CHAPTER 4

AVR I/O PORT PROGRAMMING

OBJECTIVES

Upon completion of this chapter, you will be able to:

**>>	List all the ports of the AVR**
**>>	Describe the dual role of AVR pins**
**>>	Code Assembly language to use the ports for input or output**
**>>	Explain the dual role of Ports A, B, C, and D**
**>>	Code AVR instructions for I/O handling**
**>>	Code I/O bit-manipulation programs for the AVR**
**>>	Explain the bit-addressability of AVR ports**

This chapter describes I/O port programming of the AVR with many examples. In Section 4.1, we describe I/O access using byte-size data, and in Section 4.2, bit manipulation of the I/O ports is discussed in detail.

SECTION 4.1: I/O PORT PROGRAMMING IN AVR

In the AVR family, there are many ports for I/O operations, depending on which family member you choose. Examine Figure 4-1 for the ATmega32 40-pin chip. A total of 32 pins are set aside for the four ports PORTA, PORTB, PORTC, and PORTD. The rest of the pins are designated as VCC, GND, XTAL1, XTAL2, RESET, AREF, AGND, and AVCC. They are discussed in Chapter 8.

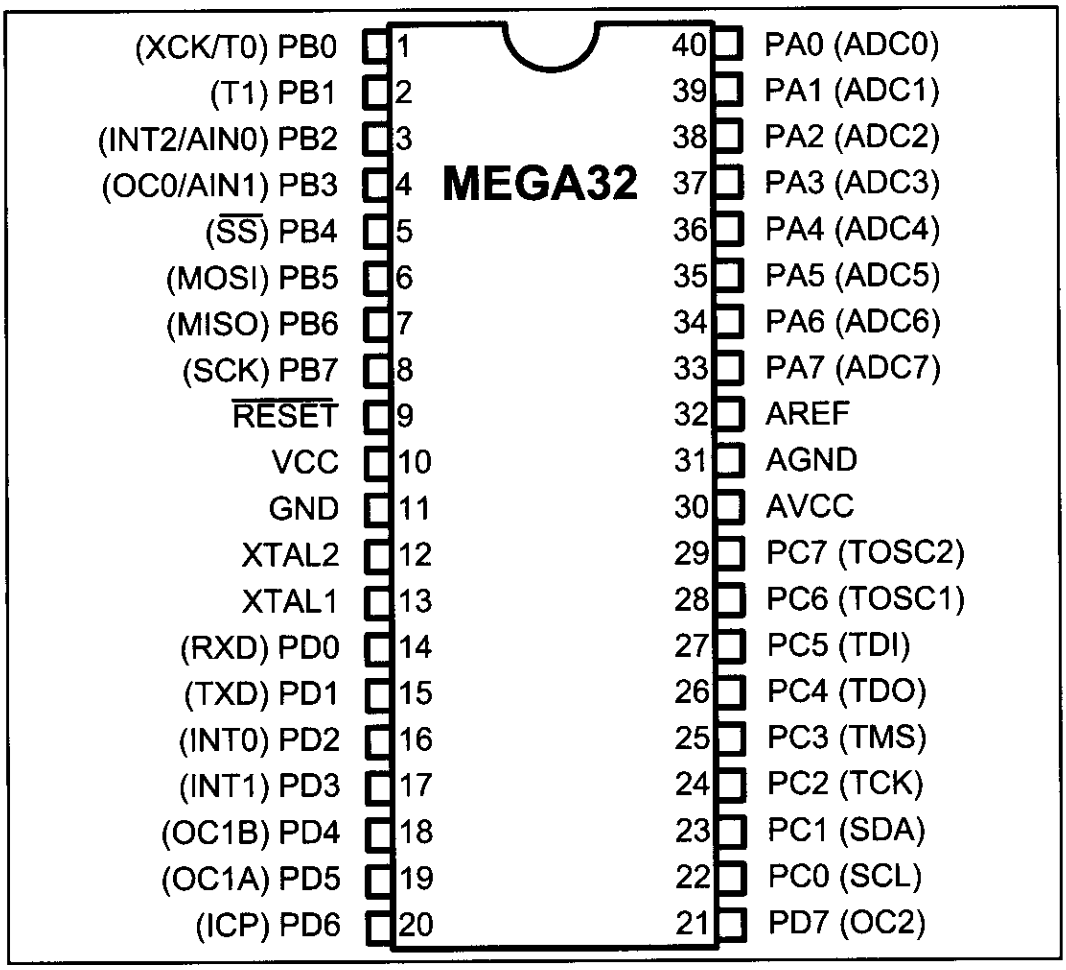


Figure 4-1. ATmega32 Pin Diagram

I/O port pins and their functions

The number of ports in the AVR family varies depending on the number of pins on the chip. The 8-pin AVR has port B only, while the 64-pin version has ports A through F, and the 100-pin AVR has ports A through L, as shown in Table 4-1. The 40-pin AVR has four ports. They are PORTA, PORTB, PORTC, and PORTD. To use any of these ports as an input or output port, it must be programmed, as we will explain throughout this section. In addition to being used for simple I/O, each

Table 4-1: Number of Ports in Some AVR Family Members

Pins	8-pin	28-pin	40-pin	64-pin	100-pin
Chip	**ATtiny25/45/85**	**ATmega8/48/88**	**ATmega32/16**	**ATmega64/128**	**ATmega1280**
Port A			X	X	X
Port B	6 bits	X	X	X	X
Port C		7 bits	X	X	X
Port D		X	X	X	X
Port E				X	X
Port F				X	X
Port G				5 bits	6 bits
Port H					X
Port J					X
Port K					X
Port L					X

Note: X indicates that the port is available.

port has some other functions such as ADC, timers, interrupts, and serial communication pins. Figure 4-1 shows alternate functions for the ATmega32 pins. We will study all these alternate functions in future chapters. In this chapter we focus on the simple I/O function of the AVR family. Not all ports have 8 pins. For example, in the ATmega8, Port C has 7 pins. Each port has three I/O registers associated with it, as shown in Table 4-2. They are designated as PORTx, DDRx, and PINx. For example, for Port B we have PORTB, DDRB, and PINB. Notice that DDR stands for Data Direction Register, and PIN stands for Port INput pins. Also notice that each of the I/O registers is 8 bits wide, and each port has a maximum of 8 pins; therefore each bit of the I/O registers affects one of the pins (see Figure 4-2; the content of bit 0 of DDRB represents the direction of the PB0 pin, and so on). Next, we describe how to access the I/O registers associated with the ports.

