Chapter 09

Programming in Assembly

Lesson 03

Programming Approach for Main and Interrupt Service Routines in 8051

Program Approach for programming Main Program Instructions

Main program initial instructions

- Instruction to define the register bank being used
- Instruction to define stack top
- Remember default Settings for the A, B, PSW and DPTR on Reset and change if required

Instruction to write the control bits and mode bits

- Instruction to write into the required SFRs
- Keep default settings in view
- Write into TCON, TMOD, TH0, TL0, TH1, TL0, TH2, TL2, RCAP2H, RCAP2L if timers being used
- Write into SCON if serial interface is being used.

Setting Interrupt priorities

- Instructions to define the Interrupt priorities
- This is to minimize the worst case interrupt latencies of the possible sources of interrupts in the system
- Set or reset the bits in the IP SFR

Setting Interrupt priorities for Latency control

- Latency is the waiting period in the response to an interrupt
- Latency when multiple devices in the MCU used
- Interrupts from each takes place at the different instances
- Some ISRs have lower latency (shorter code) and some high (longer code)
-

Latency Control

- Required because some interrupts have to be responded by executing ISR fast compared to other

Enabling (un-mask) selected sources of the interrupts

- Instruction to do this is the set or reset bits in the IE SFR
- First set or reset the EA (enable all) bit in the IE SFR
- SETB 0AFH is the instruction
- Then the corresponding source interrupt enabled (unmasked)

Use of the Timer, Serial Port and INT pin ISRs

- The interrupt service routines written for the timers, serial port and external interrupts

Program Approach for programming ISR Instructions

Execution Interval of an ISR

- **M**ultiple interrupts from same source (for example timer overflow)
- There can be interrupts from different sources at different instances
- ISR execution time must be less than the interval of next interrupt from the same source

Disabling and Enabling of an interrupt

- When one routine or ISR runs there might be need to disable all or selected source of interrupts
- Disabling of a source of interrupt for a give interval increases the latency of that interrupt
- Programming approach such that the latency of higher priority interrupt doe not increase
- significantly and no source of interrupt misses the deadline for the service

Resetting of Interrupt flags if not done automatically by hardware

- Some flags such as TI and RI don’t reset automatically.
- Use instructions to reset those in case present service routine for service of those interrupts

Polling

- Write instructions to check occurrences of an interrupt in a interrupt sources group if the ISR is for servicing more than one interrupt source

.

Push Instruction

- A routine on interrupt must use separate set of registers in another register bank in case the currently use set of registers in bank of 8 registers is required to be protected in the interrupted main or routine program
- Therefore PSW needs to be pushed onto stack if another register bank is required in an ISR

Push PSW Instruction

- Write the instructions, if feasible, without the need for using the PSW
- If required then Push the PSW at the beginning of the ISR
- This done when the ISR uses any of the flags CY, AC, OV or P and the currently set flags are required to be protected in the interrupted main program or routine.
- PUSH 0D0H [D0H is address of PSW]

Push Instruction

- Write the instructions, if feasible, without the need for using the A register. If not feasible, then PUSH A also
- PUSH 0E0H
- For example, use the logic operations between direct and data in the ISR in place of logic operations on A.

Push Instruction

- Write the instructions, if feasible, without the need for using those SFRs and internal memory addresses which are required to be saved in the main program or routine
- If not feasible, then PUSH those SFRs or internal memory RAM addresses
- Use PUSH *direct*

Use of Pointers R0 and R1 and bank registers

- Use pointer registers and registers in the register bank as much as possible, because instructions using registers *shorter* and taking *smaller number of instruction cycles*
- For example, use MOV using @Ri; CJNE Rn, #data8, Rel; and CJNE @Ri, #data8 , Rel;

Use of Combined Instructions

- Use combined instruction
- DJNZ Rn, Rel;

