Microprocessors and Microcontrollers

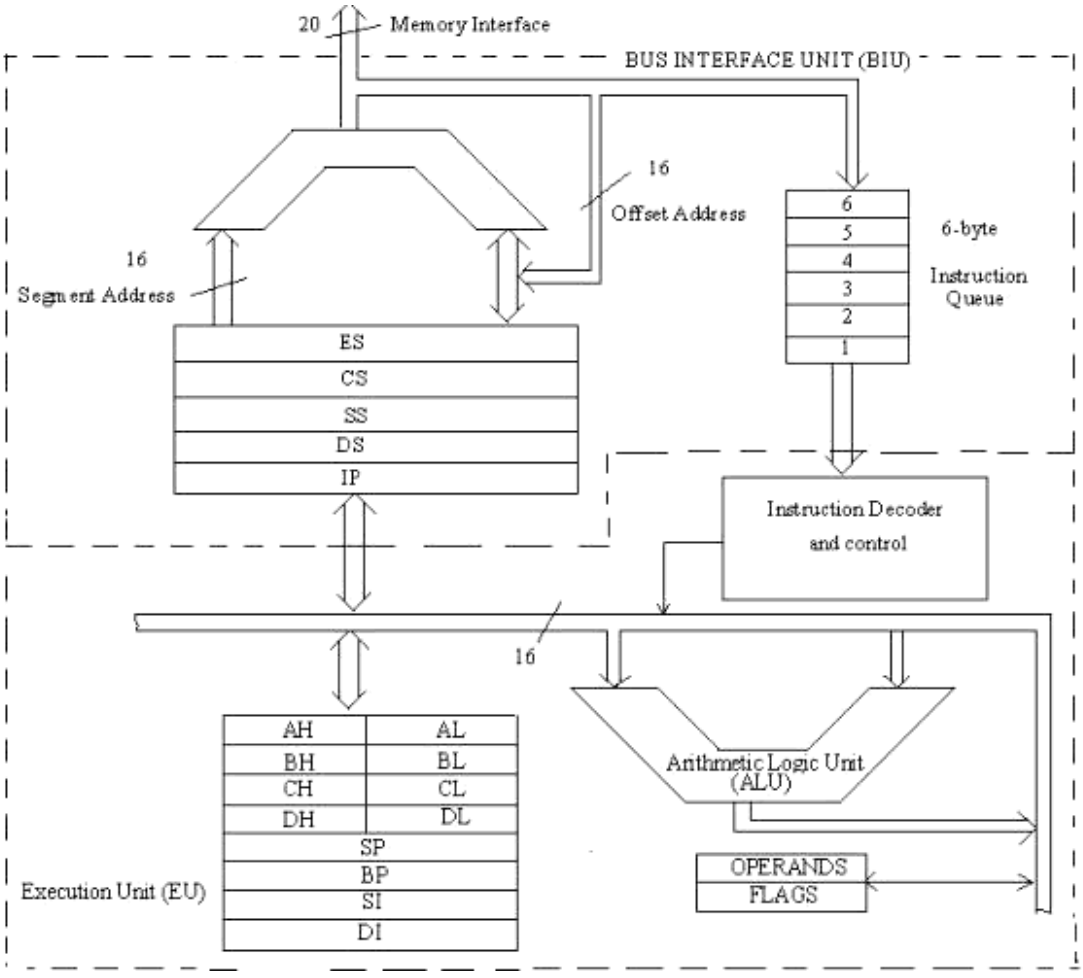
Unit I

Syllabus: 8086 Architecture: 8086 Architecture-Functional diagram, Register Organization, Memory Segmentation, Programming Model, Memory addresses, Physical Memory Organization, Signal descriptions of 8086- Common Function Signals, Timing diagrams, Interrupts of 8086.

8086 Architecture:

Intel 8086 is a 16-bit integer processor. It has 16-bit data bus and 20-bit address bus. The lower 16-bit address lines and 16-bit data lines are multiplexed (AD0-AD15). Since 20-bit address lines are available, 8086 can access up to 2^{20} or 1 Giga byte of physical memory.

The internal architecture of Intel 8086 is divided into two units, viz., Bus Interface Unit (BIU) and Execution Unit (EU).



Bus Interface Unit (BIU)

The Bus Interface Unit (BIU) generates the 20-bit physical memory address and provides the interface with external memory (ROM/RAM). As mentioned earlier, 8086 has a single memory interface. To speed up the execution, 6-bytes of instruction are fetched in advance

and kept in a 6-byte Instruction Queue while other instructions are being executed in the Execution Unit (EU). Hence after the execution of an instruction, the next instruction is directly fetched from the instruction queue without having to wait for the external memory to send the instruction. This is called pipe-lining and is helpful for speeding up the overall execution process.

8086's BIU produces the 20-bit physical memory address by combining a 16-bit segment address with a 16-bit offset address. There are four 16-bit segment registers, viz., the code segment (CS), the stack segment (SS), the extra segment (ES), and the data segment (DS). These segment registers hold the corresponding 16-bit segment addresses. A segment address is the upper 16-bits of the starting address of that segment. The lower 4-bits of the starting address of a segment is always zero. The offset address is held by another 16-bit register. The physical 20-bit address is calculated by shifting the segment address 4-bit left and then adding that to the offset address.

For Example:
Code segment Register CS holds the segment address which is 4569 H Instruction pointer IP holds the offset address which is 10A0 H The physical 20-bit address is calculated as follows.

Segment address	:	45690 H
Offset address	:	+ 10A0 H
Physical address	:	46730 H

Register Organization

The registers AX, BX, CX, and DX are the general 16-bit registers.

AX Register: Accumulator register consists of two 8-bit registers AL and AH, which can be combined together and used as a 16- bit register AX. AL in this case contains the low-order byte of the word, and AH contains the high-order byte. Accumulator can be used for I/O operations, rotate and string manipulation.

BX Register: This register is mainly used as a base register. It holds the starting base location of a memory region within a data segment. It is used as offset storage for forming physical address in case of certain addressing mode.

CX Register: It is used as default counter or count register in case of string and loop instructions.

DX Register: Data register can be used as a port number in I/O operations and implicit operand or destination in case of few instructions. In integer 32-bit multiply and divide instruction the DX register contains high-order word of the initial or resulting number.

Segment registers:

To complete 1Mbyte memory is divided into 16 logical segments. The complete 1Mbyte memory segmentation is as shown in fig 1.5. Each segment contains 64Kbyte of memory. There are four segment registers.

Code segment (CS) is a 16-bit register containing address of 64 KB segment with processor instructions. The processor uses CS segment for all accesses to instructions referenced by instruction pointer (IP) register. CS register cannot be changed directly. The CS register is automatically updated during far jump, far call and far return instructions. It is used for addressing a memory location in the code segment of the memory, where the executable program is stored.

Stack segment (SS) is a 16-bit register containing address of 64KB segment with program stack. By default, the processor assumes that all data referenced by the stack pointer (SP) and base pointer (BP) registers is located in the stack segment. SS register can be changed directly using POP instruction. It is used for addressing stack segment of memory. The stack segment is that segment of memory, which is used to store stack data.

Data segment (DS) is a 16-bit register containing address of 64KB segment with program data. By default, the processor assumes that all data referenced by general registers (AX, BX, CX, DX) and index register (SI, DI) is located in the data segment. DS register can be changed directly using POP and LDS instructions. It points to the data segment memory where the data is resided.

Extra segment (ES) is a 16-bit register containing address of 64KB segment, usually with program data. By default, the processor assumes that the DI register references the ES segment in string manipulation instructions. ES register can be changed directly using POP and LES instructions. It also refers to segment which essentially is another data segment of the memory. It also contains data.

Pointers and index registers.

The pointers contain within the particular segments. The pointers IP, BP, SP usually contain offsets within the code, data and stack segments respectively

Stack Pointer (SP) is a 16-bit register pointing to program stack in stack segment.

Base Pointer (BP) is a 16-bit register pointing to data in stack segment. BP register is usually used for based, based indexed or register indirect addressing.

Source Index (SI) is a 16-bit register. SI is used for indexed, based indexed and register indirect addressing, as well as a source data addresses in string manipulation instructions.

Destination Index (DI) is a 16-bit register. DI is used for indexed, based indexed and register indirect addressing, as well as a destination data address in string manipulation instructions.

Conditional Flags

Conditional flags are as follows:

Carry Flag (CY): This flag indicates an overflow condition for unsigned integer arithmetic. It is also used in multiple-precision arithmetic.

Auxiliary Flag (AC): If an operation performed in ALU generates a carry/barrow from lower nibble (i.e. D0 – D3) to upper nibble (i.e. D4 – D7), the AC flag is set i.e. carry given by D3 bit to D4 is AC flag. This is not a general-purpose flag, it is used internally by the Processor to perform Binary to BCD conversion.

Parity Flag (PF): This flag is used to indicate the parity of result. If lower order 8-bits of the result contains even number of 1's, the Parity Flag is set and for odd number of 1's, the Parity flag is reset.

Zero Flag (ZF): It is set; if the result of arithmetic or logical operation is zero else it is reset.

Sign Flag (SF): In sign magnitude format the sign of number is indicated by MSB bit. If the result of operation is negative, sign flag is set.

Control Flags

Control flags are set or reset deliberately to control the operations of the execution unit.

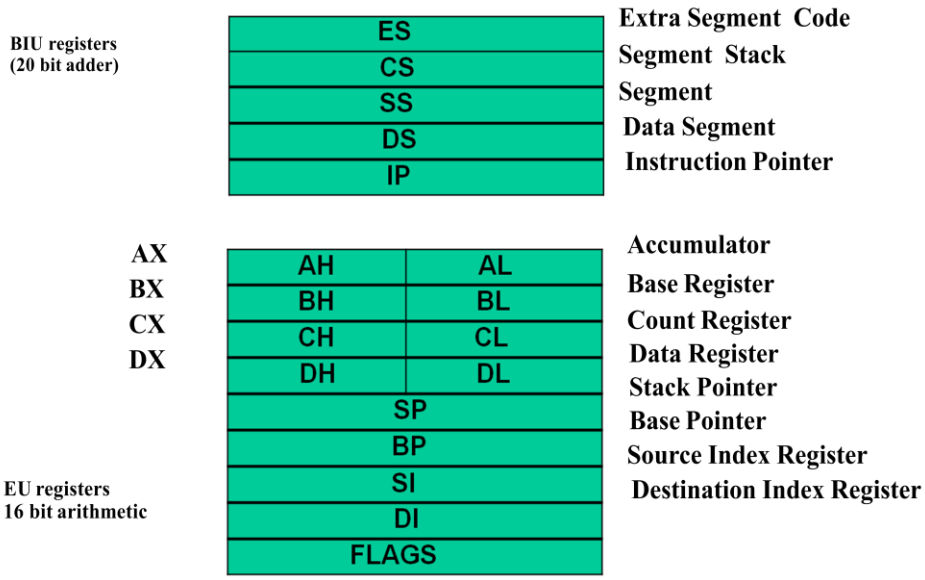
Control flags are as follows:

Trap Flag (TF): It is used for single step control. It allows user to execute one instruction of a program at a time for debugging. When trap flag is set, program can be run in single step mode.

Interrupt Flag (IF): It is an interrupt enable/disable flag. If it is set, the maskable interrupt of 8086 is enabled and if it is reset, the interrupt is disabled. It can be set by executing instruction sit and can be cleared by executing CLI instruction.

Direction Flag (DF): It is used in string operation. If it is set, string bytes are accessed from higher memory address to lower memory address. When it is reset, the string bytes are accessed from lower memory address to higher memory address.

8086 Programmer's Model

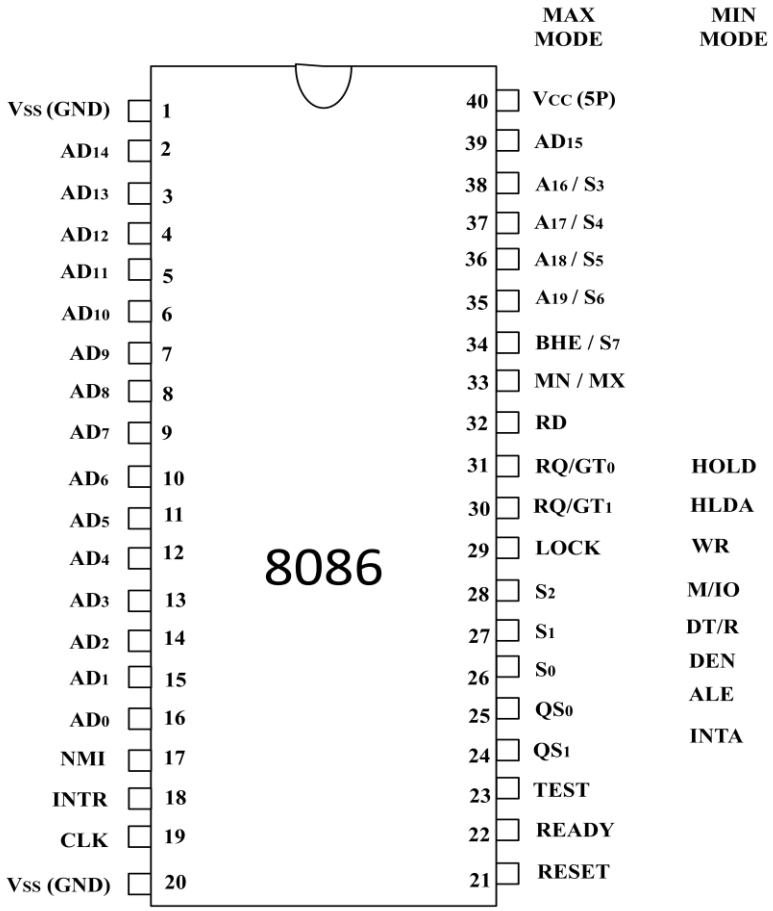


Memory Segmentation

In memory the data is stored as bytes. Each byte considered as memory location. (Here study about segment registers which is discussed above)

Signal Descriptions of 8086

Important 8086 Pin Diagram/Description



The 8086 Microprocessor is a 16-bit CPU available in 3 clock rates, i.e. 5, 8 and 10MHz, packaged in a 40 pin CERDIP or plastic package. The 8086 Microprocessor operates in single processor or multiprocessor configurations to achieve high performance. The pin configuration is as shown in fig1. Some of the pins serve a particular function in minimum mode (single processor mode) and others function in maximum mode (multiprocessor mode) configuration.

The 8086 signals can be categorized in three groups. The first are the signals having common functions in minimum as well as maximum mode, the second are the signals which have special functions in minimum mode and third are the signals having special functions for maximum mode

The following signal descriptions are **common** for both the **minimum and maximum modes**.

AD15-AD0: These are the time multiplexed memory I/O address and data lines. Address remains on the lines during T1 state, while the data is available on the data bus during T2, T3, TW and T4. Here T1, T2, T3, T4 and TW are the clock states of a machine cycle. TW is a wait state. These lines are active high and float to a tristate during interrupt acknowledge and local bus hold acknowledge cycles.

A19/S6, A18/S5, A17/S4, A16/S3: These are the time multiplexed address and status lines. During T1, these are the most significant address lines or memory operations. During I/O operations, these lines are low. During memory or I/O operations, status information is available on those lines for T2, T3, TW and T4. The status of the interrupt enable flag bit(displayed on S5) is updated at the beginning of each clock cycle. The S4 and S3 combinedly indicate which segment register is presently being used for memory accesses as shown in Table below.

S4	S3	Indication
0	0	Alternate Data
0	1	Stack
1	0	Code or none
1	1	Data

These lines float to tri-state off (tristated) during the local bus hold acknowledge. The status line S6 is always low (logical). The address bits are separated from the status bits using latches controlled by the ALE signal.

BHE/S7-Bus High Enable/Status: The bus high enable signal is used to indicate the transfer of data over the higher order (D15-D8) data bus as shown in Table below. It goes low for the data transfers over D15-D8 and is used to derive chip selects of odd address memory bank or peripherals. BHE is low during T1 for read, write and interrupt acknowledge cycles, whenever a byte is to be transferred on the higher byte of the data bus. The status information is available during T2, T3 and T4. The signal is active low and is tri-stated during 'hold'. It is low during T1 clock pulse of the interrupt acknowledges.

BHE	A0	Indication
0	0	Whole Word
0	1	Upper Byte from or to odd address
1	0	Lower Byte from or to Even address
1	1	None

RD: Read: Read signal, when low, indicates the peripherals that the processor is performing a memory or I/O read operation. RD is active low and shows the state for T2, T3, TW of any read cycle. The signal remains tri-stated during the 'hold acknowledge'.

READY: This is the acknowledgement from the slow devices or memory that they have completed the data transfer.

INTR: Interrupt Request: This is a level triggered input. This is sampled during the last clock cycle of each instruction to determine the availability of the request. If any interrupt request is pending, the processor enters the interrupt acknowledge cycle. This can be internally masked by resetting the interrupt enable flag. This signal is active high and internally synchronized.

TEST: This input is examined by a 'WAIT' instruction. If the TEST input goes low, execution will continue, else, the processor remains in an idle state. The input is synchronized internally during each clock cycle on leading edge of clock.

NMI: Non-maskable Interrupt: This is an edge-triggered input which causes a Type2 interrupt. The NMI is not maskable internally by software. A transition from low to high initiates the interrupt response at the end of the current instruction. This input is internally synchronized.

RESET: This input causes the processor to terminate the current activity and start execution from FFFF0H. The signal is active high and must be active for at least four clock cycles. It restarts execution when the RESET returns low. RESET is also internally synchronized.

CLK: Clock Input: The clock input provides the basic timing for processor operation and bus control activity. Its an asymmetric square wave with 33% duty cycle. The range of frequency for different 8086 versions is from 5MHz to 10MHz.

VCC: +5V power supply for the operation of the internal circuit. GND ground for the internal circuit.

MN/MX: The logic level at this pin decides whether the processor is to operate in either minimum (single processor) or maximum (multiprocessor) mode.

The following pin functions are for the minimum mode operation of 8086.

M/IO -Memory/IO: This is a status line logically equivalent to S2 in maximum mode. When it is low, it indicates the CPU is having an I/O operation, and when it is high, it indicates that the CPU is having a memory operation. This line becomes active in the previous T4 and remains active till final T4 of the current cycle. It is tri-stated during local bus "hold acknowledge".

INTA: Interrupt Acknowledge: This signal is used as a read strobe for interrupt acknowledge cycles. In other words, when it goes low, it means that the processor has accepted the interrupt. It is active low during T2, T3 and TW of each interrupt acknowledge cycle.

ALE-Address latch Enable: This output signal indicates the availability of the valid address on the address/data lines, and is connected to latch enable input of latches. This signal is active high and is never tri-stated.

DT /R -Data Transmit/Receive: This output is used to decide the direction of data flow through the transceivers (bidirectional buffers). When the processor sends out data, this signal is high and when the processor is receiving data, this signal is low. Logically, this is

equivalent to S1 in maximum mode. Its timing is the same as M/I/O. This is tri-stated during 'hold acknowledge'.

DEN-Data Enable: This signal indicates the availability of valid data over the address/data lines. It is used to enable the transceivers (bidirectional buffers) to separate the data from the multiplexed address / data signal. It is active from the middle ofT2 until the middle of T4 DEN is tri-stated during 'hold acknowledge' cycle.

HOLD, HLDA-Hold/Hold Acknowledge: When the HOLD line goes high it indicates to the processor that another master is requesting the bus access. The processor, after receiving the HOLD request, issues the hold acknowledge signal on HLDA pin, in the middle of the next clock cycle after completing the current bus (instruction) cycle. At the same time, the processor floats the local bus and control lines. When the processor detects the HOLD line low, it lowers the HLDA signal. HOLD is an asynchronous input, and it should be externally synchronized. If the DMA request is made while the CPU is performing a memory or I/O cycle, it will release the local bus during T 4 provided:

1. The request occurs on or before T 2 state of the current cycle.

2. The current cycle is not operating over the lower byte of a word (or operating on an odd address).

3. The current cycle is not the first acknowledge of an interrupt acknowledge sequence.

4. A Lock instruction is not being executed.

The following pin functions are applicable for maximum mode operation of 8086.

S2, S1, S0 -Status Lines: These are the status lines which reflect the type of operation, being carried out by the processor. These become active during T4 of the previous cycle and remain active during T1 and T2 of the current bus cycle. The status lines return to passive state during T3 of the current bus cycle so that they may again become active for the next bus cycle during T4. Any change in these lines during T3 indicates the starting of a new cycle, and return to passive state indicates end of the bus cycle. These status lines are encoded in table below.

S2	S1	S0	Indication
0	0	0	Interrupt Acknowledge
0	0	1	Read I/O Port
0	1	0	Write I/O Port
0	1	1	Halt
1	0	0	Code Access
1	0	1	Read Memory
1	1	0	Write memory
1	1	1	Passive

LOCK: This output pin indicates that other system bus masters will be prevented from gaining the system bus, while the LOCK signal is low. The LOCK signal is activated by the 'LOCK' prefix instruction and remains active until the completion of the next instruction. This floats to tri-state off during "hold acknowledge". When the CPU is executing a critical instruction which requires the system bus, the LOCK prefix instruction ensures that other processors connected in the system will not gain the control of the bus. The 8086, while executing the prefixed instruction, asserts the bus lock signal output, which may be connected to an external bus controller.

QS1, QS0-Queue Status: These lines give information about the status of the code prefetch queue. These are active during the CLK cycle after which the queue operation is performed. These are encoded as shown in Table below.

QS1	QS0	Indication
0	0	No Operation
0	1	First byte of opcode from the queue
1	0	Empty Queue
1	1	Subsequent byte from the queue

This modification in a simple fetch and execute architecture of a conventional microprocessor offers an added advantage of pipelined processing of the instructions. The 8086 architecture has a 6-byte instruction prefetch queue. Thus even the largest (6- bytes) instruction can be prefetched from the memory and stored in the prefetch queue. This results in a faster execution of the instructions. This scheme is known as instruction pipelining. At the starting the CS:IP is loaded with the required address from which the execution is to be started. Initially, the queue will be empty and the microprocessor starts a fetch operation to bring one byte (the first byte) of instruction code, if the CS:IP address is odd or two bytes at a time, if the CS:IP address is even. The first byte is a complete opcode in case of some instructions (one byte opcode instruction) and it is a part of opcode, in case of other instructions (two byte long opcode instructions), the remaining part of opcode may lie in the second byte. But invariably the first byte of an instruction is an opcode. These opcodes along with data are fetched and arranged in the queue. When the first byte from the queue goes for decoding and interpretation, one byte in the queue becomes empty and subsequently the queue is updated. The microprocessor does not perform the next fetch operation till at least two bytes of the instruction queue are emptied. The instruction execution cycle is never broken for fetch operation. After decoding the first byte, the decoding circuit decides whether the instruction is of single opcode byte or double opcode byte. If it is single opcode byte, the next bytes are treated as data bytes depending upon the decoded instruction length otherwise, the next byte in the queue is treated as the second byte of the instruction opcode. The second byte is then decoded in continuation with the first byte to decide the instruction length and the number of subsequent bytes to be treated as instruction data. The queue is updated after every byte is read from the queue but the fetch cycle is initiated by BIU only if at least, two bytes of the queue are empty and the EU may be concurrently executing the fetched instructions. The next byte after the instruction is completed is again the first opcode byte of the next instruction. A

similar procedure is repeated till the complete execution of the program. The main point to be noted here is that the fetch operation of the next instruction is overlapped with the execution of the current instruction. As shown in the architecture, there are two separate units, namely, execution unit and bus interface unit. While the execution unit is busy in executing an instruction, after it is completely decoded, the bus interface unit may be fetching the bytes o(the next instruction from memory, depending upon the queue status.

RQ/GT0, RQ/GT1-ReQuest/Grant: These pins are used by other local bus masters, in maximum mode, to force the processor to release the local bus at the end of the processor's current bus cycle. Each of the pins is bidirectional with RQ/GT0 having higher priority than RQ/ GT1, RQ/GT pins have internal pull-up resistors and may be left unconnected. The request grant sequence is as follows:

1. A pulse one clock wide from another bus master requests the bus access to 8086.

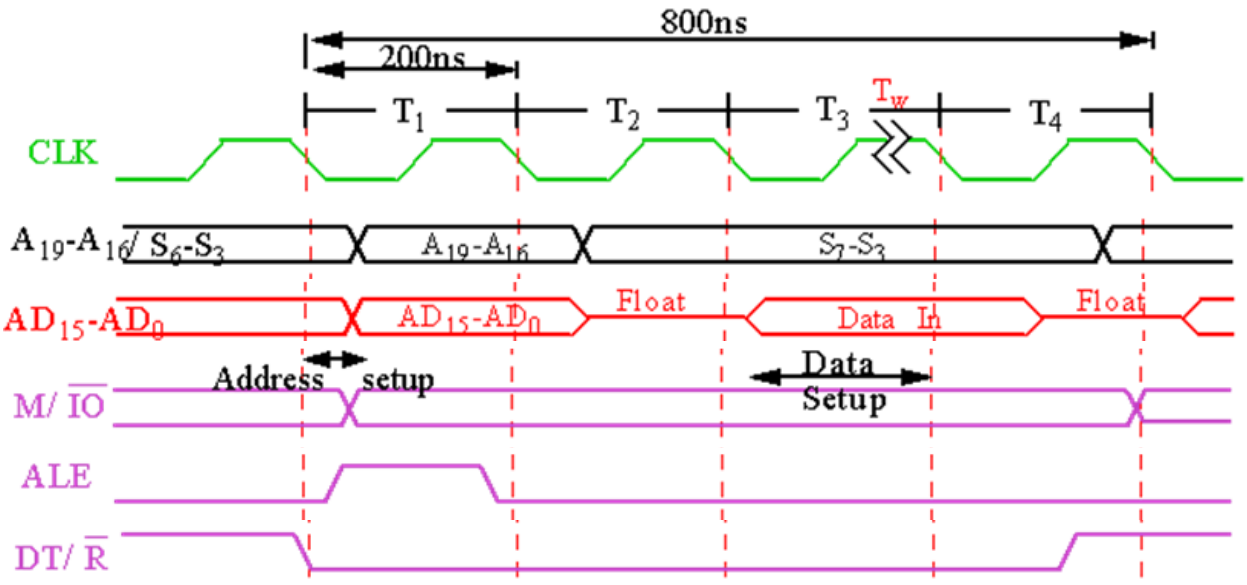
2. During T4 (current) or T1 (next) clock cycle, a pulse one clock wide from 8086 to the requesting master, indicates that the 8086 has allowed the local bus to float and that it will enter the "hold acknowledge" state at next clock cycle. The CPU's bus interface unit is likely to be disconnected from the local bus of the system.

