Introduction to 8051

Instruction Set Architecture

Suryakant Toraskar

Research Scholar

Computer Architecture and Dependable Systems Lab

Department of Electrical Engineering,

Indian Institute of Technology Bombay

Slides courtesy : Prof. Virendra Singh

http://www.ee.iitb.ac.in/~viren/

E-mail: viren@ee.iitb.ac.in

Lecture (05 June 2023)

Think it. Make it. Prove it.

INTRODUCTION (Microprocessors and Microcontrollers)

Need for Microprocessor

- Complex integrated circuit design
  - high fixed cost component.
  - cost can be made low by making it in large volumes
- Sensible to make a design which can perform different functions in different products

  => Use in variety of applications

  => Making its requirement in large quantities.
- The functionality of the device is selected by a digital input,

  => an **Instruction**
- The device needs to be provided a series of instructions to perform a specific function.

  => Such specific sequence of instructions is a **Program**.

Need for Processor…

- The programmable digital device which can run a program

  =>a processor.

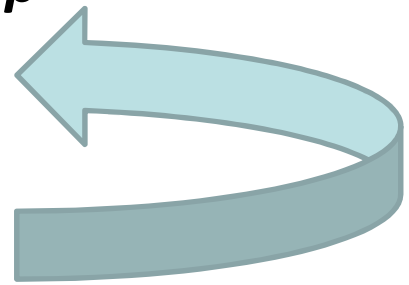
  which can be large and complex.
- A simple processor which is implemented on a single chip using VLSI technology

  => **Microprocessor**.
- Current days
  - Term is applied even to complex processors
  - e.g. Pentium has more than 100 million transistors!

Microprocessor

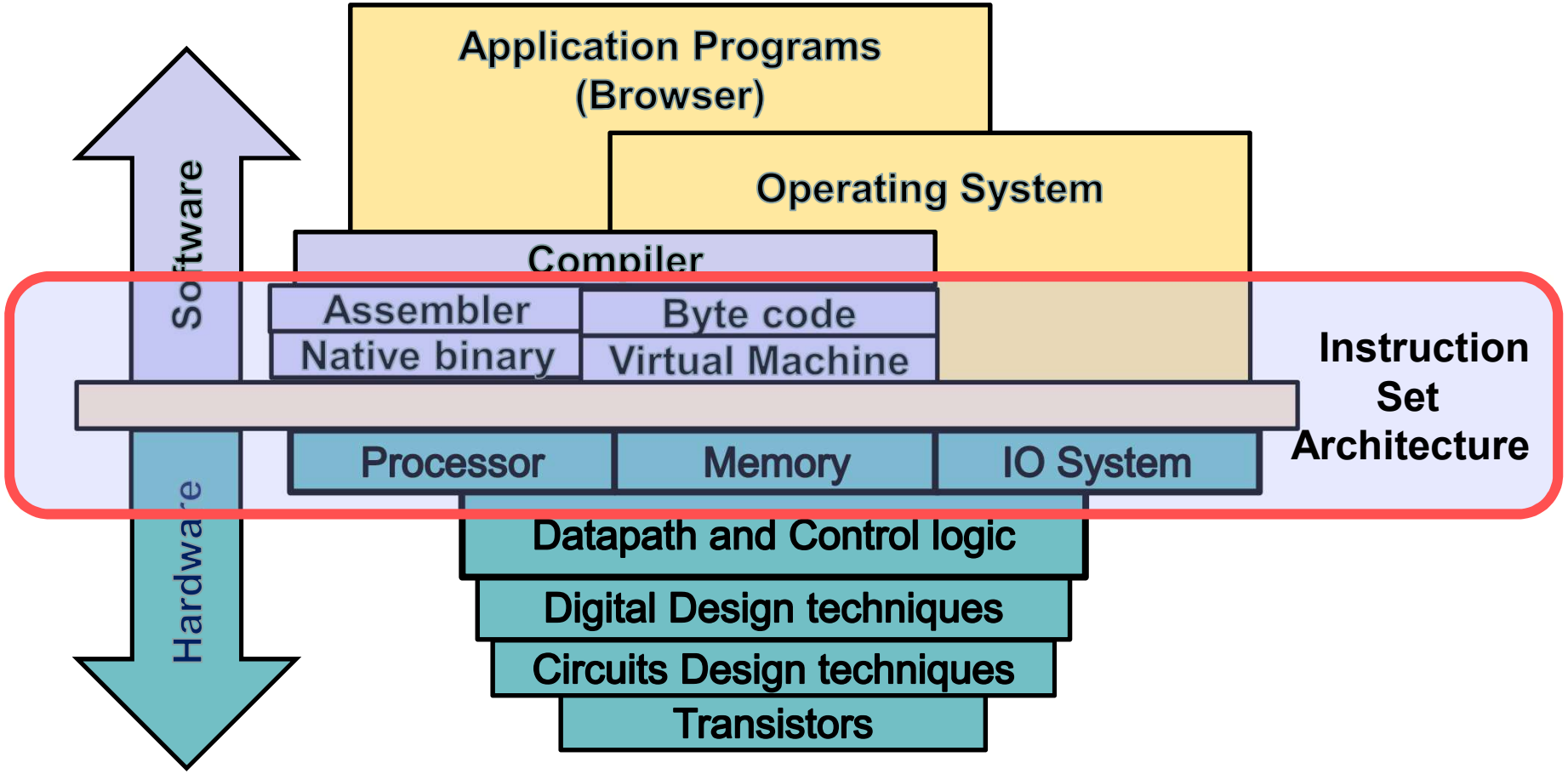
- What does a microprocessor do ?

- The circuitry in any ***microprocessor performs a few basic operations in an endless loop***
  - **Fetch** next instruction
  - **Decode** it
  - **Execute** the instruction



- When does it stop – only when power is removed

INSTRUCTION SET ARCHITECTURE



Coordination of many *levels of abstraction*

Instruction Set Architecture

- Instruction Set Architecture (ISA) :
  - **Structure (layer) of the computer** which the **machine language programmer must understand** to write a correct (timing independent) program for that machine
  - Its also the **machine description** that a **hardware designer must understand** to design a correct implementation of the computer
  - Its an interface which allows the hardware and software levels to work together

Evolution of Instruction Sets

- Major advancement in processor architecture are associated with decisions related to instruction set design
  - e.g. : **Stack** vs **GPR** (System 360)
- Design decisions need to take following into account
  - Technology
  - Machine organization
  - Programming language
  - Compiler technology
  - Operating system
- And they in turn influence these…

Components of ISA

- Sometimes also referred to as **The programmer's model** of the machine
- **Storage cells**
  - General and Special purpose registers in the CPU
  - Many general purpose of same size in memory
  - Storage associated with IO devices
- The **machine instruction set**
  - is the entire repertoire of machine operations
  - It makes use of the storage cells, formats and results of the fetch/execute cycle

Components of ISA

- The instruction format
  - Size and meaning of fields within the instruction

- The **nature of the fetch execute cycle**
  - Things that are done before the operation code is known

Components of ISA…

- The **instruction format**
  - Size and meaning of the fields within the instruction

- C statement

  f = (g+h) –(i+j)

- Assembly instruction

  add t0, g, h

  add t1, I,j

  sub f, t0, t1

- The **Assembly instruction format**

Opcode/ Mnemonic Operand list with source/destinations

Why not Bigger Instructions

- Why not a single assembly instruction for the C statement

  f = (g+h) –(i+j)

- Church's thesis: A very primitive computer can compute anything that a fancy computer can compute – one needs only arithmetic/logical functions, read and write to memory, and data dependent decisions
- The ISA design is based on practical reasons for Performance and cost, not computability
- Regularity / Patterns tends to increase both

Hardware to handle arbitrary number of operands is complex and slow and UNNECESSARY

What Must an Instruction Specify

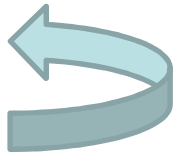
Data flow ←

- Which Operation to perform : add r0, r1, r3
  - Ans : Opcode : add, load, store, branch etc.
- Where to find the Operands : add r0, r1, r3
  - Ans : In CPU Registers, memory cells, IO locations or as part of instruction
- Where to store the Result : add r0, r1, r3
  - Ans : In CPU Registers or memory cells

What Must an Instruction Specify

- Location of the next instruction : **loop** : add r0, r1, r3

  br loop



  - Ans : Almost always the memory cell pointed by the Program counter (PC)

- Sometimes there *is* no operand, or no result, or no next instruction. Can you think of examples?

Instruction Classes

- **Data movement instructions :**

Move data from a memory location or register to another memory location or register without changing its form.

  - Load: source is a memory location and destination is a register
  - Store: source is a register and destination is a memory location

Instruction Classes

- **Arithmetic amd Logic (ALU) instructions :**

Change the form of one or more operands to produvce a result to be stored in another location.

  - **Add, Sub Shift** etc.

- **Branch (Control flow) instructions :**

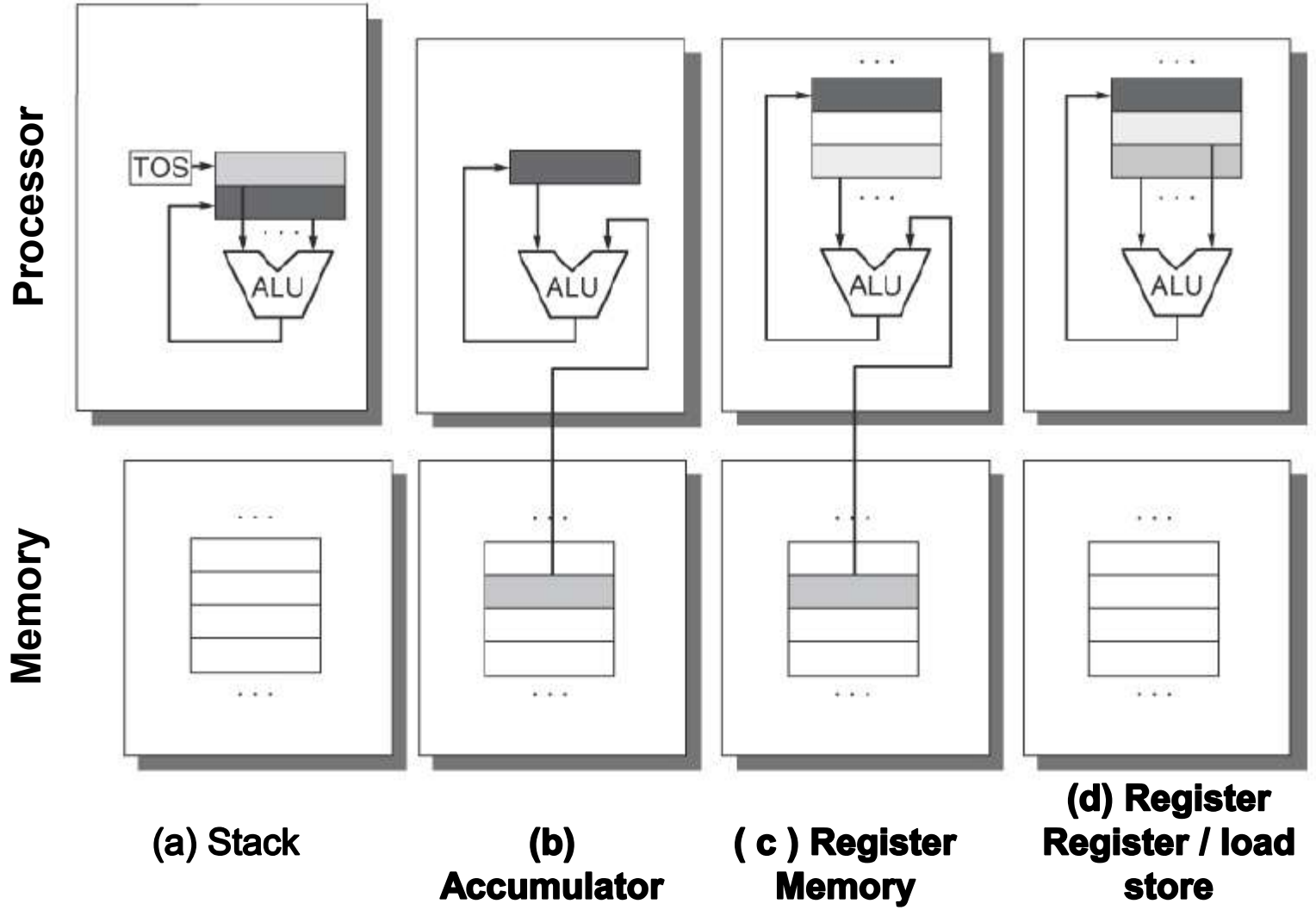
Alter the flow of control from executing the next instruction in sequence

  - Br Loc, Brz Loc2 Unconditional and Conditional branches.