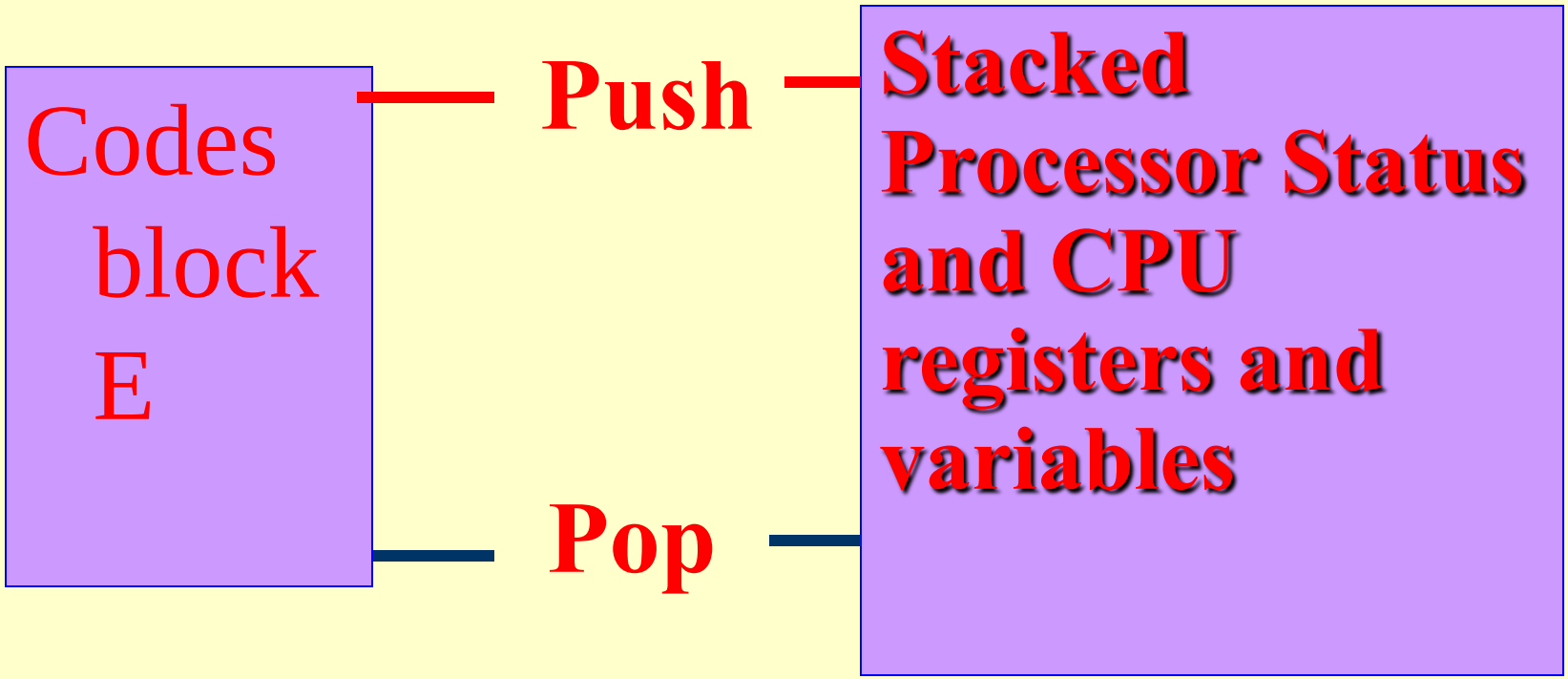
POP Instructions

- POP all registers and memory addresses before the return from the ISR that were pushed at the beginning of ISR
- POPs must be in last in first out way
- For example, if *PSW* first and then *A* were pushed then POP *A* first then *PSW*
-

