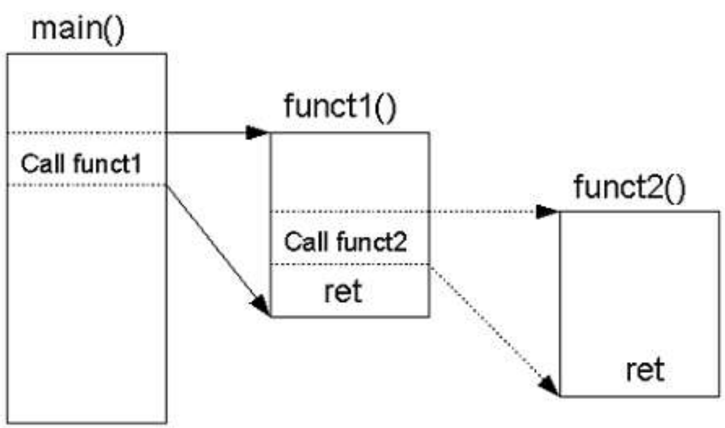
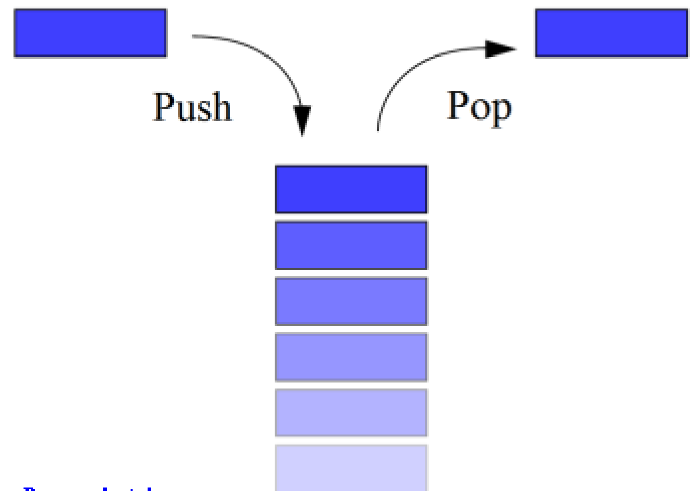
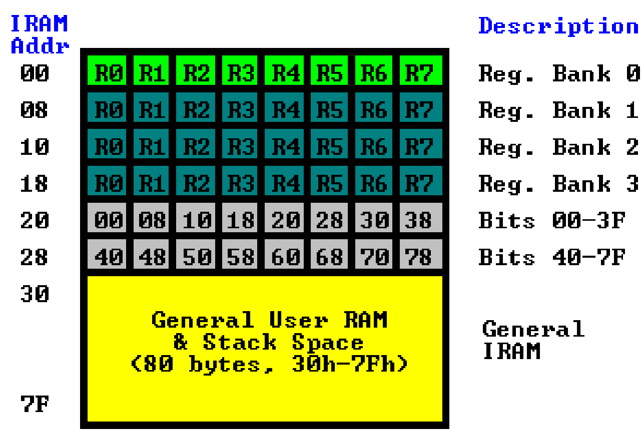
Subroutines, the stack, memory addressing







Overview

- Subroutines
  - Passing parameters
  - Estimating runtime of program
  - How it effects the stack
- Stack
  - Pushing and popping values
- Addressing modes
  - Ways to specify values or memory locations

Subroutines

- Subroutines
  - Code that executes a specific task
  - Returns back to instruction after subroutine finishes
  - Examples:
    - Subroutine that displays number in binary on LEDs
    - Subroutine that pauses for X0 millisecond
  - Stack pointer (SP) tracks return address
    - SP defaults to 07h
    - Best to set it to higher address to avoid register banks

Subroutines

- CALL *address16*
  - Calls subroutine specified by address
    - CALL → translated to ACALL (2 bytes) or LCALL (3 bytes)
    - Push PC + 2/3 (2 for ACALL, 3 for LCALL) onto stack
    - PC set to address of CALL operand
    - Increments Stack Pointer (SP) by 2
- RET
  - All subroutines must end with RET
    - PC set to top two-bytes from stack
    - Decrements Stack Pointer (SP) by 2

