

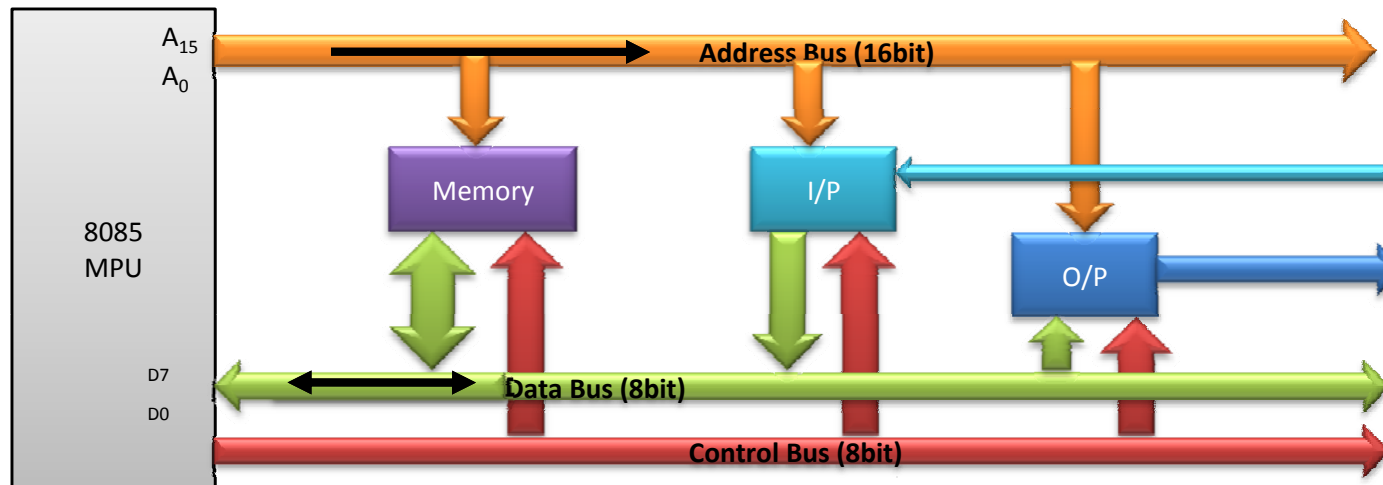
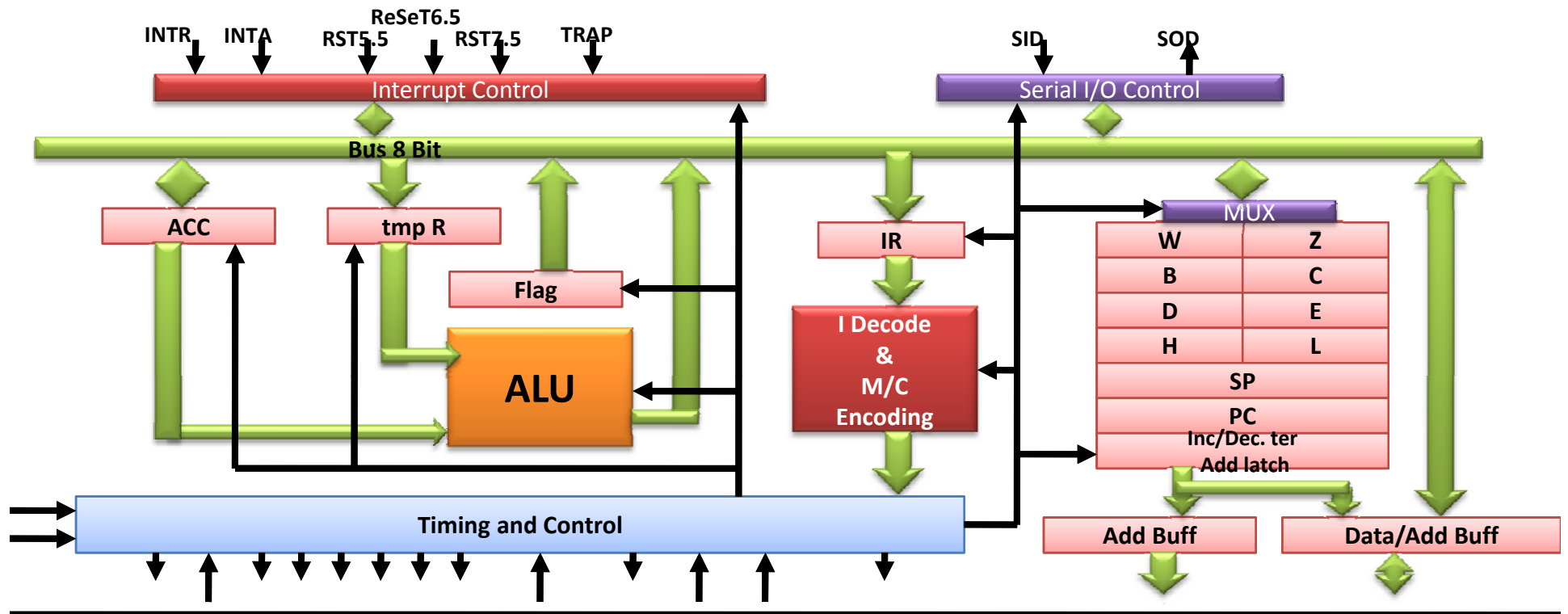
8085 Architecture & Its Assembly language programming

