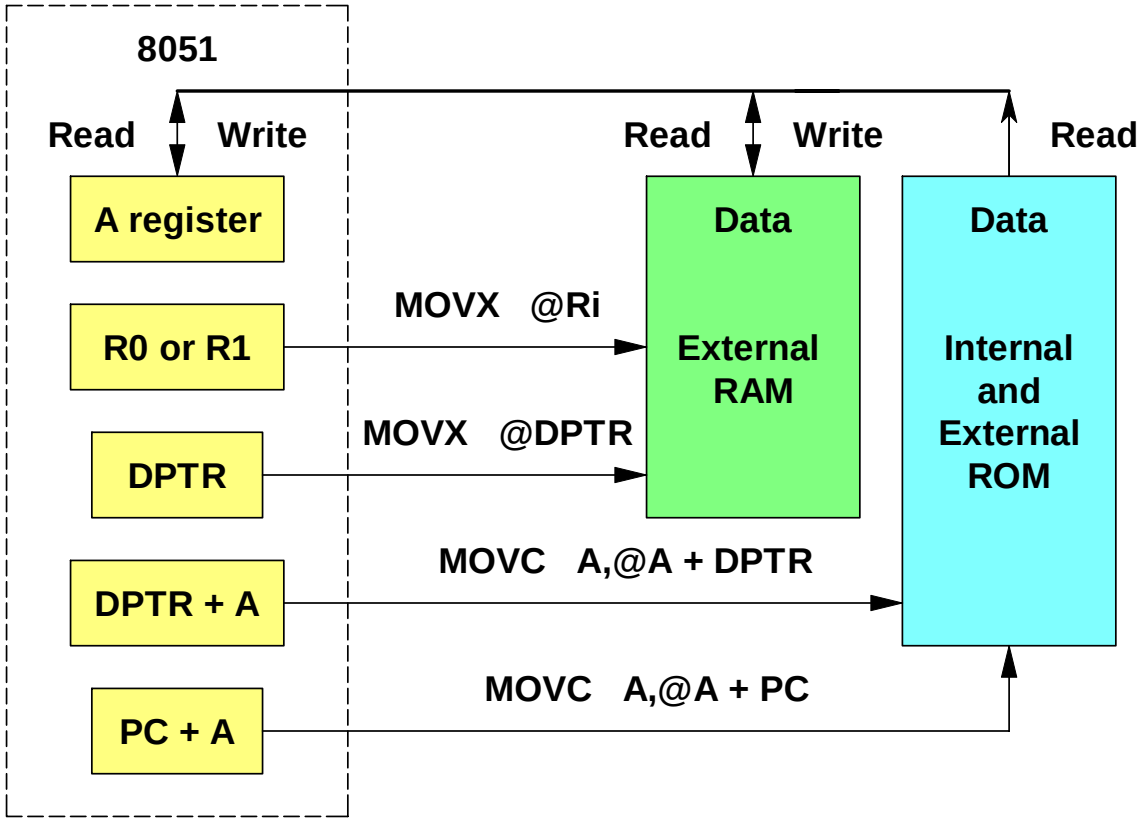
The MOVC Instruction

- MOVC is usually used with internal or external ROM and can address 4K of internal or 64K bytes of external code.
- This opcode is used to access the program memory for reading look-up tables.
- It uses either the program counter or the data pointer as the base register and the accumulator as offset.

Look-Up Table Example

```
                MOV     A,#EntryNumber
                ACALL   LookUp
LookUp:         INC     A
                MOVC    A,@A + PC
                RET
TABLE:          DB      7Eh, 30h, 6Dh, ...
```

