|  |
| --- |
| **PART –B**  **Microcontroller Programming** |

**Experiment No. 6**

**Title:** 8-Bit array addition with 8051.

**Aim:** Write 8051 ALP to add n, 8 bits numbers found in internal ram location 40H onwards

And store results in R6 and R7.

**Objective:** 1. Study how to write ALP for Microcontroller.

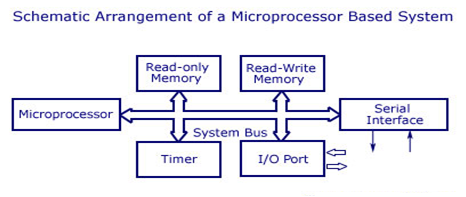
2. To accesses an internal memory and perform arithmetic operations of microcontroller.

**Theory**-

**Introduction to Microcontrollers**

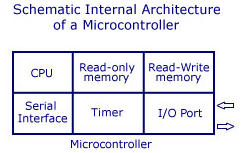
**Microprocessor is a General purpose machine. (Many task –single machine)**

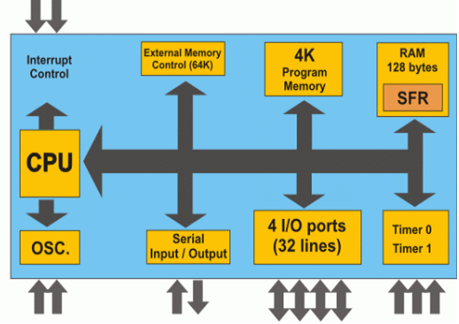
* It is CPU only
* Needs many external ICs to implement a small system (RAM,ROM,ADC,Timers, PPI 8255 etc)
* Many times Program and data memory is not separate.

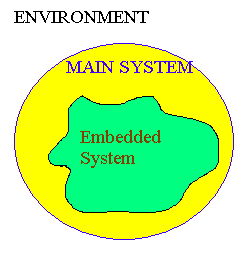


**Microcontrollers:**

* **Specific function machine (single task –single machine)**
* It is CPU plus RAM, ROM, ADC, Timers, serial ports etc on single chip.
* To implement a small system no need of external IC’S such as RAM, ROM, ADC, Timers, serial ports etc.
* Separate Program and data memory.





**** Fig: Block diagram of microcontroller (all devices on single chip)

**Embedded System (8051 Applications)**

An embedded system is closely integrated with the main system

It may not interact directly with the Environment

For example – A microcomputer in a car ignitioncontrol

* + An embedded product uses a microprocessor or microcontroller to do one task only.
  + There is only one application software that is typically burned into ROM

**Examples of Embedded Systems(Applications of microcontroller).**

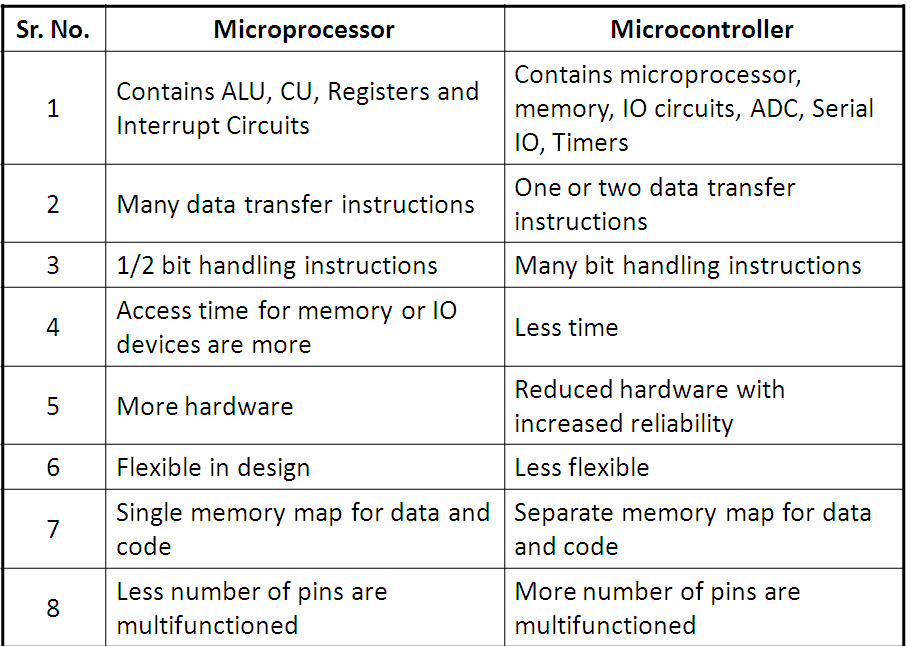
1. Keyboard
2. Printer
3. video game player
4. MP3 music players
5. Embedded memories to keep configuration information
6. Mobile phone units
7. Domestic (home) appliances(washing m/c etc)
8. Data switches
9. Automotive controls

And many more ………….

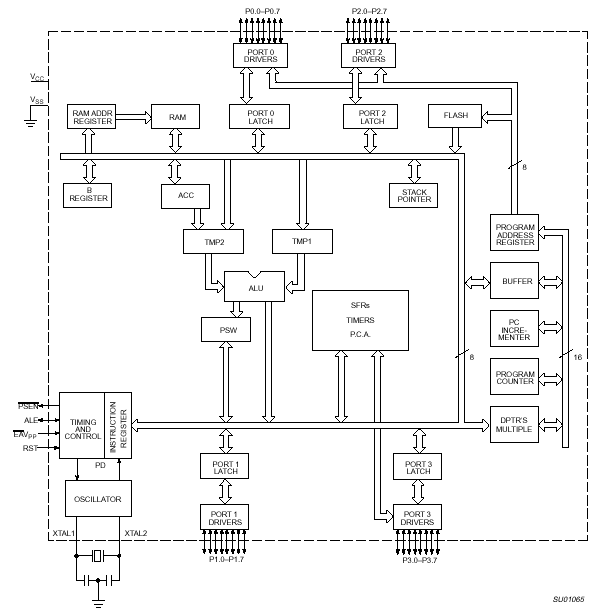
**Features of 8051 Microcontroller:**

1. Specific function machine (single task –single machine)
2. separate Program and data memory
3. It is a 8-bits Microcontroller means It’s Data bus is 8 bits
4. On chip 4K bytes internal ROM
5. On chip 128 bytes internal RAM
6. On chip Four 8-bit I/O ports (P0 - P3).
7. On chip Two 16-bit timers/counters
8. On chip One serial interface
9. only 1 On chip **oscillator** (external crystal)
10. 6 interrupt sources (2 external , 3 internal, Reset) 64K external **code** (program) memory(**only read)PSEN**64K external data memory(**can be read and write**) by **RD,WR**Code memory is selectable by **EA** (internal or external)
11. We may have External **memory** as **data** and **code**

