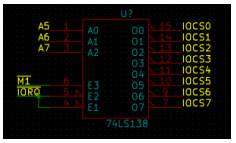
Z80 Journal

Z80 I/O Space

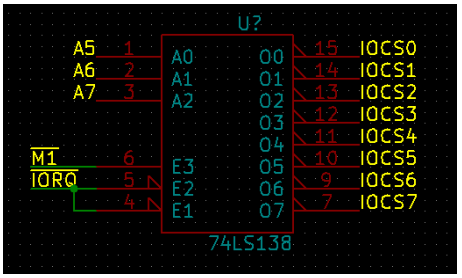
Posted on October 7, 2015 by Matt Cook



A very interesting feature of the Z80 microprocessor is that it has separate memory and I/O space. Separated memory and I/O space means that instead of having to treat your peripherals as a memory address in between your other actual memory devices you can instead address them in a different way. This may not seem like a big deal, but it is very convenient.

To address I/O devices there are only two instructions that you have to know 'OUT' and 'IN'. These two instructions handle the heavy lifting to exchange data back and forth from I/o devices. You are probably asking yourself well how is that any different from a 'LD'
instruction? Well it is all about the control signals. In memory-mapped instructions like 'LD' the control signals used are 'MREQ', 'RD', and 'WR' whereas in I/O instructions the control signals used are 'IOREQ', 'M1', 'RD', and 'WR'. The 'IN' and 'OUT' instructions then act very similarly to the 'LD' instruction using the proper control signals to exchange data with the peripheral. The major difference however is that the addressable I/O space is only made up to the first half of the address bus A0 to A7. This means that only 256 I/O devices can be addressed without any special tricks. This goes lower if like many peripherals the device takes up more than one address. For instance, a 16550 UART will take up eight addresses to address all of the registers within the device.

To start an I/O cycle the /IORQ goes low and /M1 goes high. You need these two signals /IORQ and /M1 because during an INTACK cycle /IORQ goes low and /M1 goes low. So the only distinguishing feature between knowing if it is an I/O cycle or an INTACK cycle is the state of M1. The way to decode an I/O cycle is to use a chip which has multiple select lines for enabling like the 74LS138 1-of-8 decoder which has two active-low enable pins which can be used to spot /IOREQ going low and one active-high enable pin which can be used to spot /M1 being high. When both conditions are met the decoder will enable one of its 8 outputs based off of the three inputs you supply. For instance, if you were to supply the A7, A6, and A5 address lines of the Z80 to this chip's inputs you would effectively have 8 I/O chip selects which can be addressed each $20 addresses and would have 16 addresses (registers) to select per device. This satisfies all of the conditions to decode a
peripheral's addresses and is what I actually use in my computer's design. As stated above a chip like the 16550 would worked under this type of addressing. Its chip select would use 1 of the 8 I/O chip select outputs and if we used the first chip select the UART would be
enabled when addressing I/O address $00->$1F. Since the 16550 chip has eight sub-addresses we could use addresses $00->$08 to utilize all internal registers. The next peripheral would then need to be addressed at the base address $20 and under the above scheme we "lose" $17 addresses since the UART does not need all of the



There are many other methods for I/O decoding and they each have advantages and disadvantages based off of how many peripherals you want to support and how many registers each of your peripherals employ. It is somewhat of art-form to maximize the amount of peripherals you can support.

Now on to programming in I/O space. Like I said above there are two instructions needed for I/O space 'IN' and 'OUT'. The structure of these instructions will, based off of the main Z80 instructions, take the form below.

```
IN A,(*)
OUT (*),A
```

Notice how both instructions use the accumulator (A) which means you need to load into the accumulator or have the accumulator ready for loading into. In an 'OUT' instruction you would need to have the accumulator loaded with the data you want to send to the I/O device. In an 'IN' instruction you would need to have the accumulator ready to accept the data coming from the I/O device. In both cases the * between the parenthesis indicates the I/O address to send the data to. This would be between 00H and FFH. It is highly recommended that if your assembler accepts pre-defined labels that you use these to make your code more clear as to what is happening instead of using magic
numbers. This can be seen in the following example.

```
UART0: .EQU 00H ; DATA IN/OUT
UART5: .EQU 05H ; LINE STATUS

SEND_CHAR_UART:
         IN A,(UART5) ; Get the line status register's contents
         BIT 5,A ; Test BIT 5, it will be set if UART ready
         JP Z,SEND_CHAR_UART
         LD A,B ; Get character from B Register obtained earlier          OUT (UART0),A ; Send charac
         RET
```

That is really all there is to it. Most I/O devices have address inputs, and chip select lines built in. You just need to connect these up to the correct address lines of the Z80, decide which I/O address you want to put the device at, decode the address to enable the devices
chip select in the appropriate manner, and program your software to send data 'IN' and 'OUT' of the device via the connected data bus to the device. Simple once you wrap your head around the process.

About Matt Cook
Computer Engineer
View all posts by Matt Cook →

This entry was posted in Uncategorized and tagged Assembly, Code, DIY Electronics, Hardware, Z80, Z80 Microprocessor. Bookmark the permalink.

Z80 Journal