Table 4-2: Register Addresses for ATmega32 Ports

Port	Address	Usage
PORTA	$3B	output
DDRA	$3A	direction
PINA	$39	input
PORTB	$38	output
DDRB	$37	direction
PINB	$36	input
PORTC	$35	output
DDRC	$34	direction
PINC	$33	input
PORTD	$32	output
DDRD	$31	direction
PIND	$30	input

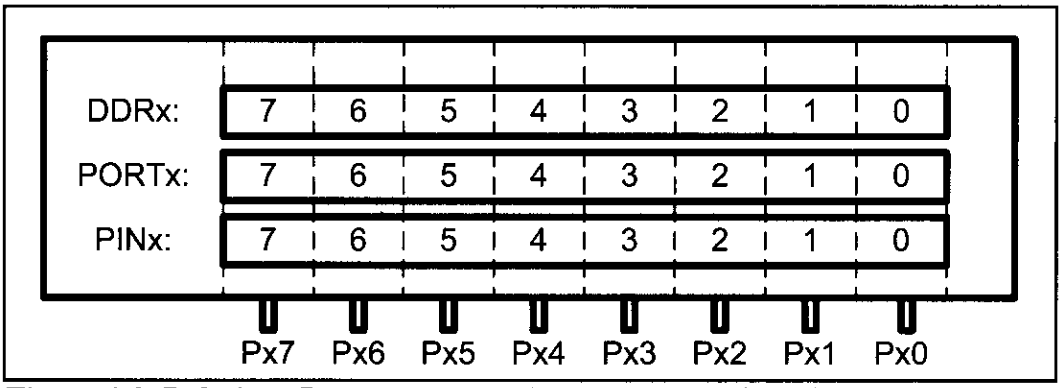


Figure 4-2. Relations Between the Registers and the Pins of AVR

DDRx register role in outputting data

Each of the ports A–D in the ATmega32 can be used for input or output. The DDRx I/O register is used solely for the purpose of making a given port an input or output port. For example, to make a port an output, we write 1s to the DDRx register. In other words, to output data to all of the pins of the Port B, we must first put 0b11111111 into the DDRB register to make all of the pins output.

The following code will toggle all 8 bits of Port B forever with some time delay between "on" and "off" states:

```
      LDI   R16,0xFF     ;R16 = 0xFF = 0b11111111
      OUT   DDRB,R16     ;make Port B an output port (1111 1111)
L1:   LDI   R16,0x55     ;R16 = 0x55 = 0b01010101
      OUT   PORTB,R16    ;put 0x55 on port B pins
      CALL  DELAY
      LDI   R16,0xAA     ;R16 = 0xAA = 0b10101010
      OUT   PORTB,R16    ;put 0xAA on port B pins
      CALL  DELAY
      RJMP  L1
```

It must be noted that unless we set the DDRx bits to one, the data will not go from the port register to the pins of the AVR. This means that if we remove the first two lines of the above code, the 0x55 and 0xAA values will not get to the pins. They will be sitting in the I/O register of Port B inside the CPU.

To see the role of the DDRx register in allowing the data to go from Portx to the pins, examine Figure 4-3. For more information about the internal circuitry of I/O ports, see Appendix C.

DDR register role in inputting data

To make a port an input port, we must first put 0s into the DDRx register for that port, and then bring in (read) the data present at the pins. As an aid for remembering that the port is input when the DDR bits are 0s, imagine a person who has 0 dollars. The person can only get money, not give it. Similarly, when DDR contains 0s, the port gets data.

Notice that upon reset, all ports have the value 0x00 in their DDR registers. This means that all ports are configured as input as we will see next.

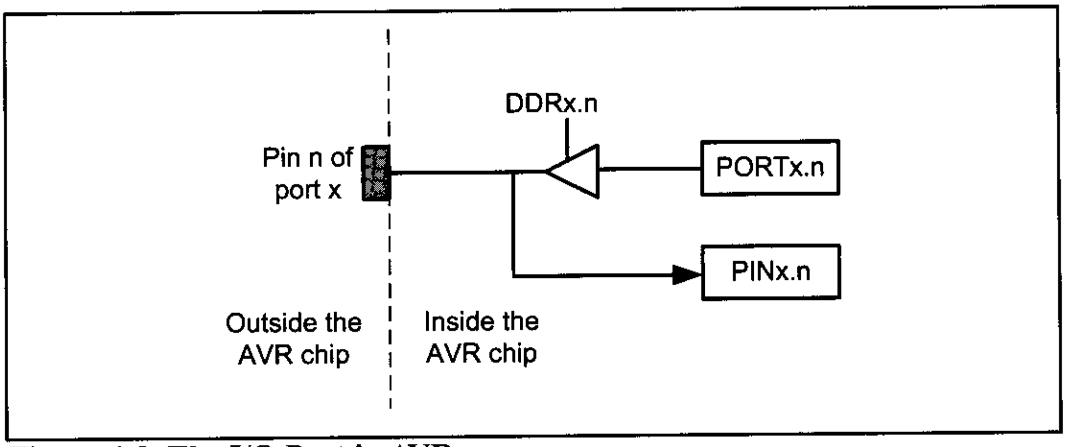


Figure 4-3. The I/O Port in AVR

PIN register role in inputting data

To read the data present at the pins, we should read the PIN register. It must be noted that to bring data into CPU from pins we read the contents of the PINx register, whereas to send data out to pins we use the PORTx register.

PORT register role in inputting data

There is a pull-up resistor for each of the AVR pins. If we put 1s into bits of the PORTx register, the pull-up resistors are activated. In cases in which nothing is connected to the pin or the connected devices have high impedance, the resistor pulls up the pin. See Figure 4-4.

If we put 0s into the bits of the PORTx register, the pull-up resistor is inactive.

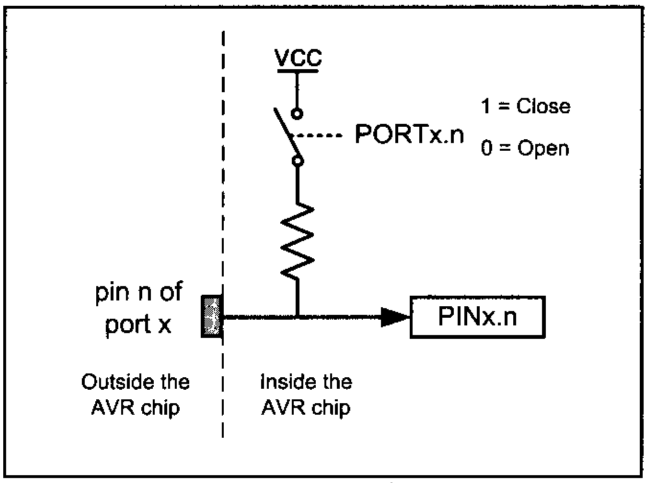


Figure 4-4. The Pull-up Resistor

The following code gets the data present at the pins of port C and sends it to port B indefinitely, after adding the value 5 to it:

```
.INCLUDE "M32DEF.INC"
      LDI   R16,0x00      ;R16 = 00000000 (binary)
      OUT   DDRC,R16      ;make Port C an input port
      LDI   R16,0xFF      ;R16 = 11111111 (binary)
      OUT   DDRB,R16      ;make Port B an output port(1 for Out)
L2:   IN    R16,PINC      ;read data from Port C and put in R16
      LDI   R17,5
      ADD   R16,R17       ;add 5 to it
      OUT   PORTB,R16     ;send it to Port B
      RJMP  L2            ;continue forever
```