Interrupts Enable

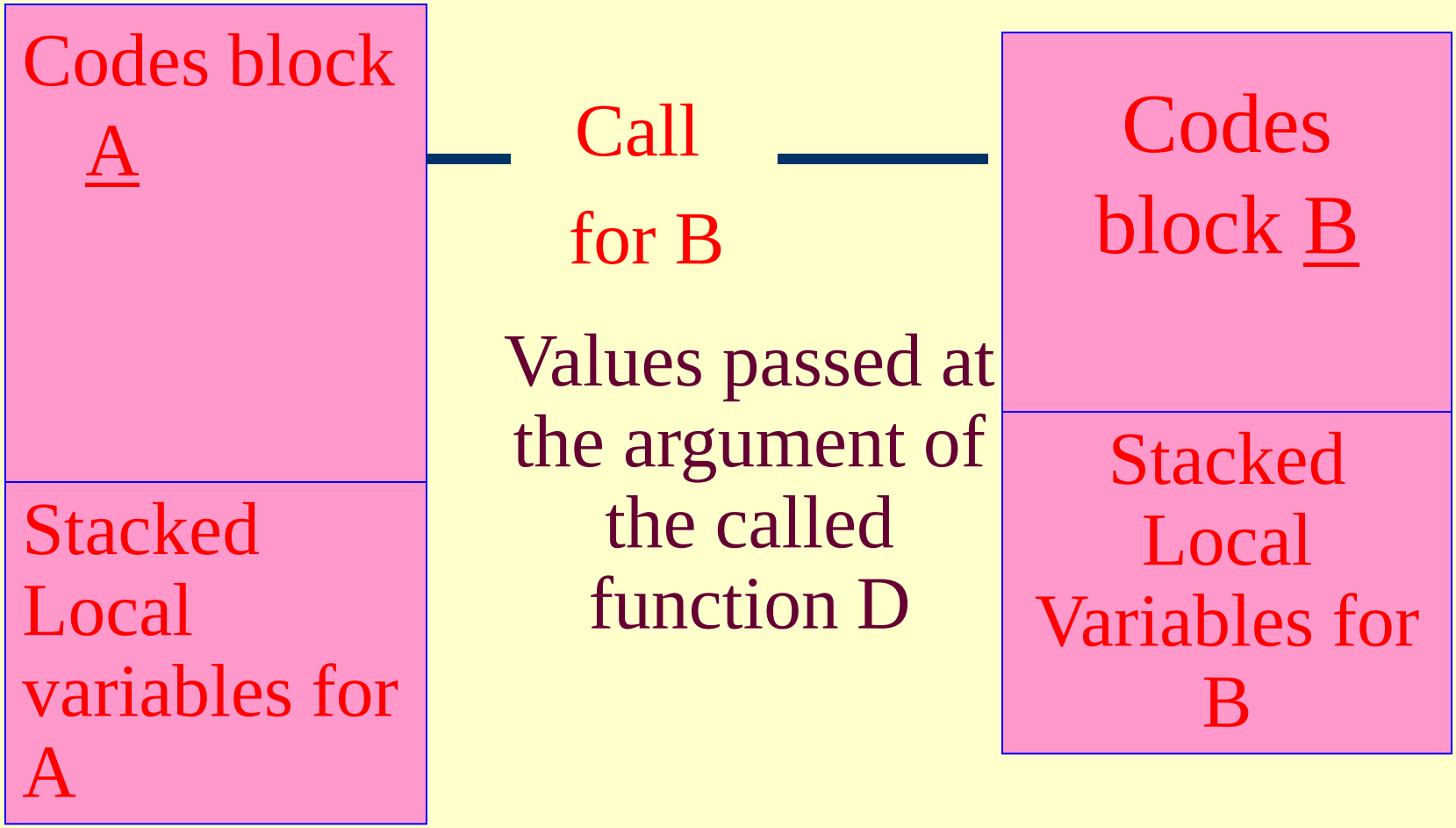
- Enable all or selected interrupts, in case they were disabled at the beginning of the ISR
- Use in 8051 the instruction for SETB 0x0AFH. [IE^7= 1; for enabling interrupts handling of un-masked interrupts.]