ISA Classification

- Internal storage type is the most basic differentiator
- **Stack Architecture**
- **Accumulator Architecture**
- **General Purpose Register Architecture**
- **Memory-Memory Architecture**

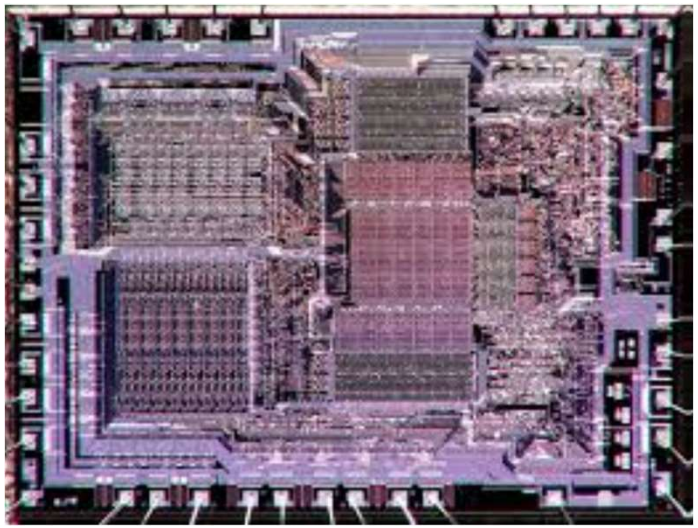
Basic Machine Organizations



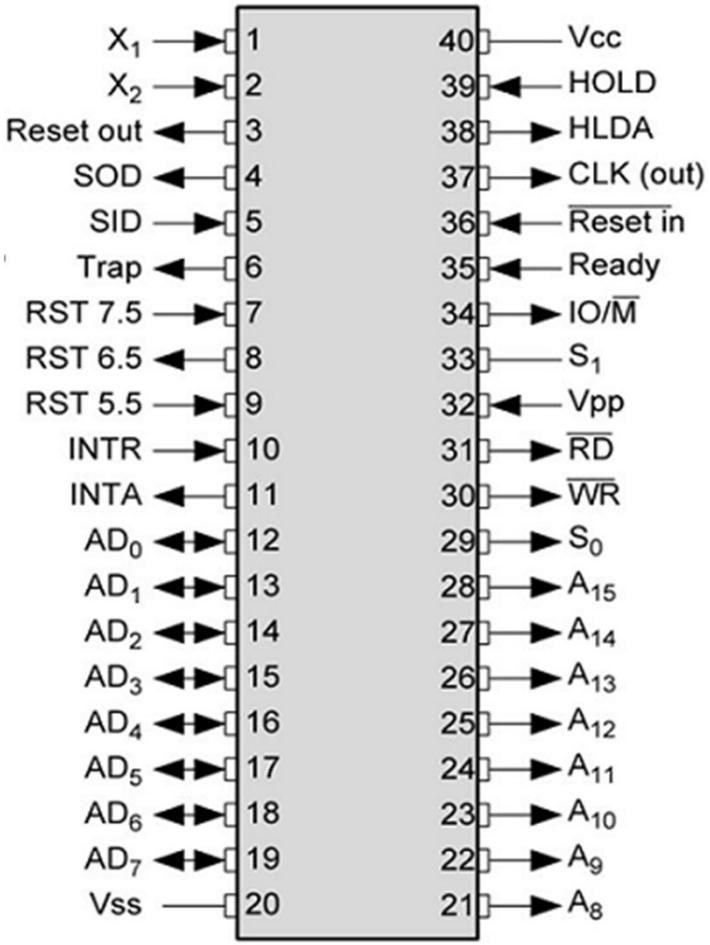
Addressing Modes

- Register
- Immediate
- Register Indirect
- Base + Displacement
- Direct Absolute
- Memory Indirect
- Auto Increment
- Auto decrement

