6502 Input Output Programming

by Ted Kosan

Part of The Professor And Pat series
(professorandpat.org)

Table of Contents

Monitoring And Controlling The Physical World

A shower of sparks fell to the floor as I ground the burrs off a steel part in my shop. I had earplugs in because the noise was deafening and so I did not hear Pat enter the shop and walk next to me to observe what I was doing. However, a sixth sense told me Pat was there so I grabbed a pair of safety glasses from a box and handed them to Pat as I continued to grind.

After I finished deburring the steel part, I turned off the grinder and said "Hello Pat, how are you today?"

"Fine." said Pat. "What are you working on?"

"A part for a robot" I replied.

"A robot?" asked Pat, with more than just a little bit of excitement. "What kind of robot?"

"A 3 axis plasma cutting robot." I said.

"A what!?" cried Pat.

"I think it would be easier to show you than explain it to you." I said. "Follow me to the other side of the shop."

I led Pat to an item that was about 4 feet wide, 4 feet long, and 4 feet tall which was covered by a silver tarp. As I slowly removed the tarp, the following yellow robot was revealed:

"Wow!" said Pat "Is this a real robot?"

"Yes." I replied.

"What does it do?" asked Pat.

"It automatically cuts shapes out of sheets of metal using a plasma cutting torch." I said.

"What's a plasma cutting torch?" asked Pat.

"A plasma cutting torch shoots a stream of compressed air through a nozzle at a plate of metal. The stream of air has an arc of electricity sent through it that turns the air into a hot plasma which melts the steel and cuts it." I said. "Would you like to see it operate?"

"Oh yes!" replied Pat.

I then gave Pat a demonstration of the robot. A movie of the robot in action that was taken during its initial testing phase can be found here:

http://jautomation.dev.java.net/jautomation.mpg

When the shape the robot had cut out had cooled, I picked it up and handed it to Pat.

"Wow!" said Pat. "I didn't know robots could do things like this. Did you build this robot?"

"Most of it." I replied. "I purchased the steel parts of the robot from the Internet and then I assembled and painted them, added motors and drive electronics, and interfaced the drive electronics to a computer."

"How is a computer able to control robot motors?" asked Pat.

"Do you remember what the 3 types of memory are that are present in a computer's memory map?" I asked.

"Yes, I remember," replied Pat "they are RAM, ROM, and I/O."

"What does I/O stand for?" I said."

"Input Output" said Pat.

"And what is I/O memory used for?" I asked.

"It allows a computer to talk to things outside itself, like a keyboard, mouse, hard drive, monitor, and network connection." said Pat.

"This is correct," I said "and I/O memory is also used to allow a computer to control robot motors."

Pat's mouth dropped open with surprise. "I/O memory can be used to control robot motors?" asked Pat. "How?"

"It will take some time to explain to you all the details of how this is done," I replied "but I can give you an overview and then I can show you how to start doing I/O programming with the 6502 emulator."

"The emulator can do I/O programming!?" asked Pat.

"Yes." I said.

"Can we do it right now?" said Pat.

"Sure." I said. "Help me cover the robot back up and then we can go to the electronics room."

Computer Interfacing

When we arrived at the electronics room I said to Pat "The process of attaching a computer to devices in the physical world so the the computer can monitor and control them is called **computer interfacing**. Computer interfacing is done using computer **output ports** and **input ports**. The word **port** is used because computer ports are similar to the port holes in the side of a ship.

An **output port** contains 2 parts:

1) Special electronics that transform bits in I/O memory locations into electronic signals that can be used to control a device in the physical world.

2) A connector somewhere on the computer that the device can be attached to so that the electronic signals can be sent to it.

An **input port** also contains 2 parts:

1) Special electronics that transform electronic signals coming from a device in the physical world into bits in an I/O memory location.

2) A connector somewhere on the computer that the device can be attached to so that the electronic signals from the device can be sent to the computer.

Robot motors are too complex for computer interfacing beginners to start with, so we will use **LEDs** as output devices and **simple switches** as input devices. Also, you will have to learn fundamental electronics before we can discuss the details of how the computer is interfaced to physical devices."