Summary of External Addressing Using MOVX and MOVC



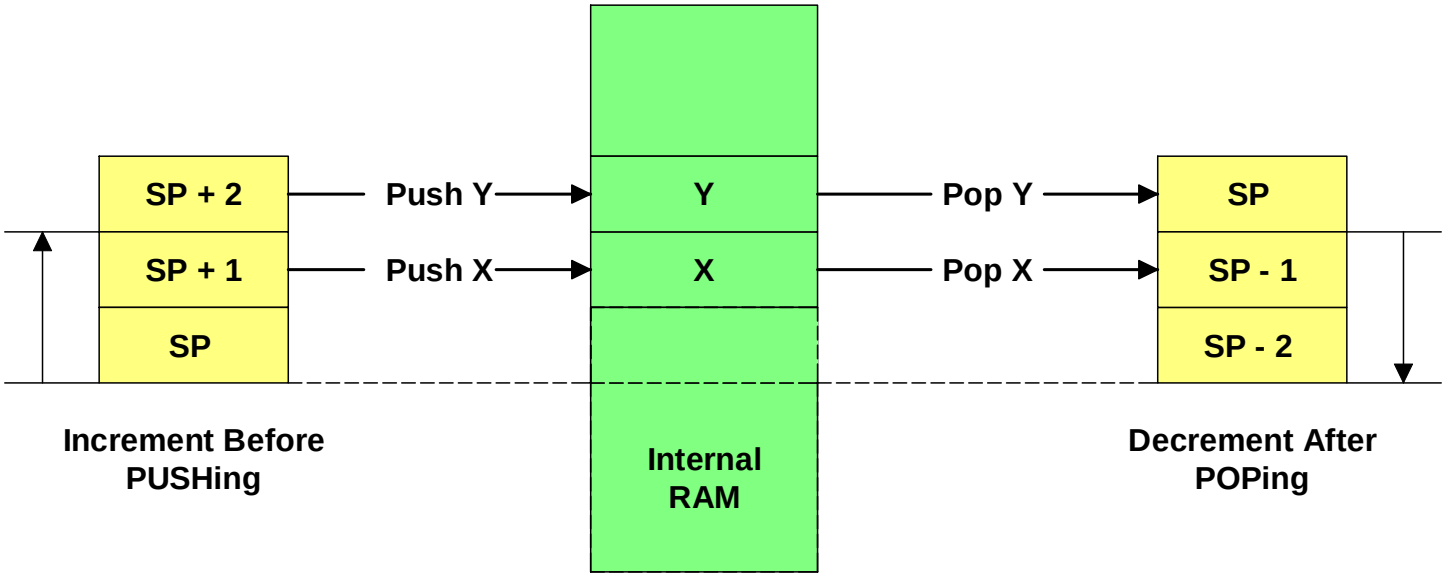
PUSH and POP Instructions

- These two instructions are used for stack operations.
- The 8051's stack resides in on-chip RAM and grows upward in memory.
- The stack pointer (SP) points to the current highest occupied location of the stack.
- The PUSH instruction first increments the stack pointer, then copies the byte into the stack.
- The POP instruction copies the top of the stack into the destination, then decrement the stack pointer.

PUSH and POP Instructions

- The SP register is set to 07h when the 8051 is reset. This is the same direct address as register R7 in bank 0.
- Unless SP is changed, the first PUSH opcode will write data to R0 in bank 1.
- The SP should be initialized by the programmer to point to an internal RAM address above the highest address likely to be used by the program.
- It is usually set at addresses above the register banks.
- When the SP reaches FFh it “rolls over” to 00h (R0).
- RAM ends at address 7Fh; PUSHes above 7Fh result in errors.

PUSH and POP Examples




```
MOV   81h,#30h    ;Copy immediate data 30h to the SP
MOV   R0,#0ACh    ;Copy the immediate data ACh to R0
PUSH  00h         ;SP = 31h and address 31h contains ACh
PUSH  00h         ;SP = 32h and address 32h contains ACh
POP   01h         ;SP = 31h and register R1 now contains ACh
POP   80h         ;SP = 30h and port 0 latch now contains ACh
```

XCH and XCHD Instructions

- XCH causes the accumulator and the addressed byte to exchange data.
- XCHD cause the lower-nibbles of the accumulator and the addressed byte to be exchanged.
- Syntax:

```
XCH     A,Rn
XCH     A,direct
XCH     A,@Ri
XCHD    A,@Ri
```

- All exchanges use register A and are internal to the 8051.
- When using XCHD, the upper nibble of A and the upper nibble of the address location in Ri do not change.

For Wednesday

- Review today's lecture notes and Chapters 3 and 5 of your textbook.
- Begin assignment 6.
- Read Chapters 4,6 and 7.