**Differences between Microprocessors and Microcontrollers**

****

**8051 Internal Block Diagram**

****

**8051 CPU Registers:**

1. A (Accumulator)
2. B (B-register (temporary)
3. PSW (Program Status Word)
4. SP (Stack Pointer)
5. PC (Program Counter)
6. DPTR (Data Pointer)

**A**

**B**

**R0**

**R1**

**R3**

**R4**

**R2**

**R5**

**R7**

**R6**

**DPH**

**DPL**

**PC**

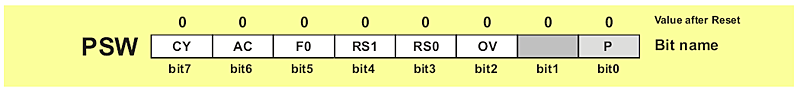
**DPTR**

**PC**

**Some 8051 16-bit Register**

**Some 8-bit Registers of the 8051**

**Program Status Word (PSW) Register (flag Register):**



PSW register is one of the most important SFRs. It contains several status bits that reflect the current state of the CPU. Besides, this register contains Carry bit, Auxiliary Carry, two register bank select bits, Overflow flag, parity bit and user-definable status flag.

**P - Parity bit.** If a number stored in the accumulator is even then this bit will be automatically set (1), otherwise it will be cleared (0). It is mainly used during data transmit and receive via serial communication.

**- Bit 1.** This bit is intended to be used in the future versions of microcontrollers.

**OV Overflow** occurs when the result of an arithmetical operation is larger than 255 and cannot be stored in one register. Overflow condition causes the OV bit to be set (1). Otherwise, it will be cleared (0).

**RS0, RS1 - Register bank select bits.** These two bits are used to select one of four register banks of RAM. By setting and clearing these bits, registers R0-R7 are stored in one of four banks of RAM.

|  |  |  |
| --- | --- | --- |
| **RS1** | **RS2** | **SPACE IN RAM** |
| 0 | 0 | Bank0 00h-07h |
| 0 | 1 | Bank1 08h-0Fh |
| 1 | 0 | Bank2 10h-17h |
| 1 | 1 | Bank3 18h-1Fh |

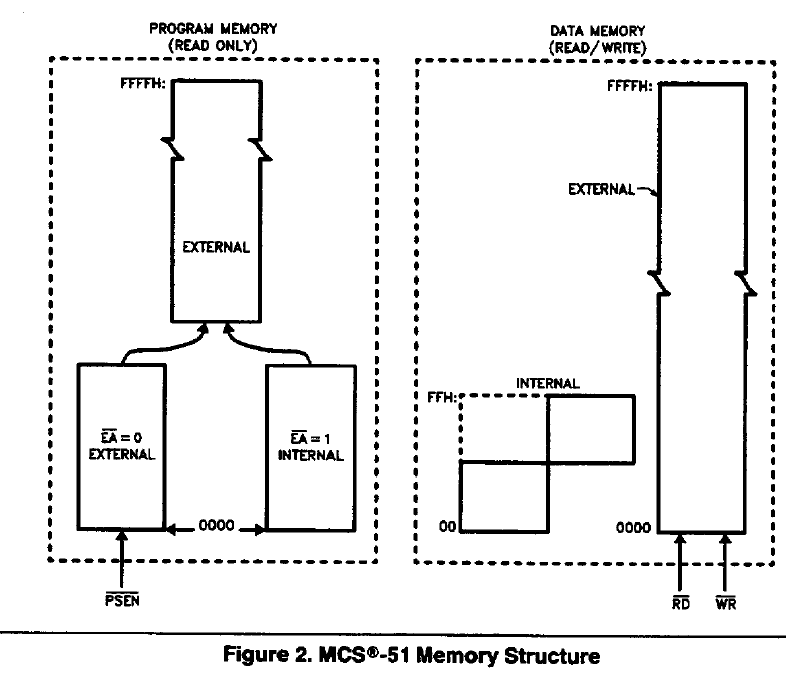
**F0 - Flag 0.** This is a general-purpose bit available for use.

**AC - Auxiliary Carry Flag** is used for BCD operations only.

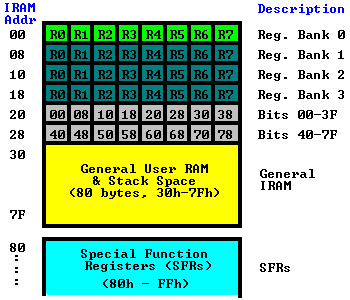
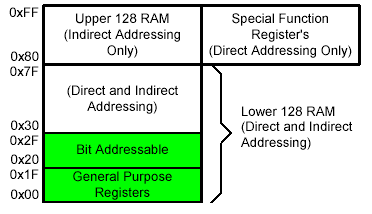
**CY - Carry Flag** is the (ninth) auxiliary bit used for all arithmetical operations and shift instructions.

**Memory organization in 8051:** The 8051 memory divided into following and physical parts as shown in fig.

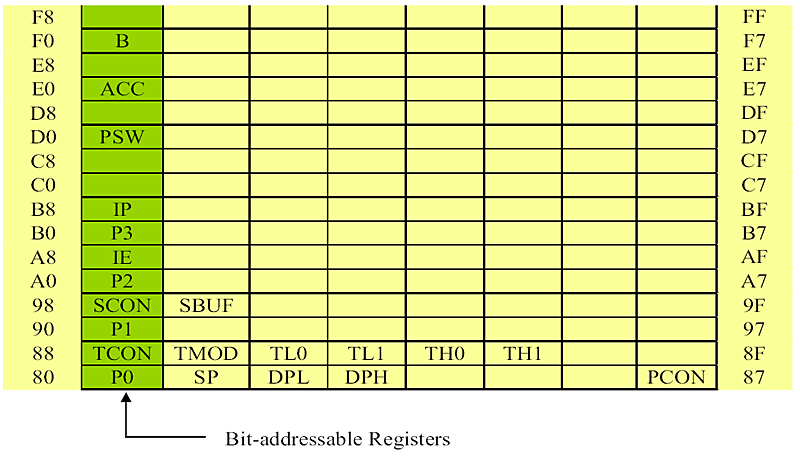
1. Internal RAM
2. Internal special function registers.
3. External RAM
4. Internal & External RAM

****

**On-Chip Memory (Internal RAM)** :



**Special Function Registers (SFRs)**: Special Function Registers (SFRs) are a sort of control table used for running and monitoring the operation of the microcontroller. Each of these registers as well as each bit they include, has its name, address in the scope of RAM and precisely defined purpose such as timer control, interrupt control, serial communication control etc.