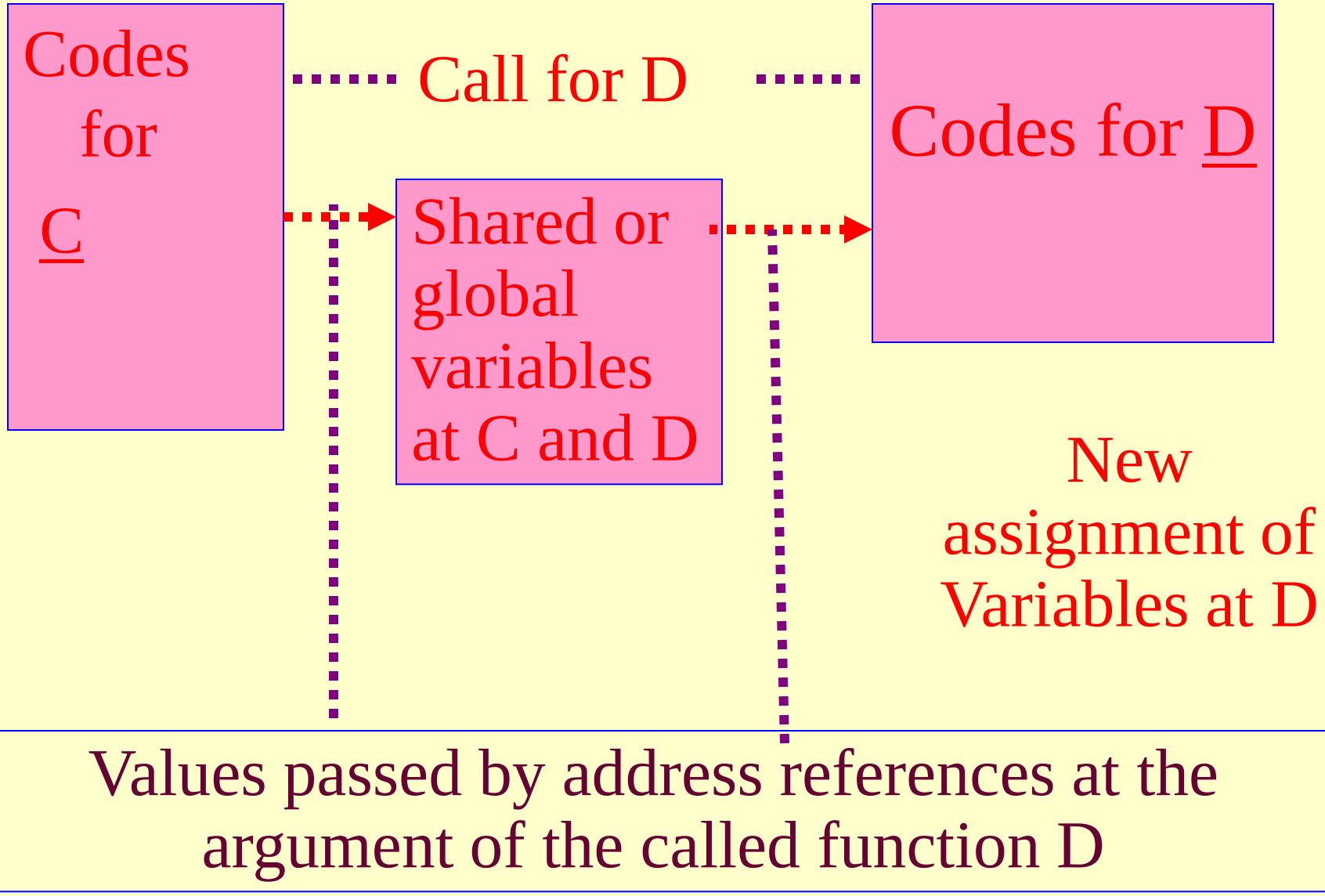
Stack



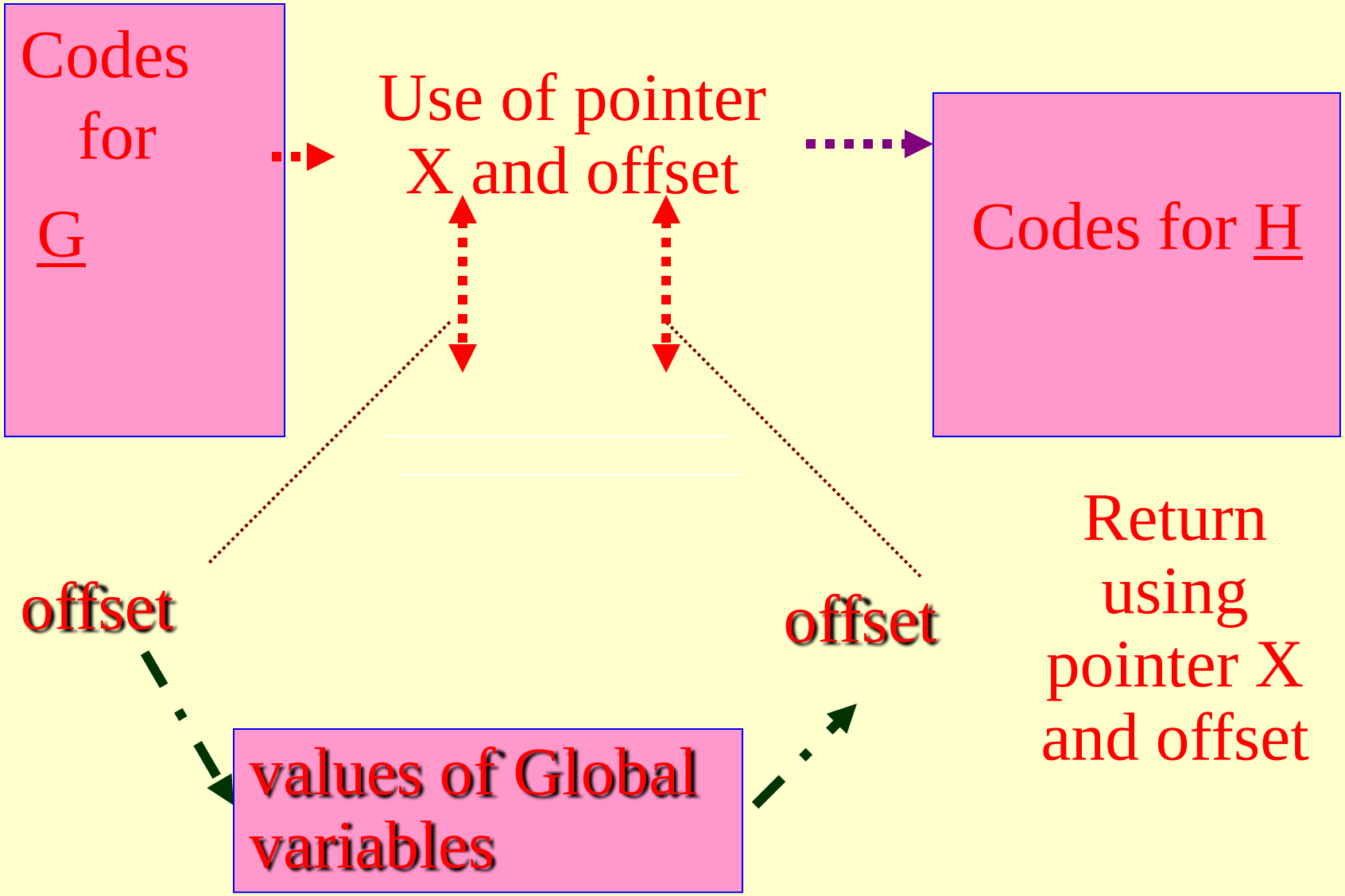
Used in LIFO mode

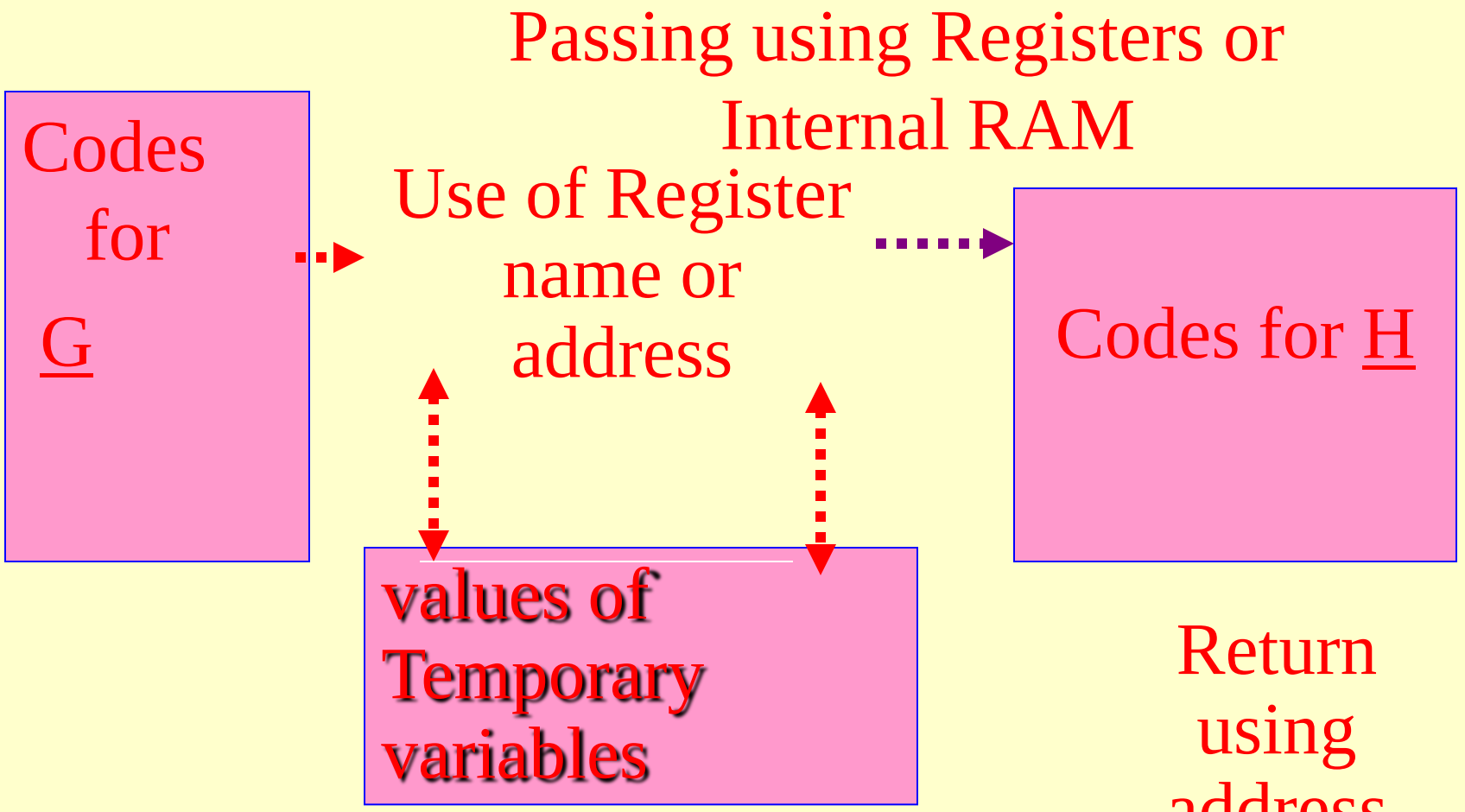
Calling a routine from other routine


Codes block A
Stacked Local variables for A
Call for B
Values passed at the argument of the called function D
Codes block B
Stacked Local Variables for B


Codes for C
Call for D
Codes for D
Shared or global variables at C and D
New assignment of Variables at D
Values passed by address references at the argument of the called function D

Passing by reference




Passing using Registers or Internal RAM
Codes for G
Use of Register name or address
Codes for H
values of Temporary variables
Return using address or register name

Reentrant Function

Reentrant Function

- When interrupted in-between, it

Enters into identical state of variables and CPU registers on return from another

Reentrant Function

- No shared data problem
- Always gets the values on pass by values, not by reference
- Always call reentrant function only
- Always uses local variables only

Summary

We learnt

- Program Initial Instructions in Main
- Program Instructions in ISR
- Use Pointers
- Use Register Banks
- Use combine Instructions

We learnt

- Status and CPU registers saved on stack
- Local variables stacked and passed to called routine by values
- Shared variables passed to called routine by reference to address

End of Lesson 03 on

Programming Approach for Main and Interrupt Service Routines in 8051