Output Ports And LEDs

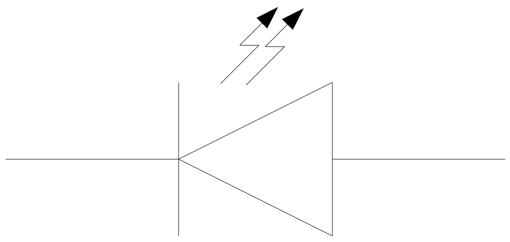
"What's an LED?" asked Pat

"**LED** stands for **Light Emitting Diode** and it is an electronic device which sends out light when electricity is applied to it." I replied. "LEDs are used as indicators in a wide variety of electronic devices including stereos, DVD players, and computers. The most popular ones emit red, green, yellow, or white light." I then opened a parts drawer, picked up an LED, touched Pat's hand with my pinky finger, and then gave the LED to Pat.

Pat looked at the LED then said "I know what you are talking about now!" said Pat. "LEDs are used all over the place!"

"Yes, they are." I said. "Here is the symbol that is used to represent an LED in an electronic circuit diagram." I then drew the symbol for an LED on the whiteboard. (see Fig.1)

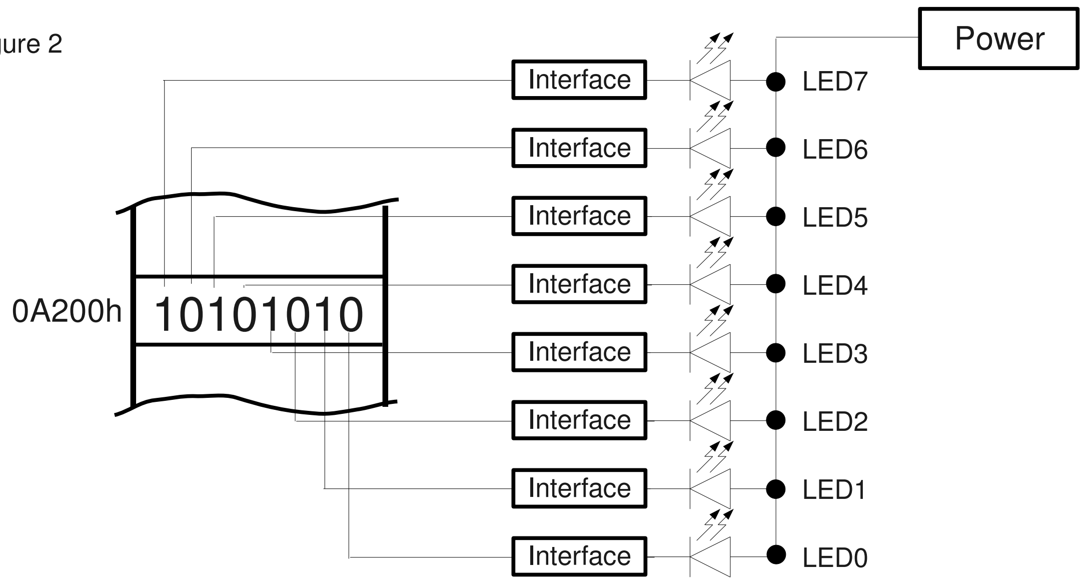
Figure 1



LED Symbol

"In the emulator, memory locations 0a200h and 0a400h are output ports and both ports have 8 LEDs interfaced to them. I will now draw a diagram which shows how each bit in output port 0a200h is attached to a simulated LED in the emulator." I then drew the diagram (see Fig. 2)

Figure 2



Output Port 0a200h

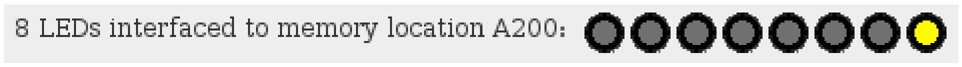
"This diagram shows that interface circuits are used to attach each bit in port 0a200h with an LED." I said. "When a given bit is set to 1, its LED turns on and when it is set to 0, its LED turns off."

"Can you show me how this works in the emulator?" asked Pat.

"Yes." I said. I then launched the emulator and used the **Enter** command to place a 00000001 binary into output port 0a200h.

```
-e a200 01
```

After entering this command, here is what was shown on the emulator's display:



"Cool!" cried Pat. "Can we turn all of the LEDs on now?"

