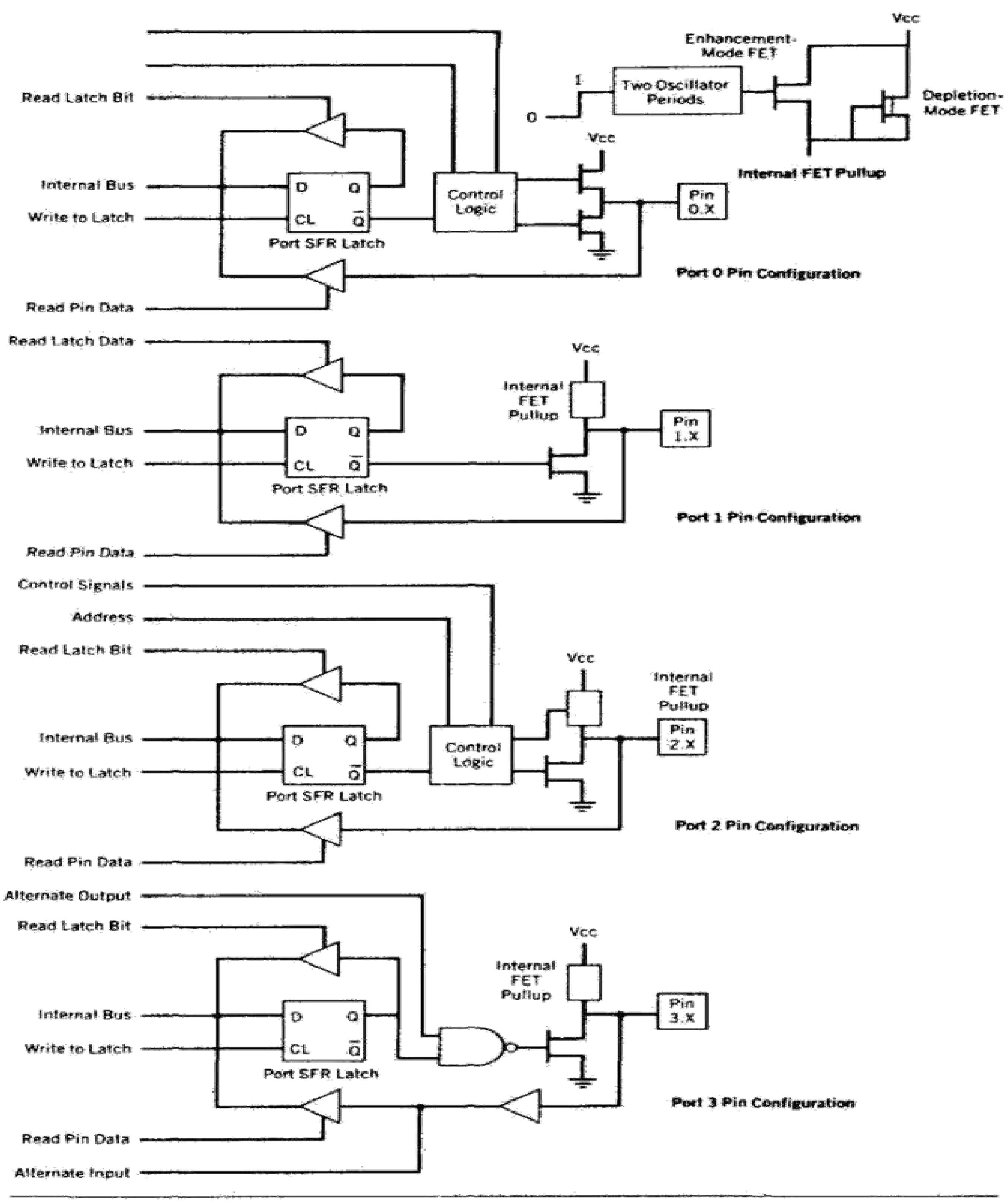
Port Pin Circuits



provide a logic low at the pin. After the address has been formed and latched into external circuits by the Address Latch Enable (ALE) pulse, the bus is turned around to become a data bus. Port 0 now reads data from the external memory and must be configured as an input, so a logic 1 is automatically written by internal control logic to all port 0 latches.

**Port l**

Port 1 pins have no dual functions. Therefore, the output latch is connected directly to the gate of the lower FET, which has an FET circuit labeled "Internal FET Pull up" as an active pull up load.