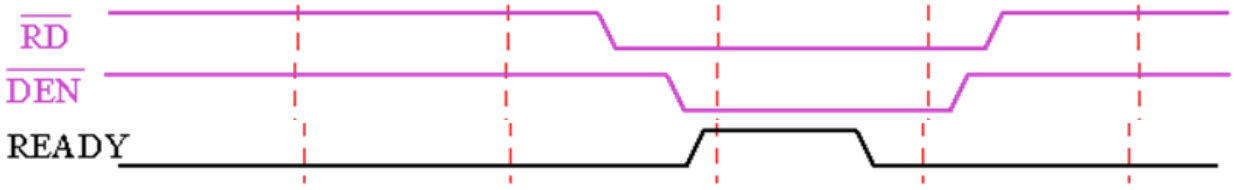
3. A one clock wide pulse from the another master indicates to 8086 that the 'hold' request is about to end and the 8086 may regain control of the local bus at the next clock cycle.

Thus each master to master exchange of the local bus is a sequence of 3 pulses. There must be at least one dead clock cycle after each bus exchange. The request and grant pulses are active low. For the bus requests those are received while 8086 is performing memory or I/O cycle, the granting of the bus is governed by the rules as discussed i~ case of HOLD, and HLDA in minimum mode.

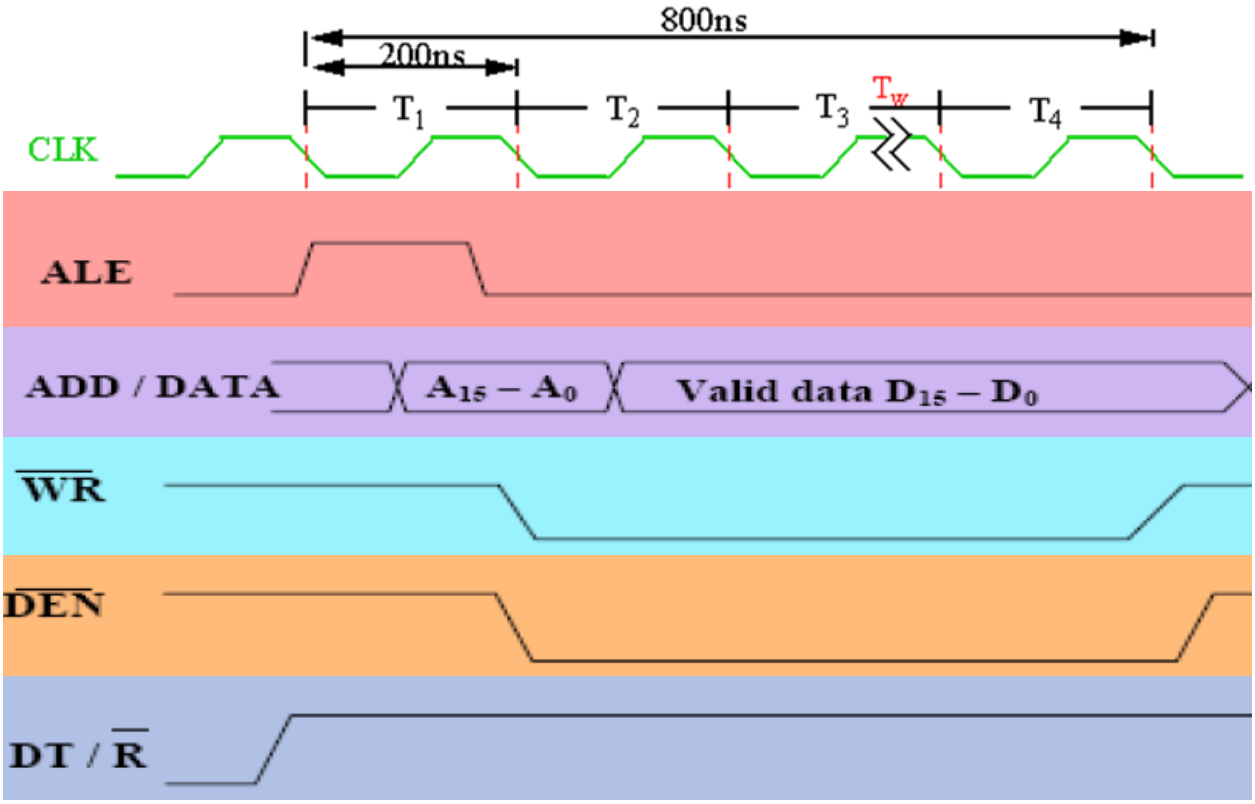
Timing Diagram:

Minimum Mode timing diagram Read operation

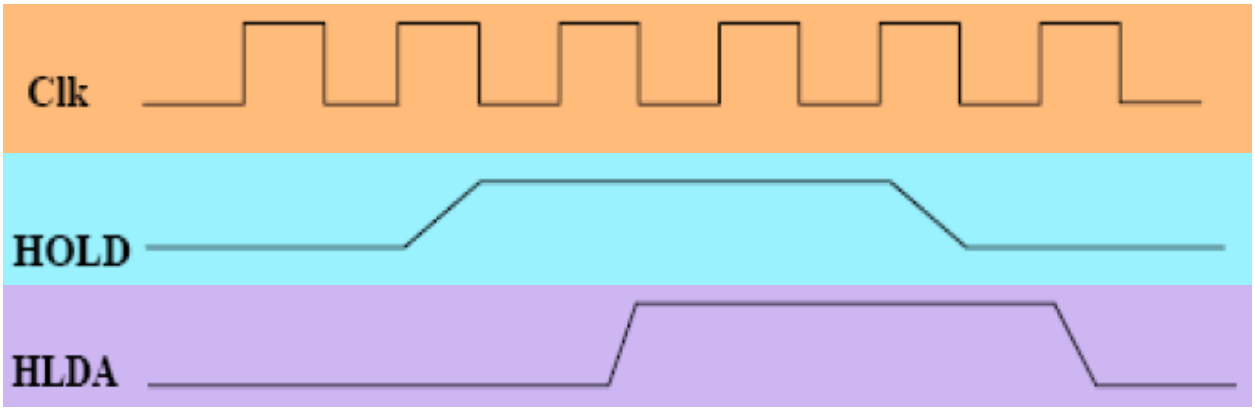




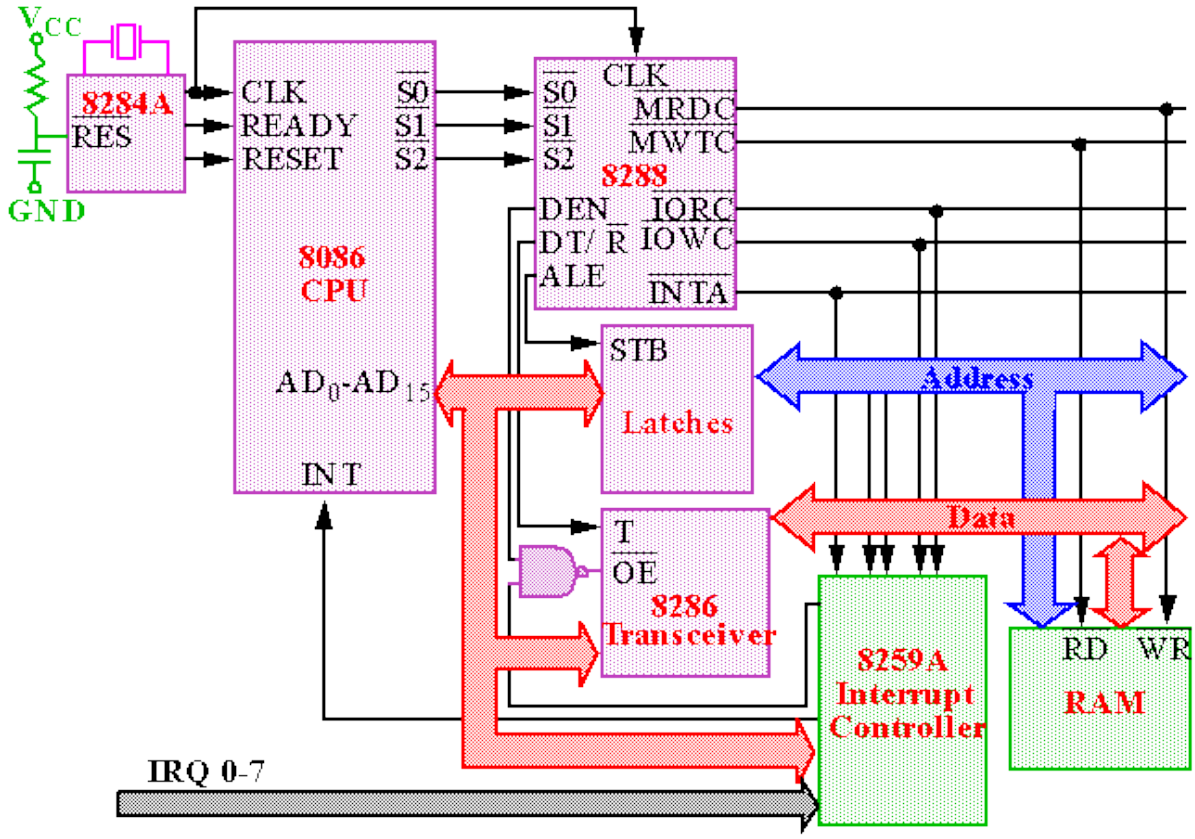
Minimum Mode timing diagram Write operation



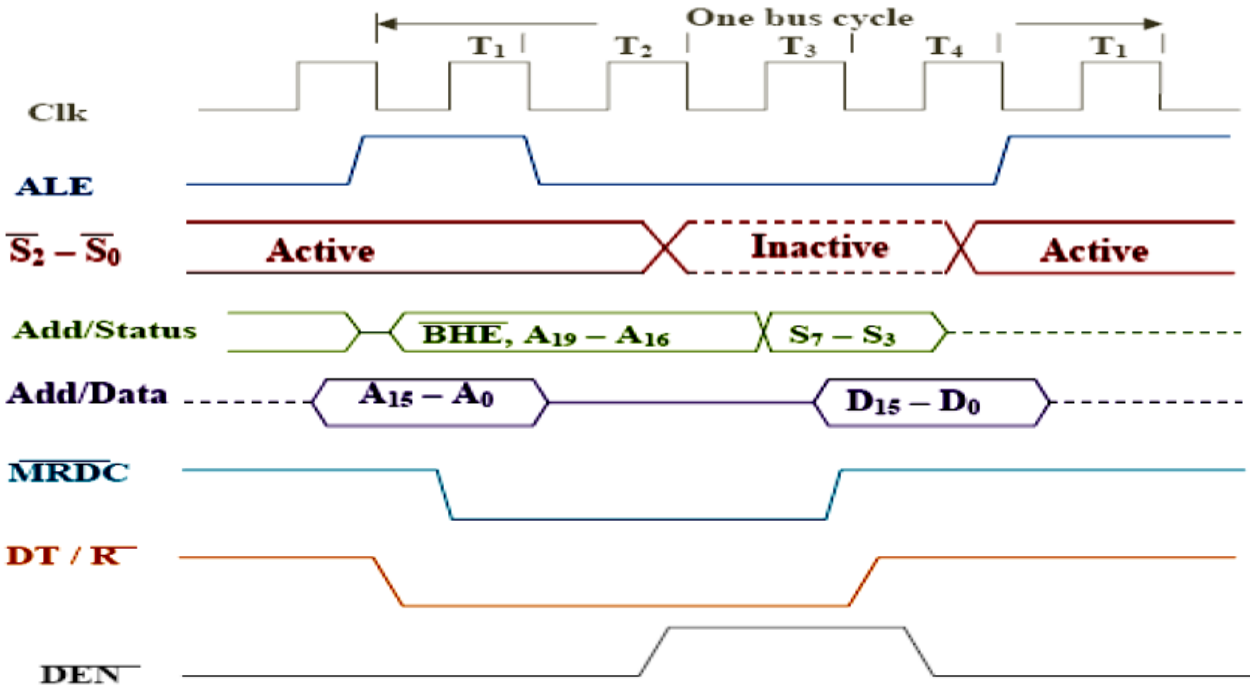
Hold response in min. mode



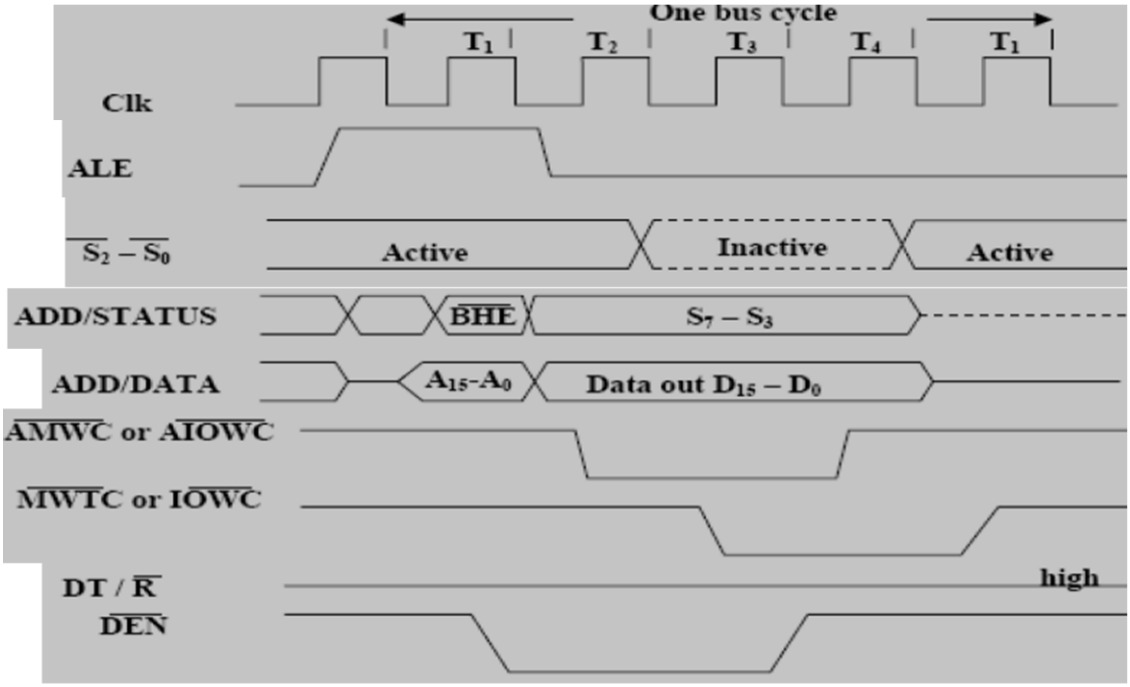
Maximum mode of 8086



Maximum mode Memory Read Operation



Maximum mode Write Operation



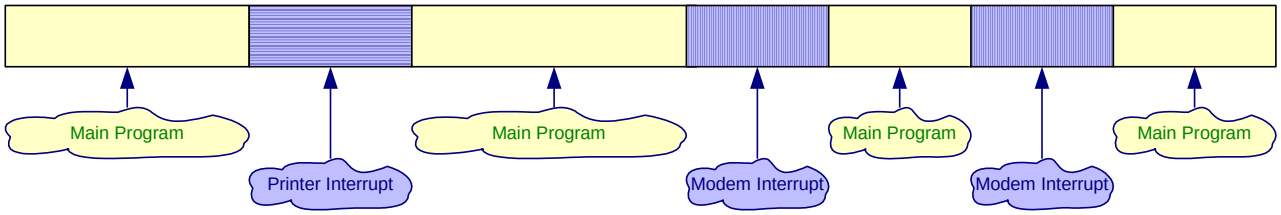
Interrupts of 8086

- Interrupt Types
    - Hardware Interrupts: External event
    - Software Interrupts: Internal event (Software generated)
    - Maskable and non-maskable interrupts
    - Interrupt priority
- Interrupt Vectors and Interrupt Handlers
- Interrupt Controllers

The Purpose of Interrupts

- Interrupts are useful when interfacing I/O devices with low data-transfer rates, like a keyboard or a mouse, in which case polling the device wastes valuable processing time
- The peripheral interrupts the normal application execution, requesting to send or receive data.
- The processor jumps to a special program called *Interrupt Service Routine* to service the peripheral

- After the processor services the peripheral, the execution of the interrupted program continues.



Interrupt pins: Set of pins used in hardware interrupts

Interrupt Service Routine (ISR) or Interrupt handler: code used for handling a specific interrupt

Interrupt priority: In systems with more than one interrupt inputs, some interrupts have a higher priority than other

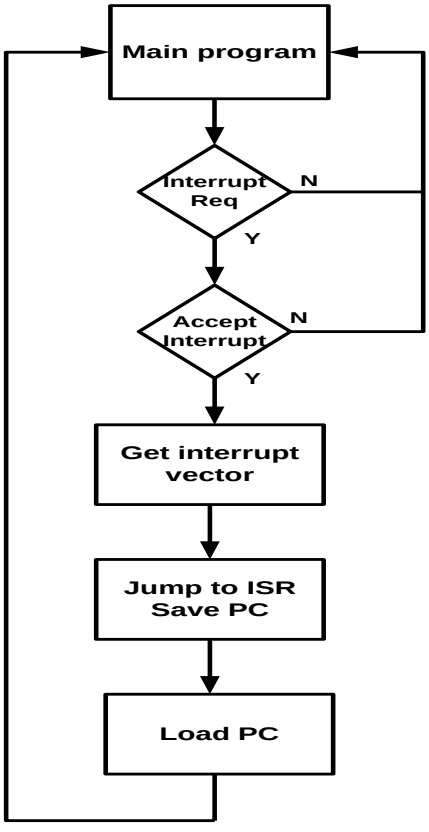
They are serviced first if multiple interrupts are triggered simultaneously

Interrupt vector: Code loaded on the bus by the interrupting device that contains the Address (segment and offset) of specific interrupt service routine

Interrupt Masking: Ignoring (disabling) an interrupt

Non-Maskable Interrupt: Interrupt that cannot be ignored (power-down)

Interrupt processing flow



Interrupt Vectors

- The processor uses the interrupt vector to determine the address of the ISR of the interrupting device.
- In the 8088/8086 processor as well as in the 80386/80486/Pentium processors operating in Real Mode (16-bit operation), the interrupt vector is a pointer to the Interrupt Vector Table.
    - The Interrupt Vector Table occupies the address range from 00000H to 003FFH (the first 1024 bytes in the memory map).
    - Each entry in the Interrupt Vector Table is 4 bytes long:
        - The first two represent the offset address and the last two the segment address of the ISR.
    - The first 5 vectors are reserved by Intel to be used by the processor.
    - The vectors 5 to 255 are free to be used by the user.
- The *Interrupt Vector* contains the address of the interrupt service routine
- The *Interrupt Vector Table* is located in the first 1024 bytes of memory at address 000000H-0003FFH.
- It contains 256 different 4-byte interrupt vectors, grouped in 18 types

    000H: Type 0 (Divide error)

    004H: Type 1 (Single-step)

    008H: Type 2 (NMI)

    00CH: Type 3 (1-byte breakpoint)

    010H: Type 4 (Overflow)

    014H: Type 5 (BOUND)

    018H: Type 6 (Undefined opcode)

    01CH: Type 7 (Coprocessor not available)

    020H: Type 8 (Double fault)

    024H: Type 9 (Coprocessor segment overrun)

    028H: Type 10 (Invlid task state segment)

    02CH: Type 11 (Segment not present)

030H: Type 12 (Stack segment overrun)

034H: Type 13 (General protection)

038H: Type 14 (Page fault)

03CH: Type 15 (Unassigned)

040H: Type 16 (Coprocessor error)

044H-07CH: Type 14-31 (Reserved)

080H: Type 32-255 (User)

1. Type 0: Divide error – Division overflow or division by zero
2. Type 1: Single step or Trap – After the execution of each instruction when trap flag set
3. Type 2: NMI Hardware Interrupt – '1' in the NMI pin
4. Type 3: One-byte Interrupt – INT3 instruction (used for breakpoints)
5. Type 4: Overflow – INTO instruction with an overflow flag
6. Type 5: BOUND – Register contents out-of-bounds
7. Type 6: Invalid Opcode – Undefined opcode occurred in program
8. Type 7: Coprocessor not available – MSW indicates a coprocessor
9. Type 8: Double Fault – Two separate interrupts occur during the same instruction
10. Type 9: Coprocessor Segment Overrun – Coprocessor call operand exceeds FFFFH
11. Type 10: Invalid Task State Segment – TSS invalid (probably not initialized)
12. Type 11: Segment not present – Descriptor P bit indicates segment not present or invalid
13. Type 12: Stack Segment Overrun – Stack segment not present or exceeded
14. Type 13: General Protection – Protection violation in 286 (general protection fault)
15. Type 14: Page Fault – 80386 and above
16. Type 16: Coprocessor Error – ERROR´ = '0' (80386 and above)
17. Type 17: Alignment Check – Word / Double word data addressed at odd location (486 and above)
18. Type 18: Machine Check – Memory Management interrupt (Pentium and above)

UNIT -II:

Instruction Set and Assembly Language Programming of 8086:

8086 INSTRUCTIONS

An instruction given to the computer it performs a specified operation on given data. The instruction set of a microprocessor is the collection of the instructions that the microprocessor is designed to execute.

The instructions of Intel 8086 have been classified into the following groups.

1. Data transfer instructions.

2. Arithmetic instructions.

3. Bit manipulation (logical) instructions.

4. String instructions.

5. Program execution transfer instructions.

6. Processor control instructions.

DATA TRANSFER INSTRUCTIONS

General purpose byte or word transfer instructions:

MOV	Copy byte or word from specified source to specified destination.
PUSH	Copy specified word to top of stack
POP	Copy word from top of stack to specified location.
XCHG	Exchange bytes or exchange words.
XLAT	Translate a byte in AL using a table in memory.

Simple input and output port transfer instructions:

IN	Copy a byte or word from specified port to accumulator.
OUT	Copy a byte or word from accumulator to specified port.

Special address transfer instructions:

LEA	Load effective address of operand into specified register.
LDS	Load DS register and other specified registers from memory.
LES	Load ES register and other specified register from memory.

Flag transfer instructions:

LAHF Load (copy to) AH with the low byte of the flag register.

SAHF Store (copy) AH Register to low byte of flag register.

PUSHF Copy flag register to top of stack.

POPF Copy word at top of stack to flag register.

ARITHMETIC INSTRUCTIONS

Addition instructions:

ADD Add specified byte to byte or specified word to word.

ADC Add byte + byte + carry flag or word + word + carry flag

INC Increment specified byte or specified word by 1.

AAA ASCII adjust after addition.

DAA Decimal (BCD) adjust after addition.

Subtraction instructions

SUB Subtract byte from byte or word from word.

SBB Subtract byte and carry flag from byte or word and carry flag from word

DEC Decrement specified byte or specified word by 1

NEG Negate – invert each bit of a specified byte or word and add 1 (form 2's compliment).

CMP Compare two specified bytes or two specified words.

AAS ASCII adjust after subtraction.

DAS Decimal (BCD) adjust after subtraction.

Multiplication instructions :

MUL Multiply unsigned byte by byte or unsigned word by word.

IMUL Multiply signed byte by byte or signed word by word.

AAM ASCII adjust after multiplication.

Division instructions:

DIV Divide unsigned word by byte or unsigned double word by word.

IDIV Divide signed word by byte or signed double word by word.

AAD ASCII adjust before division.

CBW Fill upper byte or word with copies of sign bit of lower byte.

CWD Fill upper word of double word with sign bit of lower word.

BIT MANIPULATION INSTRUCTIONS

Logical instructions:

NOT Invert each bit of a byte or word

AND AND each bit in a byte or word with the corresponding bit in another byte or word.

OR OR each bit in a byte or word with the corresponding bit in another byte or word.

XOR Exclusive OR each bit in a byte or word with the corresponding bit in another byte or word.

TEST AND operands to update flags, but don't change operands.

Shift instructions:

SHL/SAL Shift bits of word or byte left, put zero(s) in LSB(s).

SHR Shift bits of word or byte right, put zero(s) in MSB(s).

SAR Shift bits of word or byte right, copy old MSB into new MSB.

Rotate instructions:

ROL Rotate bits of byte or word left, MSB to LSB and to CF.

ROR Rotate bits of byte or word right, LSB to MSB and to CF.

RCL Rotate bits of byte or word left, MSB to CF and CF to LSB.

RCR Rotate bits of byte or word right LSB to CF and CF to MSB.

STRING INSTRUCTIONS

A string is a series of byte or a series of words in sequential memory locations. A string often consists of ASCII character codes. In the list, a "/" is used to separate different mnemonics for the same instructions. Use the mnemonic which most clearly describe the function of the instruction in a specific application. A "B" in a mnemonic is used to specifically indicate that a string of bytes is to be acted upon. A "W" in the mnemonic is used to indicate that a string of words is to be acted upon.

REP An instruction prefix. Repeat following instruction until CX=0.

REPE/REPZ An instruction prefix. Repeat instruction until CX = 0 or zero flag ZF = 1.

REPNE/REPNZ An instruction prefix. Repeat until CX = 0 or ZF = 1.

MOVS/MOVSB/MOVSW Move byte or word from one string to another

COMPS/COMPSB/COMPSW Compare two string bytes or two string words

SCAS/SCASB/SCASW Scan a string. Compare a string byte with a byte in AL or a string word with a word in AX

LODS/LODSB/LODSW Load string byte into AL or string word into AX

STOS/STOSB/STOSW Store byte from AL or word from AX into string

PROGRAM EXECUTION TRANSFER INSTRUCTIONS

These instructions are used to tell the 8086 to start fetching instruction from some new address, rather than continuing in sequence.

Unconditional transfer instructions:

CALL Call a procedure (subprogram) save return address on stack.

RET Return from procedure to calling program.

JMP Go to specified address to get next instruction.

A "/" is used to separate two mnemonics which represent the same instruction. Use the mnemonic which most clearly describes the decision condition in a specific program. These instructions are often used after a compare instruction. The term below and above refers to unsigned binary numbers. Above means larger in magnitude. The terms greater than or less than refer to signed binary numbers. Greater than means more positive.