Dr A Sahu
Dept of Computer Science &
Engineering
IIT Guwahati

Outline

- 8085
 - Block diagram (Data Path)
 - Instruction Set of 8085
- Sample program of 8085
- Counter & Time Delay
- Stack and Sub Routine
- Assignment on 8085
- Introduction to 8086 and 30x86 architecture

8085 Microprocessor Architecture



Assumption

- RAM Memory is interfaced
- Instructions are stored in memory
- One I/O display port is interfaced to display data of ACC

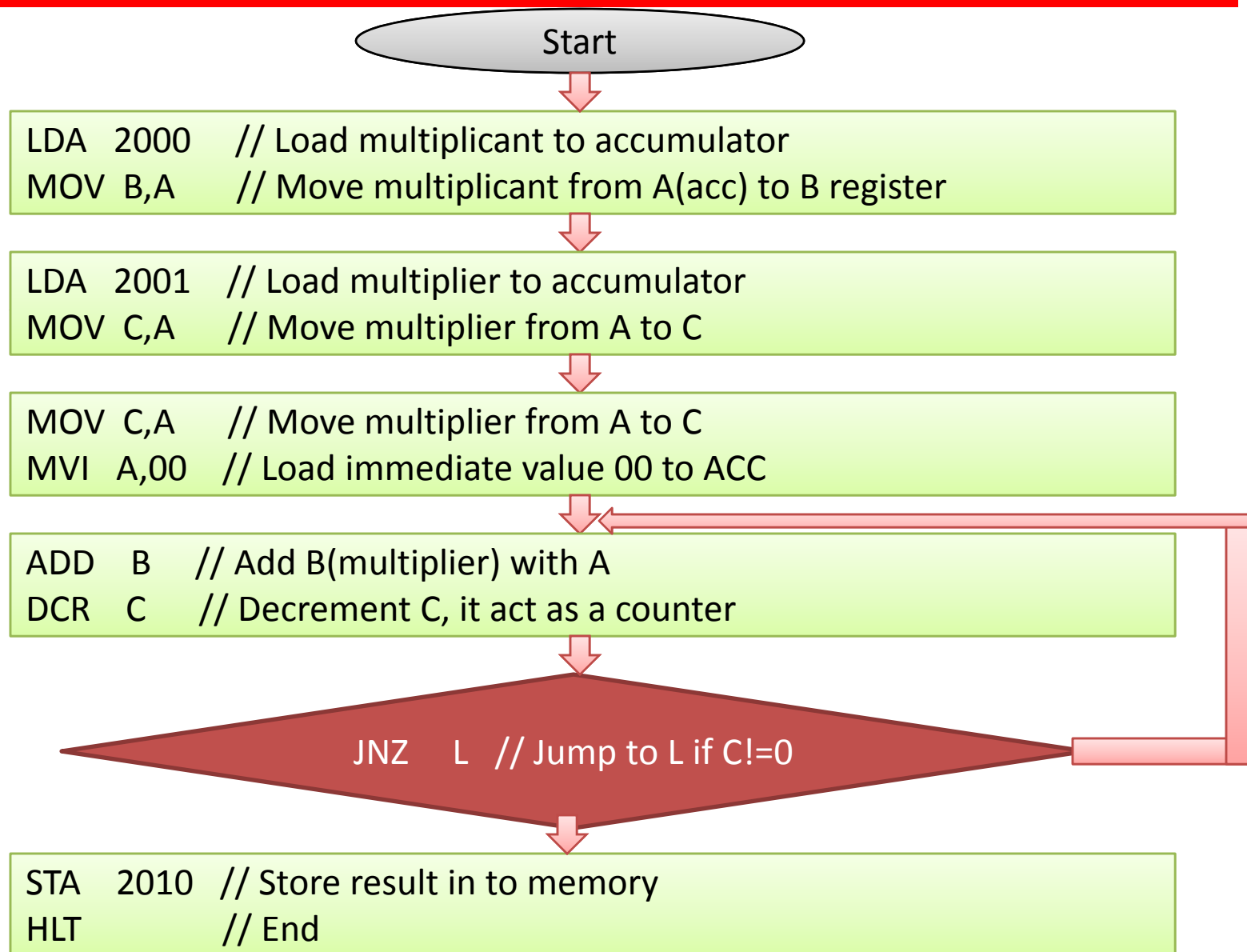
Simple Assembly Program

```
MVI  A, 24H    // load Reg ACC with 24H
MVI  B , 56H    // load Reg B with 56H
ADD  B         // ACC= ACC+B
OUT  01H       // Display ACC contents on port 01H
HALT          // End the program
```