**Fig: Special Function Registers (SFRs)**

**Addressing modes:**

An "addressing mode" refers to how you are addressing a given memory location. In summary, the addressing modes are as follows, with an example of each:

|  |  |
| --- | --- |
| **Immediate Addressing** | MOV A,#20h |
| **Direct Addressing** | MOV A,30h |
| **Indirect Addressing** | MOV A,@R0 |
| **External Direct** | MOVX A,@DPTR |
| **Code Indirect** | MOVC A,@A+DPTR |

**Instruction used-**

1. **MOV DES,SRC-**

This instruction copies a byte from source to destination. It does not affect the flags.

Ex.MOV Rx,Ax copy data from accumulator to Register.

1. **ADD Ax,SRC-**

The instruction adds source byte the content in accumulator source operand must also be one byte. All addition is done with registers as destination of the result.

ADD Ax,Rx

ADD A to register Rx and put the result in A.

1. **INC byte-**

This instruction add one to the register as memory location specified by operand is not affected even if value FF increase by 1 to 00. It supports accumulator register direct and registers indirect addressing modes.

INC A: ADD 01 to register A

INC Rx: ADD o to register Rx.

1. **JNZ Target-**

Jump if no carry the subtraction examining carry flag and if it 0 it will jump to target address.

Ex. JNZ add

**Algorithm**:-

1. Initialize pointer to the initial position for pointing first element of an array.

2. Initialize counter with number of array elements and for carry record (Initially it is Zero).

3. Start with first element and storage variable of added result.

4. Increment the pointer and Decrement the counter.

5. After each addition check whether carry is generated, if yes then increment carry counter by one.

6. Go on adding numbers in internal memory till counter becomes zero. If not then go for cumulative addition and incrimination of pointer and decrement of counter, till counter of array count becomes zero.

7. Display the result of addition on R6 and R7.

**INPUT:** An array of n-8 bit numbers stored in internal memory (where n is any integer value)

**OUTPUT:** Result of addition of n-8 bit numbers displayed on R6 and R7.

**CONCLUSION:** Internal memory access are studied by performing addition of 8 bit numbers stored in an internal memory

**Outcome expected from assignment:**

1. Student should able to write ALP for 8051 Microcontroller.

2. Student will be able to input values in internal memory as array element.

3. Student will be able to access internal memory for each array element addition.

4. Student will be able to justify the usage of instruction for particular program.

5. Student will be able differentiate between different addressing mode.

**ENHANCEMENTS / MODIFICATIONS:**

Write a program to add n, 16 bits numbers

**Experiment No. 6(b)**

**Title:** ALP using Microcontroller 8051 for a block transfer.

**Aim:** Write an ALP using Microcontroller 8051 for a block transfer for internal/external Memory.

**Objective:** To access and transferred the block of n-element from one location to other by Using data transfer group of instruction and with the help of special register.

**Theory**-

**Addressing modes:**

An "addressing mode" refers to how you are addressing a given memory location. In summary, the addressing modes are as follows, with an example of each:

|  |  |
| --- | --- |
| **Immediate Addressing** | MOV A,#20h |
| **Direct Addressing** | MOV A,30h |
| **Indirect Addressing** | MOV A,@R0 |
| **External Direct** | MOVX A,@DPTR |
| **Code Indirect** | MOVC A,@A+DPTR |

**INSTRUCTION USED:**

MOV A @, R1: move indirect ROM to accumulator.

MOV @ R1,A: move accumulator to internal RAM.

DJNZ R0, rel: decrement register &jump if not equal to zero.

**Algorithm**:-

1. Initialize the pointers and counters with starting or ending location with array size in one

General purpose registers.

2. Access each memory location one by one sequentially

3. Move the data from internal memory pointed by register to accumulator.

4. Move the data from accumulator to other memory location pointed by register.

5. Increment pointers register (input, output) to point to next location.

6. Decrement counter register by one and check for block is finish or not i.e count is zero, if not

Jump back block element copy and storage.

7. If counter value is null, then shift the pointer registers to point other memory location i.e

Destination block transferred location.

8. Access each byte from this block one by one to display the result.

9. Stop

**INPUT:** The number of bytes in the block to be transfer at source address location. Toward

Destinations address location all location contents are initially zero or undefined.

**OUTPUT:** The data has been transferred from source address location to destination Address location. Display the result of all byte transferred successfully.

**CONCLUSION:** Internal memory access and External memory access are studied by performing Block transfer operation on internal memory to External memory and vice versa.

**Outcome expected from assignment:**

1. Student should able to write ALP for 8051 Microcontroller.

2. Student will be able to input values in internal memory/External memory as a Block.

3. Student will be able to access internal memory/ External memory for each element.

4. Student will be able to justify the usage of instruction for particular program.

5. Student will be able differentiate between different addressing mode.

**ENHANCEMENTS / MODIFICATIONS:**

Write a program to Transfer, 16 bits numbers

**Experiment No. 7**

**Title:** ALP writing for multiplication with 8051.

**Aim:** Write 8051 ALP to multiply 16 bit number by 8 bit number and store the result in internal memory location.

**Objective:** To access and transferred the block of n-element from one location to other by Using data transfer group of instruction and with the help of special register.

**Theory**-

Consider that byte is present at memory location 9000H and second is present at memory location 9001H.We will multiply bytes present at above two memory location using MUL instruction. As MUL instruction operator only on the A and B registers we will used two no result of multiplication is stored in accumulator while digit is stored is ‘B’ register.

**Instructions used**:

**1) MUL:**

Multiplication operation use register A and B as both source and destination address from operation. The unsigned no. in register as follows.

**MUL AB**

Multiply A and B put low order byte of product in A put low order byte of product in A put higher bytes in B the overflow flag will in set if AB > FEH. It signed if no. larger than 8 bit and programmer need in set register B for higher order byte of multiplication operator. The carry flag is always cleared to ‘0’.