JA/JNBE Jump if above/Jump if not below or equal

JAE/JNB Jump if above or equal / Jump if not below

JB/JNE Jump if below / Jump if not above or equal

JBE/JNA Jump if below or equal / Jump if not above

JC Jump if carry flag CF = 1

JE/JZ Jump if equal / Jump if zero flag ZF = 1

JG/JNLE Jump if greater / Jump if not less than or equal

JGE/JNL Jump if greater than or equal / jump if not less than

JL/JNGE	Jump if less than / Jump if not greater than or equal
JLE/JNG	Jump if less than or equal / Jump if not greater than
JNC	Jump if no carry (CF = 0)
JNE/JNZ	Jump if not equal / Jump if not zero (ZF = 0)
JNO	Jump if no overflow (overflow flag OF = 0)
JNP/JPO	Jump if not parity / Jump if parity odd (PF = 0)
JNS	Jump if not sign (sign flag SF = 0)
JO	Jump if over flow flag OF = 1
JP/JPE	Jump if parity / Jump if parity even (PF = 1)
JS	Jump if sign (SF = 1)

ITERATION CONTROL STATEMENTS

These instructions can be used to execute a series of instructions some number of times. Here mnemonics separated by a "/" represent the same instruction. Use the one that best fits the specific application.

LOOP	Loop through a sequence of instructions until CX = 0.
LOOPE/LOOPZ	Loop through a sequence of instructions while ZF = 1 and CX □ 0
LOOPNE/LOOPNZ	Loop through a sequence of instructions while ZF = 0 and CX □ 0.
JCXZ	Jump to specified address if CX = 0.

Interrupt instructions:

INT	Interrupt program execution, call service procedure
INTO	Interrupt program execution if OF = 1
IRET	Return from interrupt service procedure to main program

PROCESSOR CONTROL INSTRUCTIONS

Flag set/clear instructions:

STC	Set carry flag CF to 1
CLC	Clear carry flag CF to 0
CMC	Compliment the state of the carry flag CF
STD	Set direction flag DF to 1 (decrement string pointer)

CLD Clear direction flag DF to 0

STI Set interrupt enable flag to 1 (enable INTR input)

CLI Clear interrupt enable flag to 0 (disable INTR input)

External hardware synchronization instructions:

HLT Halt (do nothing) until interrupt or reset

WAIT Wait (do nothing) until signal on the TEST pin is low

ESC Escape to external co-processor such as 8087 or 8089

LOCK An instruction prefix. Prevents another processor from taking the bus while the adjacent instruction executes

No operation instruction:

NOP No action except fetch and decode.

Instruction formats

opcode	D	W	MOD	REG	R/M	Low byte displacement	High byte displacement	Low byte immediate data	Higher byte immediate data
6	1	1	2	3	3	8	8	8	8

Different Instructions

1) One byte instructions

2) Register to register instructions

3) Register to memory with no displacement

4) Register to memory with displacement

5) Immediate operand to register

6) Immediate operand to memory with 16 bit displacement

1) One byte instruction: This format is only one byte long and may have the implied data or register operands. The least significant 3 bits are used for specifying the register operand if any. Otherwise all the 8 bits form an opcode and the operands are implied.

One byte instruction format:

opcode	D	W

6	1	1

Ex) STC, CLC, STD, CLD.,

2) Register to register instruction : This format is a 2 byte long. The first byte of the code specifies the operation code and width of the operand specified by W bit. The second byte of the code shows the register operands and R?M fields as shown.

Register to register instruction format:

opcode	D	W	MOD	REG	R/M
6	1	1	2	3	3

Ex) MOV AX,BX

MOV AL,BL

3)Register to memory with no displacement : This format also 2 bytes long and similar to register to register format except for the MOD field.

Register to memory with no displacement instruction format:

opcode	D	W	MOD	REG	R/M
6	1	1	2	3	3

Ex) MOV AL, [DX]

4) Register to memory with displacement: This type of instruction format contains one or two additional bytes for displacement along with the format of the register to/from memory without displacement.

Register to memory with displacement

opcode	D	W	MOD	REG	R/M	Low byte displacement	High byte displacement
6	1	1	2	3	3	8	8

EX) MOV AL,[4587]

5)Immediate Operand to Register : In this format, the first byte as well as the 3-bits from secod byte which are used for REG field are used for opcode.

Immediate operand to register format:

opcode	D	W	MOD	REG	R/M	Low byte displacement	High byte displacement
6	1	1	2	3	3	8	8

EX) MOV AL,45H

MOV AX,4568H

6) Immediate operand to memory with 16 bit displacement: This type of instruction format requires 5 or 6 bytes for coding. The first 2 bytes contain the information regarding OPCODE, MOD and R/M. The remaining 4 bytes contain 2 bytes of displacement and 2 bytes of data.

Opcode	D	W	MOD	REG	R/M	Low byte displacement	High byte displacement	Low byte immediate data	Higher byte immediate data
6	1	1	2	3	3	8	8	8	8

EX) MOV DEST [DI],4567H

The opcode usually appears in the first byte, but in a few instructions, in some instructions the 3-bits in the second byte also used for opcode.

The assembler instruction formats of intel 8086 are shown. The description of instruction format is as follows:

D- represents a 1 bit field identifying the direction. If D is 0 the register specified is the source, otherwise destination.

W-bit : This indicates whether the instruction is to be operate over an 8-bit data or 16-bit data. If w bit is 0, the operand is of 8-bits and if w is 1, the operand is of 16-bits.

MOD is a 2-bit field defines addressing mode.

REG is a 3-bit field identifies a register

R/M is a 3-bit field specifies a register or memory address.

Displacement is an 8-bit or 16-bit value contained in an instruction.

The REG code of the different registers either as source or as destination operands in the opcode byte are assigned with binary codes. The segment registers are only 4. Hence 2 binary bits are sufficient to code them. The other registers are 8 in number, so at least 3 bits are required for coding them.

Addressing modes

Each instruction requires certain data on which it has to operate. There are various techniques to specify the address of the data. The different ways that a processor can access data are referred to as its addressing modes.

For example the MOV instruction has the format

MOV destination, source

When executed, this instruction copies a word or a byte from the specified source location to the specified destination location. The source can be a number written directly in the instruction, a specified register, or a memory location.

The destination can be a specified register or a memory location.

The source and destination cannot both be memory locations in an instruction.

The various addressing modes can be classified into the following groups.

1. Immediate addressing mode.

2. Register addressing mode.

3. Addressing modes for accessing data in memory.

    a. Direct Addressing mode

    b. Register indirect Addressing mode

    c. Based Addressing mode

    d. Indexed Addressing mode

    e. Based indexed Addressing mode

    f. String Addressing mode.

4. Addressing modes for accessing I/O ports.

5. Relative addressing mode.

6. Implied addressing mode.

1.Immediate Addressing Mode:-

Suppose if you want to put a number 437BH in the CX register,

MOV CX, 437BH

instruction can be used. This instruction will put the immediate hexadecimal number 437BH in the 16-bit CX register. This is referred to as immediate addressing mode, because the number to be loaded into CX register will be put in two memory locations immediately following the code for the MOV instruction.

CH ← 43H

CL ← 7BH

The instruction MOV CL, 48H can be used to load the 8-bit immediate number 48H into the 8-bit CL register.

2. Register Addressing Mode :-

Register addressing mode means that a register is the source of an operand for the instruction.

For example

MOV CX, AX

Copies the contents of the 16-bit AX register into the 16-bit CX register. The previous contents of CX are written over, but the contents of AX are not changed. If CX contains 2A84H and AX contains 4971H before MOV CX, AX instruction executes. After the instruction executes CX will contains 4971H and AX will still contains 4971H.

You can move any 16-bit register to any 16-bit register, or you can move any 8-bit register to any 8-bit register. But you cannot move an 8-bit register to a 16-bit register.

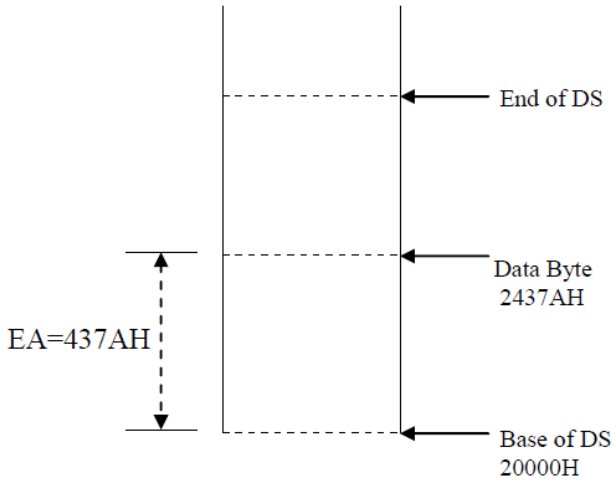
MOV AL,BL ; Content of register BL is copied to register AL.

3a) Direct Addressing Mode: -

MOV CL, [437AH]

The square brackets around the 437AH indicate that "the contents of the memory location at a displacement from the segment base". When this instruction executed the contents of the memory location, at a displacement of 437AH from the data segment base copied into CL register.

The actual 20-bit physical memory address will be produced by shifting the data segment base in DS four bit left and adding the effective address 437AH to the result.



MOV BX, [437AH]

This instruction copies a word from memory into the BX register. Since each memory address of 8086 represents a byte of storage, the word must come from two memory locations.

DS=	2	0	0	0	0	H
EA=		4	3	7	A	H
Physical Address	2	4	3	7	A	H

The byte at a displacement of 437AH from the data segment base will be copied into BL. The contents of next higher address, displacement 437BH, will be copied into the BH register.

[BL] ← [437AH]

[BH] ← [437BH]

The instruction MOV [437AH], BX will copy the contents of the BX register to two memory locations in the data segment. The contents of BL will be copied to the memory location at a displacement of 437AH. The contents of BH will be copied to the memory location at a displacement of 437BH.

[437AH ← [BL]

[437BH ← [BH]

3b) Register Indirect Addressing Mode :-

In this mode, the effective address is specified in either a pointer register or an index register. The pointer register can be either base register BX or a base pointer register BP and index register can be either source index (SI) register or destination Index (DI) register.

The 20-bit physical address is computed using DS and EA.

Example:- MOV [DI], BX

The destination operand of the above instruction is in register indirect mode; while the source operand is in register mode. The instruction moves the 16-bit content of BX into a memory location offset by the value of EA specified in DI from the current contents in DS.

DS = 5004H 50040H

DI = 0020H 0020H

BX= 2456H 50060H

After MOV[DI], BX the content of BX (2456) is moved to memory locations 50060 and 50061.

3c) Based Addressing Mode:-

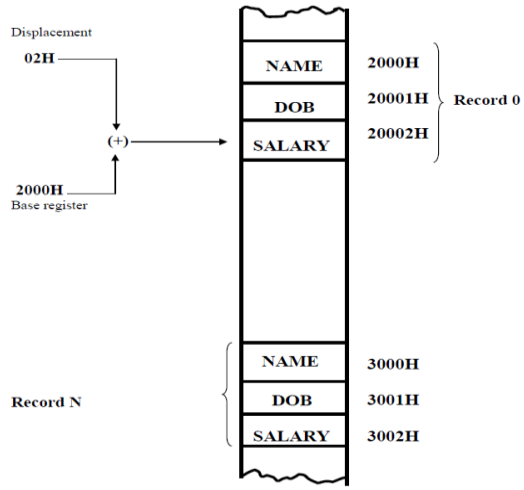
In this mode EA is obtained by adding a displacement value to the contents of BX or BP. The segment registers used are DS and SS. When memory is accessed, the 20-bit physical address is computed from BX and DS, on the other hand when the stack is accessed, the 20-bit physical address is computed from BP and SS. This allows the programmer to access the stack without changing SP contents.

MOV AL, START [BX]

The source operand in the above instruction is in based mode. EA is obtained by adding the value of start and [BX]. The 20-bit physical address is produced from DS and EA.

The 8-bit content of this memory location is moved to AL. The displacement START can be either unsigned 16-bit or signed 8-bit.

MOV AL, START [BX]



BX		2	0	0	0 H
START				0	2 H
EA		2	0	0	2 H
DS	2	0	0	0	0 H
Physical Address	2	2	0	0	2 H

Based addressing provides a convenient way to address structure which may be stored at different places in memory.

For example the element salary in record 0 of employee name 0 can be loaded into an 8086 internal register such as AL using the instruction MOV AL, ALPHA[BX], where ALPHA is the 8-bit displacement 02H and BX contains the starting address of record 0. Now in order to access the salary of record N, the programmer simply changes the contents of base register to 3000H.

Indexed Addressing Mode :-

In this mode, the effective, address is calculated by adding the unsigned 16-bit or signed 8-bit displacement and the contents of SI or DI.

MOV BH, START [SI]

Move the contents of 20-bit address computed from the displacement START, SI and DS into BH.

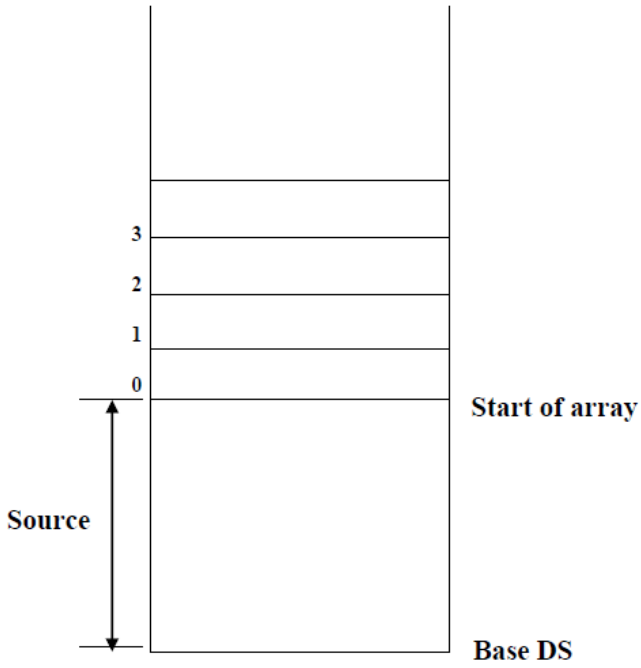
The 8-bit displacement is provided by the programmer using the assembler pseudo instruction such as EQU.

Index addressing mode can be used to access a single table(Single dimensional array). The displacement can be the starting address of the table. The contents of SI or DI can then be used as an index from the starting address to access a particular element in the table.

The instruction

MOV AL, Source [SI]

Sends elements of array to register AL Where Source is the name of the array SI gives the position. I f SI=0, the first element will be copied into register AL, if SI=1, the second element will be copied into register AL, and so on.



Based Indexed Addressing Mode:-

In this mode EA is computed by adding a base register [BX OR BP], an index register [SI or DI] and a displacement.

MOVE ALPHA [SI][BX],CL

[BX] = 0200H

ALPHA = 08H

[SI] = 1000H

[DS] = 3000H

[DS] ---------- |3|0|0|0|0|H

[BX] ---------- | |0|2|0|0|H

ALPHA ---------- | | | |0|8|H

[SI] ---------- |3|1|2|0|8|H

Then 8-bit content of CL is moved to 20-bit physical address 31208.

Based index addressing mode provides a convenient way for a subroutine to address an array allocated on stack. Register BP can be loaded with the offset in the segment SS. The displacement can be the value which is the difference between the top of the stack and the beginning of the array. An index register can then be used to access individual array elements.

String Addressing Mode:-

This mode uses index registers. The string instructions automatically assume SI to point the first byte or word of the source operand and DI to point to the first byte or word of the destination operand. The contents of SI and DI are automatically incremented; or decremented to point to the next byte or word.

Consider MOVSB instruction

If [DF]=0 [DS]=2000H, [SI]=0500H, [ES]=4000H, [DI]=0300H,

[20500H]=38H and [40300H]=45H

LEA SI, Source

LEA DI, Destination

MOV CX, 0004H

CLD

REP MOVSB

Then after execution of MOVS BYTE instruction

[40300H]=38H and [SI]=0501H [DI]=0301H.

MOVSB instruction moves one byte from source to destination increases SI and DI, Decreases CX. REP checks whether count in CX is zero or not. If it is not zero it repeats

MOVSB until CX becomes zero. The contents of other registers and memory locations are unchanged. Note that SI and DI can be used in either source or destination operand of a two operand instruction.

4. ADDRESSING MODES FOR ACCESSING I/O PORTS :-

Standard I/O uses port addressing modes. For memory mapped I/O, memory addressing modes are used. There are two types of port addressing modes.

Direct & Indirect

In direct port mode, the port number is an 8-bit immediate operand. This allows fixed access to ports numbered 0 to 255.

For example OUT 05H, AL outputs content of AL to 8-bit port 05H.

IN AL,05H ;input data from port address 05H to register AL

In indirect port mode, the port number is taken from DX allowing 64K 8-bit ports or 32K 16-bit ports.

For example if [DX] = 5040H then

IN AL, DX

Inputs the 8-bit content of port 5040H into AL.

On the other hand IN AX,DX inputs the 8-bit contents of ports 5040H and 5041H into AL and AH respectively. Note that 8-bit and 16-bit I/O transfers must take place via AL and AX respectively.

OUT DX, AL

This instruction sends the contents of register AL to the port addressed by DX.

5. Relative Addressing Mode :-

Instructions using this mode specify the operand as a signed 8-bit displacement relative to Program Counter/instruction pointer.

JNC START

This instruction means that if carry = 0 then PC/IP is loaded with current PC/IP contents + 8-bit signed value of START. Otherwise the next instruction is executed.

MOV AL, N1

MOV AH, 00H

MOV BL, N2

ADD AL, BL

JNC FORWARD

INC AH

FORWARD: MOV RES, AX

HLT

Other Examples

JC : Jump on Carry

JNC= Jump on no carry

JE/JZ: Jump if equal or jump on zero

JNE/JNZ: Jump if not equal/jump if result is not zero

JNP/JPO: Jump if no parity/jump if odd parity

JS: Jump if sign flag is set.

6) Implied Addressing Mode:-

Instructions using this mode have no operands. An example is

CLC which clears the carry Flag to zero.

STC-Set carry Flag

CMC-Complement carry Flag

STI-Set Interrupt Flag

CLI-Clear Carry Flag

STD-Set Direction Flag

CLD-Clear Direction Flag

Assembler Directives

Assembly language programs are composed of two types of statements.

1) The Instructions which can be translated to machine code by the assembler.

2) The directives that directs the assembler during the assembly process for which no machine code is generated.

Assembler Directives are instructions entered into the source code along with the assembly language. Pseudo instructions (Assembler Directives) do not get translated into object code,

but are used as special instructions to the assembler to perform some special functions. The directives control the generation of machine code and organization of the program.

Ex: SEGMENT, ENDS, ASSUME, DB, DW, DD, DQ, DT, END, PROC, ENDP, MACRO, ENDM, EQU, LENGTH, SIZE, OFFSET, EVEN, ORG, LABEL, INCLUDE

The assembler directives are classified into the following categories based on the functions performed by them. They are

a) Data definition and storage allocation directives

b) Program organization directives

c) Alignment Directives

d) Program end Directive

e) Value returning attribute directives

f) Procedure definition directives

g) Macro definition directives

h) Data control directives

i) Header file inclusion directives

a) Data definition and storage allocation directives

Data definition directives are used to define the program variables and allocate a specified amount of memory to them. They are of type BYTE, WORD, Double Word, Quad Word and Ten Byte and their size in bytes are 1,2,4,8 and 10 respectively. The data definition directives are DB, DW, DD, DQ, DT.

DB – [define byte]

The DB directive is used to define a byte–type variable or to set aside one or more storage locations of type byte in memory. It can be used to define single or multiple byte variables.

Ex 1) n DB 42H

tells the assembler to reserve 1 byte of memory for a variable named n and to put the value 42H in the that memory location, when the program is loaded into memory to be run.

Ex 2) num DB ?

The above statement informs the assembler to reserve one byte of memory for a variable named num. use of ? in data definition informs the assembler that the value is unknown and hence the variable num is not initialized.

Ex 3) grade DB ‘A’ or “A”

The above statement informs the assembler to reserve one byte of memory for a variable named grade and initializes with ASCII equivalent of a letter ‘A’. The ASCII character should be enclosed within single or double quotes.

Ex 4) num DB 25,50,43,76,34

The above statement reserves 5 bytes of consecutive memory locations for the variable num and initializes them with 5 values.

Ex 5) info DB ‘welcome’

The above statement defines a variable info and reserves 7 bytes of consecutive memory locations and initializes with the string ‘welcome’ during execution of program.

Ex 6) sname db 10 dup(‘-‘)

This statement defines a variable sname, reserves 10 bytes of consecutive memory locations and initializes with ASCII equivalent of character ‘-‘.

Ex 7) sum db 25 dup(?)

This statement defines a variable and reserves 25 bytes of consecutive memory locations and are not initialized.

DW – [define word]

The DW directive is used to declare a variable of type word, or to reserve storage locations of type word in memory.

Ex:- MULTIPLIER DW 347AH

num dw 023ah

sum dw ?

items dw 03abh, 0abc4h, 0bbah, 0543ch, 04a4bh

res dw 20 dup(0)

DD – [define double word]

The DD directive is used to declare a variable of type double word.

DQ – Define Quad word :

The directive DQ is used to define a quad word (8 bytes) type variable.

DT – Define Ten bytes :

The directive DT is used to define a Ten bytes type variable.

b) Program organization directives

SEGMENT : The segment directive is used to indicate the start of a logical segment. Preceding the segment directive the name you want to give the segment.

CODE SEGMENT

CODE ENDS

Indicates to the assembler the start of a logical segment called code. The SEGMENT and ENDS directives are used to identify a group of data items or a group of instructions that you want to be put together in a particular segment. A group of data statements or a group of instruction statements contained between segment and ends directives is called a logical segment. When you setup a logical segment, you give it a name of your choosing.

ENDS :- [end segment] : This directive is used with the name of a segment to indicate the end of the logical segment. ENDS is used with the segment directive to 'bracket' a logical segment containing instructions or data.

data segment

n1 db 05h

n2 db 04h

sum db ?

data ends

ASSUME :-

The assume directive is used to tell the assembler the name of the logical segment it should use for a specified segment. The 8086 program may have several logical segments. At any time the 8086 works directly with only four physical segments; a code segment, a data segment, a stack segment and an extra segment.

The statement

ASSUME CS : Code

tells the assembler that the instructions for a program are in a logical segment named code.

code segment

assume cs:code, ds:data

mov ax,data

mov ds,ax

---

code ends

data segment

---

data ends

end

c) Alignment Directives

EVEN :

Align as even memory address. The directive EVEN is used to inform the assembler to increment the location counter to the next even memory address if it is not pointing to even memory location already.

data segment

sum db 10

even

items dw 100 dup(?)

data ends

The data array items starts at the even memory location which makes access to its elements efficient; The directive even is used for the alignment of the data array item.

ORG : Originate :

The directive ORG assigns the location counter with the value specified in the directive. It helps in placing the machine code in the specified location while translating the instructions into machine codes by the assembler.

ORG 100

The above statement informs the assembler to initialize the location counter to 100.

d) Program end Directive

END – END PROGRAM

The END directive is put after the last statement of a program to tell the assembler that this is the end of the program module. A carriage return is required after the END directive. The last statement of every program must be an end directive.

code segment

---

ends

data segment

---

data ends

end

e) Value returning attribute directives

LENGTH:

The directive length informs the assembler about the number of elements in a data item such as an array. If an array is defined with DB then it returns the number of bytes allocated to the variable. If an array is defined with DW then it returns the number of words allocated to the array variable.

Ex1) MOV AX, LENGTH ITEMS

data segment

items db 08h,78h,56h,78h,98h

data ends

In the above example the number of elements assigned as array are 5. So five is the length of items which will be stored in AX.

Ex2) MOV AX, LENGTH ITEMS

data segment

items dw 0048h,0a78h,0b56h,0c78h,0d98h

data ends

In the above example the number of elements assigned as array are 5. So five is the length of items which will be stored in AX.

SIZE:

The directive SIZE is same as LENGTH except that it returns the number of bytes allocated to the data item instead of the number of elements in it.

Ex1) MOV AX,SIZE ITEMS

data segment

items DB 08h,78h,56h,78h,98h

data ends

In the above example the number of bytes occupied by the array items are 5. So 5 is the size of items which will be stored in AX.

Ex2) MOV AX, SIZE num

data segment

num DW 0008h,0078h,0a56h,0b78h,0c98h

data ends

In the above example the number of bytes occupied by the array elements 10. So 10 is the size of num which will be stored in AX.

OFFSET :

The directive OFFSET informs the assembler to determine the displacement of the specified variable with respect to the base of data segment.

Offset variable_ name

Ex:

MOV DX, OFFSET MSG

Data segment

MSG DB ‘nalanda’

Data ends

f) Procedure definition directives

PROC:

The PROC directive is used to identify the start of a procedure. The PROC directive follows a name you give the procedure. After the PROC directive the term NEAR or FAR is used to specify the type of the procedure.

Factorial PROC Near

RET

Factorial ENDP

ENDP :- [end procedure]

This directive is used along with the name of the procedure to indicate the end of a procedure to the assembler.

SQUARE_ROOT PROC NEAR

SQUARE_ROOT ENDP

g) Macro definition directives

MACRO:

A macro is a group of instructions we bracket and give a name to at the start of our program. Each time we call the macro in our program, the assembler insert the defined group of instructions in place of the call.

The MACRO directive is used to identify the start of a macro. The Macro directive follows a name you give the macro.

MACRONAME MACRO ; START OF MACRO

ENDM ; END OF MACRO

ENDM :-[end macro]

This directive is used along with the name of the macro to indicate the end of a macro to the assembler.

PUSHALL MACRO

PUSHF

PUSH AX

PUSH BX

PUSH CX

PUSH DX

ENDM

EQU :-[equate]

EQU is used to give a name to some value or symbol. Each time the assembler finds the given name in the program it will replace the name with the value or symbol equated with that name.

Ex1) SUM EQU 10

The above statement declares the symbol SUM with value 10. The assembler will replace every occurrence of the symbol in the program by its value.

Ex2) MOVEMENT EQU MOV

Every occurrence of MOVEMENT will be replaced by MOV.

Ex3) DECIMAL_ADJUST EQU DAA

Every occurrence of DECIMAL_ADJUST will be replaced by DAA.

h) Data control directives

LABEL:

The label directive is used to give a name to the current value in the location counter. The label directive must be followed by a term which specifies the type you want associated with that name.

STK SEGMENT

S DW 100 DUP(0)

S_TOP LABEL WORD

STK ENDS

i) Header file inclusion directives

INCLUDE:

The header file inclusion directive is used to define an include file header. The header file inclusion directive is INCLUDE.