Used as an input, a 1 is written to the latch, turning the lower FET off; the pin and the input to the pin buffer are pulled high by the FET load. An external circuit can overcome the high impedance pull up and drive the pin low to input a 0 or leave the input high for a 1.

If used as an output, the latches containing a I can drive the input of an external circuit high through the pull up. If a 0 is written to the latch, the lower FET is on, the pull up is off, and the pin can drive the input of the external circuit low.

To aid in speeding up switching times when the pin is used as an output, the internal FET pull up has another FET in parallel with it. The second FET is turned on for two oscillator time periods during a low-to-high transition on the pin, as shown in Figure 2.7.

This arrangement provides a low impedance path to the positive voltage supply to help reduce rise times in charging any parasitic capacitances in the external circuitry.

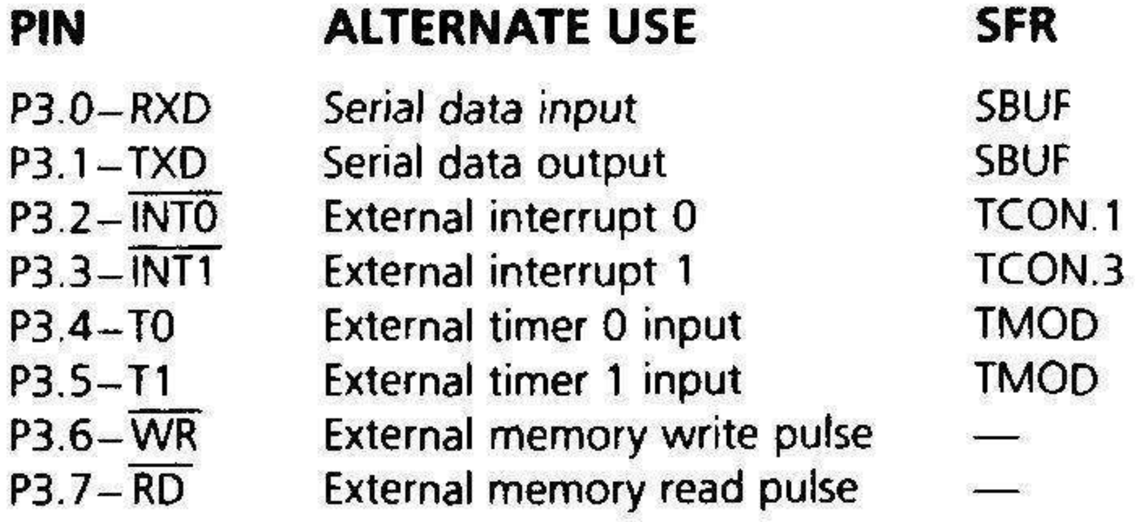
**Port 2**

Port 2 may be used as an input/output port similar in operation to port 1. The alternate use of port 2 is to supply a high-order address byte in conjunction with the port 0 low-order byte to address external memory.

Port 2 pins are momentarily changed by the address control signals when supplying the high byte of a 16-bit address. Port 2 latches remain stable when external memory is addressed, as they do not have to be turned around (set to 1) for data input as is the case for port 0.

**Port3**

Port 3 is an input/output port similar to port I. The input and output functions can be programmed under the control of the P3 latches or under the control of various other special function registers. The port 3 alternate uses are shown in the following table:-



| PIN | ALTERNATE USE | SFR |
|---|---|---|
| P3.0–RXD | Serial data input | SBUF |
| P3.1–TXD | Serial data output | SBUF |
| P3.2–$\overline{\text{INT0}}$ | External interrupt 0 | TCON.1 |
| P3.3–$\overline{\text{INT1}}$ | External interrupt 1 | TCON.3 |
| P3.4–T0 | External timer 0 input | TMOD |
| P3.5–T1 | External timer 1 input | TMOD |
| P3.6–$\overline{\text{WR}}$ | External memory write pulse | — |
| P3.7–$\overline{\text{RD}}$ | External memory read pulse | — |

Unlike ports 0 and 2, which can have external addressing functions and change all

eight port bits when in alternate use, each pin of port 3 may be individually programmed to be used either as I/O or as one of the alternate functions.