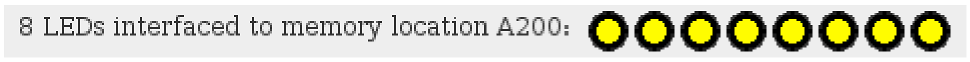
"Sure," I said "what number do I have to pass to the Enter command in order to do this?"

Pat looked at the ceiling for a few moments then said "FF hex."

I then entered the following line into the emulator:

```
-e a200 ff
```

And here was what was shown on the display:



Pat was very excited by this and one could almost see the gears turning behind those bright eyes. Finally Pat said "Can we make a program that blinks all of the LEDs on and off continuously?"

"Okay." I said. I then created the following program, assembled it, loaded it into the emulator, and executed it:

```
%uasm65,description=""
;Program Name: blink.
;
;Version: 1.02.
;
;Description: The purpose of this program is to blink
; the lights on and off continuously.
;

;**************************************
;      Program entry point.
;**************************************
      org 0200h
```

```
Main *
;Turn all the lights on and then waste some time
; so that the user can see the lights on.
      lda #11111111b
      sta 0a200h
      jsr delay

;Turn all the lights off and then waste some time
; so that the user can see the lights off.
      lda #00000000b
      sta 0a200h
      jsr delay

      jmp Main

;Exit the program.
      brk


;***************************************
;         Subroutines area.
;***************************************

;***************************************
;Delay subroutine.
;
;The purpose of this subroutine is to generate
; a delay so that the rate of the blinking
; can be controlled.
;
;Change the number that is being loaded into
; the 'A' register to change the delay time.
;***************************************
Delay *
;Save registers on the stack.
      pha
      txa
      pha
      tya
      pha

;Place 10 into the count down timer.  The count down timer
;will automatically decrement the value in memory location 0A800h
;at a rate of one dedrement per second until it reaches 0.
    lda #10d
    sta 0a800h

;Wait until the value in memory location 0a800h reaches zero.
WaitLoopTop *
```

```
    lda 0a800h
    bne WaitLoopTop

;Restore registers from the stack.
      pla
      tay
      pla
      tax
      pla

      rts


;*************************************
;        Variables area.
;*************************************

      end
%/uasm65
```

"In this program," I said "all of the LEDs are turned on then off in a continuous loop. A delay needs to occur after the lights are turned on to give the user time to see them and a delay also needs to occur after the lights are turned off so that the user can see their dark state.

The delay is performed by a subroutine which uses the **timer** circuit which is interfaced to memory location 0A800h. When a number is placed into this memory location, the timer will automatically decrement the location's contents until it reaches 0. Instead of decrementing the memory location as quickly as it can, however, it only decrements once every 100 milliseconds."

"What's a millisecond?" asked Pat.

"A millisecond is one thousandth (1/1000) of a second." I replied. "If one were to take a second and cut it into a thousand pieces of equal duration, each piece would be one thousandth as long as the original second."

Pat thought about this for a while then asked "Why does the timer use milliseconds instead of some other unit of time?"

"That is a good question." I replied. "One reason is that milliseconds are easy to build up into longer units of time. For example, how many milliseconds are in 1/2 of a second?"

Pat pondered this for a long time without coming up with an answer so I asked "Let's try a simpler question. How many milliseconds are in a second?"

Pat replied "There are 1000 milliseconds in one second."

"Correct," I said "and if 1/2 of a second is half as long as a full second, how many milliseconds are half as long as 1000 milliseconds?"

"Oh, I see!" said Pat. "500 milliseconds equals 1/2 of a second!"

"Yes." I replied. "Now, if 500 milliseconds equals 1/2 of a second, how much of a second is 100 milliseconds?"

"Hmmm." said Pat. "Since there are ten 100's in 1000, 100 milliseconds must equal 1/10 of a second. Does the timer use 100 milliseconds as its decrement rate because 1/10 of a second is an easy unit of time to work with?"

"Yes." I replied.

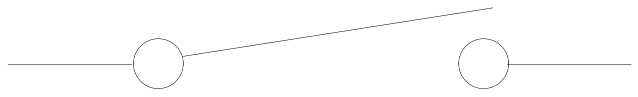
Input Ports And Switches

"I think I am starting to see how output ports work." said Pat. "Can you show me how input ports work now?"