The directive INCLUDE informs the assembler to include the statements defined in the include file. The name of the include file follows the statement INCLUDE. It is useful to place all the data and frequently used macros into a file known as include file or header file.

INCLUDE <file path specification>

Simple Programs involving Logical, Branch and Call Instructions Sorting, Evaluating Arithmetic Expressions

ALP to adding two multi byte numbers and store the result as the third number

```
DATA SEGMENT

BYTES EQU 08H

NUM1 DB 05H, 5AH, 6CH, 55H, 66H, 77H, 34H, 12H

NUM2 DB 04H, 56H, 04H, 57H, 32H, 12H, 19H, 13H

NUM3 DB 0AH DUP (00)

DATA ENDS

CODE SEGMENT

ASSUME CS: CODE, DS: DATA

START: MOV AX, DATA

 MOV DS, AX

 MOV CX, BYTES

 LEA SI, NUM1

 LEA DI, NUM2

 LEA BX, NUM3

MOV AX, 00
```

```
NEXT: MOV AL, [SI]
 ADC AL, [DI]
 MOV [BX], AL
 INC SI
 INC DI
 INC BX
 DEC CX
JNZ NEXT
INT 3H
CODE ENDS
END START
```

ALP to Subtracting two multi byte numbers and store the result as the third number

```
DATA SEGMENT
BYTES EQU 08H
NUM2 DB 05H, 5AH, 6CH, 55H, 66H, 77H, 34H, 12H
NUM1 DB 04H, 56H, 04H, 57H, 32H, 12H, 19H, 13H
NUM3 DB 0AH DUP (00)
DATA ENDS
CODE SEGMENT
ASSUME CS: CODE, DS: DATA
START: MOV AX, DATA
 MOV DS, AX
 MOV CX, BYTES
 LEA SI, NUM1
 LEA DI, NUM2
 LEA BX, NUM3
 MOV AX, 00
```

```
NEXT: MOV AL, [SI]
 SBB AL, [DI]
 MOV [BX], AL
 INC SI
 INC DI
 INC BX
 DEC CX
JNZ NEXT
INT 3H
CODE ENDS
END START
```

ALP to Multiplying two multi byte numbers and store the result as the third number

```
DATA SEGMENT
BYTES EQU 08H
NUM1 DB 05H, 5AH, 6CH, 55H, 66H, 77H, 34H, 12H
NUM2 DB 04H, 56H, 04H, 57H, 32H, 12H, 19H, 13H
NUM3 DB 0AH DUP (00)
DATA ENDS
CODE SEGMENT
ASSUME CS: CODE, DS: DATA
START: MOV AX, DATA
 MOV DS, AX
 MOV CX, BYTES
 LEA SI, NUM1
 LEA DI, NUM2
 LEA BX, NUM3
 MOV AX, 00
```

```
NEXT: MOV AL, [SI]
 MOV DL,[DI]
 MUL DL
 MOV [BX], AL
 MOV [BX+1],AH
 INC SI
 INC DI
 INC BX
 INC BX
 DEC CX
JNZ NEXT
INT 3H
CODE ENDS
END START
```

ALP to Dividing two multi byte numbers and store the result as the third number

```
DATA SEGMENT
BYTES EQU 08H
NUM2 DB 05H, 5AH, 6CH, 55H, 66H, 77H, 34H, 12H
NUM1 DB 04H, 56H, 04H, 57H, 32H, 12H, 19H, 13H
NUM3 DB 0AH DUP (00)
DATA ENDS
CODE SEGMENT
ASSUME CS: CODE, DS: DATA
START: MOV AX, DATA
 MOV DS, AX
 MOV CX, BYTES
 LEA SI, NUM1
```

```
 LEA DI, NUM2
 LEA BX, NUM3
NEXT: MOV AX, 00
MOV AL, [SI]
 MOV DL,[DI]
 MUL DL
 MOV [BX], AL
 MOV [BX+1],AH
 INC SI
 INC DI
 INC BX
 INC BX
 DEC CX
JNZ NEXT
INT 3H
CODE ENDS
END START
```

ALP to ASCII Addition:

```
CODE SEGMENT
ASSUME CS: CODE
START: MOV AL,'5'
 MOV BL,'9'
 ADD AL, BL
 AAA
 OR AX, 3030H
 INT 3H
CODE ENDS
```

```
END START
```

ALP to Converting Packed BCD to unpacked BCD:

```
DATA SEGMENT
NUM DB 45H
DATA ENDS
CODE SEGMENT
ASSUME CS: CODE, DS: DATA
START: MOV AX, DATA
 MOV DS, AX
 MOV AX, NUM
 MOV AH, AL
 MOV CL, 4
SHR AH, CL
AND AX, 0F0FH
INT 3H
CODE ENDS
END START
```

ALP to Moving a Block using strings

```
DATA SEGMENT
SRC DB ‘MALLAREDDY ENGINEERING COLLEGE’
 DB 10 DUP (?)
DST DB 20 DUP (0)
DATA ENDS
CODE SEGMENT
ASSUME CS: CODE, DS: DATA, ES: DATA
START: MOV AX, DATA
 MOV DS, AX
```

```
 MOV ES, AX
 LEA SI, SRC
 LEA DI, DST
 MOV CX, 20
 CLD
 REP MOVSB
 INT 3H
CODE ENDS
END START
```

ALP to move a string from one place to another place

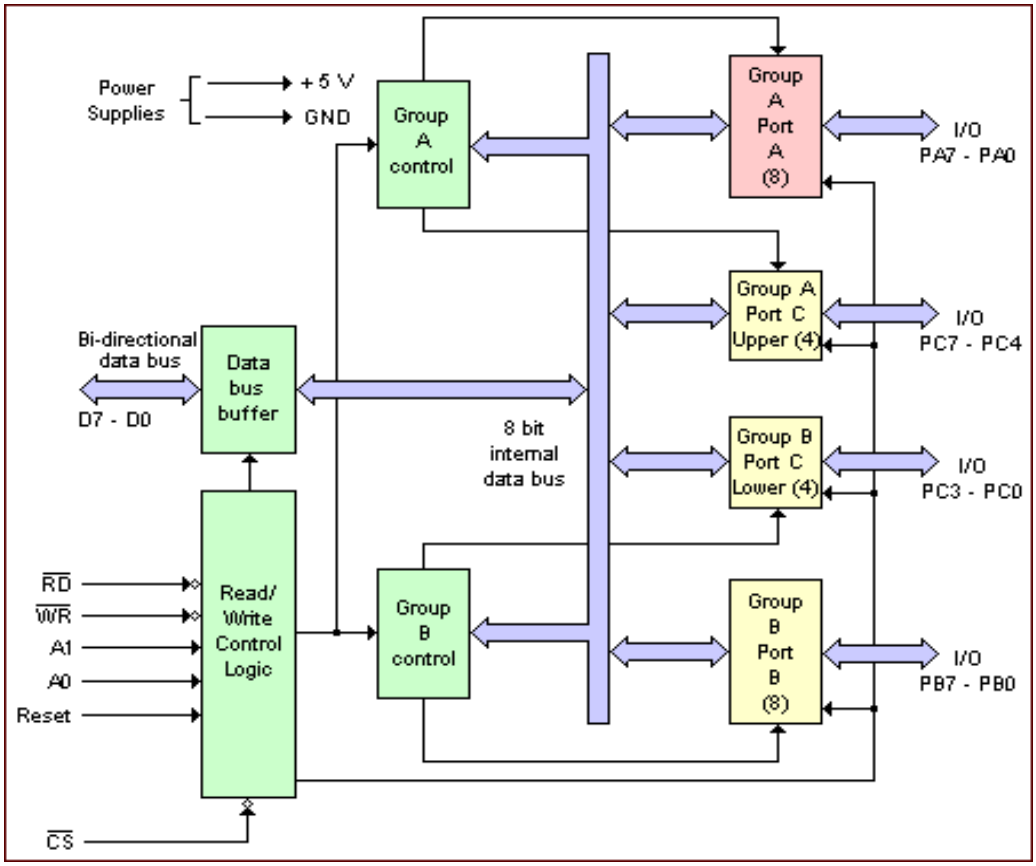
```
CODE SEGMENT
ASSUME CS:CODE, DS:DATA, ES:DATA
MOV AX,DATA
MOV DS,AX
MOV ES,AX
LEA SI,SOURCE
LEA DI,DEST
MOV CX,0007H
CLD
REPE MOVSB
HLT
CODE ENDS
DATA SEGMENT
SOURCE DB 'MALLAREDDY ENGINEERING COLLEGE'
DEST DB ?
DATA ENDS
END
```

UNIT -III:

I/O Interface:

8255 PPI, Various Modes of Operation and Interfacing to 8086

The 8255 is a widely used, programmable parallel I/O device. It can be programmed to transfer data under data under various conditions, from simple I/O to interrupt I/O. It is flexible, versatile and economical (when multiple I/O ports are required). It is an important general purpose I/O device that can be used with almost any microprocessor. The 8255 has 24 I/O pins that can be grouped primarily into two 8 bit parallel ports: A and B, with the remaining 8 bits as Port C. The 8 bits of port C can be used as individual bits or be grouped into two 4 bit ports : CUpper (CU) and CLower (CL). The functions of these ports are defined by writing a control word in the control register. 8255 can be used in two modes: Bit set/Reset (BSR) mode and I/O mode. The BSR mode is used to set or reset the bits in port C. The I/O mode is further divided into 3 modes: mode 0, mode 1 and mode 2. In mode 0, all ports function as simple I/O ports. Mode 1 is a handshake mode whereby Port A and/or Port B use bits from Port C as handshake signals. In the handshake mode, two types of I/O data transfer can be implemented: status check and interrupt. In mode 2, Port A can be set up for bidirectional data transfer using handshake signals from Port C, and Port B can be set up either in mode 0 or mode 1.



RD (Read): This signal enables the Read operation. When the signal is low, microprocessor reads data from a selected I/O port of 8255.

WR (Write): This control signal enables the write operation.

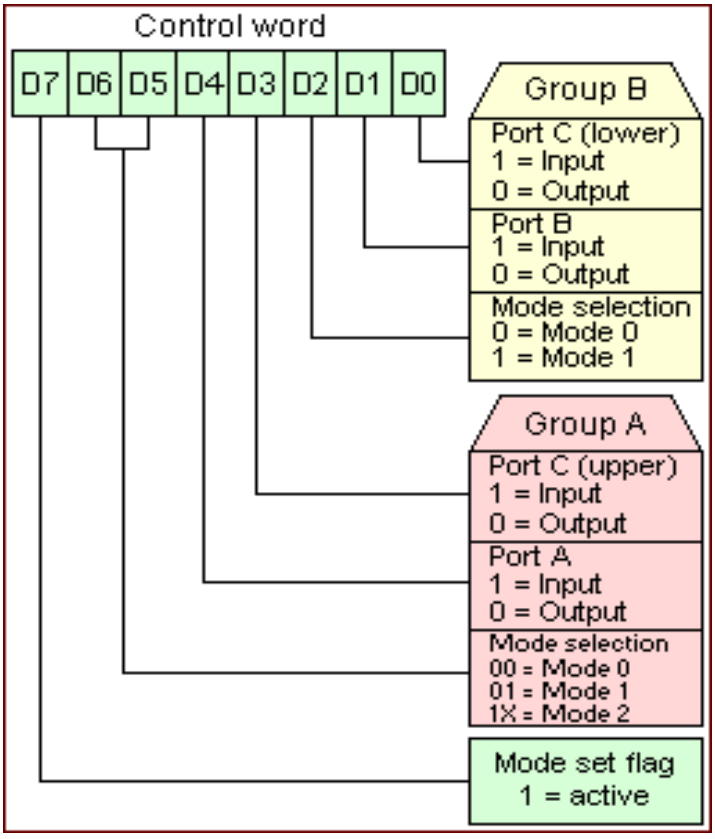
RESET (Reset): It clears the control registers and sets all ports in input mode.

CS, A0, A1: These are device select signals. is connected to a decoded address and

A0, A1 are connected to A0, A1 of microprocessor.

CS	A_0	A_1	Selected
0	0	0	Port A
0	0	1	Port B
0	1	0	Port C
0	1	1	Control Register
1	X	X	8255 Not Selected

Control Word Format of 8255:



This mode is selected by making D7 = '1'.
D0, D1, D3, D4 are for lower port C, port B, upper port C and port A respectively. When D0 or D1 or D3 or D4 are "SET", the corresponding ports act as input ports.

Eg: if D0= D4 = '1', then lower port C and port A act as input ports.

If these bits are "RESET", then the corresponding ports act as output ports.

Eg, if D1 = D3 = '0', then port B and upper port C act as output ports.

D2 is used for mode selection for group B (Port B and Lower Port C).
When D2 = '0', mode 0 is selected and when D2 = '1', mode 1 is selected.
D5, D6 are used for mode selection for group A (Upper Port C and Port A). The format is as follows:

D6	D5	mode
0	0	0
0	1	1
1	x	2

BSR Mode of 8255

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

This mode is selected by making D7='0'.
D0 is used for bit set/reset. When D0= '1', the port C bit selected (selection of a port C bit is shown in the next point) is SET, when D0 = '0', the port C bit is RESET.
D1, D2, D3 are used to select a particular port C bit whose value may be altered using D0 bit as mentioned above. The selection of the port C bits is done as follows:

D3	D2	D1	bit/pin of port C selected
0	0	0	PC0
0	0	1	PC1
0	1	0	PC2
0	1	1	PC3
1	0	0	PC4
1	0	1	PC5
1	1	0	PC6
1	1	1	PC7

D4, D5, D6 are not used.

I/O Modes of 8255

Mode 0 : Simple Input or Output

In this mode, Port A and Port B are used as two simple 8-bit I/O ports and Port C as two 4-bit I/O ports. Each port (or half-port, in case of Port C) can be programmed to function as simply an input port or an output port. The input/output features in mode 0 are: Outputs are latched, Inputs are not latched. Ports do not have handshake or interrupt capability.

Mode 1 : Input or Output with handshake

In mode 1, handshake signals are exchanged between the microprocessor and peripherals prior to data transfer. The ports (A and B) function as 8-bit I/O ports. They can be configured either as input or output ports. Each port (Port A and Port B) uses 3 lines from port C as

handshake signals. The remaining two lines of port C can be used for simple I/O functions. Input and output data are latched and Interrupt logic is supported.

Input control signals

STB (Strobe Input) : This signal (active low) is generated by a peripheral device that it has transmitted a byte of data. The 8255, in response to, generates IBF and INTR.

IBF (Input buffer full) : This signal is an acknowledgement by the 8255 to indicate that the input latch has received the data byte. This is reset when the microprocessor reads the data.

INTR (Interrupt Request) : This is an output signal that may be used to interrupt the microprocessor. This signal is generated if STB, IBF and INTE are all at logic 1.

INTE (Interrupt Enable) : This is an internal flip-flop to a port and needs to be set to generate the INTR signal. The two flip-flops INTEA and INTEB are set /reset using the BSR mode. The INTEA is enabled or disabled through PC4, and INTEB is enabled or disabled through PC2.

OBF (Output Buffer Full) : This is an output signal that goes low when the microprocessor writes data into the output latch of the 8255. This signal indicates to an output peripheral that new data is ready to be read. It goes high again after the 8255 receives a signal from the peripheral.

ACK (Acknowledge) : This is an input signal from a peripheral that must output a low when the peripheral receives the data from the 8255 ports.

INTR (Interrupt Request) : This is an output signal, and it is set by the rising edge of the signal. This signal can be used to interrupt the microprocessor to request the next data byte for output. INTE (Interrupt Enable) : This is an internal flip-flop to a port and needs to be set to generate the INTR signal. The two flip-flops INTEA and INTEB are set /reset using the BSR mode. The INTEA signal can be enabled or disabled through PC6, and INTEB is enabled or disabled through PC2.

Mode 2: Bidirectional Data Transfer

This mode is used primarily in applications such as data transfer between the two computers or floppy disk controller interface. Port A can be configured as the bidirectional port and Port B either in mode 0 or mode 1. Port A uses five signals from Port C as handshake signals for data transfer. The remaining three lines from Port C can be used either as simple I/O or as handshake signals for Port B.

Interfacing Keyboard

Getting meaningful data from a keyboard, it requires the following three major tasks:

1. Detect a key press.

2. Debounce the key press.

3. Encode the key press

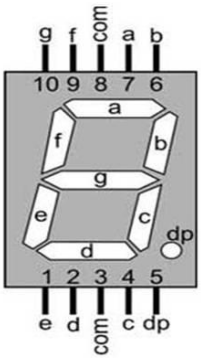
Display

7-segment display, it is composed of 8 LEDs, 7 segments are arranged as a rectangle for symbol displaying and there is an additional segment for decimal point displaying. In order to simplify connecting, anodes and cathodes of all diodes are connected to the common pin so that there are

Common Anode displays.

Common Cathode displays.

Segments are marked with the letters from A to G, plus DP, as shown in the figure below. On connecting, each diode is treated separately, which means that each must have its own current limiting resistor.



A seven segment display, as its name indicates, is composed of seven elements. Individually on or off, they can be combined to produce the representations numbers. In most applications, the seven segments are of nearly uniform shape and size, though in the case of adding machines, the vertical segments are longer and more oddly shaped at the ends in an effort to further enhance readability.

Since the seven segment display works on negative logic, we will have to provide logic 0 to the corresponding pin to make an LED glow. Table below shows the hex values used to display the different digits.

DIGIT	a	b	c	d	e	f	g	HEX Value
0	0	0	0	0	0	0	1	0x40
1	1	0	0	1	1	1	1	0xF9
2	0	0	1	0	0	1	0	0x24
3	0	0	0	0	1	1	0	0x30
4	1	0	0	1	1	0	0	0x19
5	0	1	0	0	1	0	0	0x12
6	0	1	0	0	0	0	0	0x02

7	0	0	0	1	1	1	1	0xF8
8	0	0	0	0	0	0	0	0x00
9	0	0	0	1	1	0	0	0x10

Assembly Language Program for seven segment display interface:

```
DATA SEGMENT

PORTA EQU 120H

PORTB EQU 121H

PORTC EQU 122H

CWRD EQU 123H

TABLE DB 8CH, 0C7H, 86H, 89H

DATA ENDS

CODE SEGMENT

ASSUME CS: CODE, DS:DATA

START: MOV AX, DATA; initialize data segment

MOV DS, AX

MOV AL, 80H; initialize 8255 port-b and port-c as o/p

MOV DX, CWRD

OUT DX, AL

MOV BH, 04; BH = no of digits to be displayed

LEA SI, TABLE; SI = starting address of lookup table

NEXTDIGIT: MOV CL, 08; CL = no of segments = 08

MOV AL, [SI]

NEXTBIT: ROL AL, 01

MOV CH, AL; save al

MOV DX, PORTB; one bit is sent out on portb

OUT DX, AL

MOV AL, 01
```

```
MOV DX, PORTC; one clock pulse sent on pc0

OUT DX, AL

DEC AL

MOV DX, PORTC

OUT DX, AL

MOV AL, CH; get the seven segment code back in al

DEC CL; send all 8 bits, thus one digit is displayed

JNZ NEXTBIT

DEC BH

INC SI; display all the four digits

JNZ NEXTDIGIT

MOV AH, 4CH; exit to dos

INT 21H

CODE ENDS

END START
```

INTERFACING ANALOG TO DIGITAL DATA CONVERTERS

The function of an A/D converter is to produce a digital word which represents the magnitude of some analog voltage or current. The specifications for an A/D converter are very similar to those for D/A converter. The resolution of an A/D converter refers to the number of bits in the output binary word. An 8-bit converter for example has a resolution of 1 part in 256. Accuracy and linearity specifications have the same meaning for an A/D converter as they do for a D/A converter. Another important specification for an ADC is its conversion time. This is simply the time it takes the converter to produce a valid output binary code for an applied input voltage. When we refer to a converter as high speed, we mean that it has a short conversion time.

The analog to digital converter is treated as an input device by the microprocessor that sends an initializing signal to the ADC to start the analog to digital data conversation process. The start of conversion signal is a pulse of a specific duration. The process of analog to digital conversion is a slow process, and the microprocessor has to wait for the digital data till the conversion is over. After the conversion is over, the ADC sends end of conversion (EOC) signal to inform the microprocessor that the conversion is over and the result is ready at the output buffer of the ADC. These tasks of issuing an SOC pulse to ADC, reading EOC signal

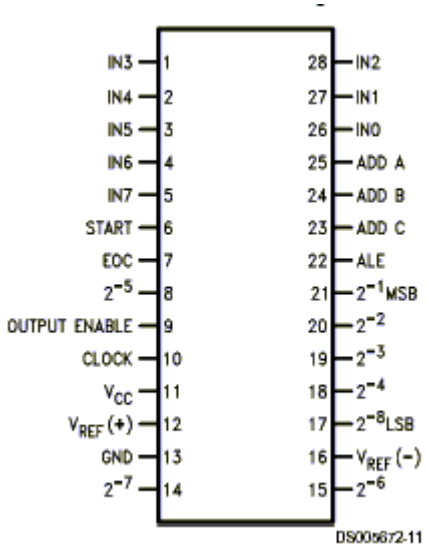
from the ADC and reading the digital output of the ADC are carried out by the CPU using 8255 I/O ports.

The time taken by the ADC from the active edge of SOC pulse (the edge at which the conversion process actually starts) till the active edge of EOC signal is called as the conversion delay of the ADC. In other words the time taken by the converter, to calculate the equivalent digital data output from the instant of the start of conversion is known as conversion delay. It may range anywhere from a few microseconds in case of fast ADCs to even a few hundred milliseconds in case of slow ADCs. A number of ADCs are available in the market, The selection of ADC for a particular application is done, keeping in mind the required speed, resolution range of operation, power supply requirements, sample and hold device requirements and the cost factors are considered.

The available ADCs in the market use different conversion techniques for the conversion of analog signals to digital signals. Parallel converter or flash converter, Successive approximation and dual slope integration techniques are the most popular techniques used in the integrated ADC chips. Whatever may be the technique used for conversion, a general algorithm for ADC interfacing contains the following steps.

1. Ensure the stability of analog input, applied to the ADC.
2. Issue start of conversion (SOC) pulse to ADC.
3. Read end of conversion (EOC) signal to mark the end of conversion process.
4. Read digital data output of the ADC as equivalent digital output.

It may be noted that analog input voltage must be constant at the input of the ADC right from the start of conversion till the end of conversion to get correct results. This may be ensured by a sample and hold circuit which samples the analog signal and holds it constant for specified time duration. The microprocessor may issue a hold signal to the sample and Hold circuit. If the applied input changes before the complete conversion process is over, the digital equivalent of the analog input calculated by the ADC may not be correct.

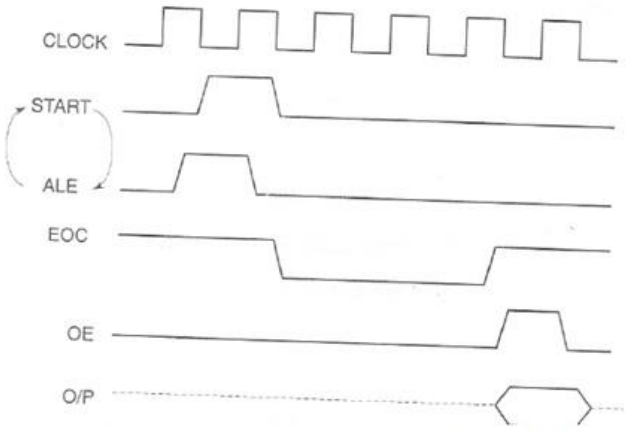


I/P0-I/P7 Analog inputs

ADD A, B, C

O_7 - O_0 Digital 8-bit output with O_7 MSB and O_0 LSB
SOC Start of conversion signal pin
EOC End of conversion signal pin
OE Output latch enable pin, if high enables output
CLK Clock input for ADC
Vcc, GND Supply pins +5V and GND
Vref+ and Vref- Reference voltage positive (+5 Volts max.) and
Reference voltage negative (0V minimum)

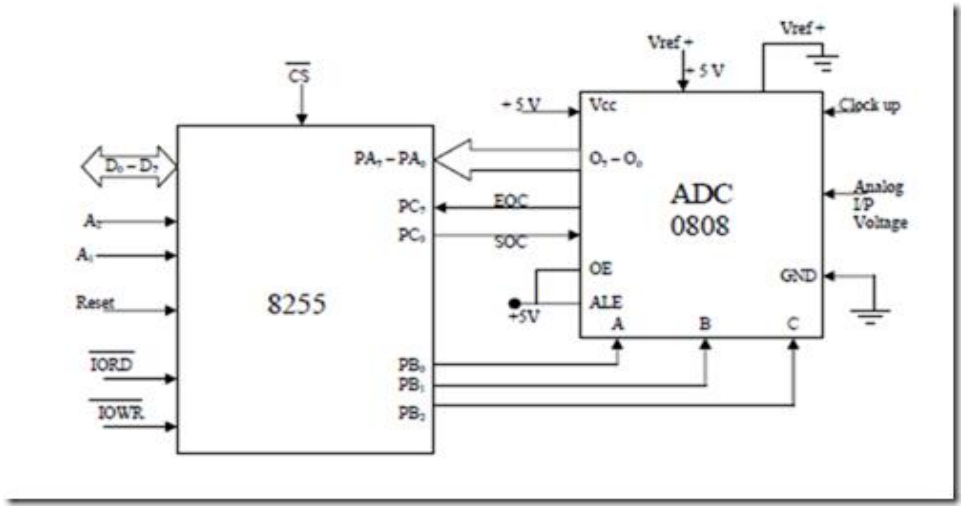
Some electrical specifications of ADC 0808/0809.
Minimum SOC pulse width 100ns
Minimum ALE pulse width 100ns
Clock frequency 10 to 1280 KHz
Conversion time 100μs at 640kHz
Resolution 8-bit
Error ±1 LSB
V_{ref}^{+} Not more than +5V
V_{ref}^{-} Not less than GND
$+V_{cc}$ supply +5V DC
Logical 1 i/p voltage minimum Vcc -1.5V
Logical 0 i/p voltage maximum 1.5V
Logical 1 o/p voltage minimum Vcc -0.4V
Logical 0 o/p voltage maximum 0.45V



Timing Diagram of ADC 0808/0809

Example: Interface ADC 0808 with 8086 using 8255 ports. Use Port A of 8255 for transferring digital data output of ADC to the CPU and Port C for control signals. Assume that an analog input is present at I/P2 of the ADC and a clock input of suitable frequency is available for ADC. Draw the schematic and write required ALP.
Solution: Following Figure shows the interfacing connections of ADC0808 with 8086 using 8255. The analog input I/P2 is used and therefore address pins A, B, C should be 0,1,0 respectively to select I/P2. The OE and ALE pins are already kept at +5V to select the ADC

and enable the outputs. Port C upper acts as the input port to receive the EOC signal while port C lower acts as the output port to send SOC to the ADC.