If we want to make the pull-up resistors of port C active, we must put 1s into the PORTC register. The program becomes as follows:

```
.INCLUDE    "M32DEF.INC"
      LDI   R16,0xFF      ;R16 = 11111111 (binary)
      OUT   DDRB,R16      ;make Port B an output port
      OUT   PORTC,R16     ;make the pull-up resistors of C active
      LDI   R16,0x00      ;R16 = 00000000 (binary)
      OUT   DDRC,R16      ;Port C an input port (0 for I)
L2:   IN    R16,PINC      ;move data from Port C to R16
      LDI   R17,5
      ADD   R16,R17       ;add some value to it
      OUT   PORTB,R16     ;send it to Port B
      RJMP  L2            ;continue forever
```

Again, it must be noted that unless we clear the DDR bits (by putting 0s there), the data will not be brought into the registers from the pins of Port C. To see the role of the DDRx register in allowing the data to come into the CPU from the pins, examine Figure 4-3.

The pins of the AVR microcontrollers can be in four different states according to the values of PORTx and DDRx, as shown in Figure 4-5.

PORTx \ DDRx	0	1
0	Input & high impedance	Out 0
1	Input & pull-up	Out 1

Figure 4-5. Different States of a Pin in the AVR Microcontroller

This is one of powerful features of the AVR microcontroller, since most of the other microcontrollers' pins (e.g., 8051) have fewer states.

Port A

Port A occupies a total of 8 pins (PA0–PA7). To use the pins of Port A as input or output ports, each bit of the DDRA register must be set to the proper value. For example, the following code will continuously send out to Port A the alternating values of 0x55 and 0xAA:

```
;toggle all bits of PORTA
.INCLUDE     "M32DEF.INC"
      LDI    R16,0xFF    ;R16 = 11111111 (binary)
      OUT    DDRA,R16    ;make Port A an output port
L1:   LDI    R16,0x55    ;R16 = 0x55
      OUT    PORTA,R16   ;put 0x55 on Port A pins
      CALL   DELAY
      LDI    R16,0xAA    ;R16 = 0xAA
      OUT    PORTA,R16   ;put 0xAA on Port A pins
      CALL   DELAY
      RJMP   L1
```

It must be noted that 0x55 (01010101) when complemented becomes 0xAA (10101010).

Port A as input

In order to make all the bits of Port A an input, DDRA must be cleared by writing 0 to all the bits. In the following code, Port A is configured first as an input port by writing all 0s to register DDRA, and then data is received from Port A and saved in a RAM location:

```
.INCLUDE     "M32DEF.INC"
      .EQU   MYTEMP 0x100      ;save it here
      LDI    R16,0x00    ;R16 = 00000000 (binary)
      OUT    DDRA,R16    ;make Port A an input port (0 for In)
      NOP                ;synchronizer delay
      IN     R16,PINA    ;move from pins of Port A to R16
      STS    MYTEMP,R16  ;save it in MYTEMP
```

Synchronizer delay

The input circuit of the AVR has a delay of 1 clock cycle. In other words, the PIN register represents the data that was present at the pins one clock ago. In the above code, when the instruction "`IN R16,PINA`" is executed, the PINA register contains the data, which was present at the pins one clock before. That is why the NOP is put before the "`IN R16,PINA`" instruction. (If the NOP is omitted, the read data is the data of the pins when the port was output.)

For more information see Section C-2.

Port B

Port B occupies a total of 8 pins (PB0–PB7). To use the pins of Port B as input or output ports, each bit of the DDRB register must be set to the proper value.

For example, the following code will continuously send out the alternating values of 0x55 and 0xAA to Port B:

```
      ;toggle all bits of PORTB
.INCLUDE    "M32DEF.INC"
      LDI   R16,0xFF    ;R16 = 11111111 (binary)
      OUT   DDRB,R16    ;make Port B an output port (1 for Out)
L1:   LDI   R16,0x55    ;R16 = 0x55
      OUT   PORTB,R16   ;put 0x55 on Port B pins
      CALL  DELAY
      LDI   R16,0xAA    ;R16 = 0xAA
      OUT   PORTB,R16   ;put 0xAA on Port B pins
      CALL  DELAY
      RJMP  L1
```

Port B as input

In order to make all the bits of Port B an input, DDRB must be cleared by writing 0 to all the bits. In the following code, Port B is configured first as an input port by writing all 0s to register DDRB, and then data is received from Port B and saved in some RAM location:

```
.INCLUDE    "M32DEF.INC"
      .EQU  MYTEMP=0x100 ;save it here
      LDI   R16,0x00    ;R16 = 00000000 (binary)
      OUT   DDRB,R16    ;make Port B an input port (0 for In)
      NOP
      IN    R16,PINB    ;move from pins of Port B to R16
      STS   MYTEMP,R16  ;save it in MYTEMP
```

Dual role of Ports A and B

The AVR multiplexes an analog-to-digital converter through Port A to save I/O pins. The alternate functions of the pins for Port A are shown in Table 4-3. We will show how to use Port A's ADC in Chapter 13. Because many projects use an ADC, we usually do not use Port A for simple I/O functions.

The AVR multiplexes some other functions through Port B to save pins.

The alternate functions of the pins for Port B are shown in Table 4-4. We will show how to use the alternate functions of Port B in future chapters.