Result: 7A (All are in Hex)

DAA operation for Decimal Adjust $A+6=10H$

Flowchart to multiply two number



Code to multiply two number

```
LDA 2000 // Load multiplicand to accumulator
MOV B,A  // Move multiplicand from A(acc) to B register
LDA 2001 // Load multiplier to accumulator
MOV C,A  // Move multiplier from A to C
MVI A,00 // Load immediate value 00 to a
L: ADD B  // Add B(multiplier) with A
DCR C    // Decrement C, it act as a counter
JNZ L    // Jump to L if C reaches 0
STA 2010 // Store result in to memory
HLT      // End
```

Delay of Instructions

- Performance/delay of each instruction

MVI C, FFH  7 T-State

LOOP: DCR C  4 T-State

JNZ LOOP  7/10 T-State

- Performance of other INS

ADD R  4 T-State







ADD M  7 T-State

CALL addr  18 T-State

- F=Fetch with 4 State, S=Fetch with 6 State,
R=Memory Read, W=Memory Write

Time Delay Loop

- Performance/delay of each instruction

MVI C, FFH	 	7 T-State
LOOP: DCR C		4 T-State
JNZ LOOP	  	7/10 T-State

- Time delay in loop

$$T_L = T \times \text{Loop T-States} \times N_{10}$$

where T = System clock period

N_{10} = Equiv. decimal value of count loaded to C

$$T_L = 0.5 \times 10^{-6} \times (14 \times 255) = 1.8 \text{ms (ignore 10 T-State)}$$