Interfacing the 0808 with 8255

The required ALP is given as follows:

```
MOV AL, 98 H ; Initialization of 8255
OUT CWR, AL ;
MOV AL, 02 H ; Select I/P 2 as analog
OUT Port B, AL ; input.
MOV AL,00H ; Give start of conversion
OUT Port C,AL ; pulse to the ADC
MOV AL,01H
OUT Port C,AL
Wait: IN AL, PortC ; Check for EOC by
RCR ; reading port C upper and rotate through carry
JNC Wait
IN AL, PortA ; If EOC, read digital equivalent in AL
HLT ; Stop
```

INTERFACING DIGITAL TO ANALOG ONVERTERS:

The digital to analog converters convert binary numbers into their analog equivalent voltages or currents. Several techniques are employed for digital to analog conversion.

i. Weighted resistor network

ii. R-2R ladder network

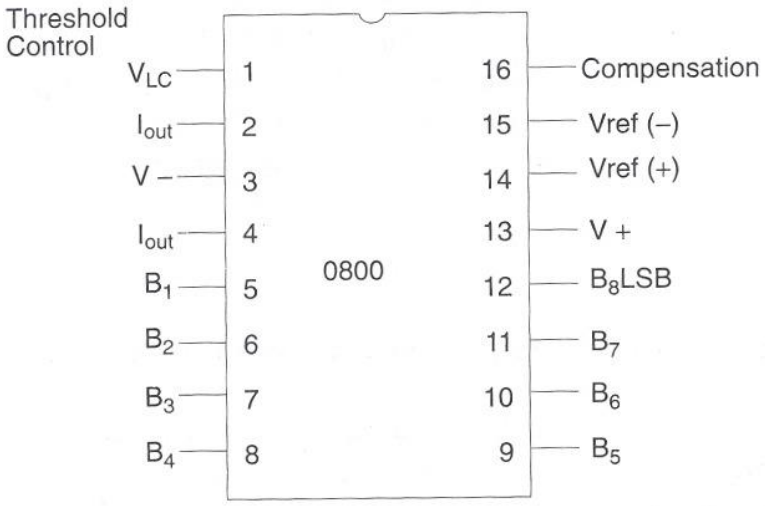
iii. Current output D/A converter

The DAC find applications in areas like digitally controlled gains, motor speed control, programmable gain amplifiers, digital voltmeters, panel meters, etc. D/A converter have many applications besides those where they are used with a microcomputer. In a compact disk audio player for example a 14-or16-bit D/A converter is used to convert the binary data read off the disk by a laser to an analog audio signal. Most speech synthesizer integrated circuits contain a D/A converter to convert stored binary data words into analog audio signals. Characteristics

1. Resolution: It is a change in analog output for one LSB change in digital input.

It is given by$(1/2^n)*V_{ref}$. If n=8 (i.e.8-bit DAC)
1/256*5V=39.06mV
2. Settling time: It is the time required for the DAC to settle for a full scale code change.

DAC 0800 8-bit Digital to Analog converter Features:
i. DAC0800 is a monolithic 8-bit DAC manufactured by National semiconductor.
ii. It has settling time around 100ms
iii. It can operate on a range of power supply voltage i.e. from 4.5V to +18V. Usually the supply V^+ is 5V or +12V. The V- pin can be kept at a minimum of -12V.
iv. Resolution of the DAC is 39.06mV



Pin Diagram of DAC0800

Communication Interface:

Serial Communication Standards

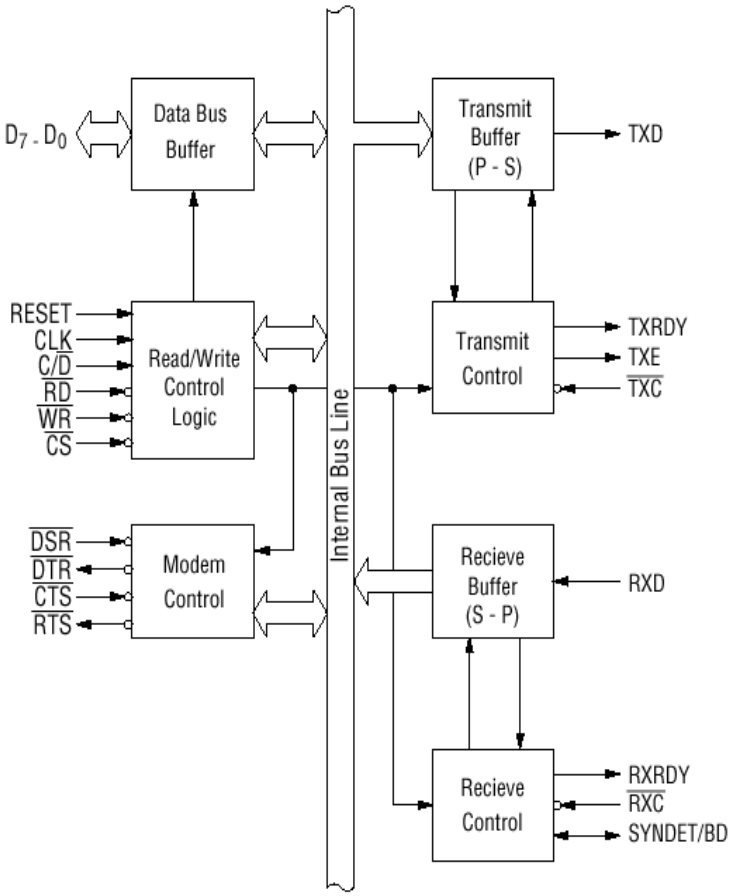
Serial Data Transfer Schemes

UART	USART
UART requires only data signal	In USART, Synchronous mode requires both data and a clock.
In UART, the data does not have to be transmitted at a fixed rate.	In USART's synchronous mode, the data is transmitted at a fixed rate.
In UART, data is normally transmitted one byte at a time.	In USART, Synchronous data is normally transmitted in the form of blocks
In UART, data transfer speed is set around specific values like 4800, 9600, 38400 bps, etc.	Synchronous mode allows for a higher DTR (data transfer rate) than asynchronous mode does, if all other factors are held constant.

UART speed is limited around 115200 bps	USART is faster than 115kb
Full duplex	Half duplex

8251 USART Architecture and Interfacing

The 8251 is a USART (Universal Synchronous Asynchronous Receiver Transmitter) for serial data communication. As a peripheral device of a microcomputer system, the 8251 receives parallel data from the CPU and transmits serial data after conversion. This device also receives serial data from the outside and transmits parallel data to the CPU after conversion.



Block diagram of the 8251 USART (Universal Synchronous Asynchronous Receiver Transmitter)

The 8251 functional configuration is programmed by software. Operation between the 8251 and a CPU is executed by program control. Table 1 shows the operation between a CPU and the device.

$\overline{CS}$	C/$\overline{D}$	$\overline{RD}$	$\overline{WR}$	
1	×	×	×	Data Bus 3-State
0	×	1	1	Data Bus 3-State
0	1	0	1	Status → CPU
0	1	1	0	Control Word ← CPU
0	0	0	1	Data → CPU
0	0	1	0	Data ← CPU

Table 1 Operation between a CPU and 8251

Control Words

There are two types of control word.
1. Mode instruction (setting of function)
2. Command (setting of operation)

1) Mode Instruction
Mode instruction is used for setting the function of the 8251. Mode instruction will be in "wait for write" at either internal reset or external reset. That is, the writing of a control word after resetting will be recognized as a "mode instruction."
Items set by mode instruction are as follows:
• Synchronous/asynchronous mode
• Stop bit length (asynchronous mode)
• Character length
• Parity bit
• Baud rate factor (asynchronous mode)
• Internal/external synchronization (synchronous mode)
• Number of synchronous characters (Synchronous mode)
The bit configuration of mode instruction is shown in Figures 2 and 3. In the case of synchronous mode, it is necessary to write one-or two byte sync characters. If sync characters were written, a function will be set because the writing of sync characters constitutes part of mode instruction.

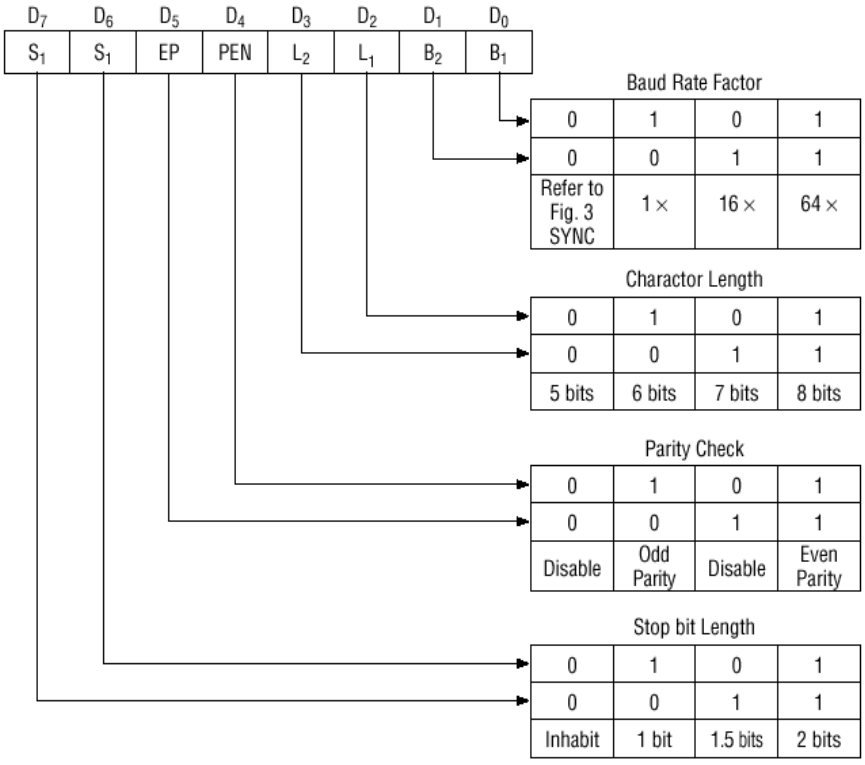


Fig. 2 Bit Configuration of Mode Instruction (Asynchronous)

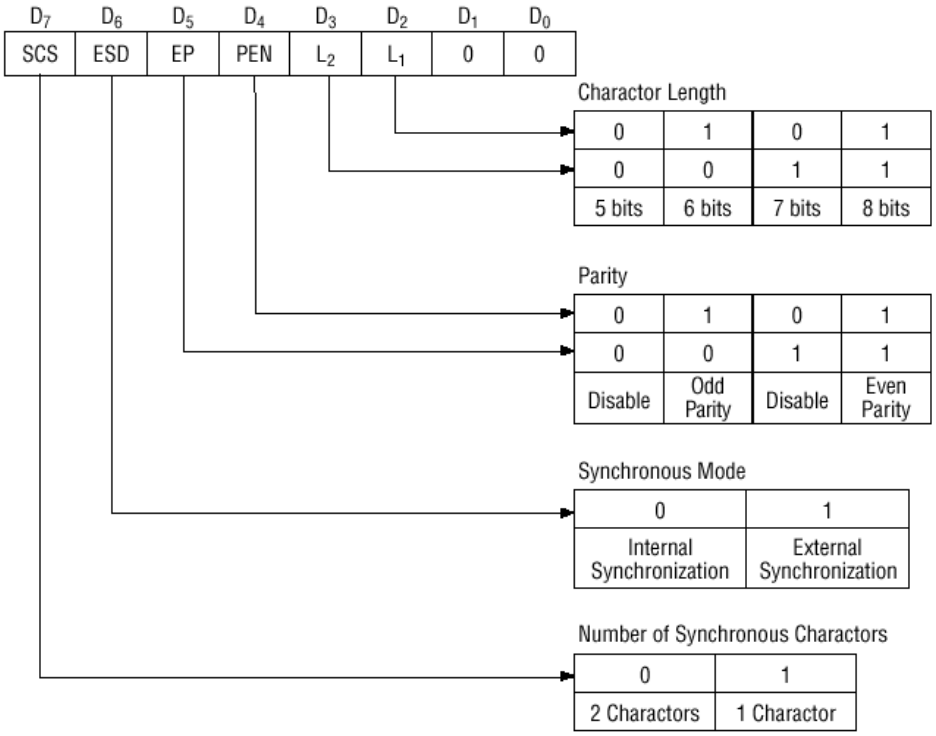


Fig. 3 Bit Configuration of Mode Instruction (Synchronous)

2) Command

Command is used for setting the operation of the 8251. It is possible to write a command whenever necessary after writing a mode instruction and sync characters.
Items to be set by command are as follows:

- Transmit Enable/Disable
- Receive Enable/Disable
- DTR, RTS Output of data.
- Resetting of error flag.
- Sending to break characters
- Internal resetting
- Hunt mode (synchronous mode)

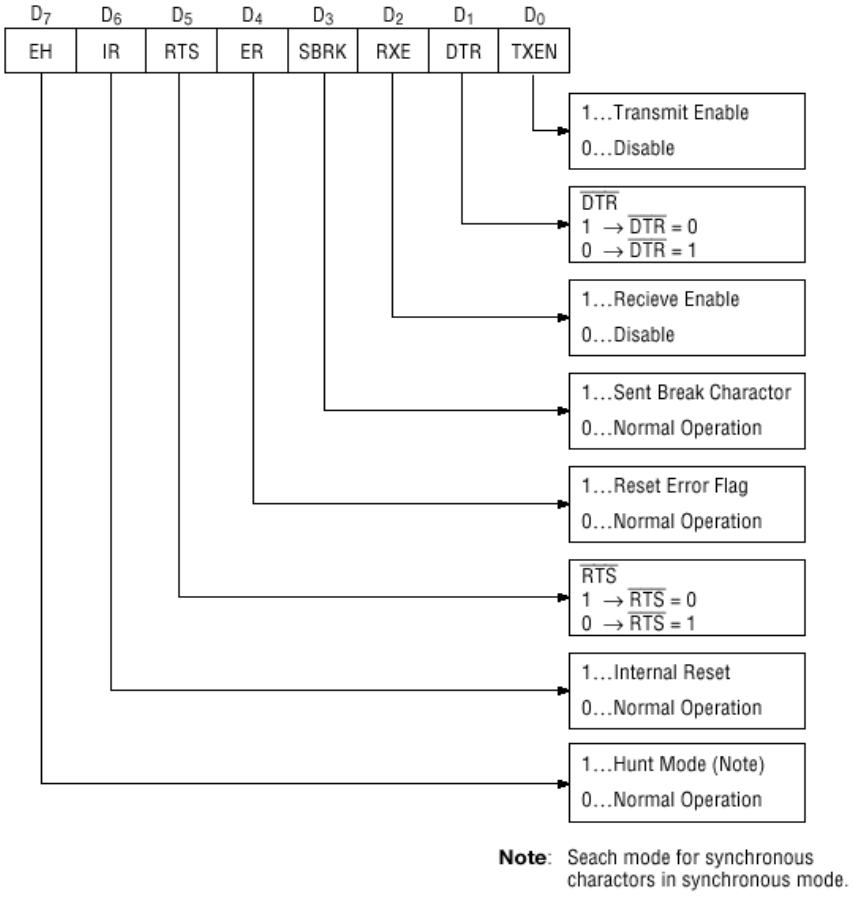


Fig. 4 Bit Configuration of Command

Status Word

It is possible to see the internal status of the 8251 by reading a status word. The bit configuration of status word is shown in Fig. 5.

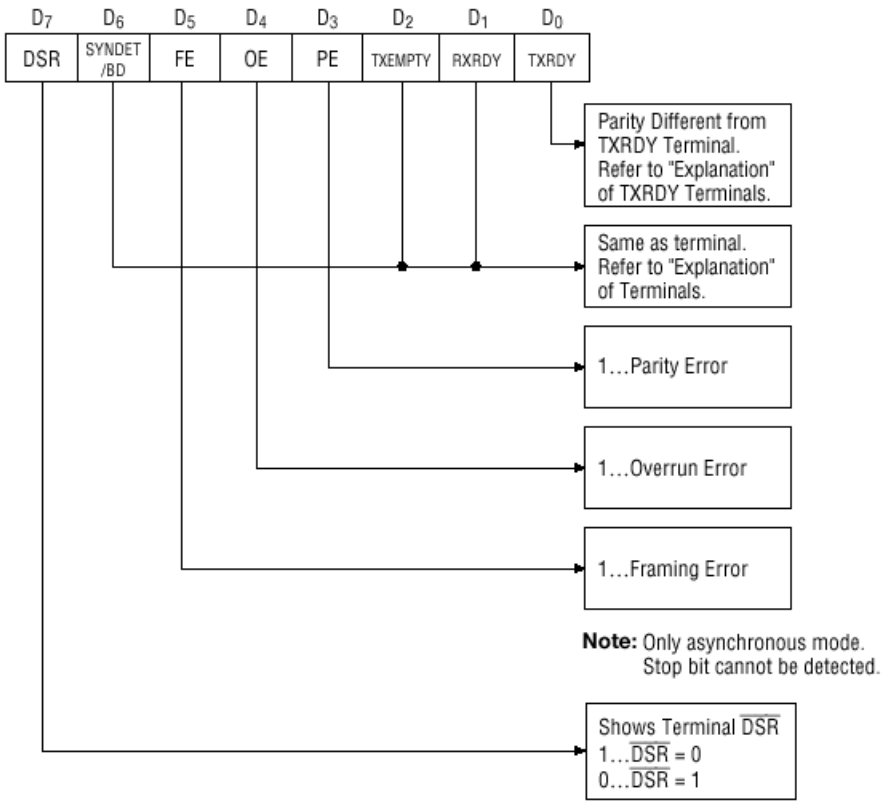


Fig. 5 Bit Configuration of Status Word

Pin Description

D 0 to D 7 (l/O terminal)

This is bidirectional data bus which receives control words and transmits data from the CPU and sends status words and received data to CPU.

RESET (Input terminal)

A "High" on this input forces the 8251 into "reset status." The device waits for the writing of "mode instruction." The min. reset width is six clock inputs during the operating status of CLK.

CLK (Input terminal)

CLK signal is used to generate internal device timing. CLK signal is independent of RXC or TXC. However, the frequency of CLK must be greater than 30 times the RXC and TXC at Synchronous mode and Asynchronous "x1" mode, and must be greater than 5 times at Asynchronous "x16" and "x64" mode.

WR (Input terminal)

This is the "active low" input terminal which receives a signal for writing transmit data and control words from the CPU into the 8251.

RD (Input terminal)

This is the "active low" input terminal which receives a signal for reading receive data and status words from the 8251.

C/D (Input terminal)

This is an input terminal which receives a signal for selecting data or command words and status words when the 8251 is accessed by the CPU. If C/D = low, data will be accessed. If C/D = high, command word or status word will be accessed.

CS (Input terminal)

This is the "active low" input terminal which selects the 8251 at low level when the CPU accesses. Note: The device won't be in "standby status"; only setting CS = High.

TXD (output terminal)

This is an output terminal for transmitting data from which serial-converted data is sent out. The device is in "mark status" (high level) after resetting or during a status when transmit is disabled. It is also possible to set the device in "break status" (low level) by a command.

TXRDY (output terminal)

This is an output terminal which indicates that the 8251is ready to accept a transmitted data character. But the terminal is always at low level if CTS = high or the device was set in "TX disable status" by a command. Note: TXRDY status word indicates that transmit data character is receivable, regardless of CTS or command. If the CPU writes a data character, TXRDY will be reset by the leading edge or WR signal.

TXEMPTY (Output terminal)

This is an output terminal which indicates that the 8251 has transmitted all the characters and had no data character. In "synchronous mode," the terminal is at high level, if transmit data characters are no longer remaining and sync characters are automatically transmitted. If the CPU writes a data character, TXEMPTY will be reset by the leading edge of WR signal. Note : As the transmitter is disabled by setting CTS "High" or command, data written before disable will be sent out. Then TXD and TXEMPTY will be "High". Even if a data is written after disable, that data is not sent out and TXE will be "High".After the transmitter is enabled, it sent out. (Refer to Timing Chart of Transmitter Control and Flag Timing)

TXC (Input terminal)

This is a clock input signal which determines the transfer speed of transmitted data. In "synchronous mode," the baud rate will be the same as the frequency of TXC. In "asynchronous mode", it is possible to select the baud rate factor by mode instruction. It can be 1, 1/16 or 1/64 the TXC. The falling edge of TXC sifts the serial data out of the 8251.

RXD (input terminal)

This is a terminal which receives serial data.

RXRDY (Output terminal)

This is a terminal which indicates that the 8251 contains a character that is ready to READ. If the CPU reads a data character, RXRDY will be reset by the leading edge of RD signal. Unless the CPU reads a data character before the next one is received completely, the preceding data will be lost. In such a case, an overrun error flag status word will be set.

RXC (Input terminal)

This is a clock input signal which determines the transfer speed of received data. In "synchronous mode," the baud rate is the same as the frequency of RXC. In "asynchronous mode," it is possible to select the baud rate factor by mode instruction. It can be 1, 1/16, 1/64 the RXC.

SYNDET/BD (Input or output terminal)

This is a terminal whose function changes according to mode. In "internal synchronous mode." this terminal is at high level, if sync characters are received and synchronized. If a status word is read, the terminal will be reset. In "external synchronous mode, "this is an input terminal. A "High" on this input forces the 8251 to start receiving data characters.

In "asynchronous mode," this is an output terminal which generates "high level"output upon the detection of a "break" character if receiver data contains a "low-level" space between the stop bits of two continuous characters. The terminal will be reset, if RXD is at high level. After Reset is active, the terminal will be output at low level.

DSR (Input terminal)

This is an input port for MODEM interface. The input status of the terminal can be recognized by the CPU reading status words.

DTR (Output terminal)

This is an output port for MODEM interface. It is possible to set the status of DTR by a command.

CTS (Input terminal)

This is an input terminal for MODEM interface which is used for controlling a transmit circuit. The terminal controls data transmission if the device is set in "TX Enable" status by a command. Data is transmitable if the terminal is at low level.

RTS (Output terminal)

This is an output port for MODEM interface. It is possible to set the status RTS by a command.

Interfacing with advanced devices:

Memory Interfacing to 8086

We have four common types of memory:

1. Read only memory (ROM)
2. Flash memory (EEPROM)
3. Static Random access memory (SARAM)
4. Dynamic Random access memory (DRAM)

Pin connections common to all memory devices are: The address input, data output or input/outputs, selection input and control input used to select a read or write operation.

Address connections: All memory devices have address inputs that select a memory location within the memory device. Address inputs are labeled from A0 to An.

Data connections: All memory devices have a set of data outputs or input/outputs. Today many of them have bi-directional common I/O pins.

Selection connections: Each memory device has an input, that selects or enables the memory device. This kind of input is most often called a chip select (CS), chip enable (CE) or simply select (S) input

RAM memory generally has at least one CS or S input and ROM at least one CE .

If the CE, CS, S input is active the memory device perform the read or write.

If it is inactive the memory device cannot perform read or write operation.

If more than one CS connection is present, all most be active to perform read or write data.

Control connections: A ROM usually has only one control input, while a RAM often has one or two control inputs.

The control input most often found on the ROM is the output enable (OE) or gate (G), this allows data to flow out of the output data pins of the ROM.

If OE and the selected input are both active, then the output is enable, if OE is inactive, the output is disabled at its high-impedance state.

The OE connection enables and disables a set of three-state buffer located within the memory device and must be active to read data.

A RAM memory device has either one or two control inputs. If there is one control input it is often called R/ W.

This pin selects a read operation or a write operation only if the device is selected by the selection input (CS).

If the RAM has two control inputs, they are usually labeled WE or W and OE or G.

(WE) write enable must be active to perform a memory write operation and OE must be active to perform a memory read operation.

When these two controls WE and OE are present, they must never be active at the same time.

The ROM read only memory permanently stores programs and data and data was always present, even when power is disconnected.

It is also called as nonvolatile memory.

EPROM (erasable programmable read only memory) is also erasable if exposed to high intensity ultraviolet light for about 20 minutes or less, depending upon the type of EPROM.

We have PROM (programmable read only memory)

RMM (read mostly memory) is also called the flash memory.

The flash memory is also called as an EEPROM (electrically erasable programmable ROM), EAROM (electrically alterable ROM), or a NOVROM (nonvolatile ROM).

These memory devices are electrically erasable in the system, but require more time to erase than a normal RAM.

EPROM contains the series of 27XXX contains the following part numbers:

2704(512 * 8), 2708(1K * 8), 2716(2K * 8), 2732(4K * 8), 2764(8K * 8), 27128(16K * 8) etc.

Each of these parts contains address pins, eight data connections, one or more chip selection inputs (CE) and an output enable pin (OE).

This device contains 11 address inputs and 8 data outputs.

If both the pin connection CE and OE are at logic 0, data will appear on the output connection. If both the pins are not at logic 0, the data output connections remain at their high impedance or off state.

To read data from the EPROM V_{pp} pin must be placed at logic 1.

SRAM

Static RAM memory device retain data for as long as DC power is applied. Because no special action is required to retain stored data, these devices are called as static memory. They are also called volatile memory because they will not retain data without power.

The main difference between a ROM and RAM is that a RAM is written under normal operation, while ROM is programmed outside the computer and is only normally read.

The SRAM stores temporary data and is used when the size of read/write memory is relatively small.

The semiconductor RAM is broadly two types – Static RAM and Dynamic RAM.

The semiconductor memories are organized as two dimensional arrays of memory locations.

For example 4K * 8 or 4K byte memory contains 4096 locations, where each locations contains 8-bit data and only one of the 4096 locations can be selected at a time. Once a location is selected all the bits in it are accessible using a group of conductors called Data bus.

For addressing the 4K bytes of memory, 12 address lines are required.

In general to address a memory location out of N memory locations, we will require at least n bits of address, i.e. n address lines where n = Log2 N.

Thus if the microprocessor has n address lines, then it is able to address at the most N locations of memory, where 2n=N. If out of N locations only P memory locations are to be interfaced, then the least significant p address lines out of the available n lines can be directly connected from the microprocessor to the memory chip while the remaining (n-p) higher order address lines may be used for address decoding as inputs to the chip selection logic.

