8051 Registers , Delays and I/O

Lecture 3

Bilal Habib

DCSE, UET

Writing to Port

The following code will continuously send out to port 1 the alternating value 55H and AAH

```
BACK:
        MOV A,#55H
        MOV P1,A
        ACALL  DELAY
        MOV A,#0AAH
        MOV P1,A
        ACALL  DELAY
SJMP BACK
```

Reading from Port

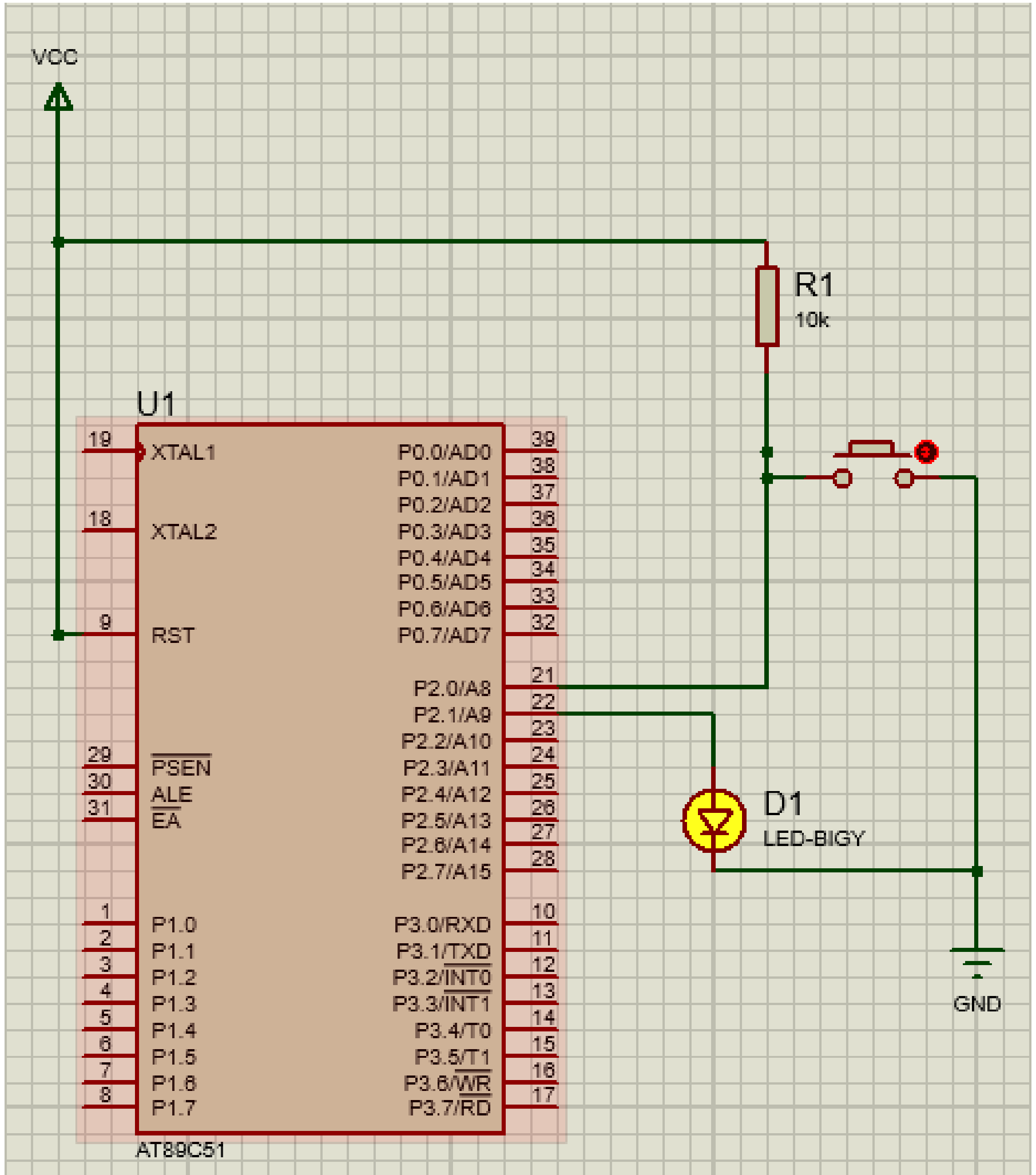
In order to make port an input, the port must be programmed by writing 1 to all the bits

Port 1 is configured first as an input port by writing 1s to it, and then data is received from that port and sent to P1

```
MOV A,#0FFH             ;A=FF hex
MOV P1,A                ;make P1 an i/p port
                        ;by writing it all 1s
BACK:
        MOV A,P1        ;get data from P1
        MOV P2,A        ;send it to port 2
SJMP BACK               ;keep doing it
```