**2)MOV Rr, #n:** Copy the 8 bit no. into register Rr of current register that.

Eg. MOV Ro # 40H

Put 8 bit memory address in register Ro where 40H is internal memory location in above example of immediate addressing mode is used.

**3) INC and DEC:**

This is simple arithmetic operation involving addition and subtraction of content by 1.

**Programming Tip**: In fact, any multiplication results in an answer which is the sum of the bits in the two multiplicands. For example, multiplying an 8-bit value by a 16-bit value results in a 24-bit value (8 + 16). A 16-bit value multiplied by another 16-bit value results in a 32-bit value (16 + 16), etc.

**For 16 bits \* 8 bits:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| . |  | **Byte 3** | **Byte 2** | **Byte 1** |
| . |  | . | **62** | **30** |
| \* |  | . |  | **2E** |
| = |  | . | 08 | A0 |
| . |  | 11 | 9C | . |
| . |  | 0C | 90 | . |
| . |  | A6 | . | . |
| = |  | **C4** | **34** | **A0** |
|  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| . |  | **Byte 3** | **Byte 2** | **Byte 1** |
| \* |  |  | R6 | R7 |
| \* |  |  |  | R5 |
| = |  | **R1** | **R2** | **R3** |

1. Multiply R5 by R7, leaving the 16-bit result in R2 and R3.
2. Multiply R5 by R6, adding the 16-bit result to R1 and R2.

**For 16-bit \* 16-bit value:**

For the sake of example, let's multiply **25,136** by **17,198**. The answer is 432,288,928. As with both addition and subtraction, let's first convert the expression into hexadecimal: **6230h** x **432Eh**.

Once again, let's arrange the numbers in columns as we did in primary school to multiply numbers, although now the grid becomes more complicated. The green section represents the original two values. The yellow section represents the intermediate calculations obtained by multipying each byte of the original values. The red section of the grid indicates our final answer, obtained by summing the columns in the yellow area.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| . | **Byte 4** | **Byte 3** | **Byte 2** | **Byte 1** |
| . | . | . | **62** | **30** |
| \* | . | . | **43** | **2E** |
| = | . | . | 08 | A0 |
| . | . | 11 | 9C | . |
| . | . | 0C | 90 | . |
| . | 19 | A6 | . | . |
| = | **19** | **C4** | **34** | **A0** |

Remember how we did this in elementary school? First we multiply 2Eh by 30h (byte 1 of both numbers), and place the result directly below. Then we multiply 2Eh by 62h (byte 1 of the bottom number by byte 2 of the upper number). This result is lined up such that the right-most column ends up in byte 2. Next we multiply 43h by 30h (byte 2 of the bottom number by byte 1 of the top number), again lining up the result so that the right-most column ends up in byte 2. Finally, we multiply 43h by 62h (byte 2 of both numbers) and position the answer such that the right-most column ends up in byte 3. Once we've done the above, we add each column, with appropriate carries, to arrive at the final answer.

Our process in assembly language will be identical. Let's use our now-familiar grid to help us get an idea of what we're doing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| . | **Byte 4** | **Byte 3** | **Byte 2** | **Byte 1** |
| \* |  |  | R6 | R7 |
| \* |  |  | R4 | R5 |
| = | **R0** | **R1** | **R2** | **R3** |

Thus our first number will be contained in R6 and R7 while our second number will be held in R4 and R5. The result of our multiplication will end up in R0, R1, R2 and R3. At 8-bits per register, these four registers give us the 32 bits we need to handle the largest possible multiplication. Our process will be the following:

1. Multiply R5 by R7, leaving the 16-bit result in R2 and R3.
2. Multiply R5 by R6, adding the 16-bit result to R1 and R2.
3. Multiply R4 by R7, adding the 16-bit result to R1 and R2.
4. Multiply R4 by R6, adding the 16-bit result to R0 and R1.

**Algorithm**:-

1. Initialize pointer to memory.

2. Copy the content from memory location and store it in corresponding internal registers.

3. Store the lower byte of the 16 bit number into accumulator A, and 8 bit number into register B

4. Use instruction MUL AB

5. Store lower byte of product in one of general purpose register.

6. Store the higher byte of product in one of the other general purpose register.

7. Get higher byte of 16 bit number in accumulator and 8 bit number into register B.

8. Use instruction MUL AB.

9. Store lower byte of product in one of general purpose register.

10. Add lower byte of the product with higher byte of previous multiplication.

11. If carry record it.

12. Add carry with the higher byte of the result generated after second multiplication.

13. Store the result into general purpose registers.

14. Addition gives most significant byte of the result.

15. Store the result in consecutive memory locations.

16. Display 3 bytes of the result one after other from memory location or on onboard LEDs.

**INPUT:** 16 bit and 8 bit number stored in internal memory locations.

**OUTPUT:** Result of 16 bit x 8 bit multiplication displayed on onboard LEDs

**CONCLUSION:** In this way we can perform 16 bits by 8 bits multiplication by writing ALP to MCS 8051 and also Internal memory access are studied.

.

**Outcome expected from assignment:**

1. Student will be able use special instruction as MUL.

2. Student will be able to store the result in three consecutive locations.

3. Student will be able to show the result at respective location successfully.

**ENHANCEMENT/MODIFICATIONS:**

1. Student can go for 16 bit by 8 bit division.

OR

2. 16 BIT X 16 BIT Number multiplication.

**Experiment No. 8(a)**

### TITLE: Timer Programming: ISR based

**AIM:** Write assembly language program to generate a square wave of 2 KHz on any port pin. Use Timer with interrupt for delay generation.

**OBJECTIVE:** 1. Understanding the Timers and the required SFRs.

2. Use of Timer to generate the time delays.

S/W & H/W USED: 1. Kiel IDE Compiler / Assembler

2. Flash magic Software for downloading

3. 8051 Trainer kit.

4. CRO

**THEORY:**

### TIME DELAY GENRATION:

Most used subroutine is one that generates a programmable time delay. Time delays may be done by using software loops that essentially do nothing for some period or by using hardware timers that count internal clock pulses. The key to writing this program is to calculate the exact time each instruction will take at the clock frequency in use. Following terms are very helpful to write a subroutine which generates desired time delay.

**T-state**: T-state is defined as one subdivision of the operation performed in one clock period. The terms T-state and clock period are often used synonymously.

**Machine Cycle:** Machine cycle in 8051 is defined as 12 oscillator periods. The 8051, take one to four machine cycles to execute an instruction. To calculate the machine cycle for the 8051, we take 1/12 of crystal frequency, and then take its inverse.