The memory address depends upon the hardware circuit used for decoding the chip select (CS). The output of the decoding circuit is connected with the CS pin of the memory chip.

The general procedure of static memory interfacing with 8086 is briefly described as follows:

Arrange the available memory chip so as to obtain 16- bit data bus width. The upper 8-bit bank is called as odd address memory bank and the lower 8-bit bank is called as even address memory bank.

Connect available memory address lines of memory chip with those of the microprocessor and also connect the memory RD and WR inputs to the corresponding processor control

signals. Connect the 16-bit data bus of the memory bank with that of the microprocessor 8086.

The remaining address lines of the microprocessor, BHE and A0 are used for decoding the required chip select signals for the odd and even memory banks. The CS of memory is derived from the o/p of the decoding circuit.

As a good and efficient interfacing practice, the address map of the system should be continuous as far as possible, i.e. there should not be no windows in the map and no fold back space should be allowed.

A memory location should have a single address corresponding to it, i.e. absolute decoding should be preferred and minimum hardware should be used for decoding.

Dynamic RAM

Whenever a large capacity memory is required in a microcomputer system, the memory subsystem is generally designed using dynamic RAM because there are various advantages of dynamic RAM.

E.g. higher packing density, lower cost and less power consumption. A typical static RAM cell may require six transistors while the dynamic RAM cell requires only a transistors along with a capacitor. Hence it is possible to obtain higher packaging density and hence low cost units are available.

The basic dynamic RAM cell uses a capacitor to store the charge as a representation of data. This capacitor is manufactured as a diode that is reverse-biased so that the storage capacitance comes into the picture.

This storage capacitance is utilized for storing the charge representation of data but the reverse-biased diode has leakage current that tends to discharge the capacitor giving rise to the possibility of data loss. To avoid this possible data loss, the data stored in a dynamic RAM cell must be refreshed after a fixed time interval regularly. The process of refreshing the data in RAM is called as Refresh cycle.

The refresh activity is similar to reading the data from each and every cell of memory, independent of the requirement of microprocessor. During this refresh period all other operations related to the memory subsystem are suspended. Hence the refresh activity causes loss of time, & results in reduce of system performance.

However keeping in view the advantages of dynamic RAM, like low power consumption, high packaging density and low cost, most of the advanced computing system are designed using dynamic RAM, at the cost of operating speed.

A dedicated hardware chip called as dynamic RAM controller is the most important part of the interfacing circuit.

The Refresh cycle is different from the memory read cycle in the following aspects.

1. The memory address is not provided by the CPU address bus, rather it is generated by a refresh mechanism counter called as refresh counter.

2. Unlike memory read cycle, more than one memory chip may be enabled at a time so as to reduce the number of total memory refresh cycles.

3. The data enable control of the selected memory chip is deactivated, and data is not allowed to appear on the system data bus during refresh, as more than one memory units are refreshed simultaneously. This is to avoid the data from the different chips to appear on the bus simultaneously.

4. Memory read is either a processor initiated or an external bus master initiated and carried out by the refresh mechanism.

Dynamic RAM is available in units of several kilobits to megabits of memory. This memory is arranged internally in a two dimensional matrix array so that it will have n rows and m columns. The row address n and column address m are important for the refreshing operation.

For example, a typical 4K bit dynamic RAM chip has an internally arranged bit array of dimension 64 * 64 , i.e. 64 rows and 64 columns. The row address and column address will require 6 bits each. These 6 bits for each row address and column address will be generated by the refresh counter, during the refresh cycles.

A complete row of 64 cells is refreshed at a time to minimizes the refreshing time. Thus the refresh counter needs to generate only row addresses. The row address are multiplexed, over lower order address lines.

The refresh signals act to control the multiplexer, i.e. when refresh cycle is in process the refresh counter puts the row address over the address bus for refreshing. Otherwise, the address bus of the processor is connected to the address bus of DRAM, during normal processor initiated activities.

A timer, called refresh timer, derives a pulse for refreshing action after each refresh interval.

Refresh interval can be qualitatively defined as the time for which a dynamic RAM cell can hold data charge level practically constant, i.e. no data loss takes place.

Suppose the typical dynamic RAM chip has 64 rows, then each row should be refreshed after each refresh interval or in other words, all the 64 rows are to refreshed in a single refresh interval.

This refresh interval depends upon the manufacturing technology of the dynamic RAM cell. It may range anywhere from 1ms to 3ms.

Let us consider 2ms as a typical refresh time interval. Hence, the frequency of the refresh pulses will be calculated as follows:

Refresh Time (per row) tr = (2 * 10 -3) / 64.

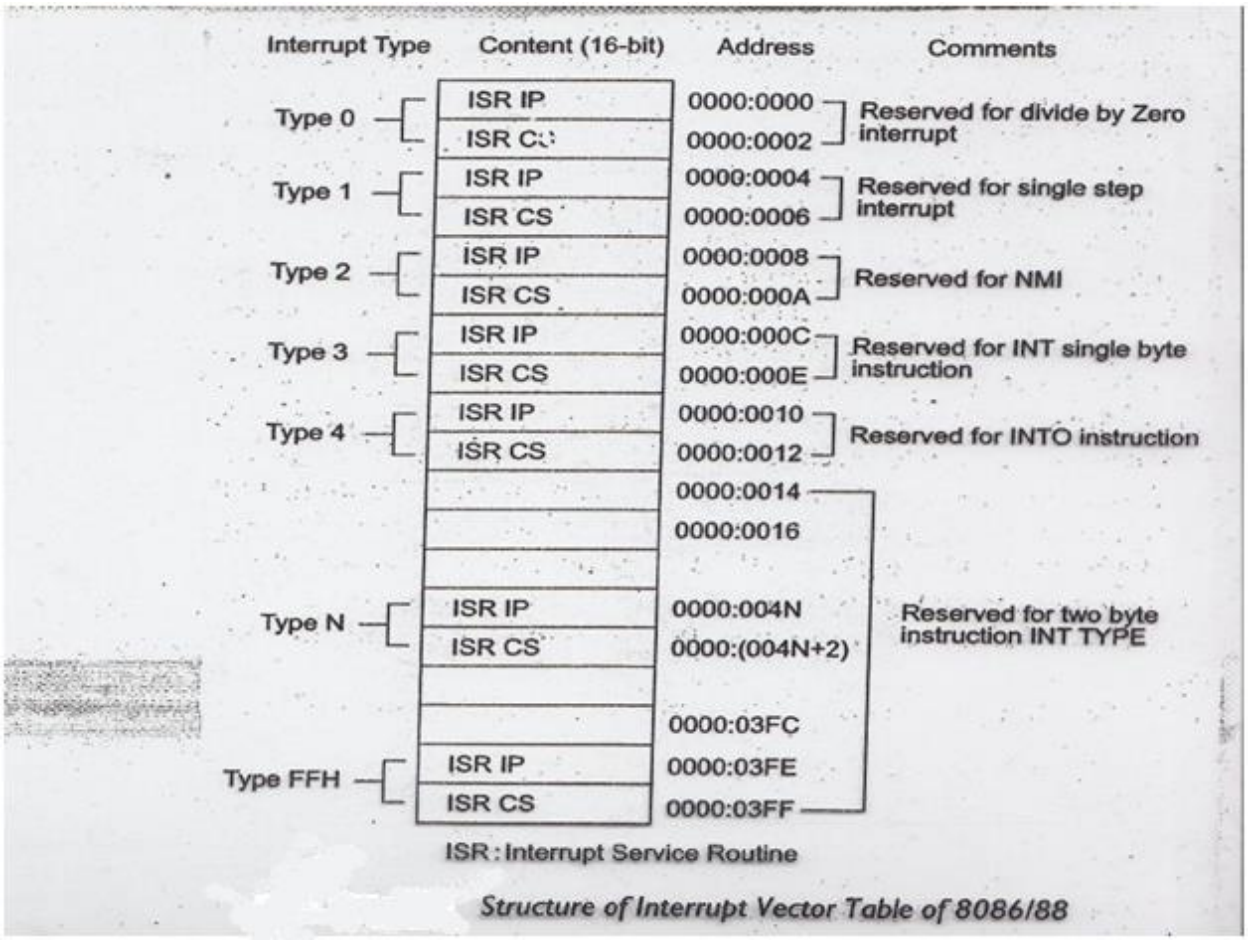
Refresh Frequency fr = 64 / (2 * 10 -3) = 32 * 103 Hz.

The following block diagram explains the refreshing logic and 8086 interfacing with dynamic RAM.

Each chip is of 16K * 1-bit dynamic RAM cell array. The system contains two 16K byte dynamic RAM units. All the address and data lines are assumed to be available from an 8086 microprocessor system.

Interrupt Structure of 8086

Vector Interrupt Table



Structure of Interrupt Vector Table of 8086/88

Type 0: Divide error – Division overflow or division by zero

Type 1: Single step or Trap – After the execution of each instruction when trap flag set

Type 2: NMI Hardware Interrupt – ‘1’ in the NMI pin

Type 3: One-byte Interrupt – INT3 instruction (used for breakpoints)

Type 4: Overflow – INTO instruction with an overflow flag

Type 5: BOUND – Register contents out-of-bounds

Type 6: Invalid Opcode – Undefined opcode occurred in program

Type 7: Coprocessor not available – MSW indicates a coprocessor

Type 8: Double Fault – Two separate interrupts occur during the same instruction

Type 9: Coprocessor Segment Overrun – Coprocessor call operand exceeds FFFFH

Type 10: Invalid Task State Segment – TSS invalid (probably not initialized)

Type 11: Segment not present – Descriptor P bit indicates segment not present or invalid

Type 12: Stack Segment Overrun – Stack segment not present or exceeded

Type 13: General Protection – Protection violation in 286 (general protection fault)

Type 14: Page Fault – 80386 and above

Type 16: Coprocessor Error – ERROR′ = '0' (80386 and above)

Type 17: Alignment Check – Word/Doubleword data addressed at odd location (486 and above)

Type 18: Machine Check – Memory Management interrupt (Pentium and above)

Type 0 interrupts: This interrupt is also known as the divide by zero interrupt. The 8086 will automatically do a type 0 interrupt if the result of a DIV operation or an IDIV operation is too large to fit in the destination register. For a type 0 interrupt, the 8086 pushes the flag register on the stack, resets IF and TF and pushes the return addresses on the stack.

Type 1 interrupts: This is also known as the single step interrupt. The use of single step feature found in some monitor programs and debugger programs. When you tell a system to single step, it will execute one instruction and stop. If they are correct we can tell a system to single step, it will execute one instruction and stop. We can then examine the contents of registers and memory locations. In other words, when in single step mode a system will stop after it executes each instruction and wait for further direction from you. The 8086 trap flag and type 1 interrupt response make it quite easy to implement a single step feature direction.

Type 2 interrupts: also known as the non-maskable NMI interrupts. The 8086 will automatically do a type 2 interrupt response when it receives a low to high transition on its NMI pin. When it does a type 2 interrupt, the 8086 will push the flags on the stack, reset TF and IF, and push the CS value and the IP value for the next instruction on the stack. It will then get the CS value for the start of the type 2 interrupt service procedure from address 0000AH and the IP value for the start of the procedure from address 00008H.

Type 3 interrupts: These types of interrupts are also known as breakpoint interrupts. The type 3 interrupt is produced by execution of the INT3 instruction. The main use of the type 3 interrupt is to implement a breakpoint function in a system. When we insert a breakpoint, the system executes the instructions up to the breakpoint and then goes to the breakpoint

procedure. Unlike the single step which stops execution after each instruction, the breakpoint feature executes all the instructions up to the inserted breakpoint and then stops execution.

Type 4 interrupts: Also known as overflow interrupts is generally existent after an arithmetic operation was performed. The 8086 overflow flag will be set if the signed result of an arithmetic operation on two signed numbers is too large to be represented in the destination register or memory location. For example, if you add the 8 bit signed number 01101100 and the 8 bit signed number 010111101, the result will be 10111101. This would be the correct result if we were adding unsigned binary numbers, but it is not the correct signed result.

The characteristics of internal interrupts are as follows:

The type of code of an interrupt is either predefined or can be contained within the instruction itself.

In case of INTR interrupt inputs the generation of complementary INTA bus cycles are not generated.

No internal interrupt can be disabled in the case of such interrupts barring single step interrupts.

Priority Wise always the internal interrupts barring single step interrupts are always given higher priority as compared to external interrupts.

Software interrupts-type 0 through 255:

The 8086 INT instruction can be used to cause the 8086 to do any one of the 256 possible interrupt types. The desired interrupt type is specified as part of the instruction. The instruction INT32, for example will cause the 8086 to do a type 32 interrupt response. The 8086 will push the flag register on the stack, reset TF and IF, and push the CS and IP values of the next instruction on the stack.

INTR INTERRUPTS-TYPES 0 THROUGH 255:

The 8086 INTR input allows some external signal to interrupt execution of a program. Unlike the NMI input, however, INTR can be masked so that it cannot cause an interrupt. If the interrupt flag is cleared, then the INTR input is disabled. IF can be cleared at any time with CLEAR instruction.

PRIORITY OF 8086 INTERRUPTS:

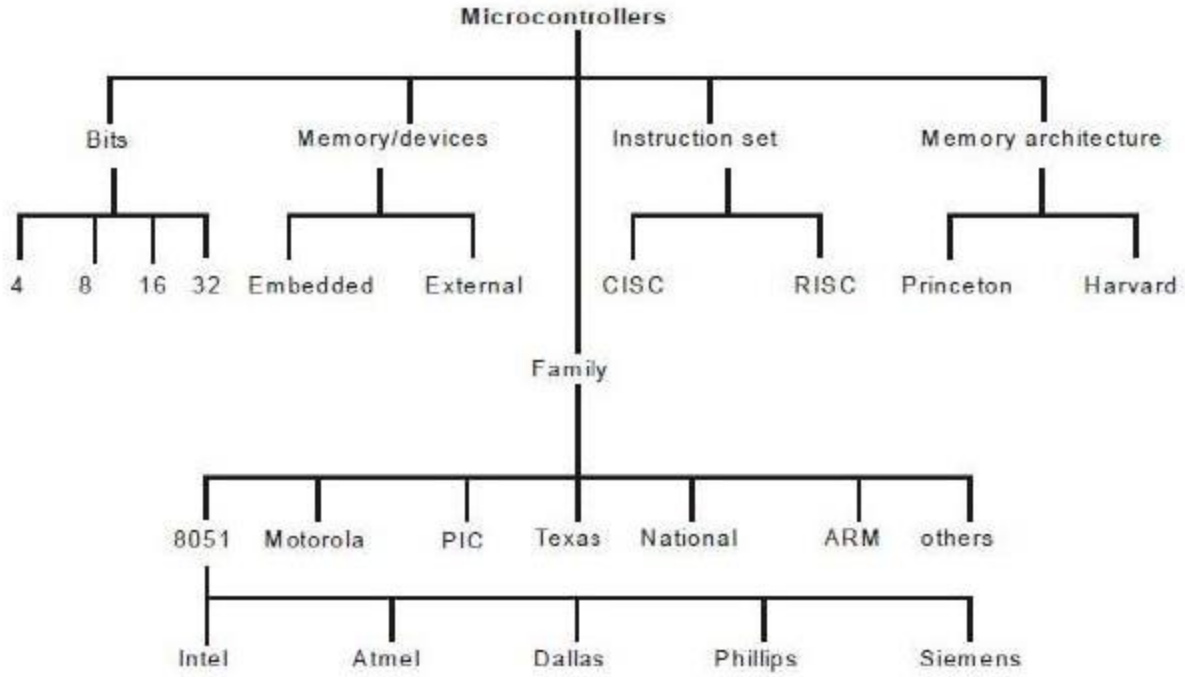
If two or more interrupts occur at the same time then the highest priority interrupt will be serviced first, and then the next highest priority interrupt will be serviced. As a example suppose that the INTR input is enabled, the 8086 receives an INTR signal during the execution of a divide instruction, and the divide operation produces a divide by zero interrupt. Since the internal interrupts-such as divide error, INT, and INTO have higher priority than INTR the 8086 will do a divide error interrupt response first.

Interrupt Service Routine.

The processor jumps to a special program called Interrupt Service Routine to service the peripheral.

UNIT -IV:

Introduction to Microcontrollers:



Overview of 8051 Microcontroller:

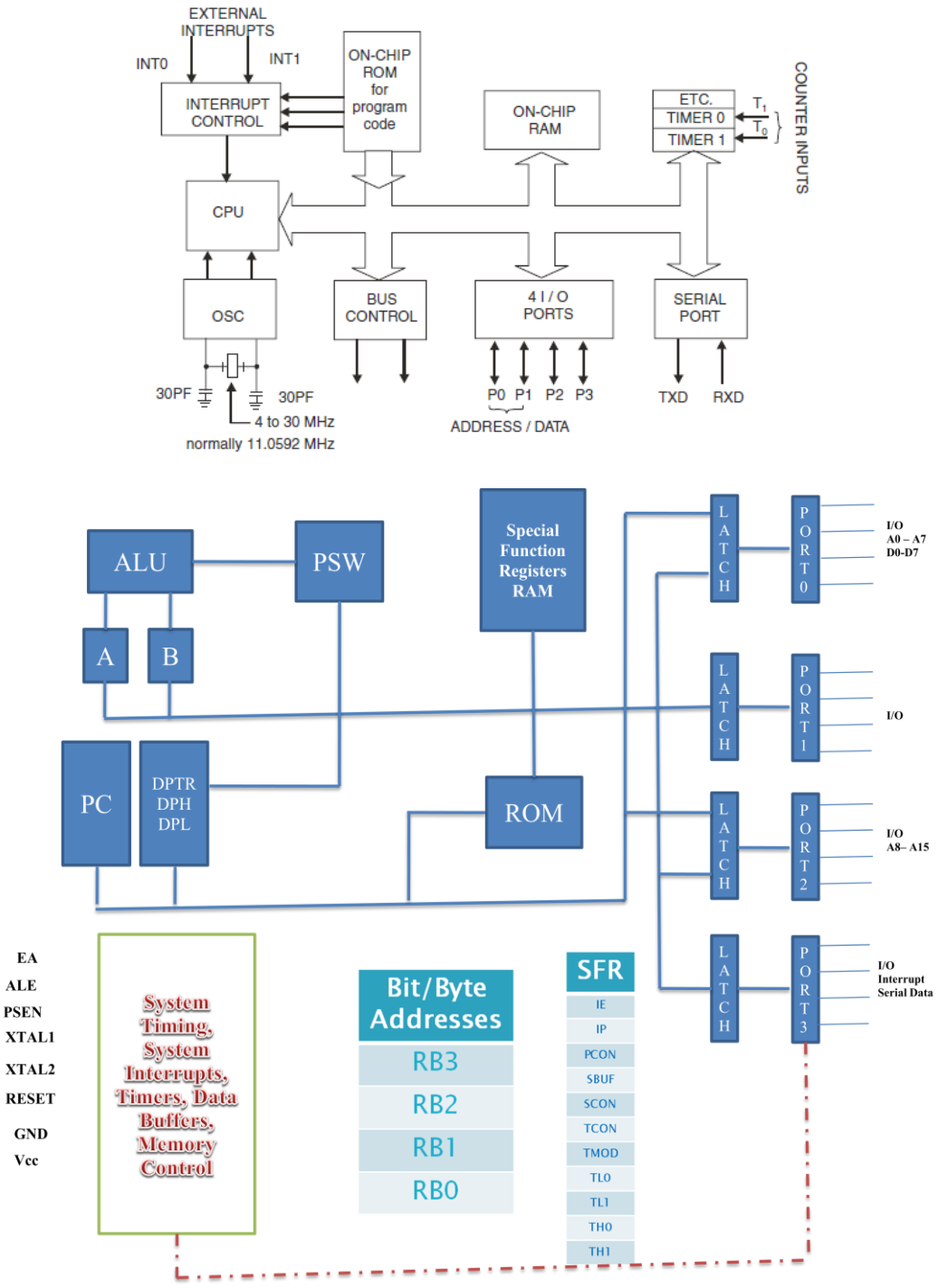
Features of Microcontroller 8051:

Following are the main features of Micro controller 8051 which makes it most efficient Microcontroller chip:

- It Includes Boolean Processing Engine. Thus internal registers and RAM can carry Boolean logic operations directly and efficiently.
- It gives us many functions in a single chip. (For example: CPU,RAM,ROM,I/O, Interrupt, Timer etc)
- It has 8 bit Data bus.
- It has 8 bit Stack Pointer.
- It has 16 bit Program Counter.
- It has 16 bit address bus which can access almost 65,536 memory locations.
- Data memory or RAM of 128 bytes. (On-Chip).
- Program Memory or ROM of 4 KB. (On Chip).
- Bi-Directional I/O port of 4 bytes.
- It has 4 separate Register Sets.
- Serial Port or UART.
- It features Power Saving Mode which saves power.
- Two Timers/Counters each of 16 bit.
- Internal and External Interrupt Sources.

- 2 level interrupt priority.

Architecture



128 Bytes RAM for Data Storage:

Random access memory is non volatile memory. During execution for storing the data the RAM is used. RAM consists of the register banks, stack for temporary data storage. It also consists of some special function register (SFR) which are used for some specific purpose

like timer, input output ports etc. Normally microcontroller has 256 byte RAM in which 128 byte is used for user space which is normally Register banks and stack.

ROM

A code of 4K memory is incorporated as on-chip ROM in 8051. The 8051 ROM is a non-volatile memory meaning that its contents cannot be altered and hence has a similar range of data and program memory, i.e, they can address program memory as well as a 64K separate block of data memory.

ALU

All arithmetic and logical functions are carried out by the ALU.

Addition, subtraction with carry, and multiplication come under arithmetic operations.

Logical AND, OR and exclusive OR (XOR) come under logical operations.

Program Counter (PC)

A program counter is a 16-bit register and it has no internal address. The basic function of program counter is to fetch from memory the address of the next instruction to be executed. The PC holds the address of the next instruction residing in memory and when a command is encountered, it produces that instruction. This way the PC increments automatically, holding the address of the next instruction.

Registers

Registers are usually known as data storage devices. 8051 microcontroller has 2 registers, namely Register A and Register B. Register A serves as an accumulator while Register B functions as a general purpose register. These registers are used to store the output of mathematical and logical instructions.

The operations of addition, subtraction, multiplication and division are carried out by Register A. Register B is usually unused and comes into picture only when multiplication and division functions are carried out by Register A. Register A also involved in data transfers between the microcontroller and external memory.

8051 microcontroller also has several Special Function Registers (SFRs). They are:

1. Serial Port Data Buffer (SBUF)

2. Timer/Counter Control (TCON)

3. Timer/Counter Mode Control (TMOD)

4. Serial Port Control (SCON)

5. Power Control (PCON)

6. Interrupt Priority (IP)

7. Interrupt Enable Control (IE)

Timers and Counters

Synchronization among internal operations can be achieved with the help of clock circuits which are responsible for generating clock pulses. During each clock pulse a particular operation will be carried out, thereby, assuring synchronization among operations. For the formation of an oscillator, we are provided with two pins XTAL1 and XTAL2 which are used for connecting a resonant network in 8051 microcontroller device. In addition to this, circuit also consists of four more pins. They are,

Internal operations can be synchronized using clock circuits which produce clock pulses. With each clock pulse, a particular function will be accomplished and hence synchronization is achieved. There are two pins XTAL1 and XTAL2 which form an oscillator circuit which connect to a resonant network in the microcontroller. The circuit also has 4 additional pins -

1. EA: External enable

2. ALE: Address latch enable

3. PSEN: Program store enable and

4. RST: Reset.

Quartz crystal is used to generate periodic clock pulses.

Four General Purpose Parallel Input/Output Ports

The 8051 microcontroller has four 8-bit input/output ports. These are:

PORT P0: When there is no external memory present, this port acts as a general purpose input/output port. In the presence of external memory, it functions as a multiplexed address and data bus. It performs a dual role.

PORT P1: This port is used for various interfacing activities. This 8-bit port is a normal I/O port i.e. it does not perform dual functions.

PORT P2: Similar to PORT P0, this port can be used as a general purpose port when there is no external memory but when external memory is present it works in conjunction with PORT PO as an address bus. This is an 8-bit port and performs dual functions.

PORT P3: PORT P3 behaves as a dedicated I/O port

PSW (Program Status Word)

Program Status Word or PSW is a hardware register which is a memory location which holds a program's information and also monitors the status of the program this is currently being executed. PSW also has a pointer which points towards the address of the next instruction to

be executed. PSW register has 3 fields namely are instruction address field, condition code field and error status field. We can say that PSW is an internal register that keeps track of the computer at every instant.

Generally, the instruction of the result of a program is stored in a single bit register called a 'flag'. The are7 flags in the PSW of 8051. Among these 7 flags, 4 are math flags and 3 are general purpose or user flags.

The 4 Math flags are:

• Carry (c)

• Auxiliary carry (AC)

• Overflow (OV)

• Parity (P)

The 3 General purpose flags or User flags are:

• FO

• GFO

• GF 1

Data Pointer (DPTR)

The data pointer or DPTR is a 16-bit register. It is made up of two 8-bit registers called DPH and DPL. Separate addresses are assigned to each of DPH and DPL. These 8-bit registers are used for the storing the memory addresses that can be used to access internal and external data/code.