Table 4-3: Port A Alternate Functions

Bit	Function
PA0	ADC0
PA1	ADC1
PA2	ADC2
PA3	ADC3
PA4	ADC4
PA5	ADC5
PA6	ADC6
PA7	ADC7

Table 4-4: Port B Alternate Functions

Bit	Function
PB0	XCK/T0
PB1	T1
PB2	INT2/AIN0
PB3	OC0/AIN1
PB4	SS
PB5	MOSI
PB6	MISO
PB7	SCK

Port C

Port C occupies a total of 8 pins (PC0–PC7). To use the pins of Port C as input or output ports, each bit of the DDRC register must be set to the proper value. For example, the following code will continuously send out the alternating values of 0x55 and 0xAA to Port C:

```
      ;toggle all bits of PORTB
.INCLUDE     "M32DEF.INC"
      LDI    R16,0xFF     ;R16 = 11111111 (binary)
      OUT    DDRC,R16     ;make Port C an output port (1 for Out)
L1:   LDI    R16,0x55     ;R16 = 0x55
      OUT    PORTC,R16    ;put 0x55 on Port C pins
      CALL   DELAY
      LDI    R16,0xAA     ;R16 = 0xAA
      OUT    PORTC,R16    ;put 0xAA on Port C pins
      CALL   DELAY
      RJMP   L1
```

Port C as input

In order to make all the bits of Port C an input, DDRC must be cleared by writing 0 to all the bits. In the following code, Port C is configured first as an input port by writing all 0s to register DDRC, and then data is received from Port C and saved in a RAM location:

```
.INCLUDE     "M32DEF.INC"
      .EQU   MYTEMP 0x100       ;save it here
      LDI    R16,0x00     ;R16 = 00000000 (binary)
      OUT    DDRC,R16     ;make Port C an input port (0 for In)
      NOP
      IN     R16,PINC     ;move from pins of Port C to R16
      STS    MYTEMP,R16   ;save it in MYTEMP
```

Port D

Port D occupies a total of 8 pins (PD0–PD7). To use the pins of Port D as input or output ports, each bit of the DDRD register must be set to the proper value. For example, the following code will continuously send out to Port D the alternating values of 0x55 and 0xAA:

```
      ;toggle all bits of PORTB
.INCLUDE     "M32DEF.INC"
      LDI    R16,0xFF     ;R16 = 11111111 (binary)
      OUT    DDRD,R16     ;make Port D an output port (1 for Out)
L1:   LDI    R16,0x55     ;R16 = 0x55
      OUT    PORTD,R16    ;put 0x55 on Port D pins
      CALL   DELAY
      LDI    R16,0xAA     ;R16 = 0xAA
      OUT    PORTD,R16    ;put 0xAA on Port D pins
      CALL   DELAY
      RJMP   L1
```

Port D as input

In order to make all the bits of Port D an input, DDRD must be cleared by writing 0 to all the bits. In the following code, Port D is configured first as an input port by writing all 0s to register DDRD, and then data is received from Port D and saved in a RAM location:

```
.INCLUDE     "M32DEF.INC"
.EQU  MYTEMP 0x100        ;save it here

      LDI    R16,0x00     ;R16 = 00000000 (binary)
      OUT    DDRD,R16     ;make Port D an input port (0 for In)
      NOP
      IN     R16,PIND     ;move from pins of Port D to R16
      STS    MYTEMP,R16   ;save it in MYTEMP
```

Dual role of Ports C and D

The alternate functions of the pins for Port C are shown in Table 4-5. We will show how to use Port C's alternate functions in future chapters. The alternate functions of the pins for Port D are shown in Table 4-6. We will show how to use Port D's alternate functions in future chapters.

Table 4-5: Port C Alternate Functions

Bit	Function
PC0	SCL
PC1	SDA
PC2	TCK
PC3	TMS
PC4	TDO
PC5	TDI
PC6	TOSC1
PC7	TOSC2

Table 4-6: Port D Alternate Functions

Bit	Function
PD0	PSP0/C1IN+
PD1	PSP1/C1IN-
PD2	PSP2/C2IN+
PD3	PSP3/C2IN-
PD4	PSP4/ECCP1/P1A
PD5	PSP5/P1B
PD6	PSP6/P1C
PD7	PSP7/P1D

Example 4-1

Write a test program for the AVR chip to toggle all the bits of PORTB, PORTC, and PORTD every 1/4 of a second. Assume a crystal frequency of 1 MHz.

Solution:

```
;tested with AVR Studio for the ATmega32 and XTAL = 1 MHz
;to select the XTAL frequency in AVR Studio, press ALT+O
.INCLUDE "M32DEF.INC"
      LDI   R16, HIGH(RAMEND)
      OUT   SPH, R16
      LDI   R16, LOW(RAMEND)
      OUT   SPL, R16    ;initialize stack pointer

      LDI   R16, 0xFF
      OUT   DDRB, R16   ;make Port B an output port
      OUT   DDRC, R16   ;make Port C an output port
      OUT   DDRD, R16   ;make Port D an output port

      LDI   R16, 0x55   ;R16 = 0x55
L3:   OUT   PORTB, R16  ;put 0x55 on Port B pins
      OUT   PORTC, R16  ;put 0x55 on Port C pins
      OUT   PORTD, R16  ;put 0x55 on Port D pins
      CALL  QDELAY      ;quarter of a second delay
      COM   R16         ;complement R16
      RJMP  L3

;---------------1/4 SECOND DELAY
QDELAY:
      LDI   R21, 200
D1:   LDI   R22, 250
D2:   NOP
      NOP
      DEC   R22
      BRNE  D2
      DEC   R21
      BRNE  D1
      RET
```

Calculations:

1 / 1 MHz = 1 μs
Delay = 200 × 250 × 5 MC × 1 μs = 250,000 μs (If we include the overhead, we will have 250,608 μs. See Example 3-18 in the previous chapter.)

Use the AVR Studio simulator to verify the delay size.

Review Questions

1. There are a total of _____ ports in the ATmega32.
2. True or false. All of the ATmega32 ports have 8 pins.
3. True or false. Upon power-up, the I/O pins are configured as output ports.
4. Code a simple program to send 0x99 to Port B and Port C.
5. To make Port B an output port, we must place _______ in register _______.
6. To make Port B an input port, we must place _______ in register _______.
7. True or false. We use a PORTx register to send data out to AVR pins.
8. True or false. We use PINx to bring data into the CPU from AVR pins.

SECTION 4.2: I/O BIT MANIPULATION PROGRAMMING

In this section we further examine the AVR I/O instructions. We pay special attention to I/O bit manipulation because it is a powerful and widely used feature of the AVR family.

I/O ports and bit-addressability

Sometimes we need to access only 1 or 2 bits of the port instead of the entire 8 bits. A powerful feature of AVR I/O ports is their capability to access individual bits of the port without altering the rest of the bits in that port. For all AVR ports, we can access either all 8 bits or any single bit without altering the rest. Table 4-7 lists the single-bit instructions for the AVR. Although the instructions in Table 4-7 can be used for any of the lower 32 I/O registers, I/O port operations use them most often. We will see the use of these instructions throughout future chapters. Table 4-8 shows the lower 32 I/O registers.

Next we describe all these instructions and examine their usage.