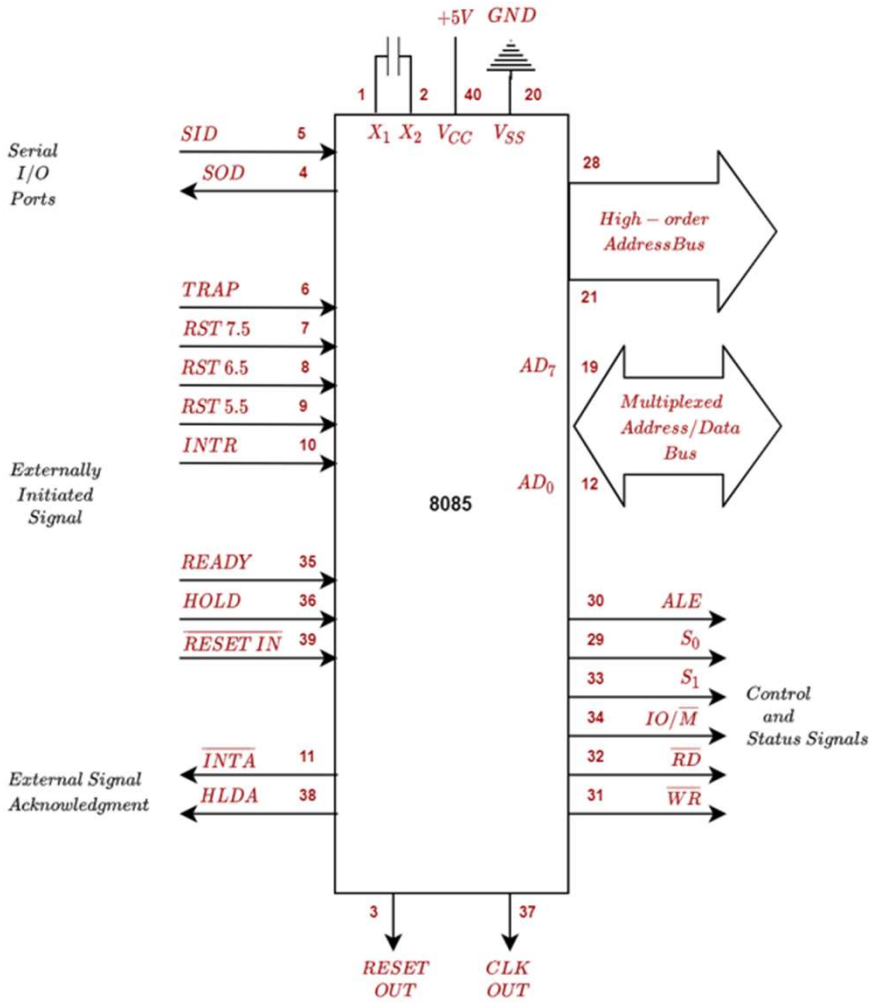
8085 ARCHITECTURE



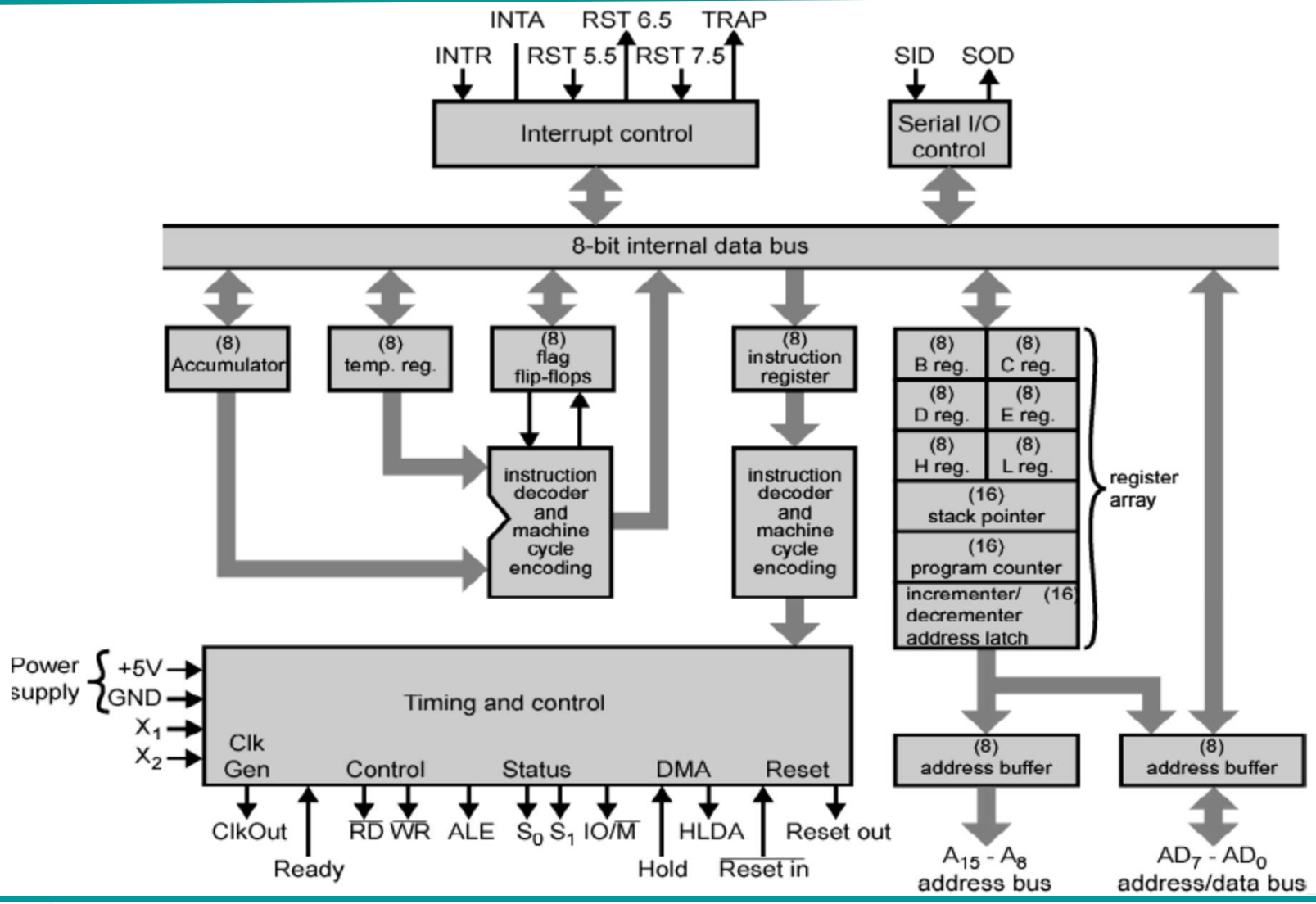
Intel 8085 pin Configuration



Intel 8085 Signal and IO pins



Intel 8085 CPU Block Diagram



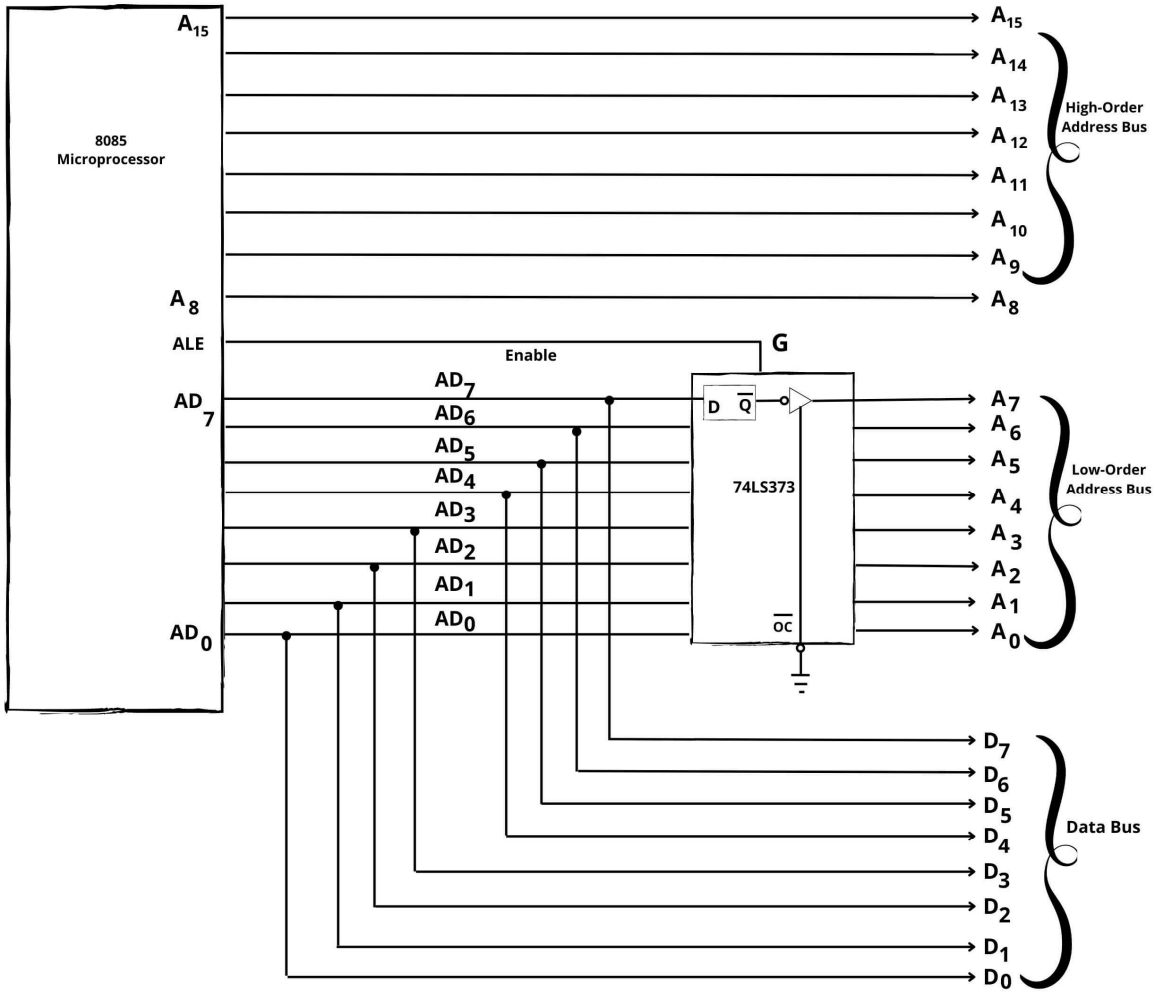
Intel 8085 and its Buses

- 8-bit general purpose microprocessor that can address 64K Byte of memory.
- 40 pins and uses +5V for power. It can run at a maximum frequency of 3 MHz.
  - The pins on the chip can be grouped into 6 groups
    - Address Bus.
    - Data Bus.
    - Control and Status Signals.
    - Power supply and frequency.
    - Externally Initiated Signals.
    - Serial I/O ports.

Address and Data bus Systems

- The address bus has 8 signal lines A8 – A15 which are unidirectional.
- The other 8 address bits are multiplexed (time shared) with the 8 data bits.
  - So, the bits AD0 – AD7 are bi-directional and serve as A0 – A7 and D0 – D7 at the same time.
    - During the execution of the instruction, these lines carry the address bits during the early part, then during the late parts of the execution, they carry the 8 data bits.
  - In order to separate the address from the data, we can use a latch to save the value before the function of the bits changes.

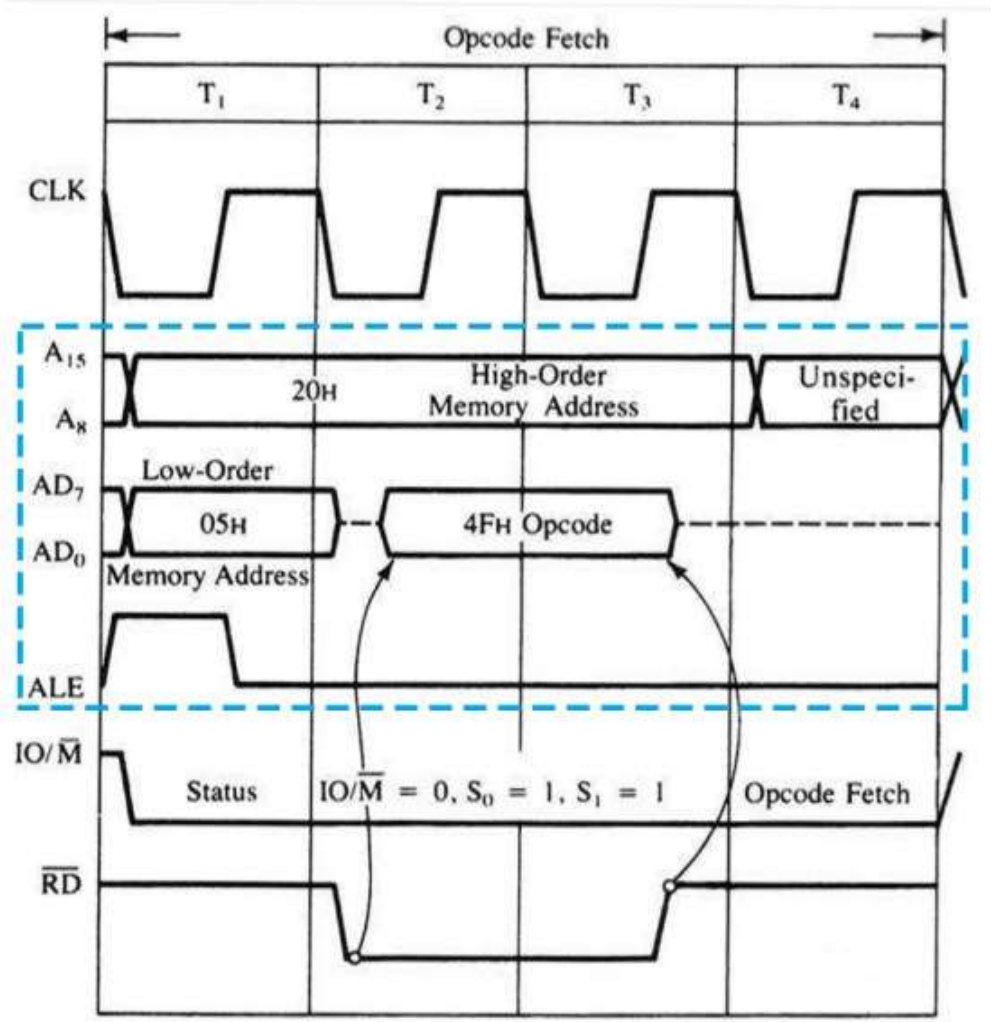
Demultiplexing Address/Data Bus



The Control and Status Signals

- 4 main control and status signals.
  - ALE: Address Latch Enable. This signal is a pulse that become 1 when the AD0 – AD7 lines have an address on them. It becomes 0 after that. This signal can be used to enable a latch to save the address bits from the AD lines.
  - RD: Read. Active low.
  - WR: Write. Active low.
  - IO/M: This signal specifies whether the operation is a memory operation (IO/M=0) or an I/O operation (IO/M=1).
  - S1 and S0 : Status signals to specify the kind of operation being performed. Usually not used in small systems.

Timing of Address/Data Bus

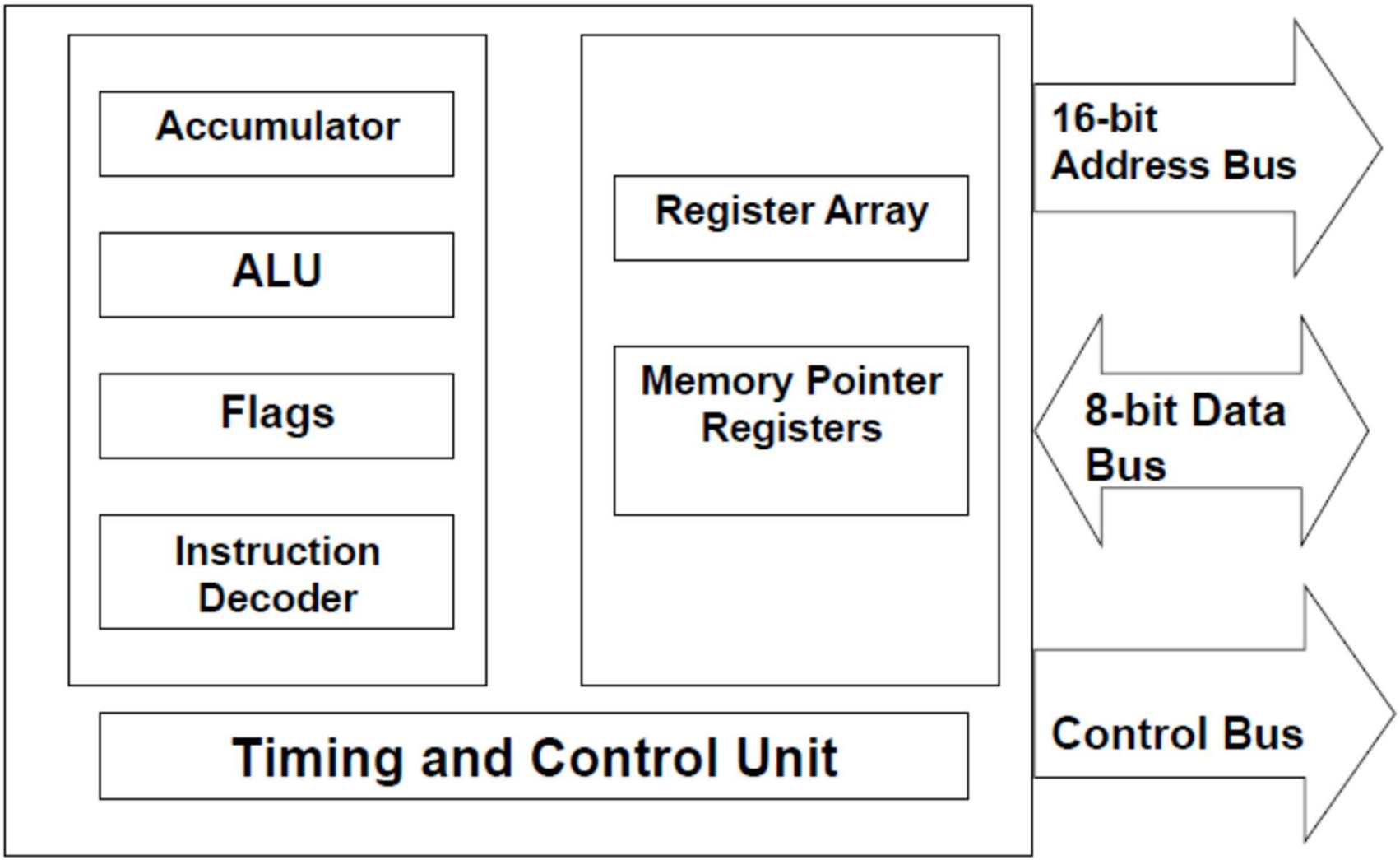


Demultiplexing Address/Data Bus

- 4 main control and status signals.
  - ALE: Address Latch Enable. This signal is a pulse that become 1 when the AD0 – AD7 lines have an address on them. It becomes 0 after that. This signal can be used to enable a latch to save the address bits from the AD lines.
  - RD: Read. Active low.
  - WR: Write. Active low.
  - IO/M: This signal specifies whether the operation is a memory operation (IO/M=0) or an I/O operation (IO/M=1).
  - S1 and S0 : Status signals to specify the kind of operation being performed. Usually not used in small systems.