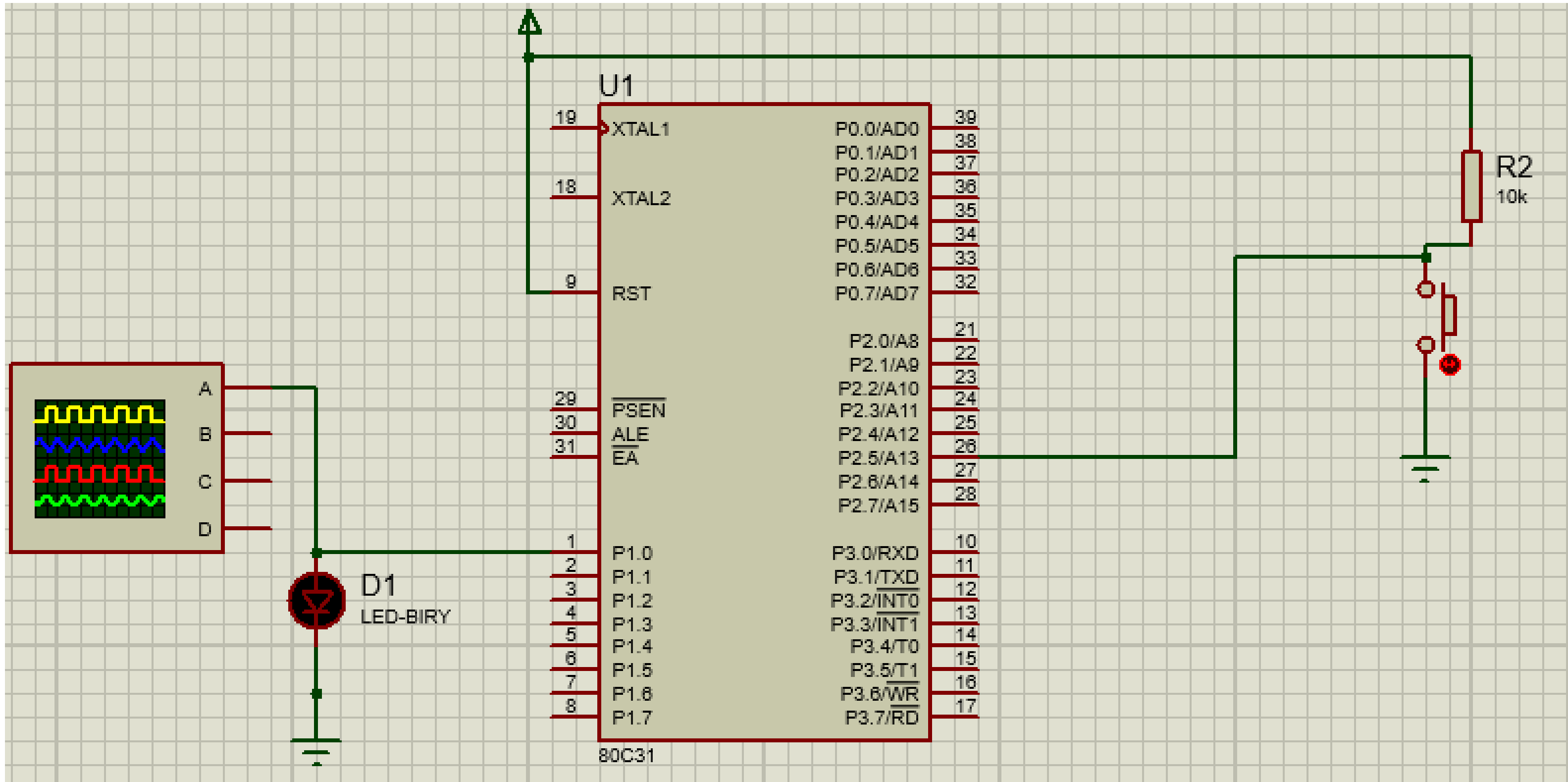
Read/Write to a Port

```
$NOMOD51
$INCLUDE (8051.MCU)

        ; Reset Vector
        org    0000h
        jmp    Start


        org    0100h
Start:
          SETB P2.0   ; Configure it as an input
Again:    MOV C, P2.0 ; Read the pin
          MOV P2.1, C ; Send to LED

          jmp Again

;===============================================
        END
```