Time Delay: Nested Loop

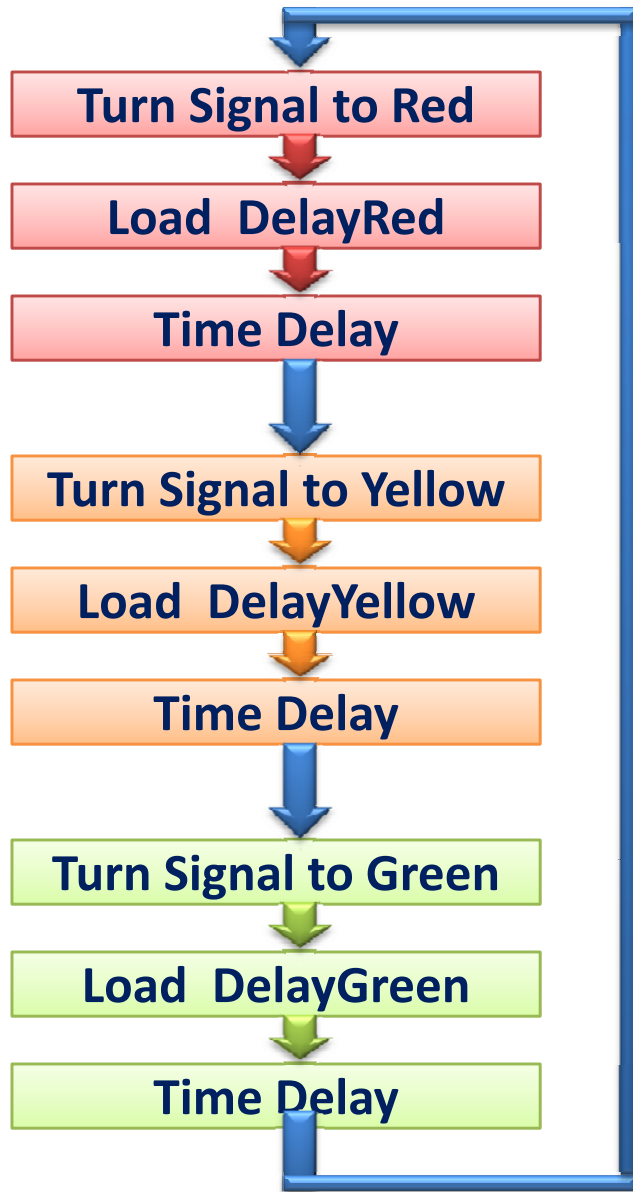
- Performance/delay of each instruction

MVI C, FFH	F	R	7 T-State
MVI D, FFH	F	R	7 T-State
LOOP1: DCR C	F		4 T-State
LOOP2: DCR D	F		4 T-State
JNZ LOOP2	F	R	7/10 T-State
JNZ LOOP1	F	R	7/10 T-State

- Time delay in Nested loop

$$T_{NL} = N1_{10} \times T \times (L1_TStates + L2_TStates \times N2_{10})$$

Traffic Light Control: Counter & Delay



LOOP: MVI A 01H
OUT 01H
LD B DELAY_RED
CALL DELAY

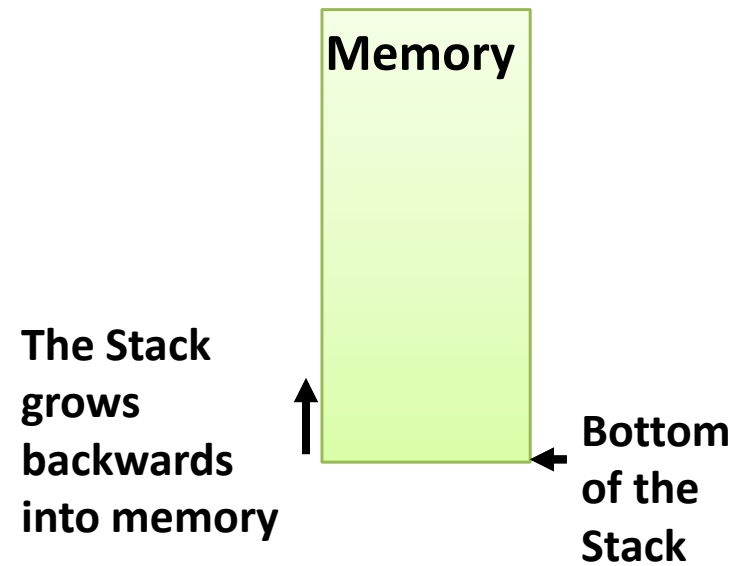
MVI A 02H
OUT 01H
LD B DELAY_YELLOW
CALL DELAY

MVI A 03H
OUT 01H
LD B DELAY_GREEN
CALL DELAY

JMP LOOP

Stack Pointer (SP) & Stack Memory

- The stack is an area of memory identified by the programmer for temporary storage of information.
- The stack is a LIFO structure.
- The stack normally grows backwards into memory.
 - Programmer can defines the bottom of the stack (**SP**) and the stack grows up into reducing address range.



