Subroutines
Interupt

Subroutines

```
Main:       ...
            acall sublabel
            ...
            ...
sublabel:   ...
            ...
            ret
```

call to the subroutine

the subroutine

Initializing Stack Pointer

- SP is initialized to 07 after reset.(Same address as R7)
- With each push operation 1st , pc is increased
- When using subroutines, the stack will be used to store the PC, so it is very important to initialize the stack pointer. Location 2Fh is often used.

```
mov SP, #2Fh
```

Subroutine - Example

```
square:  push b
         mov  b,a
         mul  ab
         pop  b
         ret
```

- 8 byte and 11 machine cycle

```
square: inc a
         movc a,@a+pc
         ret
table:  db 0,1,4,9,16,25,36,49,64,81
```

- 13 byte and 5 machine cycle

Subroutine - another example

```
; Program to compute square root of value on Port 3
; (bits 3-0) and output on Port 1.
        org 0
        ljmp Main

Main:   mov P3, #0xFF    ; Port 3 is an input
loop:   mov a, P3
        anl a, #0x0F     ; Clear bits 7..4 of A
        lcall sqrt
        mov P1, a
        sjmp loop

sqrt:   inc a
        movc a, @a + PC
        ret

Sqrs:   db  0,1,1,1,2,2,2,2,2,3,3,3,3,3,3,3
        end
```

reset service

main program

subroutine

data

Why Subroutines?

- Subroutines allow us to have "structured" assembly language programs.
- This is useful for breaking a large design into manageable parts.
- It saves code space when subroutines can be called many times in the same program.

example of delay

```
      mov a,#0aah
Back1:mov p0,a
      lcall delay1
      cpl a
      sjmp back1
Delay1:mov r0,#0ffh;1cycle
Here: djnz r0,here ;2cycle
      ret          ;2cycle
      end
```

Delay=1+255*2+2=513 cycle

```
Delay2:
      mov r6,#0ffh
back1: mov r7,#0ffh ;1cycle
Here:  djnz r7,here ;2cycle
       djnz
   r6,back1;2cycle
   ret          ;2cycle
      end
```

Delay=1+(1+255*2+2)*255+2
=130818 machine cycle

Long delay Example

```
GREEN_LED:      equ   P1.6
                org   ooh
                ljmp  Main

                org   100h
Main:           clr   GREEN_LED
Again:          acall Delay
                cpl   GREEN_LED
                sjmp  Again

Delay:          mov   R7, #02
Loop1:          mov   R6, #00h
Loop0:          mov   R5, #00h
                djnz  R5, $
                djnz  R6, Loop0
                djnz  R7, Loop1
                ret
```

reset service

main program

subroutine

Example

```
; Move string from code memory to RAM
            org 0
            mov dptr,#string
            mov r0,#10h
Loop1:      clr a
            movc a,@a+dptr
            jz stop
            mov @r0,a
            inc dptr
            inc r0
            sjmp loop1
Stop:       sjmp stop

; on-chip code memory used for string
            org 18h
String:     db 'this is a string',0
            end
```

Example

```
; p0:input  p1:output
          mov a,#0ffh
          mov p0,a
back:     mov a,p0
          mov p1,a
          sjmp back

          setb p1.2
          mov a,#45h      ;data
Again:    jnb p1.2,again ;wait for data
  request
          mov p0,a        ;enable strobe
          setb p2.3
          clr p2.3
```

Example

```
; duty cycle 50%
back:   cpl p1.2
        acall delay
        sjmp back
```