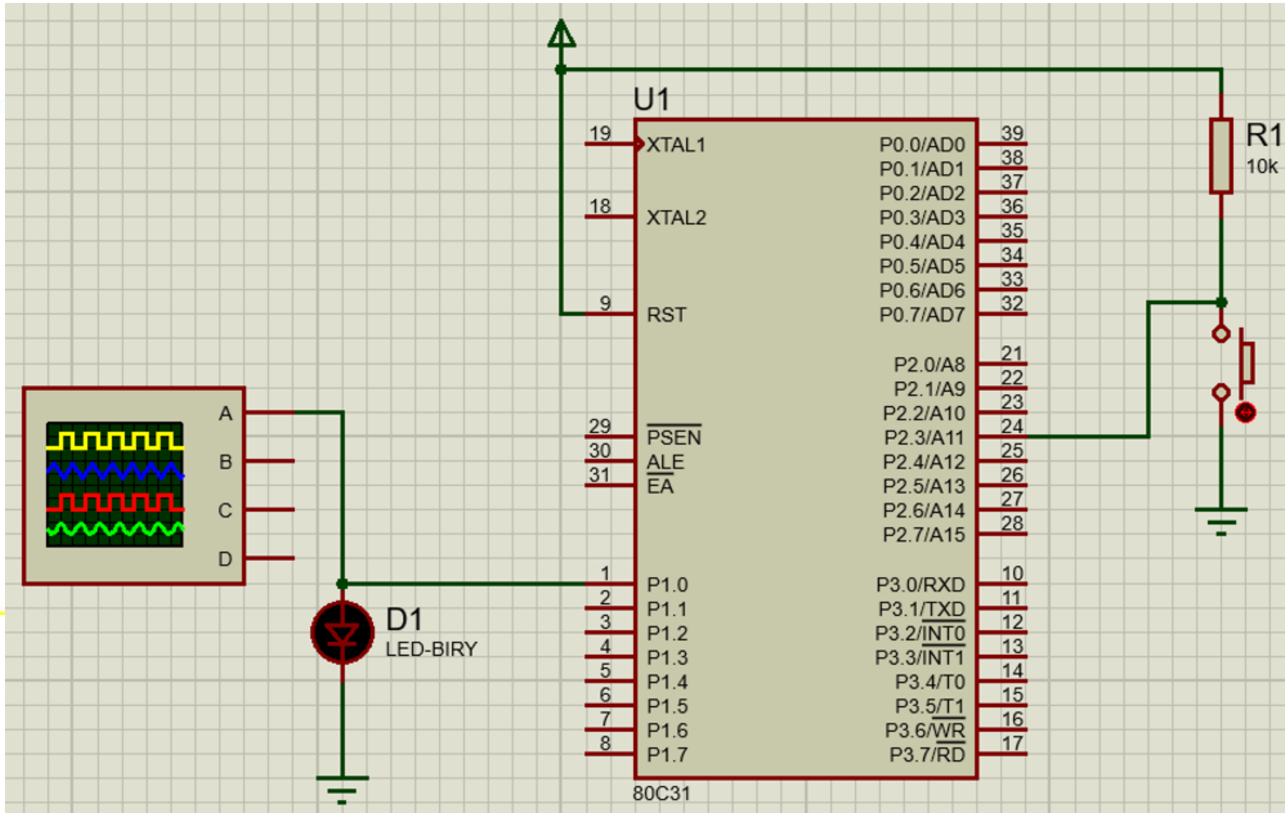


Read/Write to a Port

```c
#include <reg51.h>
#include <stdio.h>
sbit pin = P2^5;
sbit led = P1^0;
void MS_delay(unsigned int);
void main(void)
 {
        pin = 1;        //configure as input
        while(1)
        {
            if(pin == 0)
                led = 1;
            else
                led = 0;
        }
 }
```