Assume crystal frequency of 11.0592 MHz

M/C frequency = 11.0592 MHz/12 = 921.6 KHz

Machine Cycle = 1/921.6 KHz = 1.085 us (microseconds)

**Instruction Cycle:** Instruction cycle is defined as the time required for completing the execution of an instruction. One instruction cycle consists of one to four machine cycles.

e.g. 2 Machine cycles are required for instruction DJNZ R2, target to be executed. Then instruction cycle is calculated as follows.

Instruction cycle = No. Machine cycles x Machine cycle period

= 2 x 1.085 us

= 2.17 us

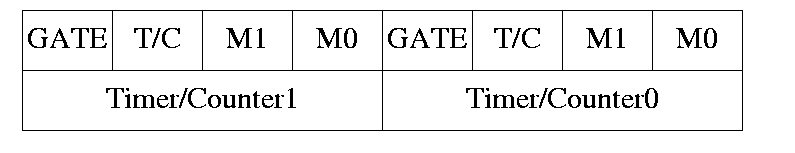
### TIMER/ COUNTER IN 8051:

8051 has 2 timer/ counter. They can be used a timers to generate a time delay or as counter to count events happening outside the microcontroller. Both Timer 0 and Timer 1 are 16 bits wide. Since the 8051 has an 8-bit architecture, each 16-bit timer register is accessed as two separate registers of low byte and high byte.

* Timer 0 can be accessed as –
  + TL0 – Timer 0 lower byte
  + TH0 – Timer 0 higher byte
* Timer 1 can be accessed as –
  + TL1 - Timer 1 lower byte
  + TH1 – Timer 1 higher byte

Both timer shares the Timer control (TCON) register, which controls the timer/ counter operation and Timer mode (TMOD) register, which is used to configure the timer for different operating modes.

***TMOD (Timer Mode Register):***

******

* Both the timers used the same 8 bit register to set various timer operation mode.
* TMOD is 8-bit register where lower 4 byte are set aside for timer 0 and higher 4 bytes for timer 1. Since, it is not bit addressable; the corresponding bit value is directly loaded into TMOD.

*Gate:*

* 8051 has both hardware and software controls to start and stop the timers.
* By the means of software controlling instruction timers are used to control to start timer or stop.

*C/T:*

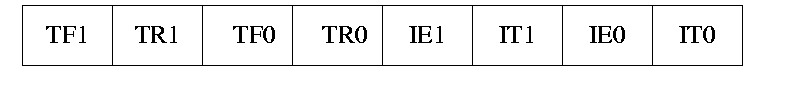
* This bit in TMOD is used to determine whether timer is to be used as delay generator or event counter.
  + If C/T = 0 – used as timer
  + If C/T = 1 – used as counter

*M1 M0:*

M1, M0 selects the timer mode.

|  |  |  |  |
| --- | --- | --- | --- |
| M1 | M0 | Mode | Operation |
| 0 | 0 | 0 | 13 bit counter, 8 bit C/T with THX and TLX as 5 bit Prescalar. |
| 0 | 1 | 1 | 16 bit counter, 8 bit C/T with THX and TLX cascaded with no Prescalar. |
| 1 | 0 | 2 | 8 bit auto reload, THX hold the value which is to be loaded into TLX after each overflow. |
| 1 | 1 | 3 | Split timer mode. |

***TCON (Timer Control Register):***

******

* Timer run control bits TR0 and TR1 and timer overflow flags TF0 and TF1 are the part of 8 bit register called TCON.
* The upper 4 bits are used to store TR and TF flags of both timer- 0 and timer 1 while the lower 4 bits are set aside for interrupt.
* Timer run control bit TR0/TR1 is used to start the corresponding timer / counter.
* Timer overflow flag bit TF0/TF1 is set when corresponding timer/ counter is overflowed i.e. count value FFFF h to 0000 h.

***Different Modes of timer/ counter –***

1. **Mode 0 (13 bit timer/counter):**

* Mode 0 is exactly like mode 1 except it is 13 bit timer.
* Hence it can hold the values from 0000H to 1FFFH in TL and TH.
* When timer rolls over from 1FFFH to 0000H, the overflow flag i.e. TFX is set.

1. **Mode 1 (16 bit Timer / counter):**

* It is 16 bit timer. Hence allowed values from 0000H to FFFFH to be loaded in TLX and THX.
* After the corresponding 16 bit value is loaded, the timer is started by setting TRX flags.
* After starting, it counts up until it reaches the limit i.e. FFFFH. When it rolls over from FFFFH to 0000H, it sets timer overflow flag i.e. TFX flag.
* After the rolling over process, the operation in mode 1 can be repeated by loading the initial value in TLX & THX and clearing TFX bit.

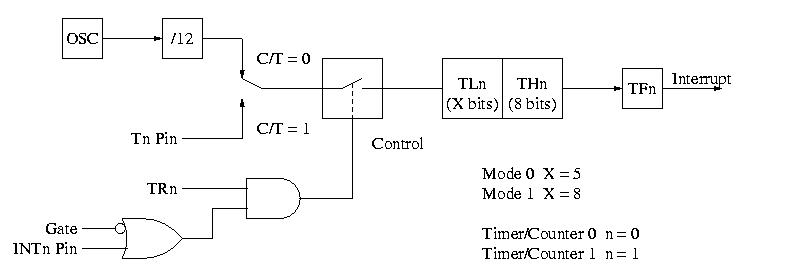


Figure 3.1: Timer/ Counter in Mode 0 and Mode 1

1. **Mode 2 (8 bit auto reload):**

* It is 8 bit timer. Hence it allows only values from 00H to FFH to be loaded in THX.
* When THX is loaded into 8 bit value, it sends a copy of it to corresponding TLX. Then timer must be started which is done by SETB TR1 for T1.
* After rolling over of TLX from FFH to 00H, overflow flag for corresponding timer i.e. TFX is set. TLX auto loaded by value present in THX.

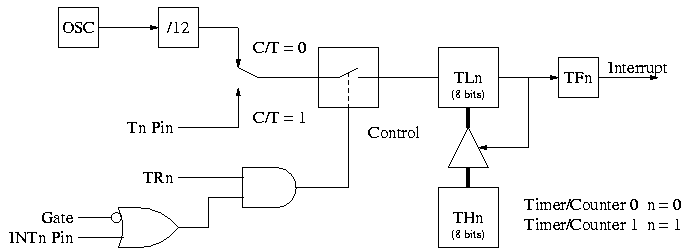


Figure 3.2: Timer/ Counter in Mode 2

1. **Mode 3 (Split timer mode):**

* In mode 3, timer0 worked as split timer i.e. two independent timer.
* TL0 uses the TR0 and TF0 bits of timer 0.
* TH0 uses the TR1 and TF1 bits of timer 1.