Table 4-7: Single-Bit (Bit-Oriented) Instructions for AVR

Instruction		Function
SBI	ioReg,bit	Set Bit in I/O register (set the bit: bit = 1)
CBI	ioReg,bit	Clear Bit in I/O register (clear the bit: bit = 0)
SBIC	ioReg,bit	Skip if Bit in I/O register Cleared (skip next instruction if bit = 0)
SBIS	ioReg,bit	Skip if Bit in I/O register Set (skip next instruction if bit = 1)

Address		Name
Mem.	I/O	
$20	$00	TWBR
$21	$01	TWSR
$22	$02	TWAR
$23	$03	TWDR
$24	$04	ADCL
$25	$05	ADCH
$26	$06	ADCSRA
$27	$07	ADMUX
$28	$08	ACSR
$29	$09	UBRRL
$2A	$0A	UCSRB
$2B	$0B	UCSRA
$2C	$0C	UDR
$2D	$0D	SPCR
$2E	$0E	SPSR
$2F	$0F	SPDR
$30	$10	PIND
$31	$11	DDRD
$32	$12	PORTD
$33	$13	PINC
$34	$14	DDRC
$35	$15	PORTC
$36	$16	PINB
$37	$17	DDRB
$38	$18	PORTB
$39	$19	PINA
$3A	$1A	DDRA
$3B	$1B	PORTA
$3C	$1C	EECR
$3D	$1D	EEDR
$3E	$1E	EEARL
$3F	$1F	EEARH

Table 4-8: The Lower 32 I/O Registers

SBI (set bit in I/O register)

To set HIGH a single bit of a given I/O register, we use the following syntax:

```
SBI ioReg, bit_num
```

where ioReg can be the lower 32 I/O registers (addresses 0 to 31) and bit_num is the desired bit number from 0 to 7. In Table 4-8 you see the list of the lower 32 I/O registers. Although the bit-oriented instructions can be used for manipulation of bits D0–D7 of the lower 32 I/O registers, they are mostly used for I/O ports. For example the following instruction sets HIGH bit 5 of Port B:

```
SBI PORTB, 5
```

In Figure 4-6, you see the SBI instruction format.

SBI a, b

1001	1010	aaaa	abbb

$0 \leq a \leq 31$

$0 \leq b \leq 7$

Figure 4-6. SBI (Set Bit) Instruction Format

CBI (Clear Bit in I/O register)

To clear a single bit of a given I/O register, we use the following syntax:

```
CBI ioReg, bit_number
```

For example, the following code toggles pin PB2 continuously:

```
      SBI   DDRB, 2          ;bit = 1, make PB2 an output pin
AGAIN:SBI   PORTB, 2         ;bit set (PB2 = high)
      CALL  DELAY
      CBI   PORTB, 2         ;bit clear (PB2 = low)
      CALL  DELAY
      RJMP  AGAIN
```

Remember that for I/O ports, we must set the appropriate bit in the DDRx register if we want the pin to be output.

Notice that PB2 is the third bit of Port B (the first bit is PB0, the second bit is PB1, etc.). This is shown in Table 4-9. See Example 4-2 for an example of bit manipulation of I/O bits.

CBI a, b

1001	1000	aaaa	abbb

$0 \leq a \leq 31$

$0 \leq b \leq 7$

Figure 4-7. CBI (Clear Bit) Instruction Format

Table 4-9: Single-Bit Addressability of Ports for ATmega32/16

PORT	PORTB	PORTC	PORTD	Port Bit
PA0	PB0	PC0	PD0	D0
PA1	PB1	PC1	PD1	D1
PA2	PB2	PC2	PD2	D2
PA3	PB3	PC3	PD3	D3
PA4	PB4	PC4	PD4	D4
PA5	PB5	PC5	PD5	D5
PA6	PB6	PC6	PD6	D6
PA7	PB7	PC7	PD7	D7

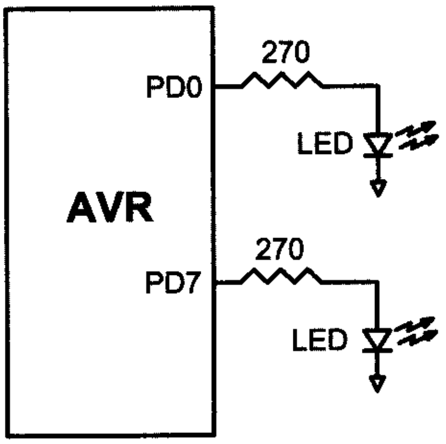
Notice in Example 4-2 that unused portions of Port C are undisturbed. This single-bit addressability of I/O ports is one of the most powerful features of the AVR microcontroller.

Example 4-2

An LED is connected to each pin of Port D. Write a program to turn on each LED from pin D0 to pin D7. Call a delay subroutine before turning on the next LED.

Solution:

```
.INCLUDE "M32DEF.INC"
LDI    R20, HIGH(RAMEND)
OUT    SPH, R20
LDI    R20, LOW(RAMEND)
OUT    SPL, R20     ;initialize stack pointer
LDI    R20, 0xFF
OUT    PORTD, R20   ;make PORTD an output port
SBI    PORTD,0      ;set bit PD0
CALL   DELAY        ;delay before next one
SBI    PORTD,1      ;turn on PD1
CALL   DELAY        ;delay before next one
SBI    PORTD,2      ;turn on PD2
CALL   DELAY
SBI    PORTD,3
CALL   DELAY
SBI    PORTD,4
CALL   DELAY
SBI    PORTD,5
CALL   DELAY
SBI    PORTD,6
CALL   DELAY
SBI    PORTD,7
CALL   DELAY
```



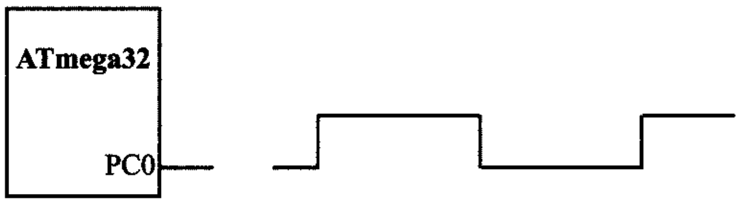
Example 4-3

Write the following programs:
(a) Create a square wave of 50% duty cycle on bit 0 of Port C.
(b) Create a square wave of 66% duty cycle on bit 3 of Port C.