"Okay." I said. "One of the simplest input devices that can be interfaced to a computer is a switch." I located a switch and gave it to Pat to look at:

"A simple normally off pushbutton switch like this one," I said "has two terminals on it that are connected electrically when the button is pressed. The symbol for a normally off pushbutton switch looks like this." I then drew the symbol on the whiteboard. (see Fig. 3)

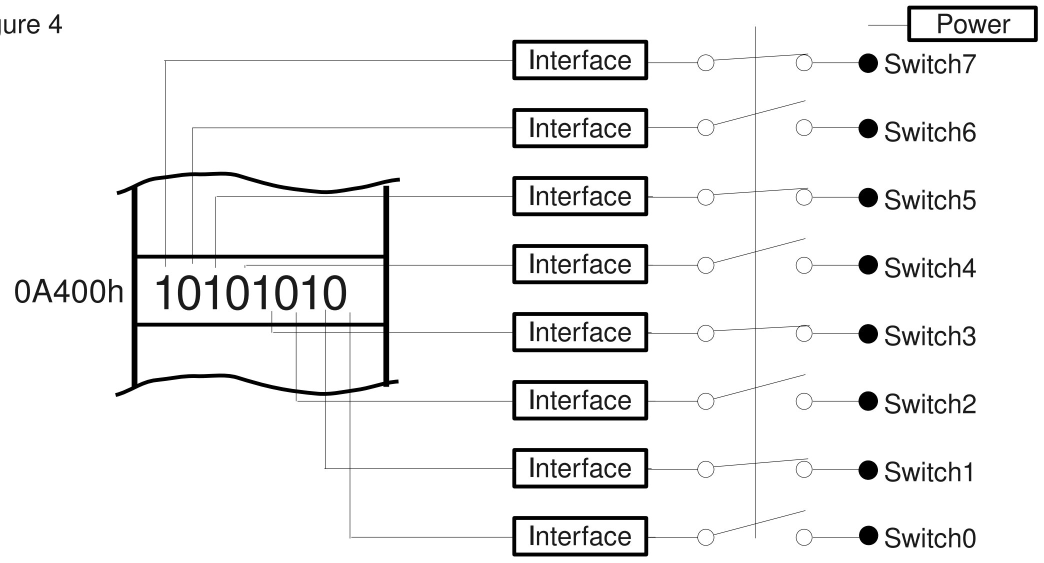
Figure 3



Switch symbol

"On the emulator, location 0a600 hex is an input port and it has 8 pushbutton switches attached to it." I said. "Each bit in location 0a600 is connected to a switch. When the switch is not pressed, the bit is set to a 1 and when the switch is pressed, the bit is set to a zero. I will draw a diagram on the whiteboard that shows port 0a600 and the switches it is connected to." I then drew the diagram. (see Fig. 4)

Figure 4



Showing The Status Of The Switches On The LEDs

"We will begin with a small program that shows the state of each switch on the LEDs attached to port 0a200h." I said. I then created the following program, assembled it, loaded it into the emulator, and executed it:

```
%uasm65
;Program Name: sw2leds.
;
;Version: 1.01.
;
;Description: The purpose of this program is to have
; the LEDs in port 0a200h reflect the state of the
; switches.

;*************************************
;      Program entry point.
;*************************************
      org 0200h

Main *
      lda 0a600h
      sta 0a200h

      jmp Main

      end
%/uasm65
```

"This program copies the value from location 0a600 hex to a200 hex in a continuous loop." I said. "After the program is running, press the switches on the emulator's GUI and each LED in output port a200 will turn on and off as the switch in the same bit position in input port 0a600 is pressed and released."

Pat pressed the switches and observed the changes this caused on the LEDs.

AND Turns Bits Off, OR Turns Bits On

After a while Pat said "In a program, how can you tell which switch is being pressed?"

"By loading the contents of location 0a600 hex into the 'A' register then using the 'AND' command to turn off all of the bits except the one you are interested in." I replied.

"What is the AND command?" asked Pat.