LED display subroutine

```
Start:
        MOV SP, #2Fh     ; Move SP away from registers, etc
        MOV R0, #13
        CALL DisplayR0   ; Display binary for 13 on LEDs
Loop:
        NOP
        JMP LOOP

; Subroutine that displays the value R0 on the LEDs.
; Handles complementing bits so binary 1 = lighted LED.
DisplayR0:
        MOV A, R0    ; Copy R0 to the A since CPL only works on A
        CPL A        ; Invert the value to make 1 = ON
        MOV P0, A    ; Copy to the LEDs
        RET
END
```

Estimating run-time

- 8052 clock speed
  - 11.0592 Mhz (11,059,200 clock ticks per second)
  - 12 clock ticks per machine cycle
  - How much time does this take?

```
; Triply nested loop to burn cycles
; Number of loops 1 * 12 * 255
      MOV R2, #1        ; for (i = 0; i < 1; i++)
Top:                    ; {
      MOV R1, #12       ;    for (j = 0; j < 12; j++)
Mid:                    ;    {
      MOV R0, #255      ;       for (k = 0; k < 255; k++)
Loop: NOP               ;       {
      DJNZ R0, Loop     ;       }
      DJNZ R1, Mid      ;    }
      DJNZ R2, Top      ; }
```

Estimating run-time

```
; Triply nested loop to burn cycles
; Number of loops 1 * 12 * 255
      MOV R2, #1         ; 1 cycle  * 1 time
Top:                     ;
      MOV R1, #12        ; 1 cycle  * 1 time
Mid:                     ;
      MOV R0, #255       ; 1 cycle  * (1 * 12 times)
Loop: NOP                ; 1 cycle  * (1 * 12 * 255 times)
      DJNZ R0, Loop      ; 2 cycles * (1 * 12 * 255 times)
      DJNZ R1, Mid       ; 2 cycles * (1 * 12 times)
      DJNZ R2, Top       ; 2 cycles * 1 time
```

```
     1
     1
    12
  3060
  6120
    24
+    2
  9220 cycles * (1 second/11059200 ticks) * 12 ticks/cycle
= 0.01000434 seconds
```

Passing parameters

- Everything really a global variable
- Passing parameters to subroutine
  - Agree where input parameters are put
    - e.g. R0 in previous example
- Returning value from subroutine
  - Agree on where output goes
    - e.g. Leave calculation in accumulator

Flexible delay subroutine

- Goal: subroutine burn N x 0.01s
- How to pass input N?
  - Dedicate 1 byte of our 256 bytes for parameter
  - Give it a friendly name with EQU
  - Allows delay of 0.00s - 2.55s
- How many loops?
  - Triply nested, outer loop use parameter
  - Dedicate another two bytes for inner counters

Flexible delay subroutine

```
DELAY_AMOUNT EQU 30h
DELAY_TEMP0  EQU 31h
DELAY_TEMP1  EQU 32h

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Subroutine to burn cycles
; DELAY_AMOUNT - hundredths of a second to pause
Delay:
       MOV DELAY_TEMP1, #12
DelayMid:
       MOV DELAY_TEMP0, #255
DelayLoop:
       NOP
       DJNZ DELAY_TEMP0, DelayLoop
       DJNZ DELAY_TEMP1, DelayMid
       DJNZ DELAY_AMOUNT, Delay
       RET
```

Using the delay subroutine

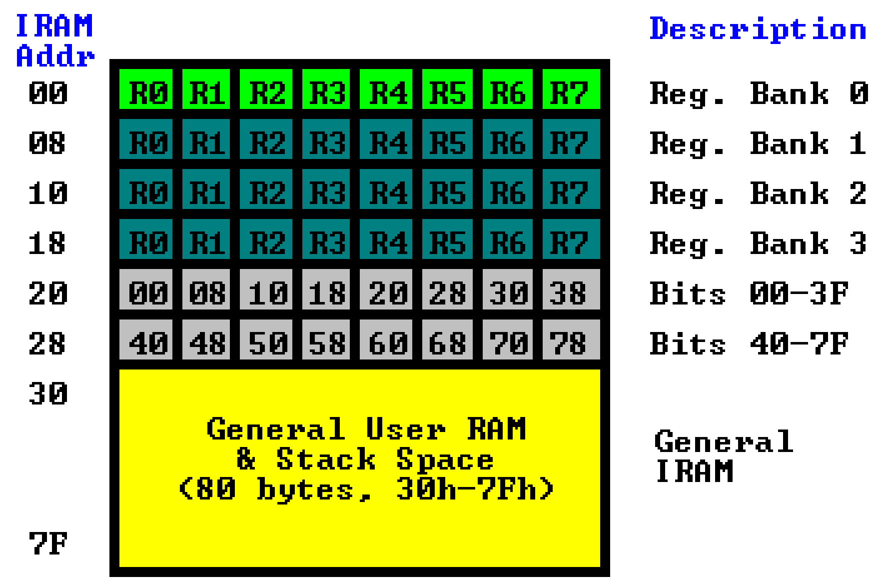
```
; Example showing usage of the flexible delay subroutine.
; Main program toggles LEDs off/on every 60 seconds.

Start:
        MOV A, #00h
Loop:
        MOV P0, A
        MOV R0, #60

Minute:
        MOV DELAY_AMOUNT, #100
        CALL Delay
        DJNZ R0, Minute
        CPL A
        JMP Loop
...
```