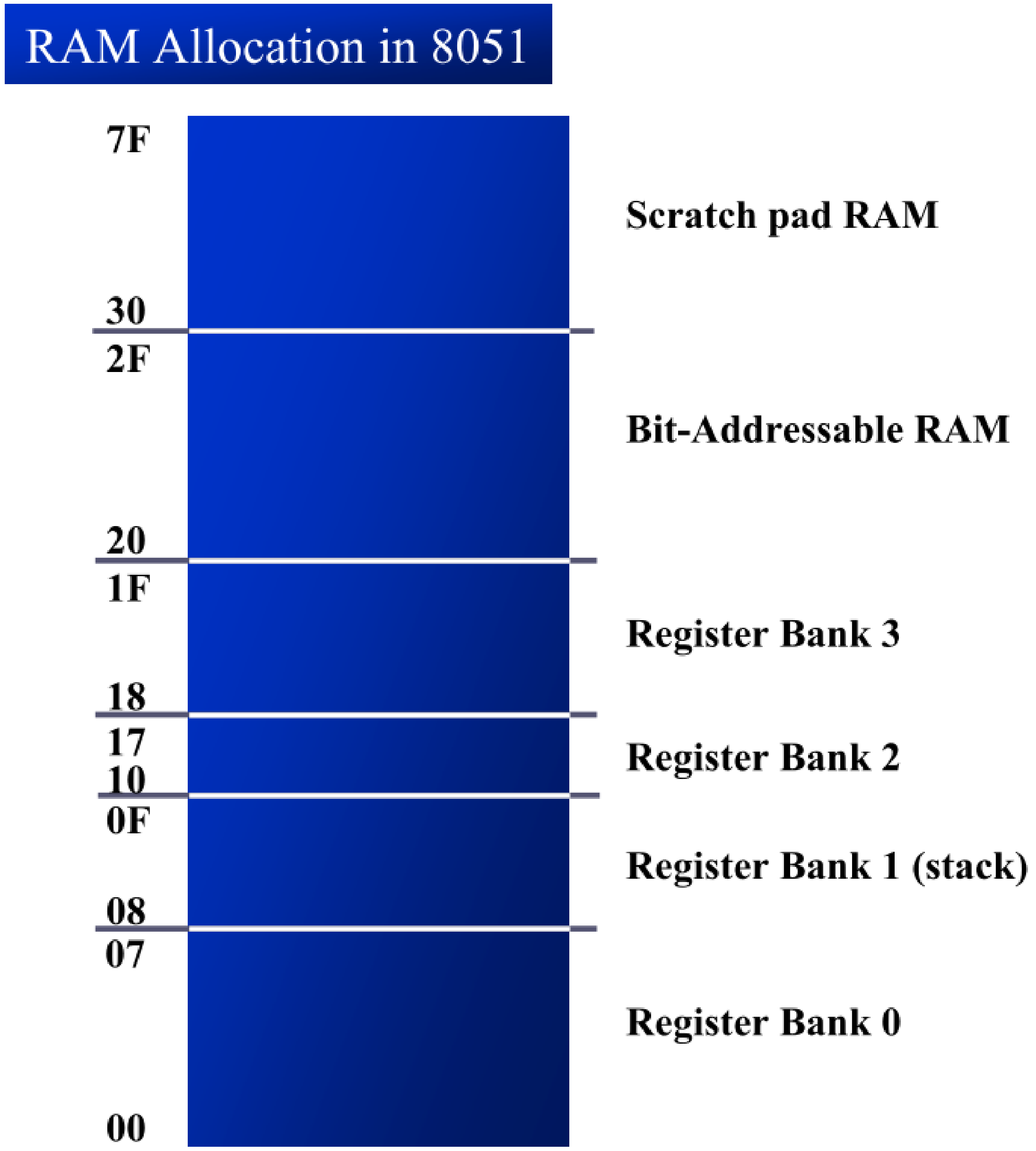


```c
#include <reg51.h>
#include <stdio.h>
sbit button = P2^3;       // Define a pin
unsigned int x = 0;                // 0 to 65535 (16 bit variable)
void delay(unsigned int);
void main(void)
 {
        while(1)
        {
            if(button == 1) // When it is not pressed
            ;               // Do nothing
            else {      // When it is pressed
            P1 = 0x01;  // 55 = 0000 0001...ON
            delay(50000);
            P1 = 0x00;  // AA = 0000 0000...Off
            delay(10000);
            }
        }
 }

void delay(unsigned int y)
{
    for(x = 0; x < y; x++);
}
```



8051 REGISTER BANKS AND STACK

RAM Memory Space Allocation (cont')



8051 REGISTER BANKS AND STACK

Stack

- The stack is a section of RAM used by the CPU to store information temporarily
  - This information could be data or an address
- The register used to access the stack is called the SP (stack pointer) register
  - The stack pointer in the 8051 is only 8 bit wide, which means that it can take value of 00 to FFH
  - When the 8051 is powered up, the SP register contains value 07
    - RAM location 08 is the first location begin used for the stack by the 8051

8051 REGISTER BANKS AND STACK

Stack (cont')

- The storing of a CPU register in the stack is called a `PUSH`
  - SP is pointing to the last used location of the stack
  - As we push data onto the stack, the SP is incremented by one
    - This is different from many microprocessors
- Loading the contents of the stack back into a CPU register is called a `POP`
  - With every pop, the top byte of the stack is copied to the register specified by the instruction and the stack pointer is decremented once

8051 REGISTER BANKS AND STACK

Pushing onto Stack

Example 2-8

Show the stack and stack pointer from the following. Assume the default stack area.

```
MOV R6, #25H
MOV R1, #12H
MOV R4, #0F3H
PUSH 6
PUSH 1
PUSH 4
```