"The AND command, and its opposite the OR command, are used to manipulate individual bits in a register or in a memory location," I said "and they implement what are called **logic** operations. **AND is used to turn bits off** and **OR is used to turn bits on**. One way to understand how these commands work is to study what is called their **truth tables**. A **truth table** is simply a table that shows the various combinations of bits that can be fed into a logic command along with the result that will be produced. I will show you the truth tables for the AND and OR commands so you can see how they work." I then drew both truth tables on the whiteboard. (see Fig. 5)

	0	1
0	0	0
1	0	1

AND

	0	1
0	0	1
1	1	1

OR

Figure 5: AND and OR truth tables

"**The AND truth table," I said "indicates that**:
a 0 bit ANDed with a 0 bit results in a 0 bit,
a 0 bit ANDed with a 1 bit results in a 0 bit,
a 1 bit ANDed with a 0 bit results in a 0 bit, and
a 1 bit ANDed with a 1 bit results in a 1 bit.

The OR truth table indicates that:
a 0 bit ORed with a 0 bit results in a 0 bit,
a 0 bit ORed with a 1 bit results in a 1 bit,
a 1 bit ORed with a 0 bit results in a 1 bit, and
a 1 bit ORed with a 1 bit results in a 1 bit."

"Ummm, okay..." said Pat "but how are AND and OR used?

I replied "Lets say we have a pattern of 8 bits and we want to set all of them to 0 except for bit 2. We want bit 2 to remain whatever it was originally. If it was a 0, it will remain a 0 and if it was a 1, it will remain a 1." I then wrote the following bit pattern on the whiteboard:

```
    10110110 - Original bit patten.
```

"What we would do is to AND this bit pattern with 8 bits which have been configured to achieve the desired result. This second bit pattern is called a **bit mask** because it acts like a halloween mask in that it allows some parts of the original pattern to show through while other parts it changes."

```
    10110110 - Original bit patten.
AND 00000100 - Bit Mask.
    --------
    00000100 - Result.
```

"Notice that each of the original bits that are ANDed with 0 in the bit mask are turned into 0 bits, but the bit that was ANDed with a 1 bit in the bit mask remained what it was." I said.

"Pat studied the AND operation I had just performed with a look of confusion then said "I'm still not getting how this is useful."

"Perhaps if I use AND in a program, it will make better sense to you." I then created the following program:

```
%uasm65,description=""
;Program Name: switchlet.
;
;Version: 1.01.
;
;Description: The purpose of this program is to output a different
; letter of the alphabet depending on which switch was toggled last.
;

;************************************************************
;              Monitor Utility Subroutine Jump Table.
;************************************************************
OutChar   equ E003h ;Output byte in A register to serial port.

GetChar   equ E006h ;Get a byte from the serial port.

GetCharW  equ E009h ;Wait and get a byte from the serial port.

PrntMess  equ E00Ch ;Print a message to the serial port.

OutSpace  equ E00Fh ;Output spaces to the serial port.

OutHex    equ E012h ;Output a HEX number to the serial port.

DgtToBin  equ E015h ;Convert an ASCII digit into binary.

GetLine   equ E018h ;Input a line from the serial port.


;**************************************
;      Program entry point.
;**************************************
      org 0200h
```

```
Main *

CkSw0 *
      lda 0a600h
      and #00000001b
      beq CkSw1
      lda #'A'
      sta LastLetter
      jmp OutLetter

CkSw1 *
      lda 0a600h
      and #00000010b
      beq CkSw2
      lda #'B'
      sta LastLetter
      jmp OutLetter

CkSw2 *
      lda 0a600h
      and #00000100b
      beq CkSw3
      lda #'C'
      sta LastLetter
      jmp OutLetter

CkSw3 *
      lda 0a600h
      and #00001000b
      beq CkSw4
      lda #'D'
      sta LastLetter
      jmp OutLetter

CkSw4 *
      lda 0a600h
      and #00010000b
      beq CkSw5
      lda #'E'
      sta LastLetter
      jmp OutLetter

CkSw5 *
      lda 0a600h
      and #00100000b
      beq CkSw6
      lda #'F'
      sta LastLetter
      jmp OutLetter
```

```
CkSw6 *
      lda 0a600h
      and #01000000b
      beq CkSw7
      lda #'G'
      sta LastLetter
      jmp OutLetter

CkSw7 *
      lda 0a600h
      and #10000000b
      beq NoSwitch
      lda #'H'
      sta LastLetter
      jmp OutLetter

NoSwitch *


OutLetter *
      lda LastLetter
      jsr OutChar
      jsr Delay
      jmp Main


;Exit the program.
      brk


;**************************************
;        Subroutines area.
;**************************************

;**************************************
;Delay subroutine.
;
;The purpose of this subroutine is to generate
; a delay so that the rate of the blinking
; can be controlled.
;
;Change the number that is being loaded into
; the 'A' register to change the delay time.
;**************************************
Delay *
;Save registers on the stack.
      pha
      txa
      pha
      tya
      pha

;Place 10 into the count down timer.  The count down timer
;will automatically decrement the value in memory location 0A800h
;at a rate of one dedrement per second until it reaches 0.
    lda #10d
    sta 0a800h

;Wait until the value in memory location 0a800h reaches zero.
WaitLoopTop *
    lda 0a800h
    bne WaitLoopTop

;Restore registers from the stack.
      pla
      tay
      pla
      tax
      pla

      rts

;*************************************
;        Variables area.
;*************************************
LastLetter dbt "*"
      end

%/uasm65
```