Data and Address Bus

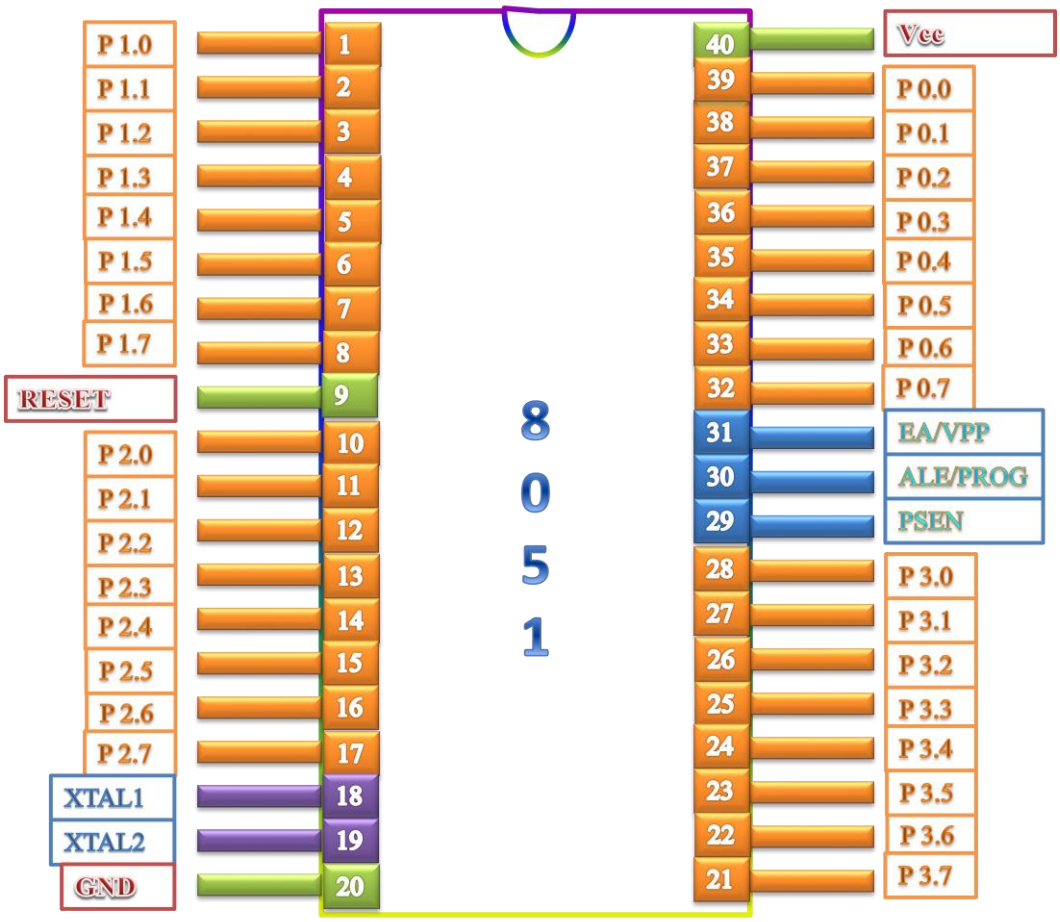
A bus is group of wires using which data transfer takes place from one location to another within a system. Buses reduce the number of paths or cables needed to set up connection between components.

There are mainly two kinds of buses - Data Bus and Address Bus

Data Bus: The purpose of data bus is to transfer data. It acts as an electronic channel using which data travels. Wider the width of the bus, greater will be the transmission of data.

Address Bus: The purpose of address bus is to transfer information but not data. The information tells from where within the components, the data should be sent to or received from. The capacity or memory of the address bus depends on the number of wires that transmit a single address bit.

Pin Diagram of 8051



Port 0(p0.0 to p0.7):

It is 8-bit bi-directional I/O port. It is bit/ byte addressable. During external memory access, it functions as multiplexed data and low-order address bus AD0-AD7.

Port 1 (p1.0 to p1.7):

It is 8-bit bi-directional I/O port. It is bit/ byte addressable. When logic '1' is written into port latch then it works as input mode. It functions as simply I/O port and it does not have any alternative function.

Port 2 (p2.0 to p2.7):

It is 8-bit bi-directional I/O port. It is bit/ byte addressable. During external memory access it functions as higher order address bus (A8-A15).

Port 3(p3.0 to port 3.7):

It is 8-bit I/O port. In an alternating function each pins can be used as a special function I/O pin.

P3.0-RxD:

It is an Input signal. Through this I/P signal microcontroller receives serial data of serial communication circuit.

P3.1-TxD:

It is O/P signal of serial port. Through this signal data is transmitted.

P3.2- (INT0):

It is external hardware interrupt I/P signal. Through this user, programmer or peripheral interrupts to microcontroller.

P3.3-(INT1):

It is external hardware interrupt I/P signal. Through this user, programmer or peripheral interrupts to microcontroller.

P3.4- T0:

It is I/P signal to internal timer-0 circuit. External clock pulses can connects to timer-0 through this I/P signal.

P3.5-T1:

It is I/P signal to internal timer-1 circuit. External clock pulses can connects to timer-1 through this I/P signal.

P3.6-[WR(bar)]:

It is active low write O/P control signal. During External RAM (Data memory) access it is generated by microcontroller. when [WR(bar)]=0, then performs write operation.

P3.7-[RD(bar)]:

It is active low read O/P control signal. During External RAM (Data memory) access it is generated by microcontroller. when [RD(bar)]=0, then performs read operation from external RAM.

XTAL1 and XTAL2:

These are two I/P line for on-chip oscillator and clock generator circuit. A resonant network as quartz crystal is connected between these two pin. 8051 microcontroller also drives from external clock, then XTAL2 is used to drive 8051 from external clock and XTAL1 should be grounded.

EA/VPP:

It is and active low I/P to 8051 microcontroller. When EA= 1, then 8051 microcontroller access from external program memory (ROM) only. When EA = 0, then it access internal and external program memories (ROMS).

PSEN:

It is active high O/P signal. It is used to enable external program memory (ROM). When PSEN= 1, then external program memory becomes enabled and microcontroller read content of external memory location. Therefore it is connected to (OE) of external ROM. It is activated twice every external ROM memory cycle.

ALE:

Address latch enable: It is active high O/P signal. When it goes high, external address latch becomes enabling and lower address of external memory (RAM or ROM) latched into it. Thus it separates A0-A7 address from AD0-AD7. It provides properly timed signal to latch lower byte address. The ALE is activated twice in every machine cycle. If external RAM & ROM is not accessed, then ALE is activated at constant rate of 1/6 oscillator frequency, which can be used as a clock pulses for driving external devices.

RESET:

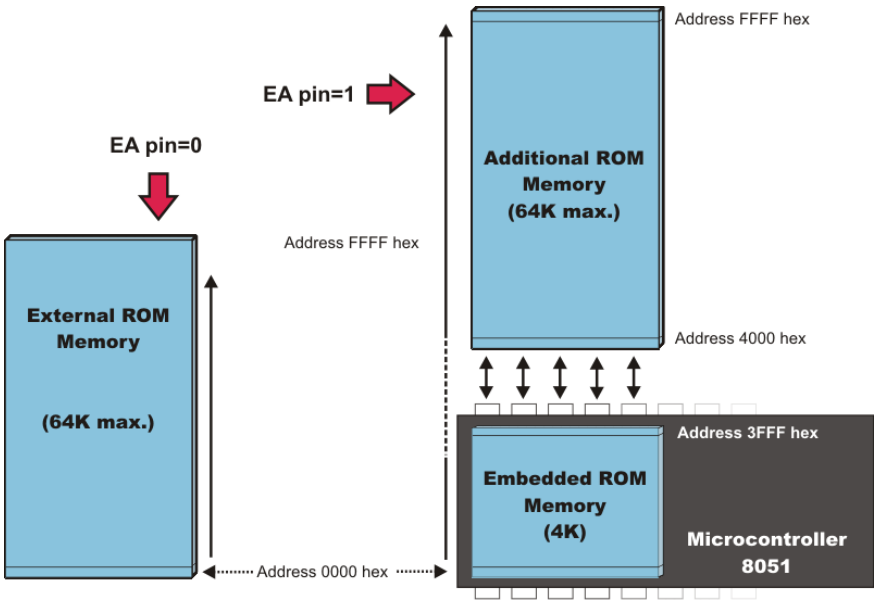
It is active high I/P signal. It should be maintained high for at least two machine cycle while oscillator is running then 8051 microcontroller resets.

Memory Organization

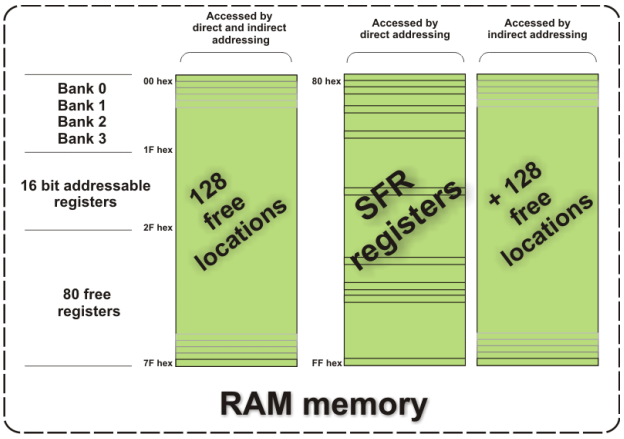
The 8051 microcontroller's memory is divided into Program Memory and Data Memory. Program Memory (ROM) is used for permanent saving program being executed, while Data Memory (RAM) is used for temporarily storing and keeping intermediate results and variables.

Program Memory (ROM)

Program Memory (ROM) is used for permanent saving program (CODE) being executed. The memory is read only. Depending on the settings made in compiler, program memory may also used to store a constant variable. The 8051 executes programs stored in program memory only. code memory type specifier is used to refer to program memory. 8051 memory organization allows external program memory to be added.



Internal Data Memory



Up to 256 bytes of internal data memory are available depending on the 8051 derivative. Locations available to the user occupy addressing space from 0 to 7Fh, i.e. first 128 registers and this part of RAM is divided in several blocks. The first 128 bytes of internal data memory are both directly and indirectly addressable. The upper 128 bytes of data memory (from 0x80 to 0xFF) can be addressed only indirectly.

Since internal data memory is used for CALL stack also and there is only 256 bytes splitted over few different memory areas fine utilizing of this memory is crucial for fast and compact code.

Memory block in the range of 20h to 2Fh is bit-addressable, which means that each bit being there has its own address from 0 to 7Fh. Since there are 16 such registers, this block contains in total of 128 bits with separate addresses (Bit 0 of byte 20h has the bit address 0, and bit 7 of byte 2Fh has the bit address 7Fh).

Three memory type specifiers can be used to refer to the internal data memory: data, idata, and bdata.

8051 RAM Memory

External Data Memory

Access to external memory is slower than access to internal data memory. There may be up to 64K Bytes of external data memory. Several 8051 devices provide on-chip XRAM space that is accessed with the same instructions as the traditional external data space. This XRAM space is typically enabled via proper setting of SFR register and overlaps the external memory space. Setting of that register must be manually done in code, before any access to external memory or XRAM space is made.

SFR Memory

The 8051 provides 128 bytes of memory for Special Function Registers (SFRs). SFRs are bit, byte, or word-sized registers that are used to control timers, counters, serial I/O, port I/O, and peripherals.

PSW: (Program Status Word)

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CY	AC	FO	RS1	RS0	OV	---	P

Bit	Symbol	Function		
0	P	Parity Flag		
1	---			
2	OV	Overflow Flag		
3	RS0	0	0	RB 0
		0	1	RB 1
4	RS1	1	0	RB 2
		1	1	RB 3
5	F0	User Flag 0		
6	AC	Auxiliary Carry Flag		
7	CY	Carry Flag		

IE (Interrupt Enable Control)

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
EA	---	ET2	ES	ET1	EX1	ET0	EX0

Bit	Symbol	Function
0	EX0	Enable external interrupt 0. Set to 1by program to enable INT0 low interrupt; cleared to disable interrupt.
1	ET0	Enable timer 0 overflow interrupt. Set to 1 by program to enable timer 0 overflow interrupt; cleared to 0 to disable timer 0 overflow interrupt.
2	EX1	Enable external interrupt 1.Set to 1 by program to enable INT1 low interrupt; cleared to disable interrupt
3	ET1	Enable timer 1 overflow interrupt. Set to1 by program to enable timer 1 overflow interrupt; cleared to 0 to disable timer 1 overflow interrupt.
4	ES	Enable serial port interrupt. Set to 1 by program to enable serial port interrupt. Cleared to 0 to disable serial port interrupt.
5	ET2	Reserved for future use.
6	---	Not implemented
7	EA	Enable interrupt bit. Cleared to 0 by program to disable all interrupts; Set to 1 to permit individual interrupts to be enable bits.

IP (Interrupt Priority)

D_7	**D_6**	**D_5**	**D_4**	**D_3**	**D_2**	**D_1**	**D_0**
---	---	PT2	PS	PT1	PX1	PT0	PX0

Bit	Symbol	Function
0	PX0	Priority of external interrupt 0. Set/cleared by program
1	PT0	Priority of timer 0 overflow interrupt. Set/cleared by program.
2	PX1	Priority of external interrupt. Set/cleared by program
3	PT1	Priority of timer 1 overflow interrupt 1. Set/cleared by program.
4	PS	Priority of serial port interrupt. Set/cleared by program
5	PT2	Reserved for future use.

6	---	Not implemented
7	---	Not implemented

SBUF (Serial Data Buffer Register)

- ✓ It is an 8 bit register used solely for serial communication in 8051.
- ✓ Byte addressable
- ✓ Used to send / receive the data via serial port.
- ✓ Any value written to SBUF will be sent to TxD pin
- ✓ Any value received by 8051 serial port's RxD pin will be delivered to the user program via SBUF.

PCON (Power Control Register)

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SMOD	**---**	**---**	**---**	**GF1**	**GF0**	**PD**	**IDL**

Bit	Symbol	Function
0	IDL	Writing 1 activate idle mode to save power
1	PD	Writing 1 will power down 8051
2	GF0	General purpose flag bit 0
3	GF1	General purpose flag bit 1
4	---	
5	---	
6	---	
7	SMOD	Serial mode bit is used to determine the serial communication port baud rate with timer 1. Baud Rate = Oscillator Frequency in Hz / N [256-TH1].

SCON (Serial Port Control Register)

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SM0	**SM1**	**SM2**	**REN**	**TB8**	**RB8**	**TI**	**RI**

Bit	Symbol	Function
0	RI	Receive Interrupt flag, set by hardware and must be cleared by software.
1	TI	Transmit Interrupt flag, set by hardware at the end of 8th bit time in mode 0, at the beginning of the stop bit in the other modes, it must be cleared by software.
2	RB8	In mode 2/3, it is the 9th bit that was received. In mode 1, if SM2=0, RB8 is the stop bit that was received, In mode 0, it is not used.
3	TB8	The 9th bit that will be transmitted in mode2/3, set/clear by software.
4	REN	Set/clear by software to enable/disable reception.
5	SM2	Enable multiprocessor communication in modes 2/3.

SM0 (7)	SM1 (6)	Mode/Description/Baud rate
0	0	0,shift register,(Frequency osc./12)
0	1	1,8 bit UART, Variable
1	0	2,9 bit UART,(Frequency osc./64) OR (Frequency osc./32)
1	1	3,9 bit UART, Variable

TCON (Timer Control Register)

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Bit	Symbol	Function
0	IT0	External interrupt 0 signal type control bit. Set to 1 by program to enable external interrupt 0 to be triggered by a falling edge signal. Set to 0 by program to enable a low level signal on external interrupt 1 to generate an interrupt.
1	IE0	External interrupt 0. Edge Set to 1 when a high to low edge signal is received on port 3 pin 3.2(INT0). Cleared when processor. Vector to interrupt service routine located at program address 0003h.Not related to timer operations bit in mode

		1.Not used in mode 0.
2	IT1	External interrupt 1 signal type control bit. Set to 1 by program to enable external interrupt 1 to be triggered by a falling edge signal. Set to 0 by program to enable a low level signal on external interrupt 1 to generate an interrupt.
3	IE1	External interrupts 1Edge flag. Set to 1 when a high to low edge signal is received on port3 pin3.3 (INT1). Cleared when processor vector to interrupt service routine located at program address 0013h. Not related to timer operator program in modes 2 and 3.Stop bit in mode1.Not used in mode 0.
4	TR0	Timer 0 run control bit. Set to 1 by program to enable timer to count; Cleared to 0 by program to halt timer. Does not reset timer.
5	TF0	Timer 0 overflow flag. Set when timer rolls from all 1s to 0s.Cleared when processor vectors to execute interrupt service routine located at program address 000Bh.
6	TR1	Timer 1 run control bit. Set to 1 by program to enable Timer to count; Cleared to 0 by program to halt timer Does not reset timer.
7	TF1	Timer 1 overflow flag. Set when timer rolls from all 1s to 0Cleared when processor vectors execute interrupt service routine located at program address 001Bh.

TMOD (Timer mode)

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
GATE	C/T	M1	M0	GATE	C/T	M1	M0
Timer 1				Timer 0			

M0	M1	Mode	Operating Mode
0	0	0	13-bit timer mode, 8-bit timer/counter THx and TLx as 5-bit prescalar.
0	1	1	16-bit timer mode, 16-bit timer/counters THx and TLx are cascaded; There are no prescalar.
1	0	2	8-bit auto reload mode, 8-bit auto reload timer/counter; THx holds a value which is to be reloaded into TLx each time it overflows.
1	1	3	Split timer mode

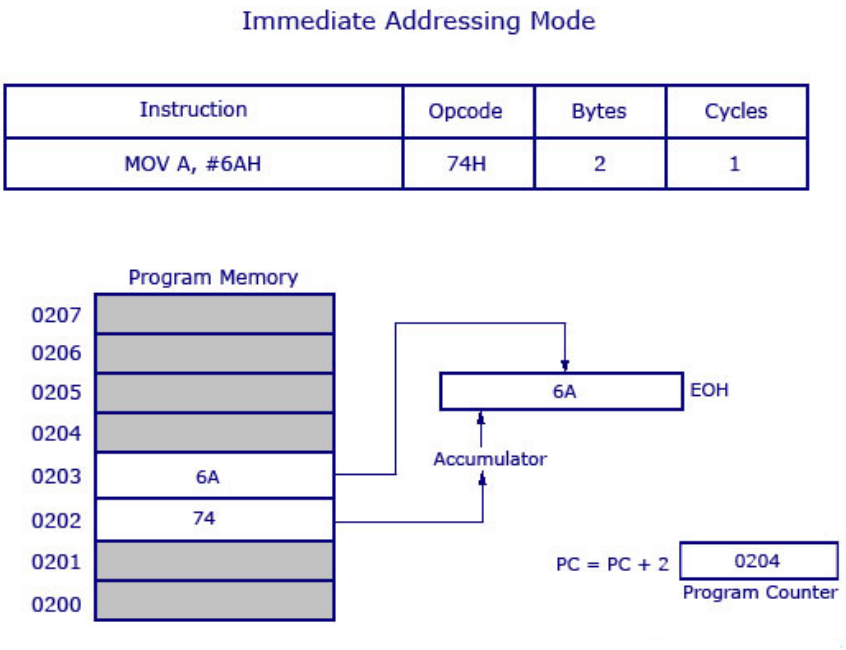
C/T bit is used to decide whether a timer is used as a time delay generator or an event counter. If this bit is 0 then it is used as a timer and if it is 1 then it is used as a counter.

GATE: Every timer has a means of starting and stopping. Some timers do this by software, some by hardware, and some have both software and hardware controls. The hardware way of starting and stopping the timer by an external source is achieved by making GATE=1 in the TMOD register. And if we change to GATE=0 then we do not need external hardware to start and stop the timers.

Addressing Modes and Instruction set of 8051

Addressing mode is a way to address an operand. Operand means the data we are operating upon (in most cases source data). It can be a direct address of memory, it can be register names, it can be any numerical data etc. The addressing modes are explained with a simple data move instruction of 8051.

Immediate Addressing Mode



MOV A, #6AH

In general we can write MOV A, #data

This addressing mode is named as “Immediate” because it transfers an 8-bit data immediately to the accumulator (destination operand).

The opcode for MOV A, # data is 74H. The opcode is saved in program memory at 0202 address. The data 6AH is saved in program memory 0203. (See, any part of the program memory can be used, this is just an example) When the opcode 74H is read, the next step taken would be to transfer whatever data at the next program memory address (here at 0203) to accumulator A (E0H is the address of accumulator). This instruction is of two bytes and is

executed in one cycle. So after the execution of this instruction, program counter will add 2 and move to o204 of program memory.

Direct Addressing Mode

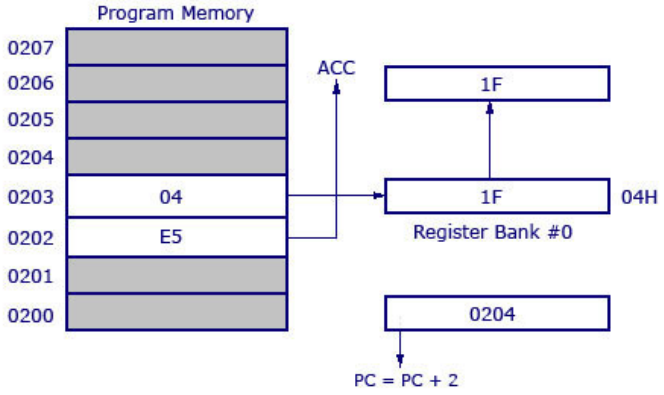
MOV A, 04H

Here 04H is the address of register 4 of register bank#0. When this instruction is executed, whatever data is stored in register 04H is moved to accumulator. In the picture below we can see, register 04H holds the data 1FH. So the data 1FH is moved to accumulator.

Note: We have not used '#' in direct addressing mode, unlike immediate mode. If we had used '#', the data value 04H would have been transferred to accumulator instead of 1FH.

Direct Addressing Mode

Instruction	Opcode	Bytes	Cycles
MOV A, #04H	E5	2	1

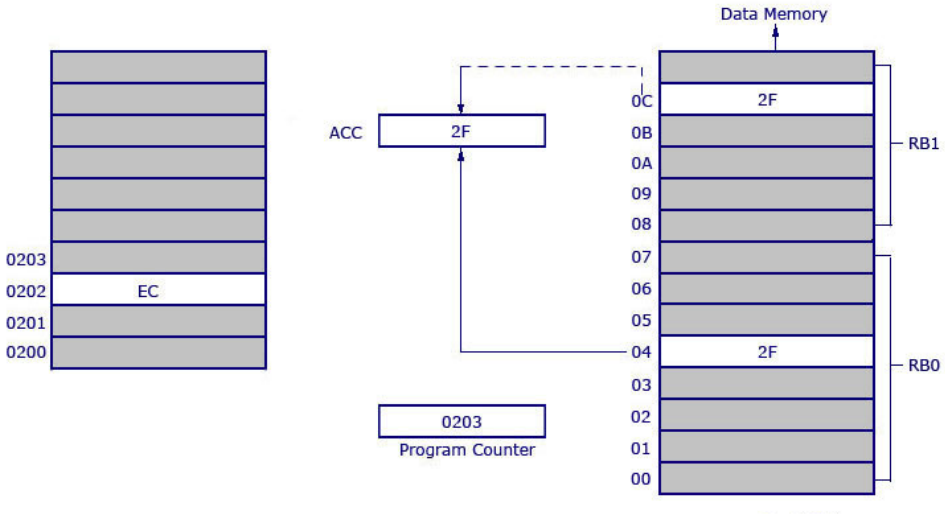


Register Direct Addressing Mode

MOV A, R4

Register Direct Addressing Mode

Instruction	Opcode	Bytes	Cycles
MOV A, R4	ECH	1	1



At a time registers can take value from R0,R1…to R7. You may already know there are 32 such registers. So how you access 32 registers with just 8 variables to address registers? Here comes the use of register banks. There are 4 register banks named 0,1,2 and 3. Each bank has 8 registers named from R0 to R7. At a time only one register bank can be selected. Selection of register bank is made possible through a Special Function Register (SFR) named Processor Status Word (PSW). PSW is an 8 bit SFR where each bit can be programmed. Bits are designated from PSW.0 to PSW.7 Register banks are selected using PSW.3 and PSW.4 These two bits are known as register bank select bits as they are used to select register banks. A picture below shows the PSW register and the Register Bank Select bits with status.

Register Indirect Addressing Mode

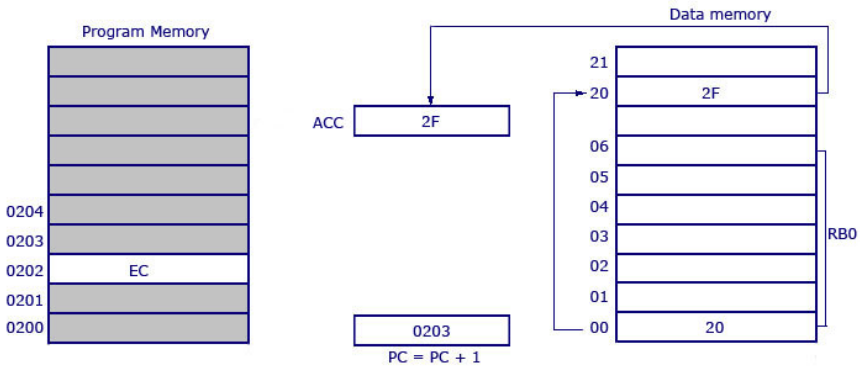
MOV A, @R0

Here the value inside R0 is considered as an address, which holds the data to be transferred to accumulator.

Example: If R0 holds the value 20H, and we have a data 2F H stored at the address 20H, then the value 2FH will get transferred to accumulator after executing this instruction.

Register Indirect Addressing Mode

Instruction	Opcode	Bytes	Cycles
MOV A, @ R0	E6H	1	1



So the opcode for MOV A, @R0 is E6H. Assuming that register bank #0 is selected. So the R0 of register bank #0 holds the data 20H. Program control moves to 20H where it locates the data 2FH and it transfers 2FH to accumulator.

This is a single byte instruction and the program counter increments 1 and moves to 0203 of program memory.

Note: Only R0 and R1 are allowed to form a register indirect addressing instruction. In other words programmer can make any instruction either using @R0 or @R1. All register banks are allowed.

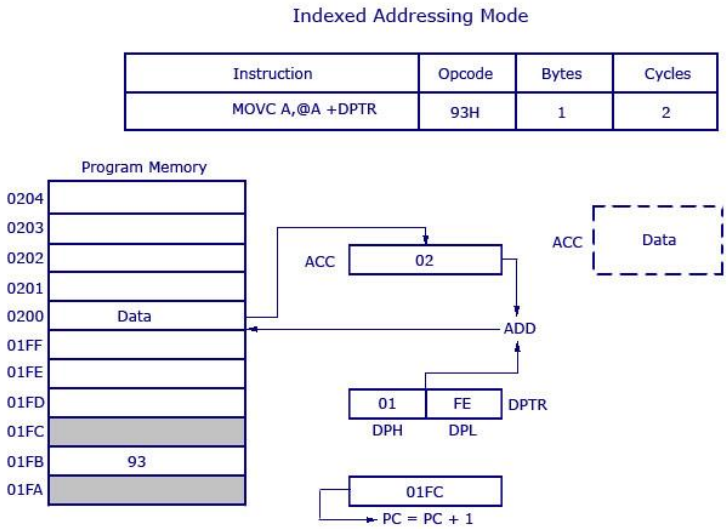
Indexed Addressing Mode

MOVC A, @A+DPTR and MOVC A, @A+PC

where DPTR is data pointer and PC is program counter (both are 16 bit registers). Lets take the first example.