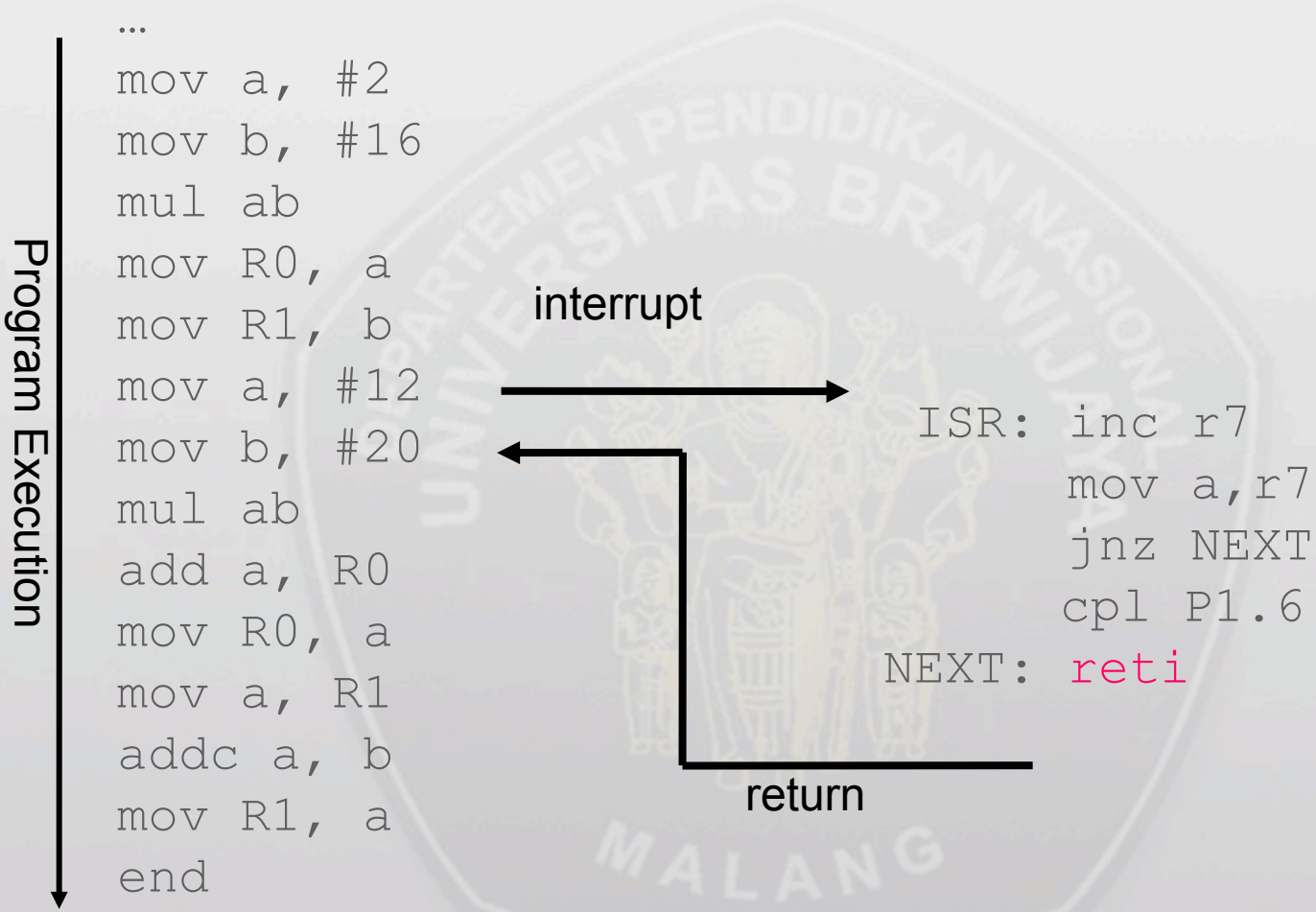
```
back:     setb p1.2
        acall delay
        Clr p1.2
        acall delay
        sjmp back
```

Example

```
; duty cycle 66%
back:    setb p1.2
         acall delay
         acall delay
         Clr p1.2
         acall delay
         sjmp back
```

8051 timer

Interrupts



Interrupt Sources

- Original 8051 has 5 sources of interrupts
  - Timer 0 overflow
  - Timer 1 overflow
  - External Interrupt 0
  - External Interrupt 1
  - Serial Port events (buffer full, buffer empty, etc)
- Enhanced version has 22 sources
  - More timers, programmable counter array, ADC, more external interrupts, another serial port (UART)

Interrupt Process

If interrupt event occurs AND interrupt flag for that event is enabled, AND interrupts are enabled, then:

1. Current PC is pushed on stack.
2. Program execution continues at the interrupt vector address for that interrupt.
3. When a RETI instruction is encountered, the PC is popped from the stack and program execution resumes where it left off.

Interrupt Priorities

- What if two interrupt sources interrupt at the same time?
- The interrupt with the highest PRIORITY gets serviced first.
- All interrupts have a default priority order.
- Priority can also be set to “high” or “low”.

Interrupt SFRs

Figure 12.9. IE: Interrupt Enable

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
EA	IEGF0	ET2	ES0	ET1	EX1	ET0	EX0	00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address:
							(bit addressable)	0xA8

Interrupt enables for the 5 original 8051 interrupts:
Timer 2
Serial (UART0)
Timer 1
External 1
Timer 0
External 0

Global Interrupt Enable – must be set to 1 for any interrupt to be enabled

1 = Enable
0 = Disable

Interrupt Vectors

Each interrupt has a specific place in code memory where program execution (interrupt service routine) begins.

```
External Interrupt 0:    0003h
Timer 0 overflow:        000Bh
External Interrupt 1:    0013h
Timer 1 overflow:        001Bh
Serial :                  0023h
Timer 2 overflow(8052+) 002bh
```

Note: that there are only 8 memory locations between vectors.

Interrupt Vectors

To avoid overlapping Interrupt Service routines, it is common to put JUMP instructions at the vector address. This is similar to the reset vector.

```
        org  009B           ; at EX7 vector
        ljmp EX7ISR
        cseg at 0x100       ; at Main program
Main:   ...                 ; Main program
        ...
EX7ISR:...                  ; Interrupt service routine
        ...                 ; Can go after main program
        reti                ; and subroutines.
```

Example Interrupt Service Routine

```
;EX7 ISR to blink the LED 5 times.
;Modifies R0, R5-R7, bank 3.
;-----------------------------------------------
   ISRBLK:   push PSW              ;save state of status word
             mov PSW,#18h          ;select register bank 3
             mov R0, #10           ;initialize counter
    Loop2:   mov R7, #02h          ;delay a while
    Loop1:   mov R6, #00h
    Loop0:   mov R5, #00h
             djnz R5, $
             djnz R6, Loop0
             djnz R7, Loop1
             cpl P1.6              ;complement LED value
             djnz R0, Loop2        ;go on then off 10 times
             pop PSW
             reti
```