Stack Memory

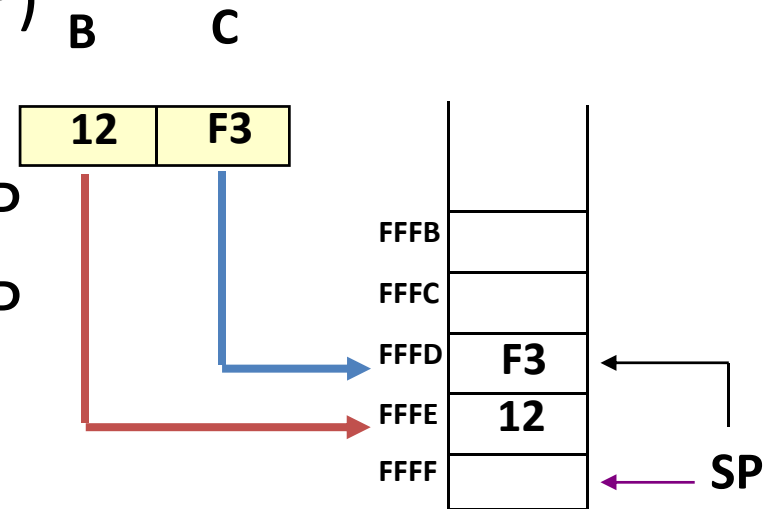
- Grows backwards into memory
- Better to place the bottom of the stack at the end of memory
- To keep it as far away from user programs as possible.
- Stack is defined by setting the SP (Stack Pointer) register.

LXI SP, FFFFH

- This sets SP to location FFFFH (end of memory for 8085).

Saving Information on the Stack

- Save information by PUSHing onto STACK
- Retrieved from STACK by POPing it off.
- PUSH and POP work with register pairs only.
- Example “PUSH B”
 - Decrement SP, Copy B to 0(SP)
 - Decrement SP, Copy C to 0(SP)
- Example “POP B”
 - Copy 0(SP) to C, Increment SP
 - Copy 0(SP) to B, Increment SP



Stack/LIFO use in CALL/RET

- Retrieve information back into its original location
 - The order of PUSHs and POPs must be opposite
- 8085 recognizes one additional register pair
 - PSW (Prog Status word) = ACC and Flag

Before any routine CALL do this

PUSH B

PUSH D

PUSH PSW

After RETURN from call do this

POP PSW

POP D

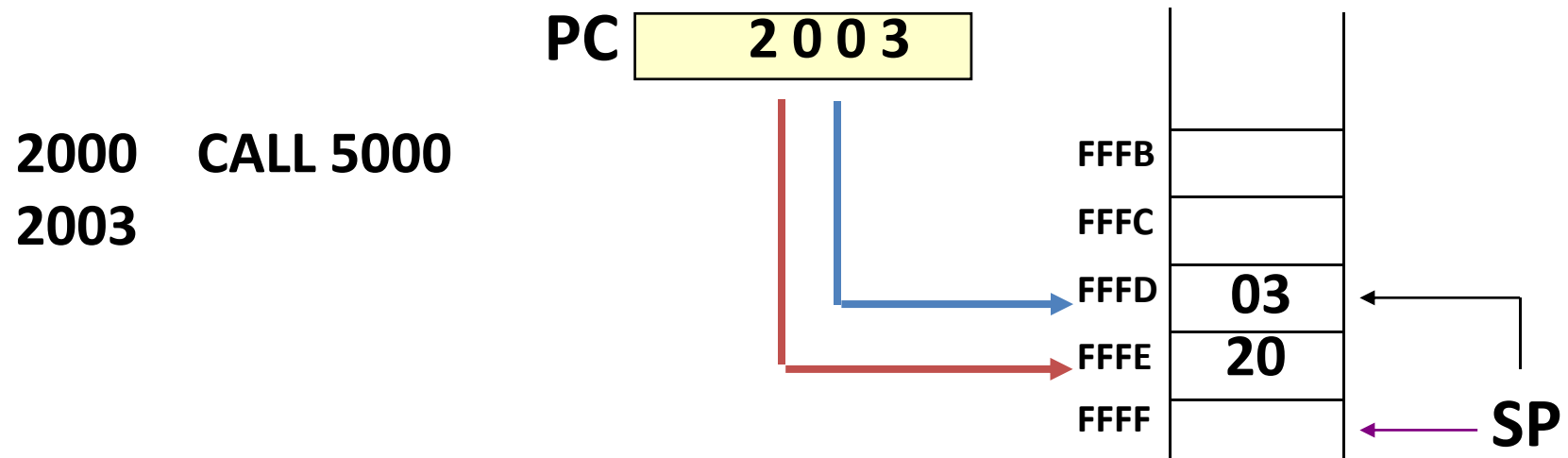
POP B

Subroutines

- A subroutine is a group of instructions
 - That is used repeatedly in different places of the program.
 - Rather than repeat the same instructions several times
 - It can be grouped into a subroutine and call from the different locations.
- Instructions for dealing with subroutines.
 - The CALL instruction is used to redirect program execution to the subroutine.
 - The RET instruction is used to return the execution to the calling routine.