**External Memory**

The system designer is not limited by the amount of internal RAM and ROM available on chip. Two separate external memory spaces are made available by the 16-bit PC and DPTR and by different control pins for enabling external ROM and RAM chips. Internal control circuitry accesses the correct physical memory, depending upon the machine cycle state and the op code being executed.

There are several reasons for adding external memory, particularly program memory, when applying the 8051 in a system. When the project is in the prototype stage, the expense—in time and money—of having a masked internal ROM made for each program "try" is prohibitive.

To alleviate this problem, the manufacturers make available an EPROM version, the 8751, which has 4K of on-chip EPROM that may be programmed and erased as needed as the program is developed. The resulting circuit board layout will be identical to one that uses a factory-programmed 8051. The only drawbacks to the 8751 are the specialized EPROM programmers that must be used to program the non-standard 40-pin part, and the limit of "only" 4096 bytes of program code. The 8751 solution works well if the program will fit into 4K bytes. Unfortunately, many times, particularly if the program is written in a high-level language, the program size exceeds 4K bytes, and an external program memory is needed. Again, the manufacturers provide a version for the job, the ROMIess 8031. The EA pin is grounded when using the 8031, and all program code is contained in an external EPROM that may be as large as 64K bytes and that can be programmed using standard EPROM programmers.

External RAM, which is accessed by the DPTR, may also be needed when 128 bytes of internal data storage is not sufficient. External RAM, up to 64K bytes, may also be added to any chip in the 8051 family.

**Connecting External Memory**

Figure 2.8 shows the connections between an 8031 and an external memory configuration consisting of I6K bytes of EPROM and 8K bytes of static RAM. The 8051 accesses external RAM whenever certain program instructions are executed. External ROM is accessed whenever the EA (external access) pin is connected to ground or when the PC contains an address higher than the last address in the internal 4K bytes ROM (OFFFh). 8051 designs can thus use internal and external ROM automatically; the 8031, having no internal ROM, must have EA grounded.

Figure 2.9 shows the timing associated with an external memory access cycle. During any memory access cycle, port 0 is time multiplexed. That is, it first provides the lower byte of the 16-bit memory address, then acts as a bidirectional data bus to write or read a byte of memory data. Port 2 provides the high byte of the memory address during the entire memory read/write cycle.

The lower address byte from port 0 must be latched into an external register to save
the byte. Address byte save is accomplished by the ALE clock pulse that provides the correct timing for the '373 type data latch. The port 0 pins then become free to serve as a data bus.

If the memory access is for a byte of program code in the ROM, the PSEN (program store enable) pin will go low to enable the ROM to place a byte of program code on the data bus. If the access is for a RAM byte, the WR (write) or RD (read) pins will go low, enabling data to flow between the RAM and the data bus.

The ROM may be expanded to 64K by using a 27512 type EPROM and connecting the remaining port 2 upper address lines AI4-A15 to the chip.

At this time the largest static RAMs available are 32K in size; RAM can be expanded to 64K by using two 32K RAMs that are connected through address A14 of port 2. The