Solution:

		After PUSH 6	After PUSH 1	After PUSH 4
0B				
0A				F3
09			12	12
08		25	25	25
	Start SP = 07	SP = 08	SP = 09	SP = 0A

8051 REGISTER BANKS AND STACK

Popping From Stack

Example 2-9

Examining the stack, show the contents of the register and SP after execution of the following instructions. All value are in hex.

```
POP     3          ; POP stack into R3
POP     5          ; POP stack into R5
POP     2          ; POP stack into R2
```

Solution:

	Start SP = 0B	After POP 3, SP = 0A	After POP 5, SP = 09	After POP 2, SP = 08
0B	54			
0A	F9	F9		
09	76	76	76	
08	6C	6C	6C	6C

Because locations 20-2FH of RAM are reserved for bit-addressable memory, so we can change the SP to other RAM location by using the instruction "`MOV SP, #XX`"

8051 REGISTER BANKS AND STACK

CALL Instruction And Stack

- The CPU also uses the stack to save the address of the instruction just below the `CALL` instruction
  - This is how the CPU knows where to resume when it returns from the called subroutine

8051 REGISTER BANKS AND STACK

Incrementing Stack Pointer

- The reason of incrementing SP after push is
  - Make sure that the stack is growing toward RAM location 7FH, from lower to upper addresses
  - Ensure that the stack will not reach the bottom of RAM and consequently run out of stack space
  - If the stack pointer were decremented after push
    - We would be using RAM locations 7, 6, 5, etc. which belong to R7 to R0 of bank 0, the default register bank

8051 REGISTER BANKS AND STACK

Stack and Bank 1 Conflict

- When 8051 is powered up, register bank 1 and the stack are using the same memory space
  - We can reallocate another section of RAM to the stack

8051 REGISTER BANKS AND STACK

Stack And Bank 1 Conflict (cont')

Example 2-10

Examining the stack, show the contents of the register and SP after execution of the following instructions. All value are in hex.

```
MOV SP, #5FH     ;make RAM location 60H
                 ;first stack location
MOV R2, #25H
MOV R1, #12H
MOV R4, #0F3H
PUSH  2
PUSH  1
PUSH  4
```

Solution:

		After PUSH 2		After PUSH 1		After PUSH 4	
63		63		63		63	
62		62		62		62	F3
61		61		61	12	61	12
60		60	25	60	25	60	25
Start SP = 5F		SP = 60		SP = 61		SP = 62	

Jump Loop and Call Instructions

LOOP AND JUMP INSTRUCTIONS

Looping

- Repeating a sequence of instructions a certain number of times is called a *loop*
  - Loop action is performed by

    `DJNZ reg, Label`

    - The register is decremented
    - If it is not zero, it jumps to the target address referred to by the label
    - Prior to the start of loop the register is loaded with the counter for the number of repetitions
    - Counter can be R0 – R7 or RAM location

A loop can be repeated a maximum of 255 times, if R2 is FFH

```
;This program adds value 3 to the ACC ten times
        MOV   A,#0      ;A=0, clear ACC
        MOV   R2,#10    ;load counter R2=10
AGAIN:  ADD   A,#03     ;add 03 to ACC
        DJNZ  R2,AGAIN  ;repeat until R2=0,10 times
        MOV   R5,A      ;save A in R5
```

LOOP AND JUMP INSTRUCTIONS

Nested Loop

- If we want to repeat an action more times than 256, we use a loop inside a loop, which is called *nested loop*
  - We use multiple registers to hold the count

Write a program to (a) load the accumulator with the value 55H, and (b) complement the ACC 700 times

```
        MOV  A,#55H    ;A=55H
        MOV  R3,#10    ;R3=10, outer loop count
NEXT:   MOV  R2,#70    ;R2=70, inner loop count
AGAIN:  CPL  A         ;complement A register
        DJNZ R2,AGAIN  ;repeat it 70 times
        DJNZ R3,NEXT
```

LOOP AND JUMP INSTRUCTIONS

Conditional Jumps

- Jump only if a certain condition is met

JZ label ;jump if A=0

```
        MOV   A,R0      ;A=R0
        JZ    OVER      ;jump if A = 0
        MOV   A,R1      ;A=R1
        JZ    OVER      ;jump if A = 0
        ...
OVER:
```

Can be used only for register A, not any other register

Determine if R5 contains the value 0. If so, put 55H in it.

```
        MOV   A,R5      ;copy R5 to A
        JNZ   NEXT      ;jump if A is not zero
        MOV   R5,#55H
NEXT:   ...
```