### 

Figure 3.3: Timer/ Counter in Mode 3

### TIME DELAY GENERATION USING TIMER:

Mode 1 and Mode 2 are widely used for most of the applications. Mode 1 is used for time delay generation and mode 2 is used to generate the baud rate in serial communication.

The following steps are taken to generate a time delay using the mode 1 and polling method.

1. Load the TMOD value indicating which timer is to be used and timer mode 1 is selected.
2. Load the registers TL and TH with initial count values. Delay generated is Depends upon the initial count value.
3. Start the timer. (SETB TR0 or SETB TR1)
4. Keep the monitoring the timer flag (TF) with the “JNB TF0, target” or “JNB TF1, target” instruction.
5. Get out of the loop when TF becomes high.
6. Stop the timer. (CLR TR0 or CLR TR1)
7. Clear the TF flag for the next round. (CLR TF0 or CLR TF1)
8. Go back to the step 2 to load TH and TL values.

The size of the time delay depends on two factors, (a) the crystal frequency and (b) the timer’s 16-bit register in mode 1. The largest delay is achieved by the making both TH and TL zero.

Formula for delay calculations using mode 1 of the timer for crystal frequency of XTAL = 11.0592 MHz, (TH, TL) = (NNNNN)10 is as follows.

Time Delay (Td) = (65536 - NNNNN) x 1.085 us.

Therefore Maximum delay = (65536 – 0000) x 1.085 us = 71 ms.

***Finding the Values to be loaded into timer for desired delay***

Assume the 11.0592 MHz as crystal frequency for 8051.

* Divide the desired time delay by 1.085us. ( n = Td/1.085us)
* Perform 65536 – n. where n is the decimal value from step 1.
* Convert the result of step 2 to hexadecimal, where yyxx is the initial hex value to be loaded into the timer’s registers.
* Set TL = xx and TH = yy.

**­­­­­­­­­­­­­­­­­­­­­­­­­­­Calculations:**

Frequency = 2 KHz

Machine Cycle = 1.085 us (microseconds)

Time period = TP = 1/2x103 = 0.5 ms

Required duty cycle is 50%

Therefore, Ton = Toff = 1/TP = 0.25 ms

Desired time delay is TD = 0.25 ms

Divide TD by 1.085x10-6 = n = 0.25x10-3/1.085x10-6 = 230

Subtract n from 65536 = 65536 – 230 = 65306

Convert above decimal value in to Hex value = FF1Ah

Load this value into Timer Register. (TH = FFh , TL = 1Ah)

**ALGORITHM: To generate the 2 KHz square wave on port pin**

1. **Main Program**
2. Load the value “10h” in TMOD register indicating Timer-1 is to be used and timer mode 1 is selected.
3. Load the registers TL and TH with initial count values i.e. FF1Ah.

(TH = FFh, TL = 1Ah)

1. Enable the Timer-1 interrupt by loading the value “88h” in IE register.
2. Start the timer by setting TR1 bit in TCON register.
3. Halt the program.
4. **ISR Routine**
5. Complement the port bit on which square wave is to be monitored.
6. Reload the registers TL and TH with initial count values i.e. FF1Ah.

(TH = FF H, TL = 1A H)

1. Return from ISR.

**INPUT:** Count Values and timer with mode values are loaded. Connect the CRO on appropriate pin on which waveform to be observed.

**OUTPUT:** 2KHZ Square waveform on C.R.O

**CONCLUSION:** Timer operations and time delay generation in 8051 are studied and generated a 2KHZ Square wave by ISR method.

**Outcome expected from assignment:**

1. Student will be able use special function registers.

2. Student will be able to write ALP for Timer programming using ISR.

3. Student will be able to Interface 8051 board and write interfacing programs.

**ENHANCEMENT/MODIFICATION:** Write the same program for different frequency.

**Experiment No. 9**

**AIM:W**rite ALP to interface 8051 with DAC and writing programs to generate triangular, trapezoidal and sine waveforms.

Objective: To learn and understand the interfacing of DAC with 8051.

Write an ALP to accept digitize input and display its equivalent on CRO.

(ASSUME DAC 0808 )

S/W & H/W USED: 1.Kiel IDE Compiler / Assembler

2. Flash magic Software for downloading

3. 8051 Trainer kit.

4. CRO

**Theory:**

**Digital-to-analog converter(DAC):**

Digital-to-analog converter (DAC or D-to-A) is a device for converting a digital (usually binary) code to an analog signal (current, voltage or electric charge).An analog-to-digital converter (ADC) performs the reverse operation.

The DAC0808 is an 8-bit monolithic digital-to-analog converter (DAC) featuring a full scale output current settling time of 150 ns while dissipating only 33 mW with ±5V supplies. No reference current (IREF) trimming is required for most applications since the full scale output current is typically ±1 LSB of 255 IREF/256. Relative accuracies of better than ±0.19% assure 8-bit monotonicity and linearity while zero level output current of less than 4 µA provides 8-bit zero accuracy for IREF>=2 mA. The power supply currents of the DAC0808 is independent of bit codes, and exhibits essentially constant device characteristics over the entire supply voltage range.The DAC0808 will interface directly with popular TTL, DTL or CMOS logic levels, and is a direct replacement for the MC1508/MC1408

Figure shows interfacing diagram of 8051 and DAC 0808. This circuit generates a Analog output according to Digital input. Output of DAC 0808 is a current. So to convert this current into voltage a OP-AMP based I to V convertor is required which is also shown in figure.