CALL/RET Instruction

- You must set the SP correctly before using CALL
- CALL 5000H
 - Push the PC value onto the stack
 - Load PC with 16-bit address supplied CALL ins.
- RET : Load PC with stack top; POP PC



Call by References

- If SR performs operations on the contents of the registers
- These modifications will be transferred back to the calling program upon returning from a subroutine.
- If this is not desired, the SR should PUSH registers and POP on return.

Stack/LIFO use in CALL/RET

- Retrieve information back into its original location
 - The order of PUSHs and POPs must be opposite
- 8085 recognizes one additional register pair
 - PSW (Prog Status word) = ACC and Flag

Before any routine CALL do this

PUSH B

PUSH D

PUSH PSW

After RETURN from call do this

POP PWD

POP D

POP B

Factorial of a number

LXI SP, 27FFH // Initialize stack pointer

LDA 2200H // Get the number

CPI 02H // Check if number is greater than 1

JC LAST

MVI D, 00H // Load number as a result

MOV E, A

DCR A

MOV C,A // Load counter one less than number

CALL FACTO // Call subroutine FACTO

XCHG // Get the result in HL // HL with DE

SHLD 2201H // Store result // store HL at 0(16bit)

JMP END

LAST: LXI H, 0001H // Store result = 01

END: **SHLD 2201H**

HLT

Sub Routine for FACTORIAL

FACTO:LXI H, 0000H

MOV B, C // Load counter

BACK: **DAD D // double add ; HL=HL+DE**

DCR B

JNZ BACK //Multiply by successive addition

XCHG // Store result in DE // HL with DE

DCR C // Decrement counter

CNZ FACTO // Call subroutine FACTO

RET // Return to main program

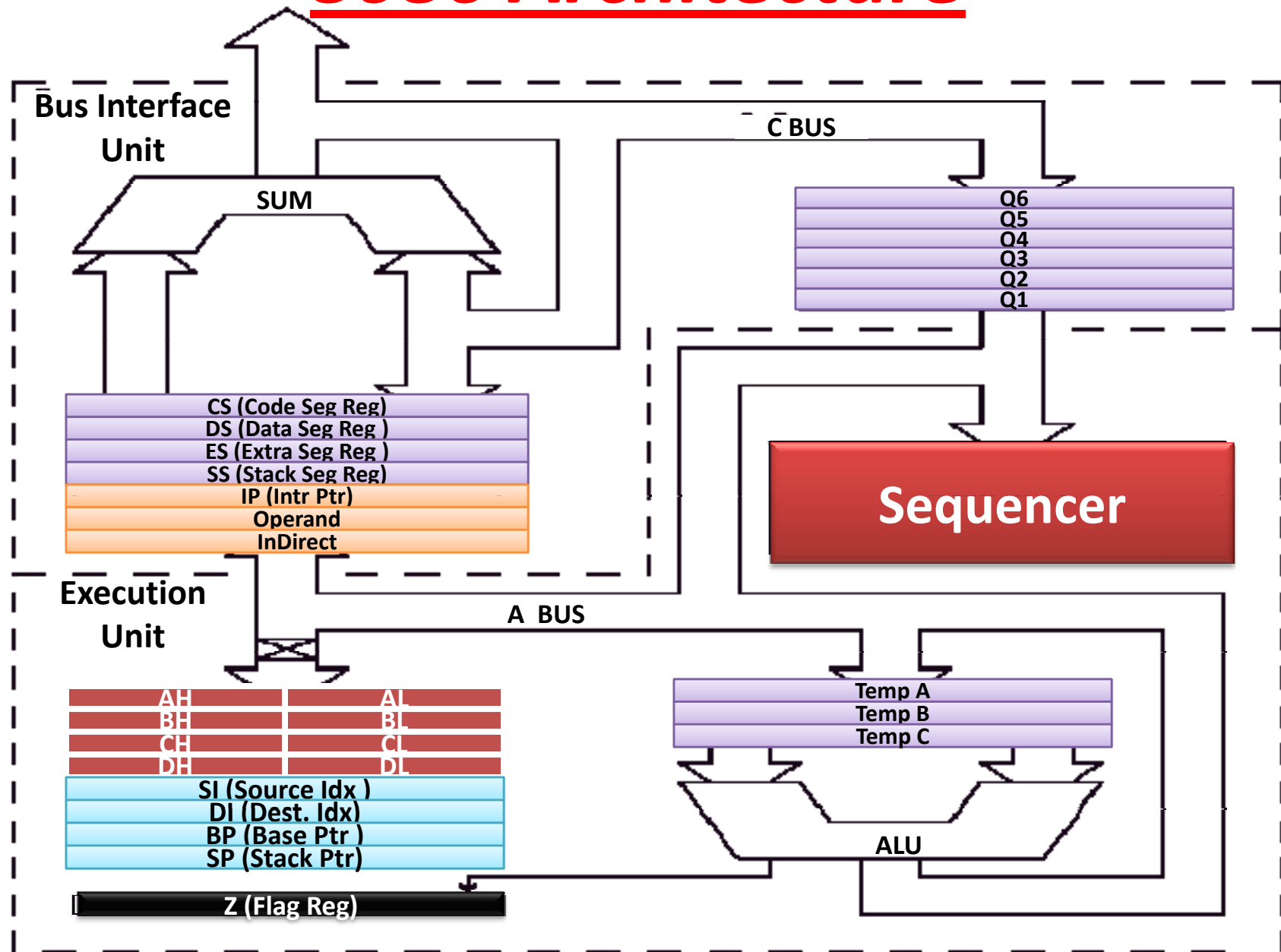
Assignment I

- Write and execute 8085 assembly language program to find value of N_{th} Fibonacci number (Recursive version: using recursive subroutine call)
- 16 bit can support up to $65356 > F_{24}$
- Deadline: 12th Aug 2010, 11.55Mid night
- After deadline grading: Max 5 out of 10
- Send TXT version of program with file name RollNo.txt to asahu@iitg.ernet.in with Assignment one as subject of email