"This program checks to see which switch, if any, is being toggled by the user and then it outputs an ASCII letter which has been associated with that switch." I said. "Toggling switch0 will output letter A's, toggling switch1 will output letter B's, and so on. The program does this by checking each bit in location 0a600 hex by isolating it with an AND instruction and then branching or not branching depending in whether it was set or not."

"I think I understand how AND works now." said Pat. "Can you show me how OR works?"

"Okay." I said "OR is used to turn bit on. Lets say we have a pattern of 8 bits and we want to set bit 2 to a 1 while allowing all the rest of the bits to remain what they were." I then wrote the following bit pattern on the whiteboard:

```
     10110010 - Original bit patten.
```

"What we would do is to OR this bit pattern with a **bit mask** which has been configured to achieve the desired result."

```
   10110010 - Original bit patten.
OR 00000100 - Bit Mask.
   --------
   10110110 - Result.
```

"Notice that each of the original bits that were ORed with 0 in the bit mask remain what they were, but the bit that was ORed with a 1 bit in the bit mask is changed to a 1." I said.

Exercises

Note: The following programs should be written as infinite loops.

1) Write a program that will turn LEDs 0 1 2 3 on while turning LEDs 4 5 6 7 off, then turn LEDs 0 1 2 3 off and LEDs 4 5 6 7 (remember, in an infinite loop).

2) Write a program that will turn all even LEDs on and all odd LEDs off, then turn all even LEDs off and all odd LEDs on.

3) Write a program that will turn LED 0 of your output port on then send this LED across the lights to LED 7 then back across the lights to LED 0.

4) Write a program that will turn LEDs 0 and 7 on, move these lights in to LEDs 3 and 4 then back out again to LEDs 0 and 7. Only 2 LEDs should be on at a time.

5) Write a program that will make your LEDs count from 0h to 0FFh then start over again at 0.

6) Write a program that will read the status of the switches and reflect this status on the LEDs attached to port 0a200h and 0a400h..

7) Write a program that will read the status of the switches and output this status as a HEX number on the computer screen (Hint: Use the monitor's OutHex utility subroutine.)

8) Write a program that will continuously output the alphabet in reverse order at a rate ranging from very slow to very fast depending upon the switch settings (Hint: Use the switches to change the timer count down value in the delay subroutine).

9) Write a program that will do the following (Hint: Use the 'and' instruction to determine which switches are pressed):
    Print "The front door is open." if switch 0 is pressed.
    Print "Your mailbox is open." if switch 1 is pressed.
    Print "The smoke alarm is on." if switch 2 is pressed.

10) Write a program that will dump the contents of memory locations 0E000h - 0E020h to the output port as an 8 bit light pattern at a rate of 1 location/second.

11) (OPTIONAL) Write a ping pong simulation that will send one light back and forth from the left side of your LEDs to the right side and back again. Each time the light crosses the LEDs, increase its speed a little bit. If the left player's switch is pressed while the leftmost LED is on

then send it back across the display, else give 1 point to the right player. If the right player's switch is pressed while the rightmost LED is on then send it back across the display, else give 1 point to the left player. The first player with 5 points wins. Keep track of the score and notify the users when either the left or the right player has won.