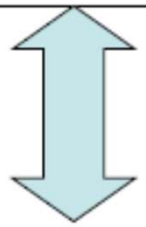
8085
INSTRUCTION SET

8085: Programming Model



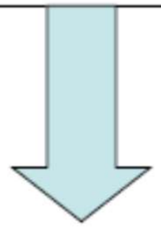
8085: Programming Model…

Accumulator (8-bit)	Flag Register (8-bit)							
	S	Z		AC		P		CY
B (8-bit)	C (8-bit)							
D (8-bit)	E (8-bit)							
H (8-bit)	L (8-bit)							
Stack Pointer (SP) (16-bit)								
Program Counter (PC) (16-bit)								



8- Lines

Bidirectional



16- Lines

Unidirectional

Instruction set of 8085

- Instructions
  - 74 operation codes, e.g. ADD
  - 246 Instructions, e.g. ADD B
- 8085 instructions can be classified as
  1. **Data Transfer (Copy)**
  2. **Arithmetic**
  3. **Logical and Bit manipulation**
  4. **Branch**
  5. **Machine Control**

8085: Addressing modes

- The various formats of specifying operands are called addressing modes
- Addressing modes of 8085
  - Register Addressing
  - Immediate Addressing
  - Memory Addressing
  - Input/Output Addressing

Register Addressing Mode

- Operands are one of the internal registers of 8085
- Examples-

MOV A, B

ADD C

Immediate Addressing Mode

- Value of the operand is given in the instruction itself
- Example-

MVI A, 20H

LXI H, 2050H

ADI 30H

SUI 10H

Memory Addressing Mode

- One of the operands is a memory location
- Depending on how address of memory location is specified, **memory** addressing is of two types
  - **Direct** addressing
  - **Indirect** addressing

Direct Addressing Mode

- 16-bit Address of the memory location is specified in the instruction directly
- Examples-

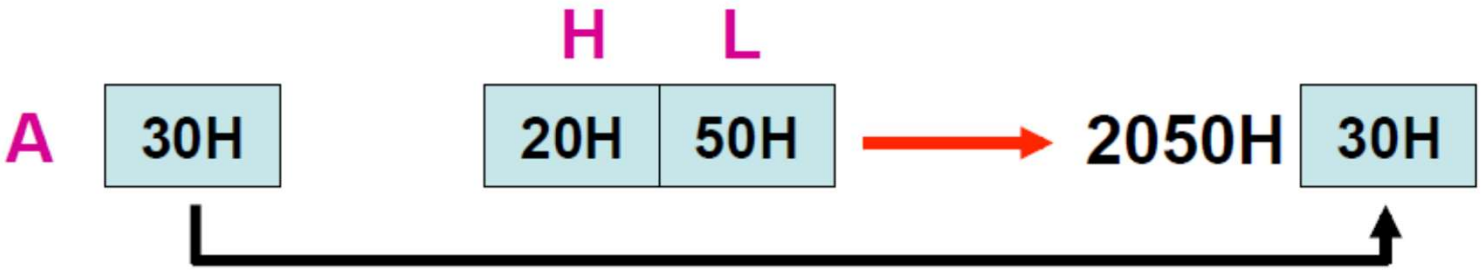
LDA 2050H

STA 3050H

Indirect Addressing Mode

- A memory pointer register is used to store the address of the memory location
- Example-

MOV M, A ;copy register A to memory location whose address is stored in register pair HL



Input Output Addressing

- 8-bit address of the port is directly specified in the instruction
- Examples-

  IN 07H

  OUT 21H

Instruction and Data Formats

Instruction classification according to size

- 1-byte Instructions
- 2-byte Instructions
- 3-byte Instructions

One byte Instruction

- Includes Opcode and Operand in the same byte
- Examples-

Opcode	Operand	Binary Code	Hex Code
MOV	C, A	0100 1111	**4F**H
ADD	B	1000 0000	**80**H
HLT		0111 0110	**76**H

Two byte Instructions

- First byte specifies Operation Code
- Second byte specifies Operand

Opcode	Operand	Binary Code	Hex Code
MVI	A, 32H	0011 1110 0011 0010	3EH 32H
MVI	B, F2H	0000 0110 1111 0010	06H F2H

Three byte Instructions

- First byte specifies Operation Code
- Second & Third byte specifies Operand

Opcode	Operand	Binary Code	Hex Code
LXI	H, 2050H	0010 0001	21H
		0101 0000	50H
		0010 0000	20H
LDA	3070H	0011 1010	3AH
		0111 0000	70H
		0011 0000	30H

8085:Data movement Instructions

Instruction	Operation
NOP	No operation
MOV *r1*, *r2*	*r1* = *r2*
MOV *r*, *M*	*r* = M[*HL*]
MOV *M*, *r*	M[*HL*] = *r*
MVI *r*, *n*	*r* = *n*
MVI *M*, *n*	M[*HL*] = *n*
LXI *rp*, Γ	*rp* = Γ
LDA Γ	*A* = M[Γ]
STA Γ	M[Γ] = *A*
LHLD Γ	*HL* = M[Γ], M[Γ+1]
SHDL Γ	M[Γ], M[Γ+1] = *HL*

Instruction	Operation
LDAX *rp*	*A* = M[*rp*] (*rp* = *BC*, *DE*)
STAX *rp*	M[*rp*] = *A* (*rp* = *BC*, *DE*)
XCHG	*DE* ↔ *HL*
PUSH *rp*	Stack = *rp* (*rp* ≠ *SP*)
PUSH PSW	Stack = *A*, flag register
POP *rp*	*rp* = Stack (*rp* ≠ *SP*)
POP PSW	*A*, flag register = Stack
XTHL	*HL* ↔ Stack
SPHL	*SP* = *HL*
IN *n*	*A* = input port *n*
OUT *n*	output port *n* = *A*

8085:Data operation Instructions

Instruction	Operation	Flags
ADD *r*	$A = A + r$	All
ADD *M*	$A = A + M[HL]$	All
ADI *n*	$A = A + n$	All
ADC *r*	$A = A + r + CY$	All
ADC *M*	$A = A + M[HL] + CY$	All
ACI *n*	$A = A + n + CY$	All
SUB *r*	$A = A - r$	All
SUB *M*	$A = A - M[HL]$	All
SUI *n*	$A = A - n$	All
SBB *r*	$A = A - r - CY$	All
SBB *M*	$A = A - M[HL] - CY$	All
SBI *n*	$A = A - n - CY$	All
INR *r*	$r = r + 1$	Not *CY*
INR *M*	$M[HL] = M[HL] + 1$	Not *CY*
DCR *n*	$r = r - 1$	Not *CY*
DCR *M*	$M[HL] = M[HL] - 1$	Not *CY*
INX *rp*	$rp = rp + 1$	None
DCX *rp*	$rp = rp - 1$	None
DAD *rp*	$HL = HL + rp$	*CY*
DAA	Decimal adjust	All

Instruction	Operation	Flags
ANA *r*	$A = A \wedge r$	All
ANA *M*	$A = A \wedge M[HL]$	All
ANI *n*	$A = A \wedge n$	All
ORA *r*	$A = A \vee r$	All
ORA *M*	$A = A \vee M[HL]$	All
ORI *n*	$A = A \vee n$	All
XRA *r*	$A = A \oplus r$	All
XRA *M*	$A = A \oplus M[HL]$	All
XRI *n*	$A = A \oplus n$	All
CMP *r*	Compare *A* and *r*	All
CMP *M*	Compare *A* and $M[HL]$	All
CPI *n*	Compare *A* and *n*	All
RLC	$CY = A_7$, $A = A_{6-0}$, A_7	*CY*
RRC	$CY = A_0$, $A = A_0$, A_{7-1}	*CY*
RAL	*CY*, *A* = *A*, *CY*	*CY*
RAR	*A*, *CY* = *CY*, *A*	*CY*
CMA	$A = A'$	None
CMC	$CY = CY'$	*CY*
STC	$CY = 1$	*CY*

8085:Program Control Instructions

Instruction	Operation
JUMP Γ	GOTO Γ
J*cond* Γ	If condition is true then GOTO Γ
PCHL	GOTO address in *HL*
CALL Γ	Call subroutine at Γ
C*cond* Γ	If condition is true then call subroutine at Γ
RET	Return from subroutine
R*cond*	If condition is true then return from subroutine
RSTn	Call subroutine at 8*n (n = 5.5, 6.5, 7.5)
RIM	*A = IM*
SIM	*IM = A*
DI	Disable interrupts
EI	Enable interrupts
HLT	Halt the CPU

Writing a Assembly Language Program

Steps to write a program

- Analyze the problem
- Develop algorithm
- Draw a flowchart
- Convert flowchart to Assembly language by picking appropriate 8085 instructions

Program : Sum of N Natural Numbers

The Algorithm of the program

```
1:      total = 0, i = 0
2:      i = i + 1
3:      total = total + i
4:      IF I /= n THEN GOTO 2
```

} n + (n - 1) + … + 1

The 8085 coding of the program

```
        LDA  n
        MOV  B, A
        XRA  A
Loop:   ADD  B
        DCR  B
        JNZ  Loop
        STA  total
```

} i = n

} sum = A XOR A = 0

} sum = sum + i

} i = i - 1

} IF I /= 0 THEN GOTO Loop

} total = sum

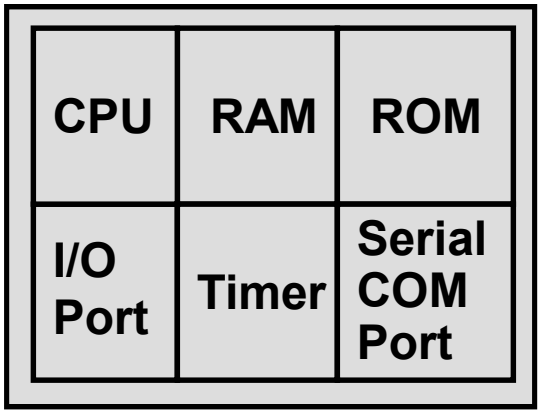
8051 MICROCONTROLLER

Criterion for Choosing a microcontroller

- Meeting the computing needs of the task efficiently and cost effectively
  - speed, the amount of ROM and RAM, the number of I/O ports and timers, size, packaging, power consumption
  - easy to upgrade
  - cost per unit
- Availability of software development tools
  - assemblers, debuggers, C compilers, emulator, simulator, technical support
- Wide availability and reliable sources of the microcontrollers