Solution:

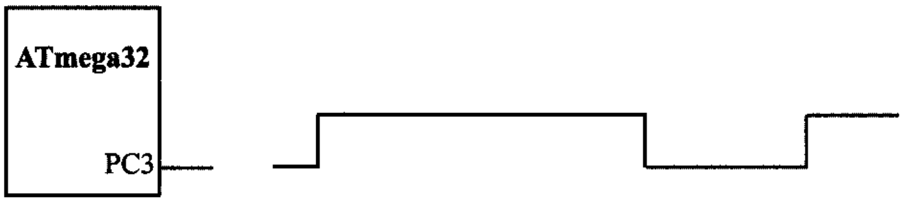
(a) The 50% duty cycle means that the "on" and "off" states (or the high and low portions of the pulse) have the same length. Therefore, we toggle PC0 with a time delay between each state.

```
.INCLUDE "M32DEF.INC"

      LDI   R20, HIGH(RAMEND)
      OUT   SPH, R20
      LDI   R20, LOW(RAMEND)
      OUT   SPL, R20    ;initialize stack pointer

      SBI   DDRC, 0     ;set bit 0 of DDRC (PC0 = out)
HERE: SBI   PORTC, 0    ;set to HIGH PC0 (PC0 = 1)
      CALL  DELAY       ;call the delay subroutine
      CBI   PORTC, 0    ;PC0 = 0
      CALL  DELAY
      RJMP  HERE        ;keep doing it
```



(b) A 66% duty cycle means that the "on" state is twice the "off" state.

```
      ....
      SBI   DDRC, 3     ;set bit 3 of DDRC (PC3 = out)
HERE: SBI   PORTC, 3    ;set to HIGH PC3 (PC3 = 1)
      CALL  DELAY       ;call the delay subroutine
      CALL  DELAY       ;call the delay subroutine
      CBI   PORTC, 3    ;PC3 = 0
      CALL  DELAY
      RJMP  HERE        ;keep doing it
```



Checking an input pin

To make decisions based on the status of a given bit in the file register, we use the SBIC (Skip if Bit in I/O register Cleared) and SBIS (Skip if Bit in I/O register Set) instructions. These single-bit instructions are widely used for I/O operations. They allow you to monitor a single pin and make a decision depending on whether it is 0 or 1. Again it must be noted that the SBIC and SBIS instructions can be used for any bits of the lower 32 I/O registers, including the I/O ports A, B, C, D, and so on.

SBIS (Skip if Bit in I/O register Set)

To monitor the status of a single bit for HIGH, we use the SBIS instruction. This instruction tests the bit and skips the next instruction if it is HIGH. See Figure 4-8. Example 4-4 shows how it is used.

SBIS a,b

1001	1011	aaaa	abbb

$0 \leq a \leq 31$

$0 \leq b \leq 7$

Figure 4-8. SBIS (Skip If Bit in I/O Register Set) Instruction Format

Example 4-4

Write a program to perform the following:
(a) Keep monitoring the PB2 bit until it becomes HIGH;
(b) When PB2 becomes HIGH, write the value $45 to Port C, and also send a HIGH-to-LOW pulse to PD3.

Solution:

```
.INCLUDE "M32DEF.INC"

            CBI   DDRB, 2      ;make PB2 an input
            LDI   R16, 0xFF
            OUT   DDRC, R16    ;make Port C an output port
            SBI   DDRD, 3      ;make PD3 an output
AGAIN:      SBIS  PINB, 2      ;skip if Bit PB2 is HIGH
            RJMP  AGAIN        ;keep checking if LOW
            LDI   R16, 0x45
            OUT   PORTC, R16   ;write 0x45 to port C
            SBI   PORTD, 3     ;set bit PD3 (H-to-L)
            CBI   PORTD, 3     ;clear bit PD3
HERE: RJMP  HERE
```

In this program, "`SBIS PINB,2`" instruction stays in the loop as long as PB2 is LOW. When PB2 becomes HIGH, it skips the branch instruction to get out of the loop, and writes the value $45 to Port C. It also sends a HIGH-to-LOW pulse to PD3.

SBIC (Skip if Bit in I/O register Cleared)

To monitor the status of a single bit for LOW, we use the SBIC instruction. This instruction tests the bit and skips the instruction right below it if the bit is LOW. See Figure 4-9. Example 4-5 shows how it is used.

```
SBIC  a,b          | 1001 | 1001 | aaaa | abbb |

                        0  ≤ a  ≤  31
                        0  ≤ b  ≤  7
```

Figure 4-9. SBIC (Skip if Bit in I/O Register Cleared) Instruction Format

Monitoring a single bit

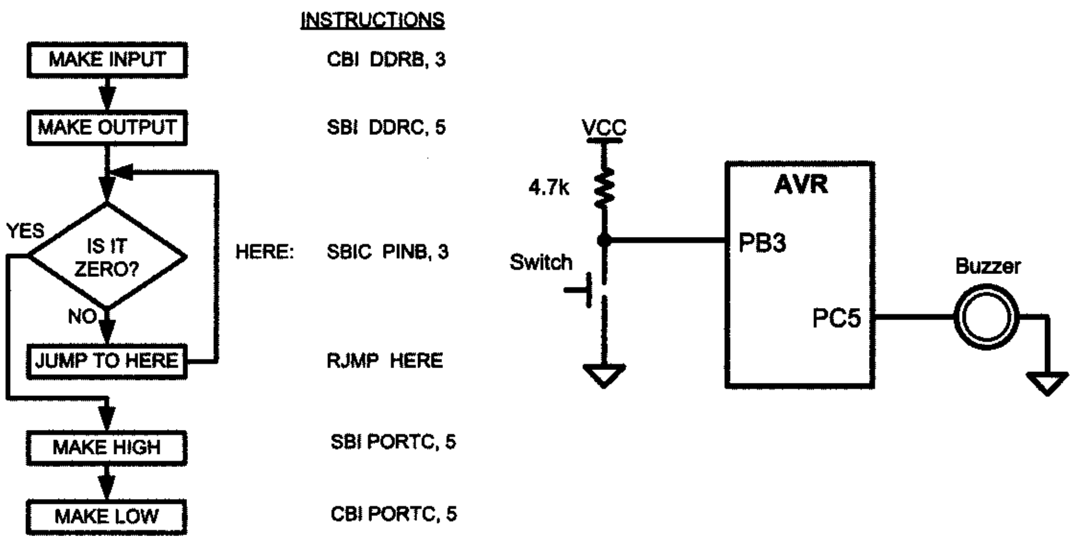
We can also use the bit test instructions to monitor the status of a single bit and make a decision to perform an action. See Examples 4-6 and 4-7.

Example 4-5

Assume that bit PB3 is an input and represents the condition of a door alarm. If it goes LOW, it means that the door is open. Monitor the bit continuously. Whenever it goes LOW, send a HIGH-to-LOW pulse to port PC5 to turn on a buzzer.