**FIGURE 2.8** External Memory Connections

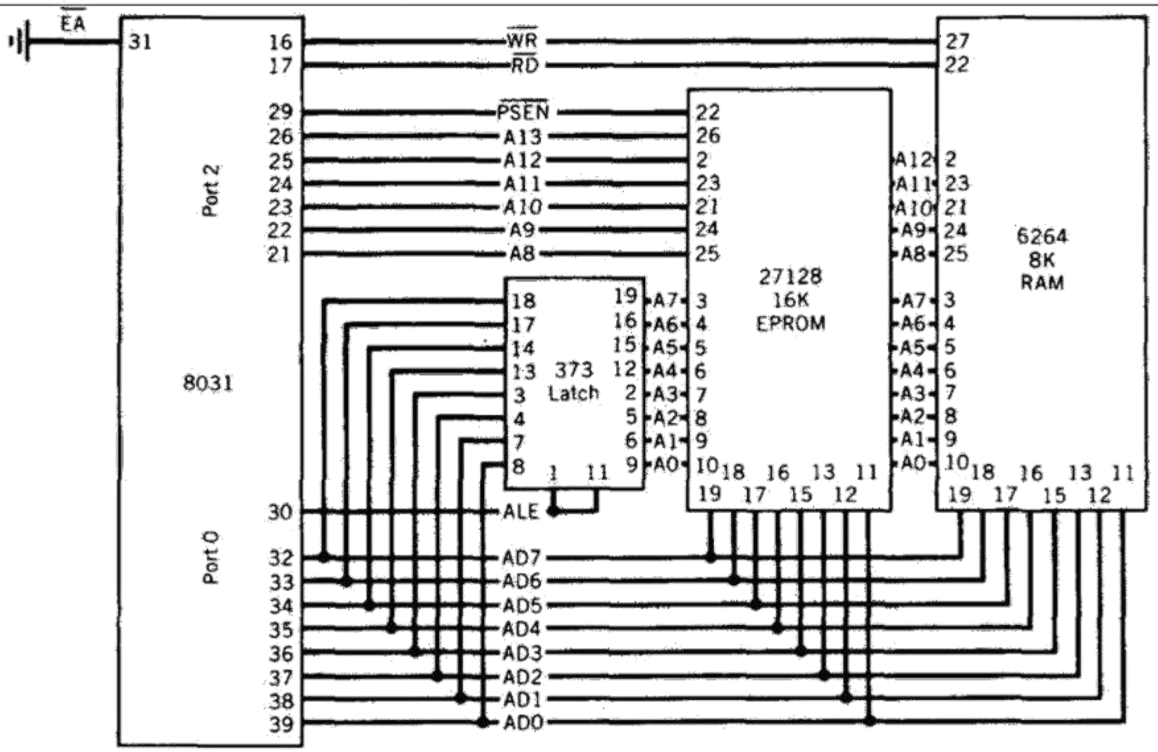
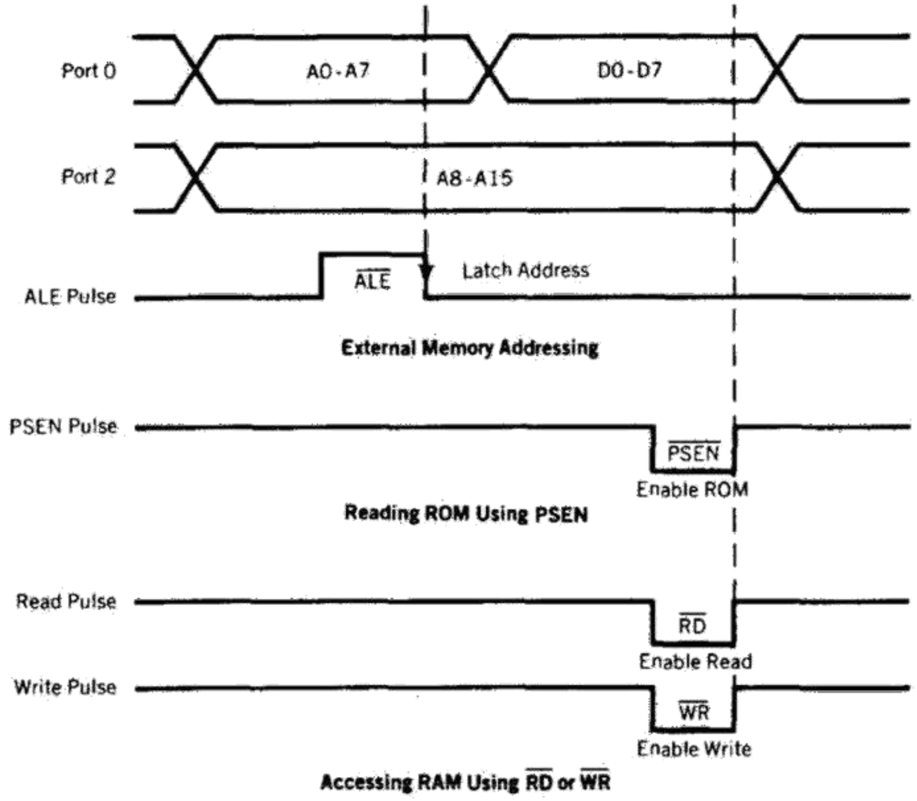


**FIGURE 2.9** External Memory Timing



**External Memory Addressing**

**Reading ROM Using PSEN**

**Accessing RAM Using RD or WR**

first 32K RAM (OOOOh-7FFFh) can then be enabled when AI5 of port 2 is low, and the second 32K RAM (SOOOh-FFFFh) when A15 is high, by using an inverter.