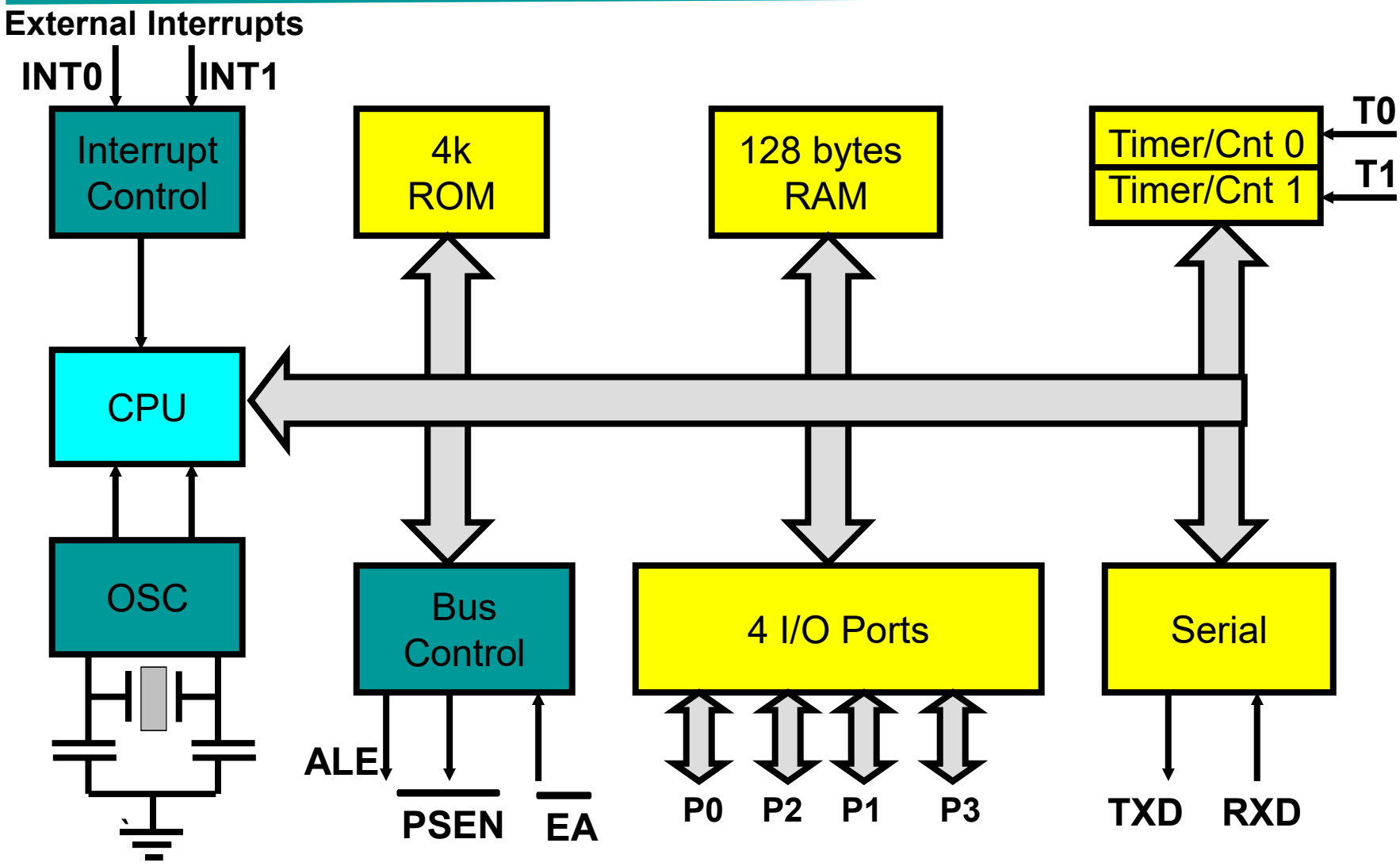
8051 Basic Component

- 4K bytes internal ROM
- 128 bytes internal RAM
- Four 8-bit I/O ports (P0 - P3).
- Two 16-bit timers/counters
- One serial interface