- Don't submit copied one: will get Negative marks

Introduction to 8086 & i386 processor

- 16 bit Microprocessor
- All internal registers as well as internal and external data buses were 16 bits wide
- 4 Main Register, 4 Index Register, 4 Segment Register, Status Reg, Instr Ptr.
- Not compatible with 8085, but with successors
- Two Unit works in parallel:
 - Bus Interface Unit (BIU)
 - Execution Unit (EI)

8086 Architecture



8086 Registers

- **AX** - the accumulator register (divided into **AH** / **AL**)
- **BX** - the base address register (divided into **BH** / **BL**)
- **CX** - the count register (divided into **CH** / **CL**)
- **DX** - the data register (divided into **DH** / **DL**)

- **SI** - source index register.
- **DI** - destination index register.
- **BP** - base pointer.
- **SP** - stack pointer.

AH	AL
BH	BL
CH	CL
DH	DL

SI (Source Idx)
DI (Dest. Idx)
BP (Base Ptr)
SP (Stack Ptr)

Z (Flag Reg)

CS (Code Seg Reg)
DS (Data Seg Reg)
ES (Extra Seg Reg)
SS (Stack Seg Reg)

IP (Intr Ptr)

8086 Architecture

- Execution Unit :
 - ALU may be loaded from three temp registers (TMPA, TMPB, TMPC)
 - Execute operations on bytes or 16-bit words.
 - The result stored into temp reg or registers connected to the internal data bus.
- Bus Interface Unit
 - BIU is intended to compute the addresses.
 - Two temporary registers
 - indirect addressing
 - four segment registers (DS, CS, SS and ES),
 - Program counter (IP - Instruction Pointer),
 - A 6-byte Queue Buffer to store the pre-fetched opcodes and data.
 - This Prefetch Queue optimize the bus usage.
 - To execute a jump instruction the queue has to be flushed since the pre-fetched instructions do not have to be executed.

Next Class Agenda

- Detail of 8086 Architecture
- Advanced 32 bit architecture (i386, Pentium, p4)
 - I know a little bit of this
 - My expertise area of work
- Programming model for x86 architecture
- 8086 Assembly language programming
- MASM / TASM /NASM (x86 assembler)
- ***If you miss the next class, will miss a lot***

Thanks