LOOP AND JUMP INSTRUCTIONS

Conditional Jumps (cont')

```
JNC label    ;jump if no carry, CY=0
```

- If CY = 0, the CPU starts to fetch and execute instruction from the address of the label
- If CY = 1, it will not jump but will execute the next instruction below JNC

Find the sum of the values 79H, F5H, E2H. Put the sum in registers R0 (low byte) and R5 (high byte).

```
        MOV   A,#0     ;A=0
        MOV   R5,A     ;clear R5
        ADD   A,#79H   ;A=0+79H=79H
;       JNC   N_1      ;if CY=0, add next number
;       INC   R5       ;if CY=1, increment R5
N_1:    ADD   A,#0F5H  ;A=79+F5=6E and CY=1
        JNC   N_2      ;jump if CY=0
        INC   R5      ;if CY=1,increment R5 (R5=1)
N_2:    ADD   A,#0E2H  ;A=6E+E2=50 and CY=1
        JNC   OVER     ;jump if CY=0
        INC   R5       ;if CY=1, increment 5
OVER:   MOV   R0,A     ;now R0=50H, and R5=02
```

MOV R5,#0

LOOP AND JUMP INSTRUCTIONS

Conditional Jumps (cont')

8051 conditional jump instructions

Instructions	Actions
JZ	Jump if A $=$ 0
JNZ	Jump if A $\neq$ 0
DJNZ	Decrement and Jump if A $\neq$ 0
CJNE A,byte	Jump if A $\neq$ byte
CJNE reg,#data	Jump if byte $\neq$ #data
JC	Jump if CY $=$ 1
JNC	Jump if CY $=$ 0
JB	Jump if bit $=$ 1
JNB	Jump if bit $=$ 0
JBC	Jump if bit $=$ 1 and clear bit

- All conditional jumps are short jumps
  - The address of the target must within -128 to +127 bytes of the contents of PC

LOOP AND JUMP INSTRUCTIONS

Unconditional Jumps

- The unconditional jump is a jump in which control is transferred unconditionally to the target location

`LJMP` (long jump)

- 3-byte instruction
  - First byte is the opcode
  - Second and third bytes represent the 16-bit target address
    - Any memory location from 0000 to FFFFH

`SJMP` (short jump)

- 2-byte instruction
  - First byte is the opcode
  - Second byte is the relative target address
    - 00 to FFH (forward +127 and backward -128 bytes from the current PC)

CALL INSTRUCTIONS

- Call instruction is used to call subroutine
  - Subroutines are often used to perform tasks that need to be performed frequently
  - This makes a program more structured in addition to saving memory space

`LCALL` (long call)

- 3-byte instruction
  - First byte is the opcode
  - Second and third bytes are used for address of target subroutine
    - Subroutine is located anywhere within 64K byte address space

`ACALL` (absolute call)

- 2-byte instruction
  - 11 bits are used for address within 2K-byte range

CALL INSTRUCTIONS

LCALL

- When a subroutine is called, control is transferred to that subroutine, the processor
  - Saves on the stack the the address of the instruction immediately below the LCALL
  - Begins to fetch instructions form the new location
- After finishing execution of the subroutine
  - The instruction RET transfers control back to the caller
    - Every subroutine needs RET as the last instruction

CALL INSTRUCTIONS

LCALL (cont')

```
        ORG   0
BACK:   MOV   A,#55H     ;load A with 55H
        MOV   P1,A       ;send 55H to port 1
        LCALL DELAY      ;time delay
        MOV   A,#0AAH    ;load A with AA (in hex)
        MOV   P1,A       ;send AAH to port 1
        LCALL DELAY
        SJMP  BACK       ;keep doing this indefinitely

;---------- this is delay subroutine ------------
        ORG   300H       ;put DELAY at address 300H
DELAY:  MOV   R5,#0FFH   ;R5=255 (FF in hex), counter
AGAIN:  DJNZ  R5,AGAIN   ;stay here until R5 become 0
        RET              ;return to caller (when R5 =0)
        END
```

Upon executing "LCALL DELAY", the address of instruction below it, "MOV A,#0AAH" is pushed onto stack, and the 8051 starts to execute at 300H.

The counter R5 is set to FFH; so loop is repeated 255 times.

When R5 becomes 0, control falls to the RET which pops the address from the stack into the PC and resumes executing the instructions after the CALL.