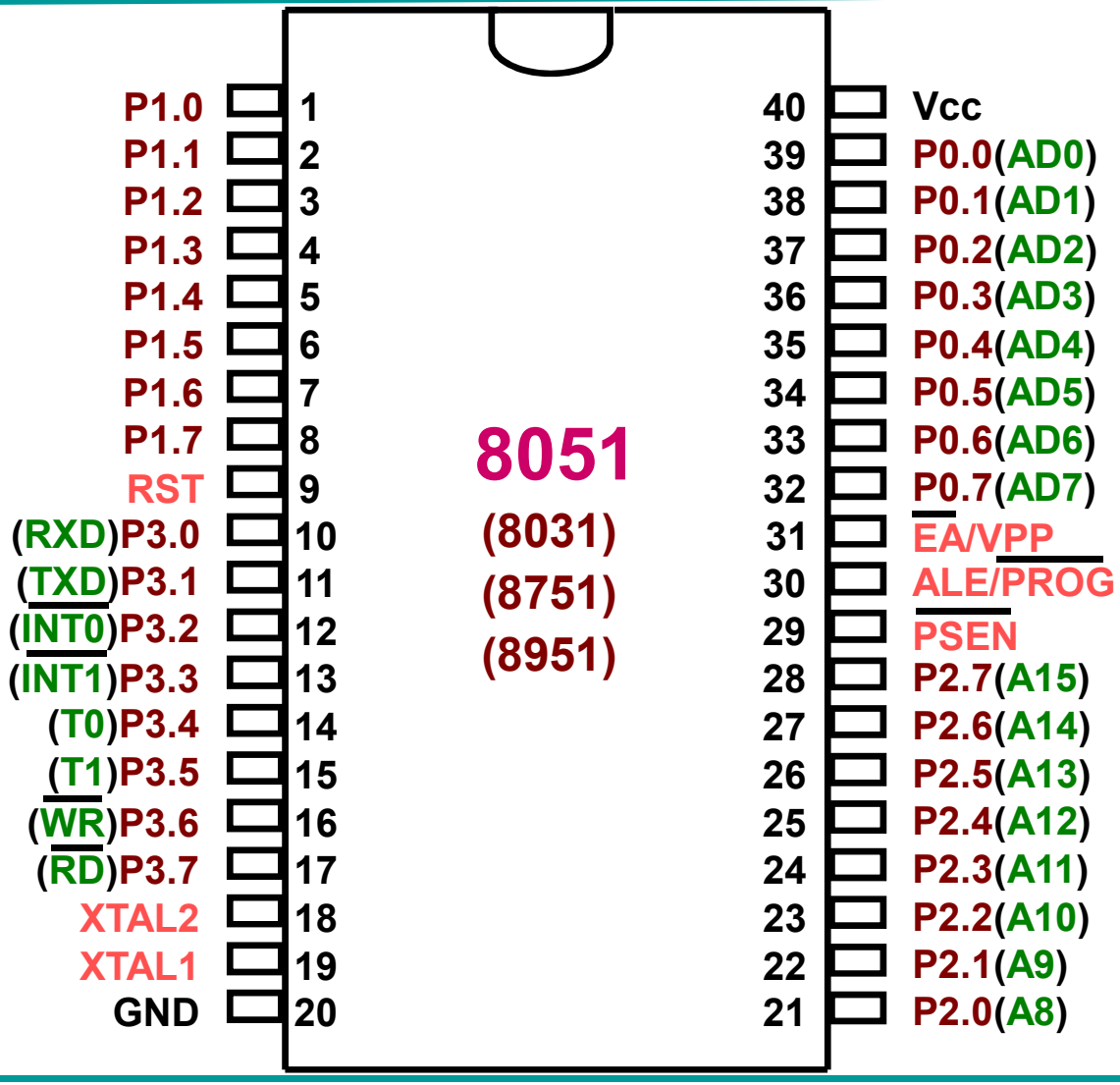


A single chip Microcontroller

Block diagram



8051 Footprint

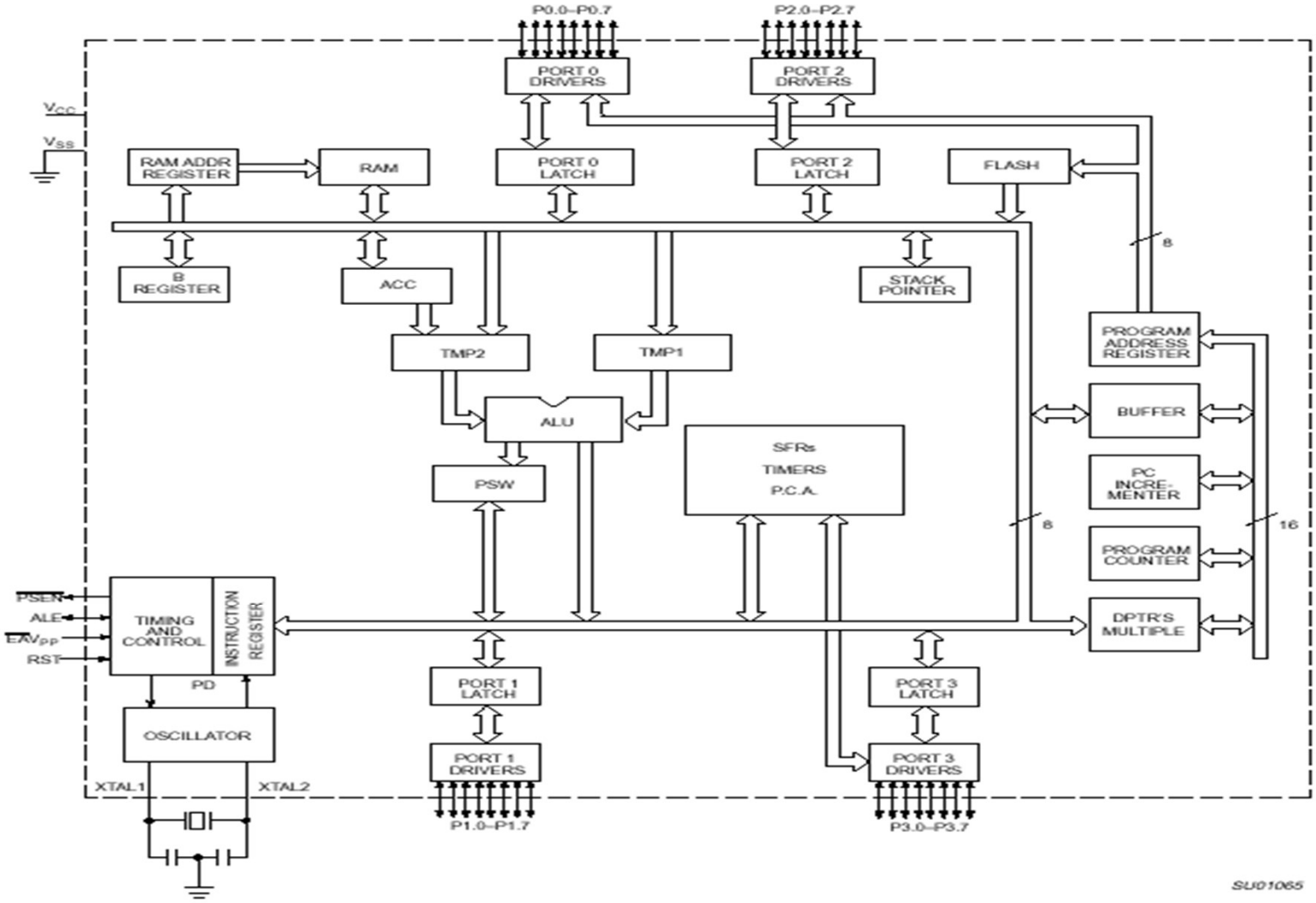


8051 ARCHITECTURE

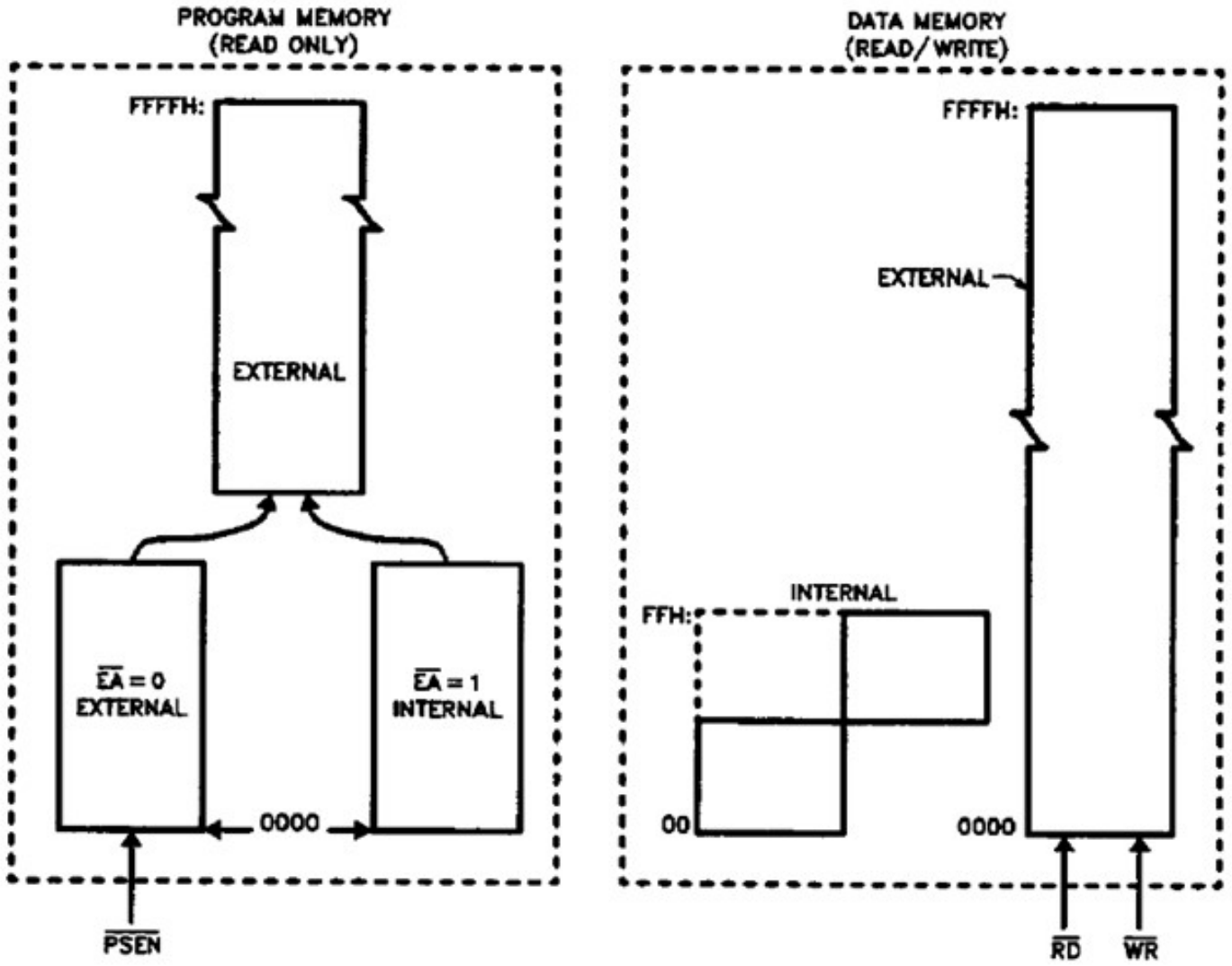
8051 Features

- Only 1 On chip oscillator (external crystal)
- 6 interrupt sources (2 external , 3 internal, Reset)
- 64K external code (program) memory(only read)PSEN
- 64K external data memory(can be read and write) by RD,WR
- Code memory is selectable by EA (internal or external)
- We may have External memory as data and code

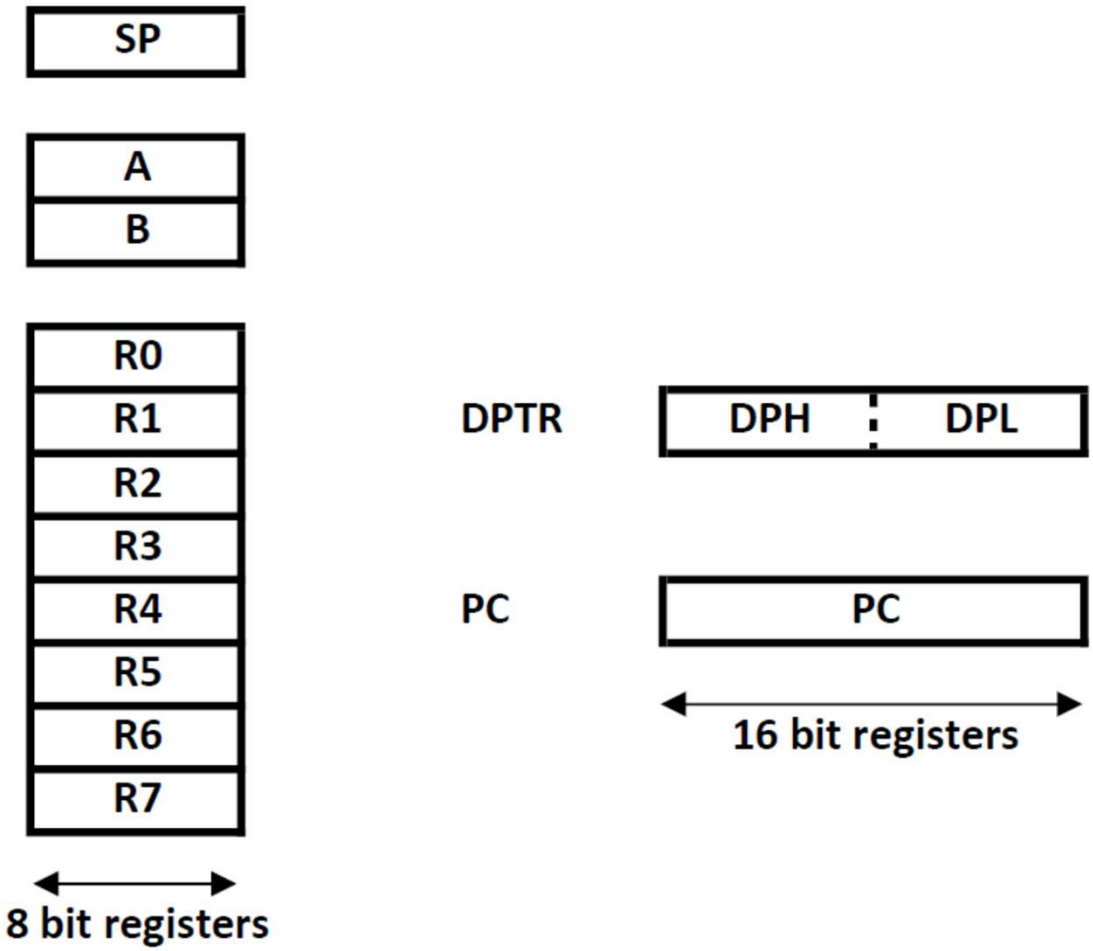
8051 Internal Block diagram



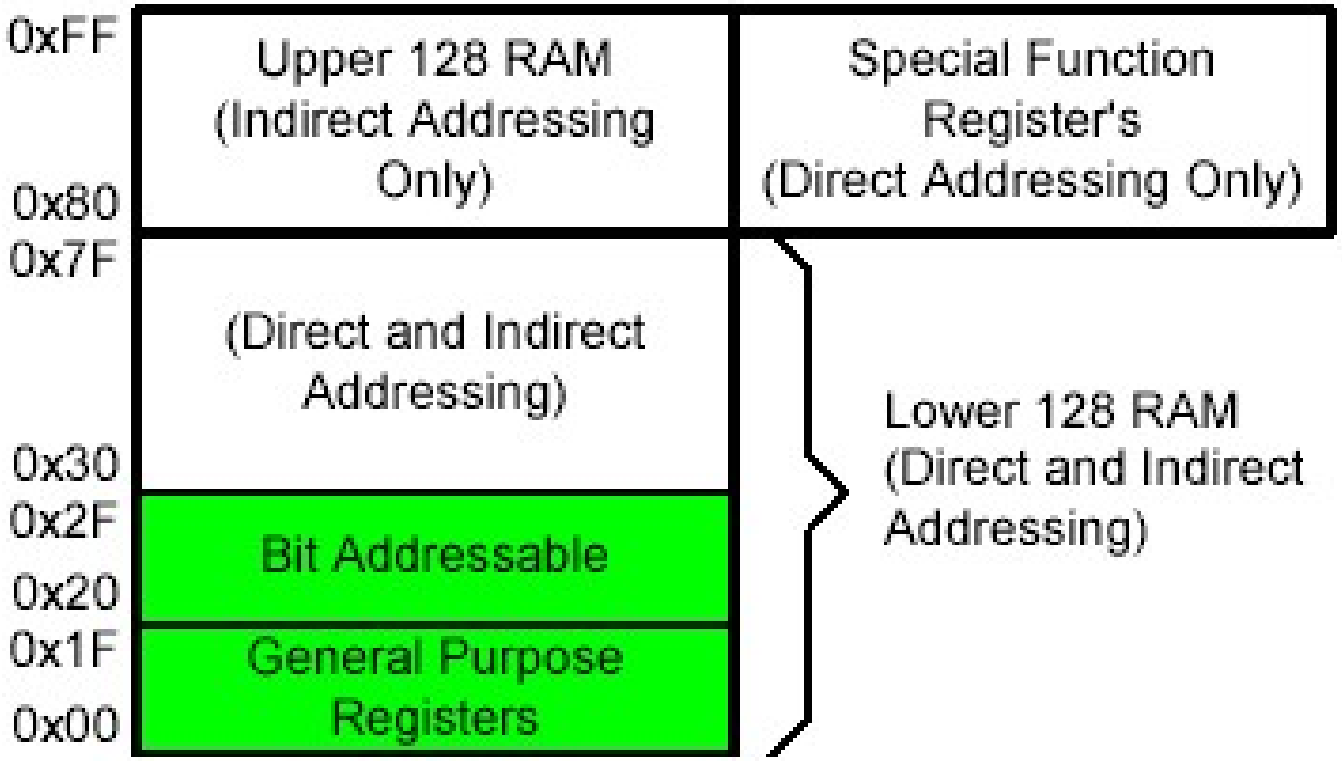
Memory Organisation



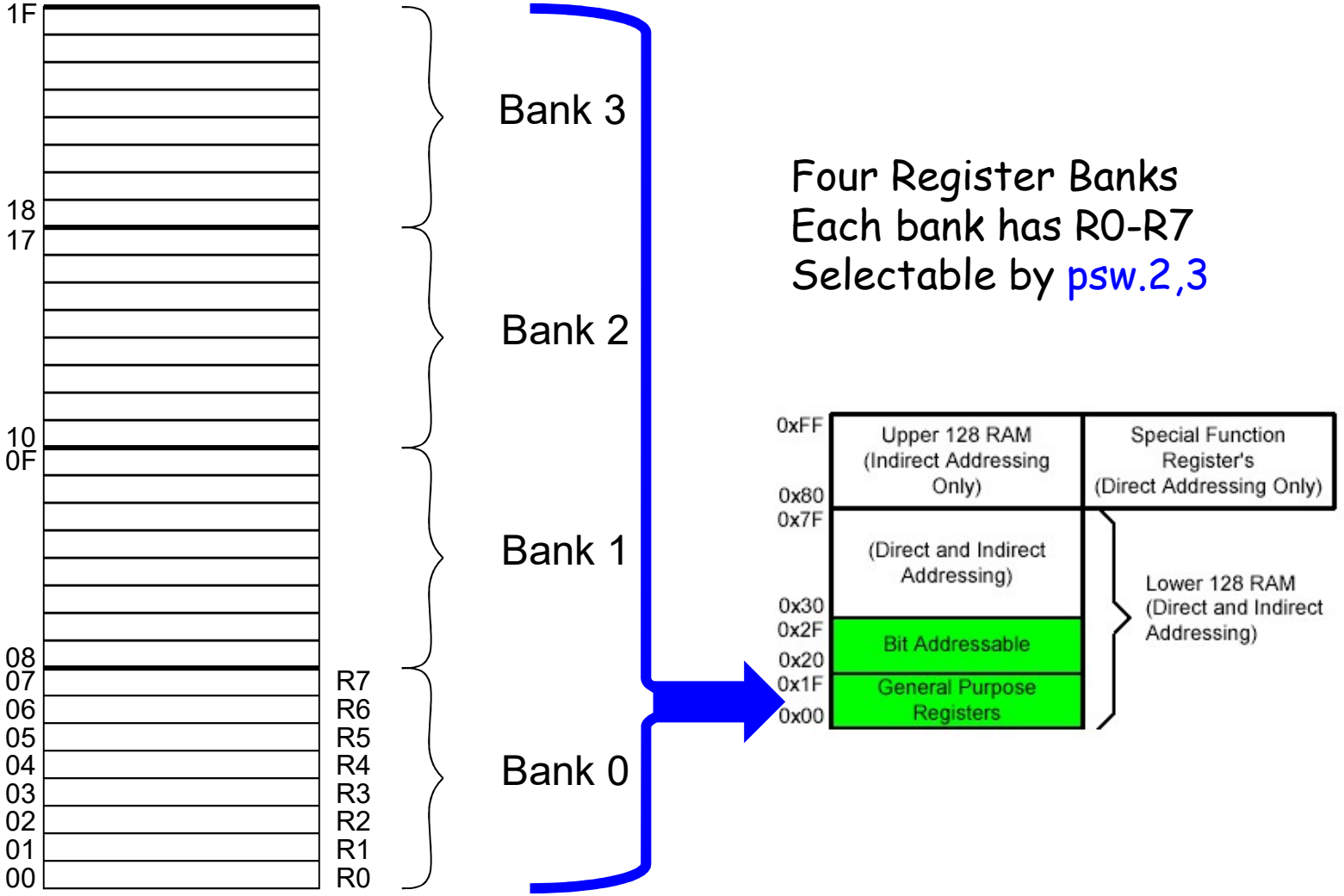
8051 Registers



On-Chip Memory: Internal RAM



Registers


1F
18
17
10
0F
08
07
06
05
04
03
02
01
00
R7
R6
R5
R4
R3
R2
R1
R0
Bank 3
Bank 2
Bank 1
Bank 0
Four Register Banks
Each bank has R0-R7
Selectable by psw.2,3
0xFF
0x80
0x7F
0x30
0x2F
0x20
0x1F
0x00
Upper 128 RAM (Indirect Addressing Only)
Special Function Register's (Direct Addressing Only)
(Direct and Indirect Addressing)
Bit Addressable
General Purpose Registers
Lower 128 RAM (Direct and Indirect Addressing)