Solution:

```
.INCLUDE "M32DEF.INC"

      CBI   DDRB, 3     ;make PB3 an input
      SBI   DDRC, 5     ;make PC5 an output
HERE: SBIC  PINB, 3     ;keep monitoring PB3 for HIGH
      RJMP  HERE        ;stay in the loop
      SBI   PORTC,5     ;make PC5 HIGH
      CBI   PORTC,5     ;make PC5 LOW for H-to-L
      RJMP  HERE
```

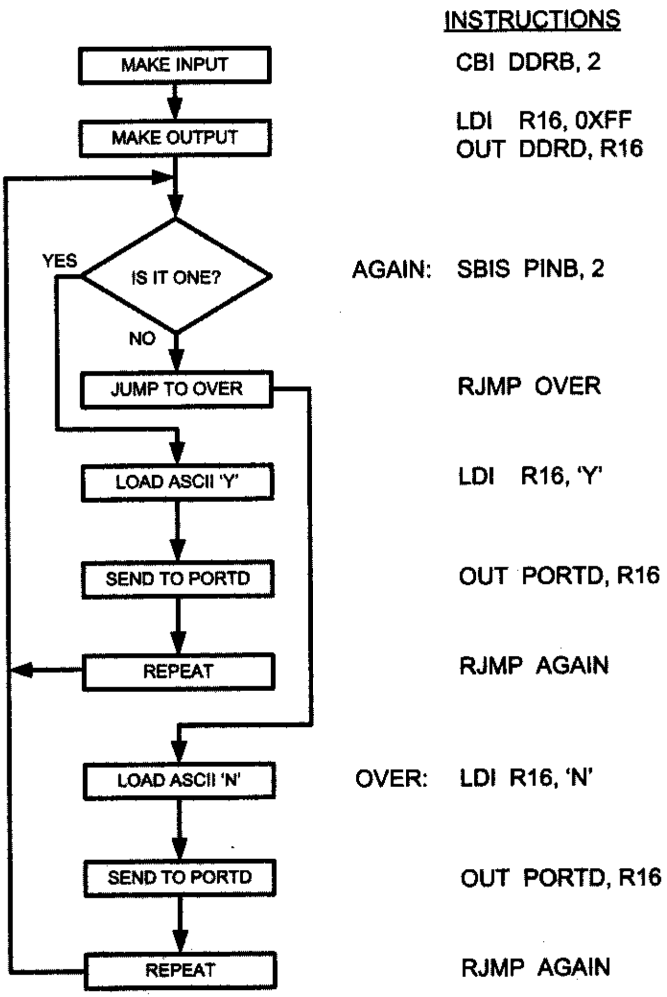


Example 4-6

A switch is connected to pin PB2. Write a program to check the status of SW and perform the following:
(a) If SW = 0, send the letter 'N' to PORTD.
(b) If SW = 1, send the letter 'Y' to PORTD.

Solution:

```
      .INCLUDE "M32DEF.INC"

      CBI   DDRB, 2      ;make PB2 an input
      LDI   R16, 0xFF
      OUT   DDRD, R16    ;make PORTD an output port

AGAIN:SBIS  PINB, 2      ;skip next if PB bit is HIGH
      RJMP  OVER         ;SW is LOW
      LDI   R16, 'Y'     ;R16 = 'Y' (ASCII letter Y)
      OUT   PORTD, R16   ;PORTD = 'Y'
      RJMP  AGAIN
OVER: LDI   R16, 'N'     ;R16 = 'N' (ASCII letter Y)
      OUT   PORTD, R16   ;PORTD = 'N'
      RJMP  AGAIN
```



INSTRUCTIONS

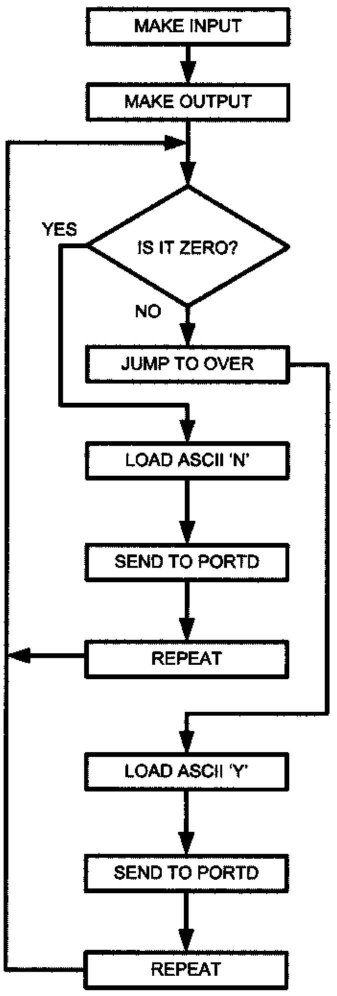
```
        CBI  DDRB, 2

        LDI  R16, 0XFF
        OUT  DDRD, R16

AGAIN:  SBIS PINB, 2

        RJMP OVER

        LDI  R16, 'Y'

        OUT  PORTD, R16

        RJMP AGAIN

OVER:   LDI  R16, 'N'

        OUT  PORTD, R16

        RJMP AGAIN
```

Example 4-7

Rewrite the program of Example 4-6, using the SBIC instruction instead of SBIS.

Solution:

```
.INCLUDE "M32DEF.INC"

       CBI    DDRB, 2      ;make PB2 an input
       LDI    R16, 0xFF
       OUT    DDRD, R16    ;make PORTD an output port

AGAIN:SBIC    PINB, 2      ;skip next if PB bit is LOW
       RJMP   OVER         ;SW is HIGH
       LDI    R16, 'N'     ;R16 = 'N' (ASCII letter N)
       OUT    PORTD, R16   ;PORTD = 'N'
       RJMP   AGAIN

OVER:  LDI    R16, 'Y'     ;R16 = 'Y' (ASCII letter Y)
       OUT    PORTD, R16   ;PORTD = 'Y'
       RJMP   AGAIN
```



INSTRUCTIONS

```
          CBI     DDRB, 2

          LDI     R16, 0XFF
          OUT     DDRD, R16

AGAIN:    SBIC    PINB, 2

          RJMP    OVER

          LDI     R16, 'N'

          OUT     PORTD, R16

          RJMP  AGAIN

OVER:     LDI     R16, 'Y'

          OUT     PORTD, R16

          RJMP    AGAIN
```

Reading a single bit

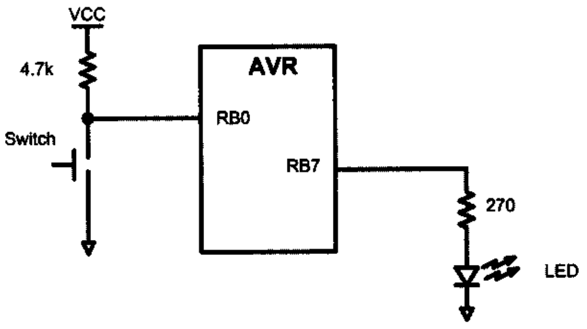
We can also use the bit test instructions to read the status of a single bit and send it to another bit or save it. This is shown in Examples 4-8 and 4-9.

Example 4-8

A switch is connected to pin PB0 and an LED to pin PB7. Write a program to get the status of SW and send it to the LED.