The amount of time delay depends on the frequency of the 8051

CALL INSTRUCTIONS

ACALL

- The only difference between ACALL and LCALL is
  - The target address for LCALL can be anywhere within the 64K byte address
  - The target address of ACALL must be within a 2K-byte range
- The use of ACALL instead of LCALL can save a number of bytes of program ROM space

TIME DELAY FOR VARIOUS 8051 CHIPS

- CPU executing an instruction takes a certain number of clock cycles
  - These are referred as to as *machine cycles*
- The length of machine cycle depends on the frequency of the crystal oscillator connected to 8051
- In original 8051, one machine cycle lasts 12 oscillator periods

Duration of 1 Machine cycle = 12 Oscillator cycles
Therefore, if Oscillator frequency (fclk) =12 MHz, then
Time period of Oscillator cycle = 1/(12MHz)
Frequency of Machine cycles = 12/(12 Mhz) = 1MHz
=> Duration of 1 Machine cycle = 1 μsec

For 8051 system of 12 MHz, find how long it takes to execute each instruction.
(a) MOV R3,#55 (b) DEC R3 (c) DJNZ R2 target
(d) LJMP (e) SJMP (f) NOP (g) MUL AB
Solution:

Machine cycles		Time to execute
(a)	1	$1 \times 1\mu s = 1\mu s$
(b)	1	$1 \times 1\mu s = 1\mu s$
(c)	2	$2 \times 1\mu s = 2\mu s$
(d)	2	$2 \times 1\mu s = 2\mu s$
(e)	2	$2 \times 1\mu s = 2\mu s$
(f)	1	$1 \times 1\mu s = 1\mu s$
(g)	4	$4 \times 1\mu s = 4\mu s$

TIME DELAY FOR VARIOUS 8051 CHIPS

Delay Calculation

Find the size of the delay in following program, if the crystal frequency is 12MHz

```
        MOV   A,#55H
AGAIN:  MOV   P1,A
        ACALL DELAY
        CPL   A
        SJMP  AGAIN
;---time delay-------
DELAY:  MOV   R3,#200
HERE:   DJNZ  R3,HERE
        RET
```

A simple way to short jump to itself in order to keep the microcontroller busy

```
HERE:  SJMP  HERE
```

We can use the following:

```
       SJMP  $
```

Solution:

```
                              Machine cycle
DELAY:  MOV   R3,#200               1
HERE:   DJNZ  R3,HERE               2
        RET                         2
```

TIME DELAY FOR VARIOUS 8051 CHIPS

Delay Calculation

Find the size of the delay in following program, if the crystal frequency is 12MHz

```
        MOV   A,#55H
AGAIN:  MOV   P1,A
        ACALL DELAY
        CPL   A
        SJMP  AGAIN
;---time delay-------
DELAY:  MOV   R3,#200
HERE:   DJNZ  R3,HERE
        RET
```

A simple way to short jump to itself in order to keep the microcontroller busy

```
HERE:  SJMP  HERE
```

We can use the following:

```
       SJMP  $
```

Solution:

```
                                  Machine cycle
DELAY:  MOV   R3,#200                 1
HERE:   DJNZ  R3,HERE                 2
        RET                           2
```

Therefore, [(200x2)+1+2]x1µs ＝403µs

Find the size of the delay in following program, if the crystal frequency is 12MHz.

		Machine Cycle
DELAY:	MOV R3,#250	1
HERE:	NOP	1
	NOP	1
	NOP	1
	NOP	1
	DJNZ R3,HERE	2
	RET	2

Solution:

Find the size of the delay in following program, if the crystal frequency is 12MHz.

		Machine Cycle
DELAY:	MOV R3,#250	1
HERE:	NOP	1
	NOP	1
	NOP	1
	NOP	1
	DJNZ R3,HERE	2
	RET	2

Solution:

The time delay inside HERE loop is

[250(1+1+1+1+2)]x1μs = 1500μs.

Adding the two instructions outside loop we have 1500μs + 3 x 1 μs = 1503μs

Find the size of the delay in following program, if the crystal frequency is 12MHz.

	Machine Cycle
DELAY: MOV R2,#200	1
AGAIN: MOV R3,#250	1
HERE: NOP	1
NOP	1
DJNZ R3,HERE	2
DJNZ R2,AGAIN	2
RET	2

Solution:

Find the size of the delay in following program, if the crystal frequency is 12MHz.