Output of DAC is Current given by

**Iout= Iref \* (D7/2 + D6/4 + D5/8 + D4/16 + D3/32 + D2/64 + D1/128 + D0/256 )**

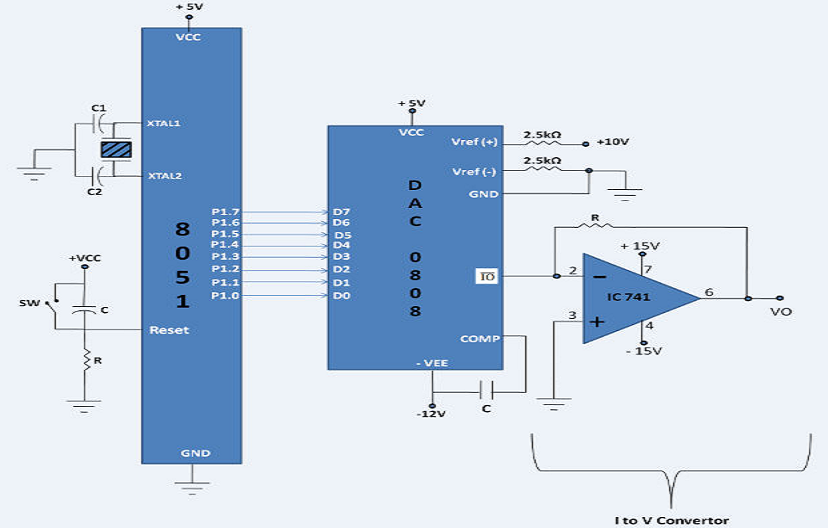
Where D7 (MSB) to D0 (LSB) bits in a Digital input

Iref = 2 mA typically

And I to V convertor gives output as a

**Vout= Iout \* Rref**

**Interfacing Diagram of DAC 0808 with Microcontroller 8051:**

****

**Fig: Interfacing Diagram of DAC 0808 with Microcontroller 8051**

**Algorithms and flowcharts:**

Write an 8051 ALP to generate following waveforms using Digital to Analog converter.

* Square wave
* Triangular wave
* Sine wave
* Trapezoidal wave .
* Ramp wave

**a) To generate a square waveform.**

With 00 as input to DAC, the analog output is 0v. Similarly, with FFh as input, the output is +5v,. Outputting digital data 00 and FF at regular intervals to DAC, results in a square wave of amplitude 5 volts.

### Algorithm

1. Clear accumulator.

2. Send A’s content on port P1.

3. Call Delay

4. Complement the content (or copy FFH) in A Register.

5. Send A’s content on port 1

6. Call Delay

7. JMP back to 1 step

**Draw flowchart**

**b) To generate a triangular waveform**

With 00 as input to DAC, the analog output is 0v. Incrementing this value by step 1, the analog voltage increases and finally it reaches to +5v (when the value is FFh). Start decrementing this value to 00h will decrease the output voltage to 0v creates triangular waveform.

### Algorithm

1. Clear accumulator.

2. Send A’s content on port P1.

3. INC Accumulator

4. Compare A with FFH.

5. If A reaches FF then Decrement A.

6. JMP back to 1 step to repeat

**Draw flowchart**

**c) To generate a sine waveform**

### Algorithm

1. START

2. Load Count Value (24h) in R2 Register.

3. Load memory address of starting lookup table into R0 and R1.

4. Load A with the content of DPTR and output it to FFC0h

5. Increment memory Pointer to point it to the next location.

6. Decrement Counter value (R2),

7. If Counter is not 00, go to step 4.

8. Jump back to step 2.

9. STOP

**Draw flowchart**

**d) To generate a Trapezoidal waveform**

1. Clear accumulator.

2. Send A’s content on port P1.

3. INC Accumulator

4. Compare A with FFH.

5. CALL delay

5. Decrement A.

6. Compare A with 00H

6. JMP back to 1 step to repeat

**e) To generate a Saw Tooth waveform**

In order to generate saw tooth

1. Clear accumulator.

2. Send A’s content on port P1.

3. Increment the content.

4. Send it again on port1.

5. If A’s content are equal to FFH.

6. Then again start from 1.

7. If the result of step 5 is uneual then jmp back to step 3.

8. JMP back to 1 step

**f) To generate a Ramp waveform**

1. Clear accumulator.

2. Send A’s content on port P1.

3. Increment the content of A by one

4. Compare it with FFH.

5. IF UNEQUAL then move to step 7

6. then decrement the content by one.

7. Resend it again on port 1.

8. JMP back to 2 step

9. If the result in step 5 is equal.

10. Decrement the content by one from accumulator

11. Compare it with zero if unequal

12. Send it on port 1

13. Jmp back step 10

**Flowcharts: (Draw )**

**Conclusion:**

In this way, we can interface 8051 with DAC kits and generate different waveforms to indicate Analog conversion.

**Experiment No. 10**

**Title:** Stepper motor interfacing

**Aim:** Interface stepper motor to 8051 and write a program to rotate motor with different

Step angles and with different speeds.

**Objective:** 1.To study the interface of stepper motor with and 8051 IC and, write an ALP

2. To rotate the same in different steps and at different speed.

S/W & H/W USED: 1. Kiel IDE Compiler / Assembler

2. Flash magic Software for downloading (if required)

3. 8051 Trainer kit

4. Stepper motor interfacing kit.

**Theory:**

Normally the motor with which we deal in our daily life are DC motors. They rotates in a constant velocity and wehttp://www.techsavvy.net76.net/index_htm_files/0.gif can not determine the angle of rotation. But a stepper motor divides a full rotation into numbers of small stepshttp://www.techsavvy.net76.net/index_htm_files/0.gif and the position and the angle of the rotor can be controlled by the user. One advantage of stepper motor is thathttp://www.techsavvy.net76.net/index_htm_files/0.gif it has no feedback mechanism to control its position. Generally we use stepper motors where precise rotation ishttp://www.techsavvy.net76.net/index_htm_files/0.gif required.

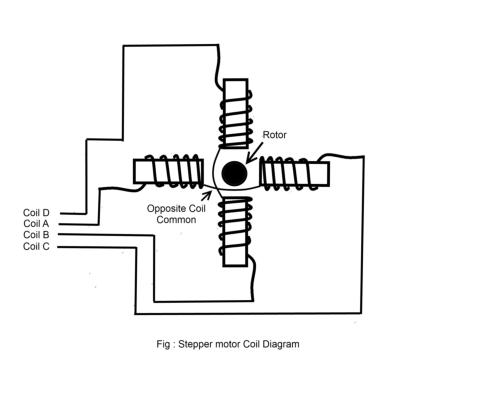
**Components Required** : 1.AT89C51 (8051 Microcontroller)http://www.techsavvy.net76.net/index_htm_files/0.gif

2. An Unipolar Stepper Motor http://www.techsavvy.net76.net/index_htm_files/0.gif

3.ULN2003 (Darlington Arrays)http://www.techsavvy.net76.net/index_htm_files/0.gif

**Description OF Stepper Motor :**

The unipolar stepper motors has four coils . The opposite two coils are connected with each other (Shorted) ashttp://www.techsavvy.net76.net/index_htm_files/0.gif you can see in the picture. You will notice while working with a stepper motor that it has six wires. Two of themhttp://www.techsavvy.net76.net/index_htm_files/0.gif are the common terminal as mentioned and they need to be connected to Vcc or GND . If you connect them tohttp://www.techsavvy.net76.net/index_htm_files/0.gif GND then a logic ‘1’ on the wire corresponding to a coil will activate it and if you connect the common terminal tohttp://www.techsavvy.net76.net/index_htm_files/0.gif Vcc then a logic ‘0’ will activate the corresponding coil.