Note that the WR and RD signals are alternate uses for port 3 pins 16 and 17. Also,

port 0 is used for the lower address byte and data; port 2 is used for upper address bits. The use of external memory consumes many of the port pins, leaving only port 1 and parts of port 3 for general I/O.

# 8051 INSTRUCTION SET

**8051 has about 111 instructions. These can be grouped into the following categories**

- Arithmetic Instructions
- Logical Instructions
- Data Transfer instructions
- Boolean Variable Instructions
- Program Branching Instructions

**The following nomenclatures for register, data, address and variables are used while write instructions**

- A: Accumulator
- B: "B" register
- C: Carry bit
- Rn: Register R0 - R7 of the currently selected register bank
- Direct: 8-bit internal direct address for data. The data could be in lower 128bytes of RAM (00 - 7FH) or it could be in the special function register (80 - FFH).
- @Ri: 8-bit external or internal RAM address available in register R0 or R1. This is used for indirect addressing mode.
- #data8: Immediate 8-bit data available in the instruction.
- #data16: Immediate 16-bit data available in the instruction.
- Addr11: 11-bit destination address for short absolute jump. Used by instructions AJMP & ACALL. Jump range is 2 kbyte (one page).
- Addr16: 16-bit destination address for long call or long jump.
- Rel: 2's complement 8-bit offset (one - byte) used for short jump (SJMP) and all conditional jumps.
- bit: Directly addressed bit in internal RAM or SFR

**Some Simple Instructions:**

```
MOV dest,source               ; dest = source
        MOV A,#72H            ;A=72H
        MOV R4,#62H ;R4=62H
        MOV     B,0F9H                ;B=the content of F9'th byte of RAM
        MOV     DPTR,#7634H
        MOV     DPL,#34H
        MOV     DPH,#76H
        MOV     P1,A          ;mov A to port 1
```

Note 1:

```
        MOV     A,#72H         ≠ MOV A,72H
```

After instruction "MOV A,72H " the content of 72'th byte of RAM will replace in Accumulator.

Note 2:

```
        MOV     A,R3  ≡       MOV A,3
```

```
ADD     A, Source             ;A=A+SOURCE
        ADD     A,#6          ;A=A+6
        ADD     A,R6          ;A=A+R6


        ADD     A,6           ;A=A+[6] or A=A+R6
        ADD             A,0F3H        ;A=A+[0F3H]
SUBB    A, Source             ;A=A-SOURCE-C
        SUBB    A,#6          ;A=A-6
        SUBB    A,R6          ;A=A+R6
```

**MUl & Div:**

- MUL AB ;B|A = A*B

```
        MOV     A,#25H
        MOV     B,#65H
        MUL     AB            ;25H*65H=0E99
                                      ;B=0EH, A=99H
```

- DIV AB ;A = A/B, B = A mod B

```
MOV    A,#25
       MOV    B,#10
       DIV    AB              ;A=2, B=5
SETB   bit            ; bit=1
CLR    bit            ; bit=0
SETB   C              ; CY=1
SETB   P0.0           ;bit 0 from port 0 =1
SETB   P3.7           ;bit 7 from port 3 =1
SETB   ACC.2          ;bit 2 from ACCUMULATOR =1
SETB   05             ;set high D5 of RAM loc. 20h
```

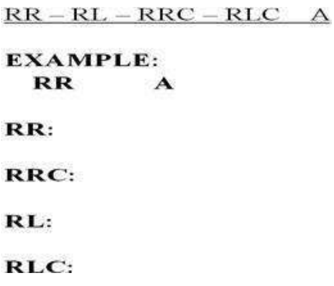
Note:

CLR instruction is as same as
SETB i.e.:

```
       CLR            C       ;CY=0
```

But following instruction is only for CLR:

```
       CLR            A       ;A=0
DEC  byte             ;byte=byte-1
INC           byte            ;byte=byte+1
INC           R7
DEC           A
DEC           40H             ; [40]=[40]-1
```



RR – RL – RRC – RLC A

EXAMPLE:
RR A

RR:

RRC:

RL:

RLC:

ANL - ORL – XRL

Bitwise Logical Operations:

AND, OR, XOR

EXAMPLE:

```
       MOV    R5,#89H
       ANL    R5,#08H
```