Solution:

```
.INCLUDE "M32DEF.INC"
      CBI    DDRB, 0      ;make PB0 an input
      SBI    DDRB, 7      ;make PB7 an output
AGAIN:SBIC   PINB, 0      ;skip next if PB0 is clear
      RJMP   OVER         ;(JMP is OK too)
      CBI    PORTB, 7
      RJMP   AGAIN        ;we can use JMP too
OVER: SBI    PORTB, 7
      RJMP   AGAIN        ;we can use JMP too
```

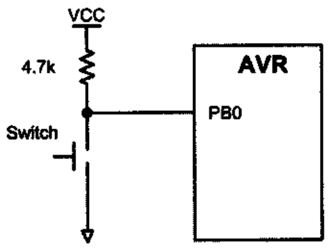


Example 4-9

A switch is connected to pin PB0. Write a program to get the status of SW and save it in location 0x200.

Solution:

```
.EQU MYTEMP = 0x200     ;set aside location 0x200
.INCLUDE "M32DEF.INC"
      CBI    DDRB, 0      ;make PB0 an input
AGAIN:SBIC   PINB, 0      ;skip next if PB0 is clear
      RJMP   OVER         ;(JMP is OK too)
      LDI    R16, 0
      STS    MYTEMP,R16   ;save it in MYTEMP
      RJMP   AGAIN        ;we can use JMP too
OVER: LDI    R16,0x1      ;move 1 to R16
      STS    MYTEMP,R16   ;save it in MYTEMP
      RJMP   AGAIN        ;we can use JMP too
```



Review Questions

1. True or false. The instruction "`SBI PORTB, 1`" makes pin PB1 HIGH while leaving other pins of PORTB unchanged, if bit 1 of the DDR bits is configured for output.
2. Show one way to toggle the pin PB7 continuously using AVR instructions.
3. Write instructions to get the status of PB2 and put it on PB0.
4. Write instructions to toggle both bits of PD7 and PD0 continuously.
5. According to Figure 4-7, what does the machine instruction $9819 do?

SUMMARY

This chapter focused on the I/O ports of the AVR. The four ports of the ATmega32, PORTA, PORTB, PORTC, and PORTD, were explored. These ports can be used for input or output. All the ports have alternate functions. The three registers associated with each port are PORTx, DDRx, and PINx. Their role in I/O manipulation was examined. Then, I/O instructions of the AVR were explained, and numerous examples were given. We also showed the bit-addressability of AVR ports.

CAUTION

We strongly recommend that you study Section C.2 (Appendix C) before connecting any external hardware to your AVR system. Failure to use the right instruction or the right connection to port pins can damage the ports of your AVR chip.

PROBLEMS

SECTION 4.1: I/O PORT PROGRAMMING IN AVR

1. The ATmega32 has a DIP package of ____ pins.
2. In ATmega32, how many pins are assigned to V_{CC} and GND?
3. In the ATmega32, how many pins are designated as I/O port pins?
4. How many pins are designated as PORTA in the 40-pin DIP package and what are their numbers?
5. How many pins are designated as PORTB in the 40-pin DIP package and what are their numbers?
6. How many pins are designated as PORTC in the 40-pin DIP package and what are their numbers?
7. How many pins are designated as PORTD in the 40-pin DIP package and what are their numbers?
8. Upon reset, all the bits of ports are configured as _____ (input, output).
9. Explain the role of DDRx and PORTx in I/O operations.

10. Write a program to get 8-bit data from PORTC and send it to PORTB and PORTD.
11. Write a program to get 8-bit data from PORTD and send it to PORTB and PORTC.
12. Which pins are for RxD and TxD?
13. Give data memory location assigned to DDR registers of Ports A–C for the ATmega32.
14. Write a program to toggle all the bits of PORTB and PORTC continuously (a) using 0xAA and 0x55 (b) using the COM instruction.

SECTION 4.2: I/O BIT MANIPULATION PROGRAMMING

15. Which ports of the ATmega32 are bit-addressable?
16. What is the advantage of bit-addressability for AVR ports?
17. Is the instruction "`COM PORTB`" a valid instruction?
18. Write a program to toggle PB2 and PB5 continuously without disturbing the rest of the bits.
19. Write a program to toggle PD3, PD7, and PC5 continuously without disturbing the rest of the bits.
20. Write a program to monitor bit PC3. When it is HIGH, send 0x55 to PORTD.
21. Write a program to monitor the PB7 bit. When it is LOW, send $55 and $AA to PORTC continuously.
22. Write a program to monitor the PA0 bit. When it is HIGH, send $99 to PORTB. If it is LOW, send $66 to PORTB.
23. Write a program to monitor the PB5 bit. When it is HIGH, make a LOW-to-HIGH-to-LOW pulse on PB3.
24. Write a program to get the status of PC3 and put it on PC4.
25. Create a flowchart and write a program to get the statuses of PD6 and PD7 and put them on PC0 and PC7, respectively.
26. Write a program to monitor the PB5 and PB6 bits. When both of them are HIGH, send $AA to PORTC; otherwise, send $55 to PORTC.
27. Write a program to monitor the PB5 and PB6 bits. When either of them is HIGH, send $AA to PORTC; otherwise, send $55 to PORTC.
28. Referring to Figure 4-8 and Table 4-8, write the machine equivalent of "SBIS PINB,3".
29. Referring to Figure 4-6 and Table 4-8, write the machine equivalent of the "SBI PORTA,2" instruction.

ANSWERS TO REVIEW QUESTIONS

SECTION 4.1: I/O PORT PROGRAMMING IN AVR

1. 4
2. True
3. False
4.
```
LDI R16,0xFF
OUT DDRB,R16
OUT DDRC,R16
LDI R16,0x99
OUT PORTB,R16
OUT PORTC,R16
```
5. $FF, DDRB
6. $00, DDRB
7. True
8. True

SECTION 4.2: I/O BIT MANIPULATION PROGRAMMING

1. True
2.
```
         CBI DDRB,7
H1:      SBI PORTB,7
         CBI PORTB,7
         RJMP H1
```
3.
```
         CBI    DDRB,2
         SBI    DDRB,0
AGAIN:   SBIS   PINB,2
         RJMP   OVER
         SBI    PORTB,0
         RJMP   AGAIN
OVER:    CBI    PORTB,0
         RJMP   AGAIN
```
4.
```
         SBI    DDRD,0
         SBI    DDRD,7
H2:      SBI    PORTD,0
         SBI    PORTD,7
         CBI    PORTD,0
         CBI    PORTD,7
         RJMP   H2
```
5. $9819 is 1001 1000 0001 1001 in binary; according to Figure 4-7, this is the CBI instruction, where a = 00011 = 3 and b = 001 = 1. According to Table 4-8, 3 is the I/O address of TWDR; thus, this is the "CBI TWDR,1" instruction and clears bit 1 of the TWDR register.