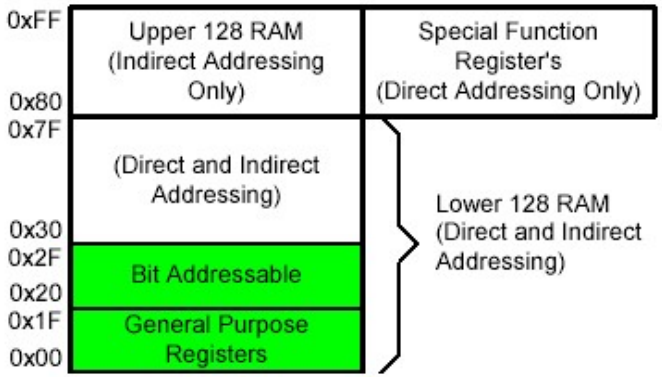
Bit Addressable Memory (RAM)

2F	7F							78
2E								
2D								
2C								
2B								
2A								
29								
28								
27								
26								
25								
24								
23						1A		
22								10
21	0F							08
20	07	06	05	04	03	02	01	00

20h – 2Fh
(16 locations X 8-bits = 128 bits)

Bit addressing:

mov C, 1Ah
or
mov C, 23h.2



- **Bit variables allow for flags needed for control applications**

Special Function Registers (SFRs)

Memory Map

8 Bytes

F8								FF
F0	B							F7
E8								EF
E0	ACC							E7
D8								DF
D0	PSW							D7
C8	T2CON		RCAP2L	RCAP2H	TL2	TH2		CF
C0								C7
B8	IP							BF
B0	P3							B7
A8	IE							AF
A0	P2							A7
98	SCON	SBUF						9F
90	P1							97
88	TCON	TMOD	TL0	TL1	TH0	TH1		8F
80	P0	SP	DPL	DPH				87

Bit ↑
Addressable

Bit Addressable SFRs

Summary of the 8051 on-chip data memory

(Special Function Registers)

Byte address	Bit address									Byte address	Bit address								
98	9F	9E	9D	9C	9B	9A	99	98	SCON	FF									
										F0	F7	F6	F5	F4	F3	F2	F1	F0	B
90	97	96	95	94	93	92	91	90	P1										
										E0	E7	E6	E5	E4	E3	E2	E1	E0	ACC
8D	not bit addressable								TH1										
8C	not bit addressable								TH0	D0	D7	D6	D5	D4	D3	D2	–	D0	PSW
8B	not bit addressable								TL1										
8A	not bit addressable								TL0	B8	–	–	–	BC	BB	BA	B9	B8	IP
89	not bit addressable								TMOD										
88	8F	8E	8D	8C	8B	8A	89	88	TCON	B0	B7	B6	B5	B4	B3	B2	B1	B0	P3
87	not bit addressable								PCON										
										A8	AF	–	–	AC	AB	AA	A9	A8	IE
83	not bit addressable								DPH										
82	not bit addressable								DPL	A0	A7	A6	A5	A4	A3	A2	A1	A0	P2
81	not bit addressable								SP										
80	87	86	85	84	83	82	81	80	P0	99	not bit addressable								SBUF

Program Status Word (PSW)

CY	AC	F0	RS1	RS0	OV	—	P

Bit Addressable

CY	PSW.7	Carry Flag.
AC	PSW.6	Auxiliary Carry Flag.
F0	PSW.5	Flag 0 available to the user for general purpose.
RS1	PSW.4	Register Bank selector bit 1 (SEE NOTE 1).
RS0	PSW.3	Register Bank selector bit 0 (SEE NOTE 1).
OV	PSW.2	Overflow Flag.
—	PSW.1	User definable flag.
P	PSW.0	Parity flag. Set/cleared by hardware each instruction cycle to indicate an odd/even number of '1' bits in the accumulator.

NOTE:

1. The value presented by RS0 and RS1 selects the corresponding register bank.

RS1	RS0	Register Bank	Address
0	0	0	00H-07H
0	1	1	08H-0FH
1	0	2	10H-17H
1	1	3	18H-1FH

8051
INSTRUCTION SET

8051 Instructions

- Data Type
- Instruction Format
- Addressing Modes
- Types of Operations
  - Data Transfer
  - Logical and Arithmetic
  - Control Flow

Addressing Modes

- Eight modes of addressing are available
- The different addressing modes determine how the operand byte is selected

Addressing Modes	Instruction
Register	MOV A, B
Direct	MOV 30H, A
Indirect	ADD A, @R0
Immediate Constant	ADD A, #80H
Relative*	SJMP AHEAD
Absolute*	AJMP BACK
Long*	LJMP FAR_AHEAD
Indexed	MOVC A, @A+PC

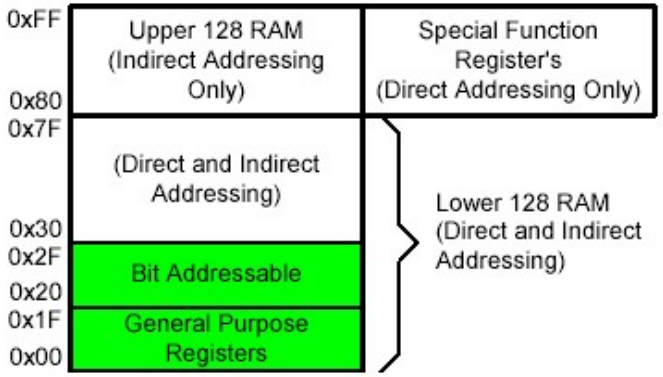
* Related to program branching instructions

8051 Instructions groups

- 8051 instructions are divided into three functional groups:
  - Data transfer operations
  - Arithmetic and Logical Instructions
  - Program branching operations

Data Transfer Instructions

- Data transfer instructions can be used to transfer data between an internal RAM location and an SFR location without going through the accumulator
- It is also possible to transfer data between the internal and external RAM by using indirect addressing
- The upper 128 bytes of data RAM are accessed only by indirect addressing and the SFRs are accessed only by direct addressing



Mnemonic	Description
MOV @Ri, direct	[@Ri] = [direct]
MOV @Ri, #data	[@Ri] = immediate data
MOV DPTR, #data 16	[DPTR] = immediate data
MOVC A, @A+DPTR	A = Code byte from [@A+DPTR]
MOVC A, @A+PC	A = Code byte from [@A+PC]
MOVX A, @Ri	A = Data byte from external ram [@Ri]
MOVX A, @DPTR	A = Data byte from external ram [@DPTR]
MOVX @Ri, A	External[@Ri] = A
MOVX @DPTR, A	External[@DPTR] = A
PUSH direct	Push into stack
POP direct	Pop from stack
XCH A, Rn	A = [Rn], [Rn] = A
XCH A, direct	A = [direct], [direct] = A
XCH A, @Ri	A = [@Rn], [@Rn] = A
XCHD A,@Ri	Exchange low order digits

Arithmetic Operations

- The appropriate status bits in the PSW are set when specific conditions are met, which allows the user software to manage the different data formats

Mnemonic	Description
ADD A, Rn	A = A + [Rn]
ADD A, direct	A = A + [direct memory]
ADD A,@Ri	A = A + [memory pointed to by Ri]
ADD A,#data	A = A + immediate data
ADDC A,Rn	A = A + [Rn] + CY
ADDC A, direct	A = A + [direct memory] + CY
ADDC A,@Ri	A = A + [memory pointed to by Ri] + CY
ADDC A,#data	A = A + immediate data + CY
SUBB A,Rn	A = A - [Rn] - CY
SUBB A, direct	A = A - [direct memory] - CY
SUBB A,@Ri	A = A - [@Ri] - CY
SUBB A,#data	A = A - immediate data - CY
INC A	A = A + 1
INC Rn	[Rn] = [Rn] + 1
INC direct	[direct] = [direct] + 1
INC @Ri	[@Ri] = [@Ri] + 1
DEC A	A = A - 1
DEC Rn	[Rn] = [Rn] - 1
DEC direct	[direct] = [direct] - 1
DEC @Ri	[@Ri] = [@Ri] - 1
MUL AB	Multiply A & B
DIV AB	Divide A by B
DA A	Decimal adjust A

- [@Ri] implies contents of memory location pointed to by R0 or R1
- Rn refers to registers R0-R7 of the currently selected register bank

Arithmetic Instructions…

Mnemonic	Description
ADD A, Rn	A = A + [Rn]
ADD A, direct	A = A + [direct memory]
ADD A,@Ri	A = A + [memory pointed to by Ri]
ADD A,#data	A = A + immediate data
ADDC A,Rn	A = A + [Rn] + CY
ADDC A, direct	A = A + [direct memory] + CY
ADDC A,@Ri	A = A + [memory pointed to by Ri] + CY
ADDC A,#data	A = A + immediate data + CY
SUBB A,Rn	A = A - [Rn] - CY
SUBB A, direct	A = A - [direct memory] - CY
SUBB A,@Ri	A = A - [@Ri] - CY
SUBB A,#data	A = A - immediate data - CY
INC A	A = A + 1
INC Rn	[Rn] = [Rn] + 1
INC direct	[direct] = [direct] + 1
INC @Ri	[@Ri] = [@Ri] + 1
DEC A	A = A - 1
DEC Rn	[Rn] = [Rn] - 1
DEC direct	[direct] = [direct] - 1
DEC @Ri	[@Ri] = [@Ri] - 1
MUL AB	Multiply A & B
DIV AB	Divide A by B
DA A	Decimal adjust A

- Multiple addressing modes for second operand

- Special arithmetic operations
- Implicit addressing

Special Arithmetic Instructions

- **MUL AB**
  - Multiplies A & B (unsigned numbers) and the 16-bit result is stored in [B15-8], [A7-0]
  - If the product is greater than 255 (FFH), the overflow flag is set; otherwise it is cleared. The carry flag is always cleared.

Special Arithmetic Instructions…

- **DIV AB**
  - Divides A by B
  - Quotient (Integer part only) is stored in A and remainder in the B register
  - Carry and OV are both cleared
  - If B contains 00H (Divide by zero), then ACC and B are undefined and an overflow flag is set. The carry flag is cleared.