MOVC A, @A+DPTR

The source operand is @A+DPTR and we know we will get the source data (to transfer) from this location. It is nothing but adding contents of DPTR with present content of accumulator. This addition will result a new data which is taken as the address of source data (to transfer). The data at this address is then transferred to accumulator.



The opcode for the instruction is 93H. DPTR holds the value 01FE, where 01 is located in DPH (higher 8 bits) and FE is located in DPL (lower 8 bits). Accumulator now has the value 02H. A 16 bit addition is performed and now 01FE H+02 H results in 0200 H. Whatever data is in 0200 H will get transferred to accumulator. The previous value inside accumulator (02H) will get replaced with new data from 0200H. New data in the accumulator is shown in dotted line box.

This is a 1 byte instruction with **2 cycles needed for execution**. The execution time required for this instruction is high compared to previous instructions (which all were 1 cycle).

The other example **MOVC A, @A+PC** works the same way as above example. The only difference is, instead of adding DPTR with accumulator, here data inside program counter (PC) is added with accumulator to obtain the target address.

Instruction Set

Depending on operation they perform, all instructions are divided in several groups:

- Arithmetic Instructions
- Branch Instructions
- Data Transfer Instructions
- Logic Instructions
- Bit-oriented Instructions

The first part of each instruction, called MNEMONIC refers to the operation an instruction performs (copy, addition, logic operation etc.). Mnemonics are abbreviations of the name of operation being executed. For example:

- INC R1 – Means: Increment register R1 (increment register R1);
- LJMP LAB5 – Means: Long Jump LAB5 (long jump to the address marked as LAB5);
- JNZ LOOP – Means: Jump if Not Zero LOOP (if the number in the accumulator is not 0, jump to the address marked as LOOP);

The other part of instruction, called OPERAND is separated from mnemonic by at least one whitespace and defines data being processed by instructions. Some of the instructions have no operand, while some of them have one, two or three. If there is more than one operand in an instruction, they are separated by a comma. For example:

- RET – return from a subroutine;
- JZ TEMP – if the number in the accumulator is not 0, jump to the address marked as TEMP;
- ADD A,R3 – add R3 and accumulator;
- CJNE A,#20,LOOP – compare accumulator with 20. If they are not equal, jump to the address marked as LOOP;

Arithmetic instructions

Arithmetic instructions perform several basic operations such as addition, subtraction, division, multiplication etc. After execution, the result is stored in the first operand. For example:

ADD A,R1 – The result of addition (A+R1) will be stored in the accumulator.

ARITHMETIC INSTRUCTIONS			
Mnemonic	**Description**	**Byte**	**Cycle**
ADD A,Rn	Adds the register to the accumulator	1	1
ADD A,direct	Adds the direct byte to the accumulator	2	2
ADD A,@Ri	Adds the indirect RAM to the accumulator	1	2
ADD A,#data	Adds the immediate data to the accumulator	2	2
ADDC A,Rn	Adds the register to the accumulator with a carry flag	1	1
ADDC A,direct	Adds the direct byte to the accumulator with a carry flag	2	2
ADDC A,@Ri	Adds the indirect RAM to the accumulator with a carry flag	1	2
ADDC A,#data	Adds the immediate data to the accumulator with a carry flag	2	2
SUBB A,Rn	Subtracts the register from the accumulator with a borrow	1	1
SUBB A,direct	Subtracts the direct byte from the accumulator with a borrow	2	2
SUBB A,@Ri	Subtracts the indirect RAM from the accumulator with a borrow	1	2
SUBB A,#data	Subtracts the immediate data from the accumulator with a borrow	2	2
INC A	Increments the accumulator by 1	1	1
INC Rn	Increments the register by 1	1	2
INC Rx	Increments the direct byte by 1	2	3
INC @Ri	Increments the indirect RAM by 1	1	3
DEC A	Decrements the accumulator by 1	1	1
DEC Rn	Decrements the register by 1	1	1
DEC Rx	Decrements the direct byte by 1	1	2
DEC @Ri	Decrements the indirect RAM by 1	2	3
INC DPTR	Increments the Data Pointer by 1	1	3
MUL AB	Multiplies A and B	1	5
DIV AB	Divides A by B	1	5
DA A	Decimal adjustment of the accumulator according to BCD code	1	1

Branch Instructions

There are two kinds of branch instructions:

Unconditional jump instructions: upon their execution a jump to a new location from where the program continues execution is executed.

Conditional jump instructions: a jump to a new program location is executed only if a specified condition is met. Otherwise, the program normally proceeds with the next instruction.

BRANCH INSTRUCTIONS			
Mnemonic	**Description**	**Byte**	**Cycle**
ACALL addr11	Absolute subroutine call	2	6
LCALL addr16	Long subroutine call	3	6
RET	Returns from subroutine	1	4
RETI	Returns from interrupt subroutine	1	4
AJMP addr11	Absolute jump	2	3
LJMP addr16	Long jump	3	4
SJMP rel	Short jump (from –128 to +127 locations relative to the following instruction)	2	3
JC rel	Jump if carry flag is set. Short jump.	2	3
JNC rel	Jump if carry flag is not set. Short jump.	2	3
JB bit,rel	Jump if direct bit is set. Short jump.	3	4
JBC bit,rel	Jump if direct bit is set and clears bit. Short jump.	3	4
JMP @A+DPTR	Jump indirect relative to the DPTR	1	2
JZ rel	Jump if the accumulator is zero. Short jump.	2	3
JNZ rel	Jump if the accumulator is not zero. Short jump.	2	3
CJNE A,direct,rel	Compares direct byte to the accumulator and jumps if not equal. Short jump.	3	4
CJNE A,#data,rel	Compares immediate data to the accumulator and jumps if not equal. Short jump.	3	4
CJNE Rn,#data,rel	Compares immediate data to the register and jumps if not equal. Short jump.	3	4
CJNE @Ri,#data,rel	Compares immediate data to indirect register and jumps if not equal. Short jump.	3	4
DJNZ Rn,rel	Decrements register and jumps if not 0. Short jump.	2	3
DJNZ Rx,rel	Decrements direct byte and jump if not 0. Short jump.	3	4
NOP	No operation	1	1

Data Transfer Instructions

Data transfer instructions move the content of one register to another. The register the content of which is moved remains unchanged. If they have the suffix “X” (MOVX), the data is exchanged with external memory.

DATA TRANSFER INSTRUCTIONS			
Mnemonic	**Description**	**Byte**	**Cycle**
MOV A,Rn	Moves the register to the accumulator	1	1
MOV A,direct	Moves the direct byte to the accumulator	2	2
MOV A,@Ri	Moves the indirect RAM to the accumulator	1	2
MOV A,#data	Moves the immediate data to the accumulator	2	2

MOV Rn,A	Moves the accumulator to the register	1	2
MOV Rn,direct	Moves the direct byte to the register	2	4
MOV Rn,#data	Moves the immediate data to the register	2	2
MOV direct,A	Moves the accumulator to the direct byte	2	3
MOV direct,Rn	Moves the register to the direct byte	2	3
MOV direct,direct	Moves the direct byte to the direct byte	3	4
MOV direct,@Ri	Moves the indirect RAM to the direct byte	2	4
MOV direct,#data	Moves the immediate data to the direct byte	3	3
MOV @Ri,A	Moves the accumulator to the indirect RAM	1	3
MOV @Ri,direct	Moves the direct byte to the indirect RAM	2	5
MOV @Ri,#data	Moves the immediate data to the indirect RAM	2	3
MOV DPTR,#data	Moves a 16-bit data to the data pointer	3	3
MOVC A,@A+DPTR	Moves the code byte relative to the DPTR to the accumulator (address=A+DPTR)	1	3
MOVC A,@A+PC	Moves the code byte relative to the PC to the accumulator (address=A+PC)	1	3
MOVX A,@Ri	Moves the external RAM (8-bit address) to the accumulator	1	3-10
MOVX A,@DPTR	Moves the external RAM (16-bit address) to the accumulator	1	3-10
MOVX @Ri,A	Moves the accumulator to the external RAM (8-bit address)	1	4-11
MOVX @DPTR,A	Moves the accumulator to the external RAM (16-bit address)	1	4-11
PUSH direct	Pushes the direct byte onto the stack	2	4
POP direct	Pops the direct byte from the stack/td>	2	3
XCH A,Rn	Exchanges the register with the accumulator	1	2
XCH A,direct	Exchanges the direct byte with the accumulator	2	3
XCH A,@Ri	Exchanges the indirect RAM with the accumulator	1	3
XCHD A,@Ri	Exchanges the low-order nibble indirect RAM with the accumulator	1	3

Logic Instructions

Logic instructions perform logic operations upon corresponding bits of two registers. After execution, the result is stored in the first operand.

LOGIC INSTRUCTIONS			
Mnemonic	**Description**	**Byte**	**Cycle**
ANL A,Rn	AND register to accumulator	1	1
ANL A,direct	AND direct byte to accumulator	2	2
ANL A,@Ri	AND indirect RAM to accumulator	1	2
ANL A,#data	AND immediate data to accumulator	2	2
ANL direct,A	AND accumulator to direct byte	2	3
ANL direct,#data	AND immediae data to direct register	3	4
ORL A,Rn	OR register to accumulator	1	1
ORL A,direct	OR direct byte to accumulator	2	2
ORL A,@Ri	OR indirect RAM to accumulator	1	2
ORL direct,A	OR accumulator to direct byte	2	3
ORL direct,#data	OR immediate data to direct byte	3	4
XRL A,Rn	Exclusive OR register to accumulator	1	1
XRL A,direct	Exclusive OR direct byte to accumulator	2	2
XRL A,@Ri	Exclusive OR indirect RAM to accumulator	1	2
XRL A,#data	Exclusive OR immediate data to accumulator	2	2
XRL direct,A	Exclusive OR accumulator to direct byte	2	3
XORL direct,#data	Exclusive OR immediate data to direct byte	3	4
CLR A	Clears the accumulator	1	1
CPL A	Complements the accumulator (1=0, 0=1)	1	1
SWAP A	Swaps nibbles within the accumulator	1	1
RL A	Rotates bits in the accumulator left	1	1
RLC A	Rotates bits in the accumulator left through carry	1	1
RR A	Rotates bits in the accumulator right	1	1

RRC A	Rotates bits in the accumulator right through carry	1	1

Bit-oriented Instructions

Similar to logic instructions, bit-oriented instructions perform logic operations. The difference is that these are performed upon single bits.

BIT-ORIENTED INSTRUCTIONS			
Mnemonic	**Description**	**Byte**	**Cycle**
CLR C	Clears the carry flag	1	1
CLR bit	Clears the direct bit	2	3
SETB C	Sets the carry flag	1	1
SETB bit	Sets the direct bit	2	3
CPL C	Complements the carry flag	1	1
CPL bit	Complements the direct bit	2	3
ANL C,bit	AND direct bit to the carry flag	2	2
ANL C,/bit	AND complements of direct bit to the carry flag	2	2
ORL C,bit	OR direct bit to the carry flag	2	2
ORL C,/bit	OR complements of direct bit to the carry flag	2	2
MOV C,bit	Moves the direct bit to the carry flag	2	2
MOV bit,C	Moves the carry flag to the direct bit	2	3

UNIT -V:

8051 Real Time Control: Programming Timer Interrupts, Programming External Hardware Interrupts, Programming the Serial Communication Interrupts, Programming 8051 Timers and Counters

The 8051 has two timers: timer0 and timer1. They can be used either as timers or as counters. Both timers are 16 bits wide. Since the 8051 has an 8-bit architecture, each 16-bit is accessed as two separate registers of low byte and high byte. First we shall discuss about Timer0 registers.

Timer0 registers is a 16 bits register and accessed as low byte and high byte. The low byte is referred as a TL0 and the high byte is referred as TH0. These registers can be accessed like any other registers.

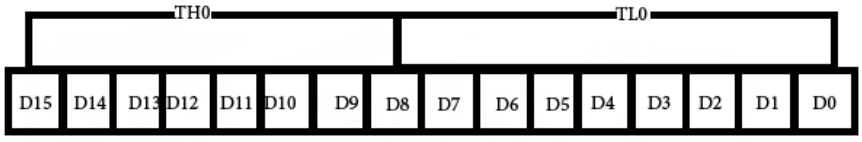


Fig. Timer0

Timer1 registers is also a 16 bits register and is split into two bytes, referred to as TL1 and TH1.

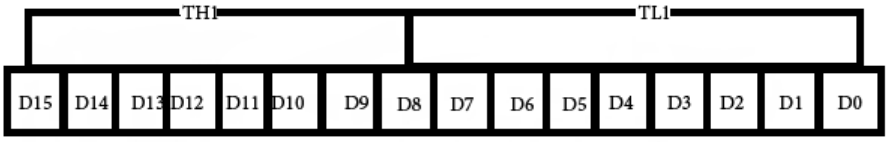


Fig. Timer1

TMOD (timer mode) Register: This is an 8-bit register which is used by both timers 0 and 1 to set the various timer modes. In this TMOD register, lower 4 bits are set aside for timer0 and the upper 4 bits are set aside for timer1. In each case, the lower 2 bits are used to set the timer mode and upper 2 bits to specify the operation.

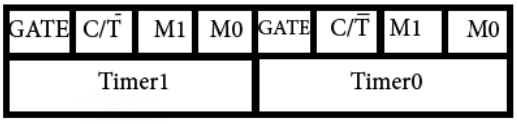


Fig. TMOD Register

In upper or lower 4 bits, first bit is a GATE bit. Every timer has a means of starting and stopping. Some timers do this by software, some by hardware, and some have both software and hardware controls. The hardware way of starting and stopping the timer by an external source is achieved by making GATE=1 in the TMOD register. And if we change to GATE=0 then we do not need external hardware to start and stop the timers.

The second bit is C/T bit and is used to decide whether a timer is used as a time delay generator or an event counter. If this bit is 0 then it is used as a timer and if it is 1 then it is used as a counter.

In upper or lower 4 bits, the last bits third and fourth are known as M1 and M0 respectively. These are used to select the timer mode.

M0	**M1**	**Mode**	**Operating Mode**
0	0	0	13-bit timer mode, 8-bit timer/counter THx and TLx as 5-bit prescalar.
0	1	1	16-bit timer mode, 16-bit timer/counters THx and TLx are cascaded; There are no prescalar.
1	0	2	8-bit auto reload mode, 8-bit auto reload timer/counter; THx holds a value which is to be reloaded into TLx each time it overflows.
1	1	3	Split Timer Mode

Mode 1- It is a 16-bit timer; therefore it allows values from 0000 to FFFFH to be loaded into the timer's registers TL and TH. After TH and TL are loaded with a 16-bit initial value, the timer must be started. We can do it by "SETB TR0" for timer 0 and "SETB TR1" for timer 1. After the timer is started, It starts count up until it reaches its limit of FFFFH. When it rolls over from FFFF to 0000H, it sets high a flag bit called TF (timer flag). This timer flag can be monitored. When this timer flag is raised, one option would be stop the timer with the instructions "CLR TR0" or CLR TR1 for timer 0 and timer 1 respectively. Again, it must be noted that each timer flag TF0 for timer 0 and TF1 for timer1. After the timer reaches its limit and rolls over, in order to repeat the process the registers TH and TL must be reloaded with the original value and TF must be reset to 0.

Mode0- Mode 0 is exactly same like mode 1 except that it is a 13-bit timer instead of 16-bit. The 13-bit counter can hold values between 0000 to 1FFFH in TH-TL. Therefore, when the timer reaches its maximum of 1FFH, it rolls over to 0000, and TF is raised.

Mode 2- It is an 8 bit timer that allows only values of 00 to FFH to be loaded into the timer's register TH. After TH is loaded with 8 bit value, the 8051 gives a copy of it to TL. Then the timer must be started. It is done by the instruction "SETB TR0" for timer 0 and "SETB TR1" for timer1. This is like mode 1. After timer is started, it starts to count up by incrementing the TL register. It counts up until it reaches its limit of FFH. When it rolls over from FFH to 00. It sets high the TF (timer flag). If we are using timer 0, TF0 goes high; if using TF1 then TF1 is raised. When Tl register rolls from FFH to 00 and TF is set to 1, TL is reloaded automatically with the original value kept by the TH register. To repeat the process, we must simply clear TF and let it go without any need by the programmer to reload the original value. This makes mode 2 auto reload, in contrast in mode 1 in which programmer has to reload TH and TL.

Mode3- Mode 3 is also known as a split timer mode. Timer 0 and 1 may be programmed to be in mode 0, 1 and 2 independently of similar mode for other timer. This is not true for mode 3; timers do not operate independently if mode 3 is chosen for timer 0. Placing timer 1 in mode 3 causes it to stop counting; the control bit TR1 and the timer 1 flag TF1 are then used by timer0.

TCON register- Bits and symbol and functions of every bits of TCON are as follows:

Fig. TCON Register

BIT	Symbol	Functions
7	TF1	Timer1 over flow flag. Set when timer rolls from all 1s to 0. Cleared When the processor vectors to execute interrupt service routine Located at program address 001Bh.
6	TR1	Timer 1 run control bit. Set to 1 by programmer to enable timer to count; Cleared to 0 by program to halt timer.
5	TF0	Timer 0 over flow flag. Same as TF1.
4	TR0	Timer 0 run control bit. Same as TR1.
3	IE1	External interrupt 1 Edge flag. Not related to timer operations.
2	IT1	External interrupt1 signal type control bit. Set to 1 by program to Enable external interrupt 1 to be triggered by a falling edge signal. Set To 0 by program to enable a low level signal on external interrupt1 to generate an interrupt.
1	TE0	External interrupt 0 Edge flag. Not related to timer operations.
0	IT0	External interrupt 0 signal type control bit. Same as IT0.

8051 Interrupts

An interrupt is some event which interrupts normal program execution. Program flow is always sequential, being altered only by those instructions which expressly cause program flow to deviate in some way. However, interrupts give us a mechanism to "put on hold" the normal program flow, execute a subroutine, and then resume normal program flow as if we had never left it. This subroutine, called an interrupt handler, is only executed when a certain event (interrupt) occurs. The event may be one of the timers "overflowing," receiving a character via the serial port, transmitting a character via the serial port, or one of two "external events."

The 8051 may be configured so that when any of these events occur the main program is temporarily suspended and control passed to a special section of code which presumably would execute some function related to the event that occurred. Once complete, control would be returned to the original program. The main program never even knows it was interrupted.

The ability to interrupt normal program execution when certain events occur makes it much easier and much more efficient to handle certain conditions. If it were not for interrupts we would have to manually check in our main program whether the timers had over flown, whether we had received another character via the serial port, or if some external event had occurred.

Besides making the main program ugly and hard to read, such a situation would make our program inefficient since wed be burning precious "instruction cycles" checking for events that usually don’t happen.

The 8051 provides five interrupt sources. These are listed below.

1. Timer 0 (TF0) and timer 1 (TF1) interrupt.

2. External hardware interrupts, INT0 and INT1.

3. Serial communication Interrupt TI and RI.

Interrupt Vector Table

In 8051, all interrupts are vectored interrupts and have vector locations as listed in the Table when interrupt is activated 8051 reads the address of interrupt service routine from the vector location.

Interrupt	**Vector Location**
External Hardware Interrupt (INT0)	0003H
Timer 0 interrupt (TF0)	000BH
External hardware Interrupt 1 (INT1)	0013H
Timer 1 interrupt (TF1 Overflow)	001BH
Serial communication interrupt (RI and TI) (Reception / Transmission of Serial Character)	0023H

Timer Interrupts and Programming

The Timer 0 and Timer 1 Interrupts are generated by TFO and TF1, which are set by a rollover in their respective Timer/Counter registers (except see Timer 0 in Mode 3). When a timer interrupt is generated, the flag that generated it is cleared by the on-chip hardware when the service routine is vectored.

As the timer flag (TF) is set (=1) when the timer rolls over. In polling method, the TF is monitored with the instruction 'JNB TF, target address'. We have to wait until the TF is raised. The problem with this polling method is that 8051 cannot do anything else until TF is set to high. This problem can be solved using interrupt method. If the timer interrupt in the IE register is enabled, TF is set whenever the timer is rolled over and the 8051 is interrupted. Thus the 8051 can perform anything else until it is interrupted. After interruption (timer

rolling over) only the 8051 remains busy in executing interrupt service routine.

Programming External Hardware Interrupts

Pins, P 3.2 (pin number 12) and P 3.3 (pin number 13) in port 3 are used as external hardware interrupts INT0 and INT1, respectively. The external Interrupts INTO and INT1 can each be either level-activated or transition-activated, depending on bits ITO and IT1 in Register TCON. The flags that actually generate these interrupts are bits IEO and IE1 in TCON. When an external interrupt is generated, the flag that generated it is cleared by the hardware when the service routine is vectored to only if the interrupt was transition-activated. If the interrupt was level- activated, then the external requesting source is what controls the request flag, rather than the on- chip hardware.

If ITx = 0, external interrupt x is triggered by a detected low at the INTx pin. If ITx = 1, external interrupt x is edge-triggered. In this mode if successive samples of the INTx pin show a high in one cycle and a low in the next cycle, interrupt request flag IEx in TCON is set. Flag bit IEx then requests the interrupt. Since the external interrupt pins are sampled once each machine cycle, an input high or low should hold for at least 12 oscillator periods to ensure sampling. If the external interrupt is transition-activated, the external source has to hold the request pin high for at least one machine cycle, and then hold it low for at least one machine cycle to ensure that the transition is seen so that interrupt request flag IEx will be set. IEx will be automatically cleared by the CPU when the service routine is called.

If the external interrupt is level-activated, the external source has to hold the request active until the requested interrupt is actually generated. Then it has to deactivate the request before the interrupt service routine is completed, or else another interrupt will be generated.

Example: Write an 8051 ALP to glow LED for a fraction of second when external interrupt INTO is activated.

Solution

```
ORG 0003H
SETB P1.0 ;              Turn ON LED
MOVE R2, #0FFH ;         Load count
DJNZ BACK ;              Decrement count and if not zero repeat
CLR P1.0 ;               Turn OFF LED
RET I ;                  Return to main program
MOV IE, #10000001B ;     Enable external interrupt 0
                         ; wait for interrupt
SJMP HERE
END
```

Serial Communication Interrupts and Programming

The Serial port Interrupt is generated by the logical OR of RI and TI. Neither of these flags is cleared by hardware when the service routine is vectored to. In fact, the service routine will normally have to determine whether it was RI or TI that generated the interrupt, and the bit will have to be cleared in software.

In this case, the 8051 can perform other tasks in addition to serial communication, i.e. sending and receiving data from serial communication port. We know that transmit interrupt (TI) flag is set (=1) when the last bit of the framed data (stop bit) is transmitted. This indicates that the SBUF register is ready to transmit the next byte. The receive interrupt (RI) flag is set (=1) when the complete frame of data (with stop bit) is received. RI indicates that the received byte needs to be picked up before it is lost by new incoming serial data.

All the above concepts are applied equally using polling or an interrupt. Only difference is in serving the serial communication needs. In polling method, the flag (TI or RI) is monitored. The 8051 cannot do anything else until this flag is set to high. This problem is solved using interrupt method. When 8051 has received a byte or is ready to send the next byte, the RI or TI flag respectively is set. Any other work can be performed while the serial communication needs are served. There is a single interrupt set aside for serial communication. If IE register (IE.4) is enabled, when RI or TI is set (= 1), the 8051 is interrupted. When interrupted, the ISR written at 0023h is executed by 8051. In ISR, the TI and RI flags must be examined to check which one caused the interrupt and according to flag the response is given.

Interrupt priority

Each interrupt source can also be individually programmed to one of two priority levels by setting or clearing a bit in Special Function Register IP. A low-priority interrupt can itself be interrupted by a high-priority interrupt, but not by another low priority interrupt. A high-priority interrupt can't be interrupted by another interrupt source.

-	-	PT2	PS	PT1	PX0	PT0	PX0

Bit	Symbol	Bit Address	Description (1=high, 0=low priority)
IP.7	- -		Undefined
IP.6	- -		Undefined
IP.5	PT2	BDH	Priority for timer 2 interrupt (8052)
IP.4	PS	BCH	Priority for serial port interrupt
IP.3	PT1	BBH	Priority for timer 1 interrupt
IP.2	PX1	BAH	Priority for external 1 interrupt
IP.1	PT0	B9H	Priority for timer 0 interrupt

IP.0 PX0 B8H Priority for external 0 interrupt

– 0= lower priority, 1= higher priority, reset IP=00H

– Lower priority ISR can be interrupted by a high priority interrupt.

– A high priority ISR cannot be interrupted.

Processing Interrupts

• When an interrupt occurs and is accepted by the CPU, the main program is interrupted. The following actions occur:

– The current instruction completes execution.

– The PC is saved on the stack.

– The current interrupt status is saved internally.

– Interrupts are blocked at the level of the interrupt.

– The PC is loaded with the vector address of the ISR

– The ISR executes. • The ISR finishes with an RETI instruction, which retrieves the old value of PC from the stack and restores the old interrupt status. Execution of the main program continues where it left off.

A 10-KHz Square Wave on P1.0 Using Timer Interrupts

```
ORG   0                  ; Reset entry point
LJMP  Main
ORG   000BH              ; T0 interrupt vector
T0ISR: CPL   P1.0        ; Toggle port bit
RETI
ORG   0030H
Main: MOVTMOD,#02H       ; T0 MODE 2
MOV TH0,#-50             ; 50 mS DELAY
SETB TR0                 ; START TIMER 0
MOV IE,#82H              ; ENABLE T0 INT
SJMP $                   ; DO NOTHING
```

Two 7-KHz & 500-Hz Square Waves Using Interrupts

```
ORG 0                    ; reset entry point
LJMP Main
ORG 000BH                ; T0 interrupt vector
LJMP T0ISR
ORG 001BH                ; T1 vector
```

```
LJMP T1ISR
ORG 0030H                ; main starts here
Main: MOV TMOD,#12H      ; T0 mode 2,
                         ; T1 mode 1
MOV TH0,#-71             ; 7-kHz using T0
SETB TR0                 ; START TIMER 0
SETB TF1                 ; force T1 int
MOV IE,#8AH              ; ENABLE T0,T1 INT
SJMP $                   ; DO NOTHING
T0ISR:CPL P1.7           ; 7-kHz on P1.7
RETI
T1ISR:CLR TR1            ; stop T1
MOV TH1,#HIGH(-1000)
MOV TL1,#LOW(-1000)      ;1 ms for T1
SETB TR1                 ; START TIMER 1
CPL P1.6                 ; 500-Hz on P1.6
RETI
```