	Machine Cycle
DELAY: MOV R2,#200	1
AGAIN: MOV R3,#250	1
HERE: NOP	1
NOP	1
DJNZ R3,HERE	2
DJNZ R2,AGAIN	2
RET	2

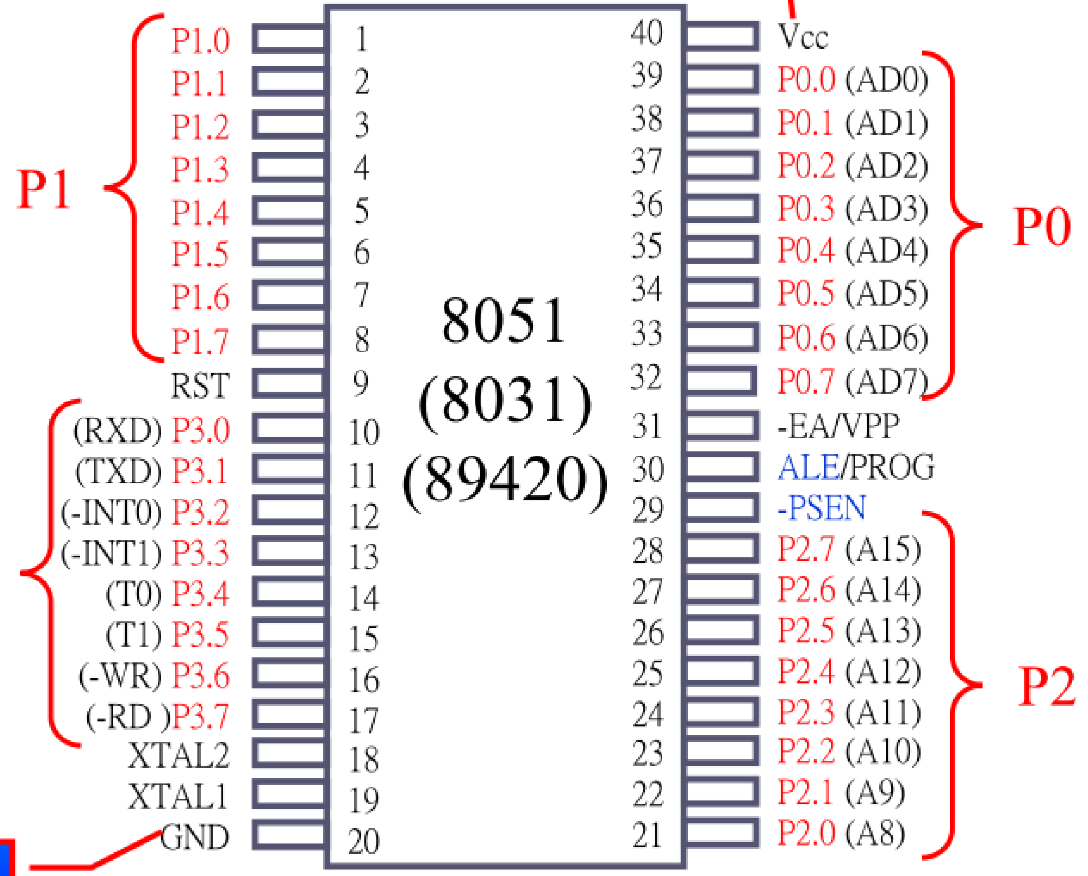
Solution:
For HERE loop, we have (4x250)x1µs＝1000µs.
For AGAIN loop repeats HERE loop 200 times, so we have 200x1000µs＝200,000µs. But "MOV R3,#250" and "DJNZ R2,AGAIN" at the start and end of the AGAIN loop add (3x200x1)=600µs.
As a result, we have 200,000+600=200600µs.
Notice in nested loop, as in all other time delay loops, the time is approximate since we have ignored the first and last instructions in the subroutine.

I/O Port Programming

8051 Pin Diagram

A total of 32 pins are set aside for the four ports P0, P1, P2, P3, where each port takes 8 pins

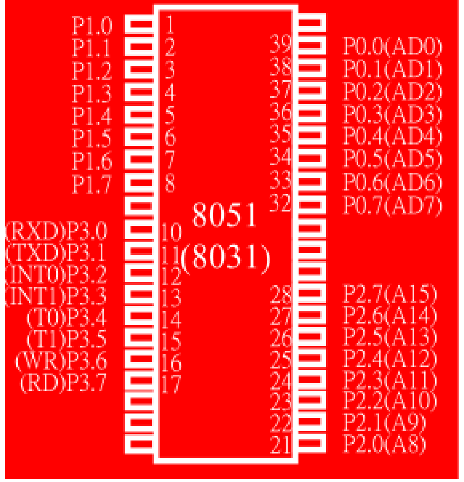
Provides +5V supply voltage to the chip



Grond

I/O PROGRAMMING

I/O Port Pins



- The four 8-bit I/O ports P0, P1, P2 and P3 each uses 8 pins
- All the ports upon RESET are configured as input, ready to be used as input ports
  - When the first 0 is written to a port, it becomes an output
  - To reconfigure it as an input, a 1 must be sent to the port
    - To use any of these ports as an input port, it must be programmed