Special Arithmetic Instructions…

- **DA A** - Decimal adjust Accumulator for Binary Coded Decimal (BCD) -to give two 4-bit BCD digits (packed BCD)
  - This instruction performs the decimal conversion by adding 00H, 06H, 60H or 66H to the accumulator,
  - depending on the initial value of ACC and PSW
  - If ACC bits A3-0 are > 9 (xxxx1010-xxxx1111), or if AC=1, then a value 6 is added to the accumulator to
  - produce a correct BCD digit in the lower order nibble.
  - If CY=1, because the high order bits A7-4 is > 9 (1010xxxx-1111xxxx), then these high order bits will be increased by 6 to produce a correct proper BCD in the high order nibble but not clear the carry

Logical Operations

- Logical instructions perform Boolean operations (AND, OR, XOR, and NOT) on data bytes on a *bit-by-bit* basis

- Examples:

```
ANL   A, #02H
;Mask bit 1

ORL   TCON, A
;TCON=TCON-OR-A
```

Mnemonic	Description
ANL A, Rn	A = A & [Rn]
ANL A, direct	A = A & [direct memory]
ANL A,@Ri	A = A & [memory pointed to by Ri]
ANL A,#data	A= A & immediate data
ANL direct,A	[direct] = [direct] & A
ANL direct,#data	[direct] = [direct] & immediate data
ORL A, Rn	A = A OR [Rn]
ORL A, direct	A = A OR [direct]
ORL A,@Ri	A = A OR [@RI]
ORL A,#data	A = A OR immediate data
ORL direct,A	[direct] = [direct] OR A
ORL direct,#data	[direct] = [direct] OR immediate data
XRL A, Rn	A = A XOR [Rn]
XRL A, direct	A = A XOR [direct memory]
XRL A,@Ri	A = A XOR [@Ri]
XRL A,#data	A = A XOR immediate data
XRL direct,A	[direct] = [direct] XOR A
XRL direct,#data	[direct] = [direct] XOR immediate data
CLR A	Clear A
CPL A	Complement A
RL A	Rotate A left
RLC A	Rotate A left (through C)
RR A	Rotate A right
RRC A	Rotate A right (through C)
SWAP A	Swap nibbles

Boolean Variable Instructions

8051 can perform single bit operations

- The operations include ***set***, ***clear***, ***and***, ***or*** and ***complement*** instructions
- Also included are bit–level moves or conditional jump instructions
- All bit accesses use direct addressing
- **Examples:**

```
SETB TR0 ;Start Timer0
POLL: JNB TR0, POLL
  ;Wait till timer
  overflows
```

Mnemonic	Description
CLR C	Clear C
CLR bit	Clear direct bit
SETB C	Set C
SETB bit	Set direct bit
CPL C	Complement c
CPL bit	Complement direct bit
ANL C,bit	AND bit with C
ANL C,/bit	AND NOT bit with C
ORL C,bit	OR bit with C
ORL C,/bit	OR NOT bit with C
MOV C,bit	MOV bit to C
MOV bit,C	MOV C to bit
JC rel	Jump if C set
JNC rel	Jump if C not set
JB bit,rel	Jump if specified bit set
JNB bit,rel	Jump if specified bit not set
JBC bit,rel	if specified bit set then clear it and jump

Program Branching Instructions

- Program branching instructions are used to control the flow of program execution

- Some instructions provide decision making capabilities before transferring control to other parts of the program (conditional branches).

Mnemonic	Description
ACALL addr11	Absolute subroutine call
LCALL addr16	Long subroutine call
RET	Return from subroutine
RETI	Return from interrupt
AJMP addr11	Absolute jump
LJMP addr16	Long jump
SJMP rel	Short jump
JMP @A+DPTR	Jump indirect
JZ rel	Jump if A=0
JNZ rel	Jump if A NOT=0
CJNE A,direct,rel	Compare and Jump if Not Equal
CJNE A,#data,rel	
CJNE Rn,#data,rel	
CJNE @Ri,#data,rel	
DJNZ Rn,rel	Decrement and Jump if Not Zero
DJNZ direct,rel	
NOP	No Operation

Program Examples

Example 1 : Addition of 16 bit Data

Add 4A86H and 3895H

```
        CLR C
        MOV A, #86H
        ADD A, #95H
        MOV R5, A
        MOV A, #4AH
        ADDC A, #38H
        MOV R6, A
STOP:   SJMP STOP
```

Example 2 : ASCII to packed BCD

Problem: Convert ASCII character to packed BCD

```
        MOV A, #'8'          ; (A) = 38 H
        MOV R1, #'6'         ; (R1) = 36 H
        ANL A, #0FH
        ANL R1, #0FH
        SWAP A
        ORL A, R1
STOP:   SJMP STOP
```

Example 3: Conversion to 3 digit BCD

Conversion of one byte binary data to 3 digit BCD

```
        MOV A, #'C'          ; Input number
        MOV B, #100
        DIV AB
        MOV R0, A            ; save upper BCD Digit in R0
        XCH A, B
        MOV B, #10
        DIV AB
        SWAP A
        ADD A, B
        MOV R1, A            ; save lower two digits in R1
STOP:   SJMP STOP
```

Example 4: Multibyte Addition

Addition of Multiple numbers stored at 50H

```
        MOV R1, #50H        ; Start address of data block
        MOV R2, #5          ; Count of numbers
        CLR A
        MOV R5, A
AGAIN:  ADD A, @R1
        JNC NEXT
        INC R5
NEXT    INC R1
        DJNZ R2, AGAIN
HERE    SJMP HERE
```

Example 5: Checksum computation

Checksum computation of 50 numbers

```
        MOV R1, #30H        ; Array address
        MOV R2, #50         ; Length of array
        CLR A
AGAIN:  XRL A, @R1          ; Checksum it
        INC R1
        DJNZ R2, AGAIN      ; decrement length
        MOV R3, A           ; save results
STOP:   SJMP STOP
```

Example 6: Sequential Search

Sequential Search

```
NUM       EQU 49H                    ; number to search
          MOV R2, #50                ; length of array
          MOV R0, #40H               ; start address of array
NEXT:     MOV A, @R0                      ; read data element
          CJNE A, #NUM, FORWARD ; compare, no, go ahead
          MOV R3, 0                   ; found , save address & stop
          SJMP STOP
FORWARD: INC R0                      ; go for next element
          DJNZ R2, NEXT              ; decrement length & repeat
          CLR A                      ; at end = not found so put 0
STOP:     SJMP STOP
```

Example 7: Bubble Sort

```
    Bubble Sort
            MOV R2, #50
LOOP2:      MOV R0, #40H
            MOV R3, 2
LOOP1:      DEC R3
            CLR C
            MOV A, @R0
            INC R0
            MOV R6, A
            SUBB A, @R0
            JC NEXT
            MOV A, R6
            XCH A, @R0
            DEC R0
            MOV @R0, A
            INC R0
NEXT:       CJNE R3, #0, LOOP1
            DEC R2
            CJNE R2, #0, LOOP2
STOP:       SJMP STOP
```

Example 7: Bubble Sort

Bubble Sort

```
        MOV R2, #50         ; length (counter)
LOOP2:  MOV R0, #40H        ; start address
        MOV R3, 2           ; get count
LOOP1:  DEC R3              ; comparision counter
        CLR C
        MOV A, @R0          ; get data
        INC R0
        MOV R6, A
        SUBB A, @R0         ; try subtraction
        JC NEXT             ; check if first number < 2nd
```

Example 7: Bubble Sort

```
            MOV A, R6           ; swap bytes
            XCH A, @R0
            DEC R0
            MOV @R0, A
            INC R0              ; point to next byte
NEXT:  CJNE R3, #0, LOOP1
            DEC R2              ; decrement pass count
       CJNE R2, #0, LOOP2
STOP:       SJMP STOP
```

Machine control Examples

Program Branching Instructions

- Program branching instructions are used to control the flow of program execution

- Some instructions provide decision making capabilities before transferring control to other parts of the program (conditional branches).

Mnemonic	Description
ACALL addr11	Absolute subroutine call
LCALL addr16	Long subroutine call
RET	Return from subroutine
RETI	Return from interrupt
AJMP addr11	Absolute jump
LJMP addr16	Long jump
SJMP rel	Short jump
JMP @A+DPTR	Jump indirect
JZ rel	Jump if A=0
JNZ rel	Jump if A NOT=0
CJNE A,direct,rel	Compare and Jump if Not Equal
CJNE A,#data,rel	
CJNE Rn,#data,rel	
CJNE @Ri,#data,rel	
DJNZ Rn,rel	Decrement and Jump if Not Zero
DJNZ direct,rel	
NOP	No Operation

Example 1: Subroutine Call

Delay Routine

```
        ORG 0
0000    MOV A, #55H
0002    MOV R4, #80H
0004    MOV R6, #98H
0006    LCALL DELAY
0009    ADD A, R4
        . . .
        . . .
```

```
        ORG 200H
DELAY:  MOV R4, #0DEH
LOOP:   MOV R6, #0FEH
HERE:   DJNZ R6, HERE
        DJNZ R4,LOOP
        RET
```

Example 1: Subroutine Call (Contd..)

Delay Routine

```
        ORG 0
0000    MOV A, #55H
00002   MOV R4, #80H
0004    MOV R6, #98H
0006    LCALL DELAY
0009    ADD A, R4
        . . .
        . . .
```

```
        ORG 200H
DELAY:  PUSH 4
        PUSH 6
        MOV R4, #0DEH
LOOP:   MOV R6, #0FEH
HERE:   DJNZ R6, HERE
        DJNZ R4, LOOP
        POP 6
        POP 4
        RET
```

Example 2: Parameter Passing using Registers

```
                ORG 0
LOOP:           MOV A, #00H          ; turn off LED
                MOV P1, A
                MOV R2, #200         ; delay count in register R2
                LCALL DELAY          ; call routine
                MOV A, #FFH          ; turn on LED
                MOV P1, A
                LCALL DELAY          ; call routine
                SJMP LOOP            ; go on forever
                ORG 100H
DELAY:          DJNZ R2, DELAY       ; decrement
                RET                  ; go abck
                END
```

Example 3: Parameter Passing using Memory

Linear Search

```
        ORG 0
NUM  EQU 45H
        MOV R2, #50 ; length
        MOV R0, #40H ; start
        MOV R3,#0   ; flag-as-fail
        LCALL SEARCH ; call
         MOV A, R3   ; save
                . . .
                . . .
```

```
        ORG 200H
SEARCH: MOV A, @R0  ; get
        CJNE A, #NUM, NEXT ; chk
        LJMP STOP ;match stop
NEXT:    INC R0     ; if not next
        DJNZ R2, SEARCH ; go on
STOP:   MOV R3, 0 ; point to result
            RET
```

Example 4: Parameter Passing using Stack

```
Delay
ORG 0
MOV R4, #96H
MOV R6, #48H
PUSH 4
PUSH 6
LCALL DELAY
. . .
. . .
        . . .
```

```
        ORG 200H
DELAY:  POP 1
        POP 3
LOOP:   MOV R2, 3
HERE:   DJNZ R2, HERE
        DJNZ R1, LOOP
        RET
```

Example 4: Parameter Passing using Stack…

```
Delay
ORG 0
MOV R4, #96H
MOV R6, #48H
PUSH 4
PUSH 6
LCALL DELAY
. . .
. . .
. . .
```

```
ORG 200H
DELAY:  DEC SP
        DEC SP
        POP 1
        POP 3
LOOP:   MOV R2, 3
HERE:   DJNZ R2, HERE
        DJNZ R1, LOOP
        INC SP
        INC SP
        RET
```

Example 4: Parameter Passing using Stack…

```
Delay
ORG 0
MOV R4, #96H
MOV R6, #48H
PUSH 4
PUSH 6
LCALL DELAY
. . .
. . .
. . .
```

```
        ORG 200H
DELAY:  DEC SP
        DEC SP
        POP 1
        POP 3
LOOP:   MOV R2, 3
HERE:   DJNZ R2, HERE
        DJNZ R1, LOOP
        INC SP
        INC SP
        INC SP
        INC SP
        RET
```

Thank You

???

- ???