Stack pointer

- Stack pointer (SP)
  - SFR at memory location 81h
  - Indicates next value to be taken from stack
  - Initialized to 07h
  - Manipulated by:
    - ACALL, LCALL, RET
    - PUSH, POP
    - RETI



Uses for the stack

- Calling subroutines
  - CALL → ACALL, LCALL
    - Push two byte address of return location on RET
    - Location is current Program Counter (PC) + size of CALL instruction (2 bytes ACALL, 3 bytes LCALL)
  - RET
    - Pop two bytes, load into Program Counter (PC)
    - Causes execution to resume after CALL
  - Subroutines can call other subroutines
    - Stack grows by 2 bytes with each CALL

Uses for the stack

- Saving and restoring data
  - Useful in Interrupt Service Routines (ISR)
    - e.g. arrival of data on serial port
  - Normal program flow suspended to run ISR
  - ISR must protect:
    - Accumulator (ACC)
    - Data Pointer SFRs (DPH/DPL)
    - Program Status Word SFR (PSW)
    - B Register (B)
    - R Registers (R0-R7)

Pushing and popping

- PUSH *direct*
  - Increments Stack Pointer (SP) by 1
  - Then pushes value at *direct* onto stack
  - 2 bytes, 2 cycles
- POP *direct*
  - Pops last value from the stack, puts into *direct*
  - Then decrements Stack Pointer (SP) by 1
  - 2 bytes, 2 cycles

PUSH / POP Example

```
; Example interrupt service routine

InterruptHandler:
        ; Save state of PSW and ACC
        PUSH ACC
        PUSH PSW
        ...
        MOV A,#00     ; Use accumulator for something
        ...
        ; Restore PSW and ACC
        POP PSW
        POP ACC

        ; Return from ISR
        RETI
```

Addressing modes

- 8052 memory addressing modes
  - Immediate MOV A, #20h
  - Direct MOV A, 30h
  - Indirect MOV A, @R0
  - External direct MOVX A, @DPTR
  - External indirect MOVX A, @R0
  - Code indirect MOVC A, @A+DPTR

Addressing modes

- Immediate addressing
  - e.g. MOV A, #20h
  - Value to be stored follows opcode
  - Specifying a literal value in decimal, octal, hex, or binary
  - Very fast, not very flexible

Addressing modes

- Direct addressing
  - e.g. MOV A, 30h
  - Value to be stored is obtained by retrieving from specified memory address
  - Lack of # symbol differentiates from immediate
  - Fast, value stored in internal RAM
  - 00h-7Fh refers to RAM (128 bytes)
  - 80h-FFh refer to Special Function Registers (SFRs)

Addressing modes

- Indirect addressing
  - e.g. MOV A, @R0
  - Read the value of R0, obtain value at memory pointed to by R0
  - Only way to get to the upper 128 bytes on 8052
  - Indirect never refers to a SFR
  - Example:

    MOV R0, #40h

    MOV A, @R0

    Register R0 holds value 40h, load accumulator with whatever is stored at RAM address 40h

Summary

- Subroutines
  - Passing parameters
  - Estimating runtime
- Stack
  - Used when we call subroutines
  - Can manually PUSH and POP values
    - We'll use in Interrupt Service Routines (ISRs)
- Addressing modes
  - Immediate, direct, indirect