**Fig : Stepper Motor  Internal Coil Connection**

As there are four coils so to operate the stepper motor correctly you should know which wire is for which coil. http://www.techsavvy.net76.net/index_htm_files/0.gif

Stepper motors consist of a permanent magnet rotating shaft, called the rotor, and electromagnets on the stationary portion that surrounds the motor, called the stator.  illustrates one complete rotation of a stepper motor. At position 1, we can see that the rotor is beginning at the upper electromagnet, which is currently active (has voltage applied to it). To move the rotor clockwise (CW), the upper electromagnet is deactivated and the right electromagnet is activated, causing the rotor to move 90 degrees CW, aligning itself with the active magnet. This process is repeated in the same manner at the south and west electromagnets until we once again reach the starting position.

It must include Motor related information.

a. Sequence related mapping

b. Clockwise rotation and anticlockwise rotation information

c. Step per second and steps per revolution equation, step angle calculation if any.

**Step Angle**

Step angle of the stepper motor is defined as the angle traversed by the motor in one step. To

Calculate step angle; simply divide 360 by number of steps a motor takes to complete one revolution. As we have seen that in half mode, the number of steps taken by the motor to complete one revolution gets doubled, so step angle reduces to half.

Step Angle ø = 360° / 4 = 90°

and in case of half mode step angle gets half so 45°.

So this way we can calculate step angle for any stepper motor. Usually step angle is given in the spec sheet of the stepper motor you are using. Knowing stepper motor's step angle helps you calibrate the rotation of motor also to helps you move the motor to correct angular position.

**Types of rotations in a stepper motor :** There are three types of rotations in a stepper motor. All of them have few advantages and disadvantages. http://www.techsavvy.net76.net/index_htm_files/0.gif

1. Wave Drive Stepping Mode.http://www.techsavvy.net76.net/index_htm_files/0.gif

2. Half Drive Stepping Mode.http://www.techsavvy.net76.net/index_htm_files/0.gif

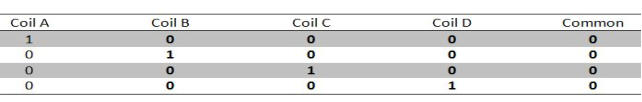
3. Full Drive Stepping Mode.http://www.techsavvy.net76.net/index_htm_files/0.gif

1. **Wave Drive Stepping Mode :**

•Only a single phase is activated at a time.

•Torque is very less.http://www.techsavvy.net76.net/index_htm_files/0.gif

The excitation table of Wave Drive Mode is given below.

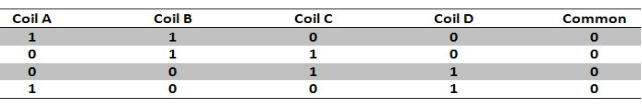


**Fig : Wave Drive Mode Coil Excitation Table**

1. **Full Drive Stepping Mode** :

* Two phases are always on.http://www.techsavvy.net76.net/index_htm_files/0.gif
* Full rated torque is available.http://www.techsavvy.net76.net/index_htm_files/0.gif

The excitation table of Full Drive Mode is given below .http://www.techsavvy.net76.net/index_htm_files/0.gif



**Fig: Full Drive Mode Coil Excitation Table**

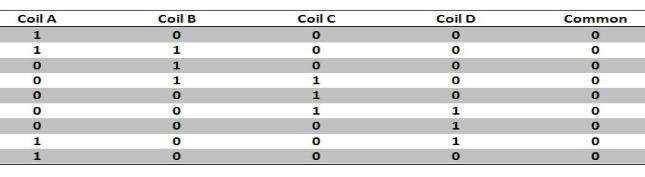
1. **Half Drive Stepping Mode:http://www.techsavvy.net76.net/index_htm_files/0.gif**

•The drive alternates between two phases on and a single phase on.http://www.techsavvy.net76.net/index_htm_files/0.gif

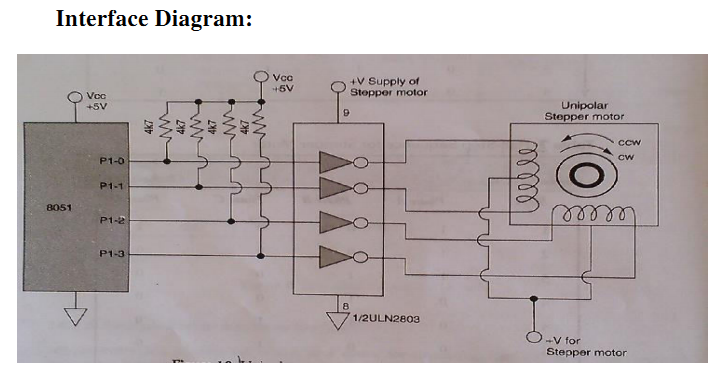
•Angular resolution increased.http://www.techsavvy.net76.net/index_htm_files/0.gif

•Generally 70% torque available.http://www.techsavvy.net76.net/index_htm_files/0.gif

The excitation table of Half Drive Mode is given below .http://www.techsavvy.net76.net/index_htm_files/0.gif



**Fig : Half Drive Mode Coil Excitation**

****

**Fig: Stepper Motor Interfacing Diagram**

**ALGORITHM:**

1. Initialize the 8051 and Stepper motor.

2. Calculate the machine coil sequence for different step

3. Load step in register A.

4. Load contents of A on P1 port.

5. Call delay.(As per decided or designed)

6. Go to step 3 .

7. Find output in terms of rotation of stepper motor.

8. For getting real movement picture add piece of paper on stepper motor.

**Conclusion:**

In this way, we can interface 8051 with Stepper motor kits and rotate motor in half stepping and full stepping mode in both directions.

**ENHANCEMENT/MODIFICATION:** Vary the step angle value and speed.