Add a Stack to Your 8008

by
Hal Chamberlin
The Computer Hobbyist
PO Box 295
Cary NC 27511

Besides higher speed, the most significant improvement offered by the 8080 is the addition of a general purpose stack capability. Using the stack, the programmer may save registers used in subroutines and interrupt service routines and then later restore them. Arguments to subroutines may also be pushed onto the stack. In the 8080 the stack is kept in main memory and addressed by means of a stack pointer register. One inconvenience of the 8080 stack is that data may be pushed and popped only in byte pairs creating wasted space if only a single register needs to be saved. Also, the stack pointer MUST be set up at the beginning of a program before any subroutines are called and kept valid at all times or very strange things may happen.

Much of the programming convenience of the 8080 stack may be had on an 8008 system with the addition of about six ICs and the use of one input and one output address. The basic stack is 16 elements deep which is generally adequate for register saving applications. Addition of more chips and substitution of 256 X 4 RAMs for the 16 X 4 RAMs gives a 256 element capacity, ample for almost any use. In either case the added hardware provides both a stack pointer and a dedicated address is given the name STPOP for STack POP. When an OUT STPSH is executed by the program, all of the existing data (or garbage) in the stack is conceptually pushed down one location and the byte in register A is written into the top location which was vacated. When an INP STPOP is executed, the contents of the top location are read into register A and then all of the lower data in the stack moves up one location and the top location is lost.

An obvious application of the stack is in writing subroutines that do their job without destroying any registers. A simple example is the exchange HL and DE subroutine in Fig. 1. First register A is pushed onto the stack. Then registers H and D and L and E are exchanged using A. Finally the original state of register A is restored by popping it off the stack and the subroutine returns. Because of the push-down nature of the stack, one subroutine that uses the stack may call another subroutine

Fig. 1. A subroutine to exchange DE and HL register pairs using one stack location and no additional registers.

```
XCDEHL OUT  STPSH     SAVE A ON THE STACK
       LAH            EXCHANGE H AND D
       LHD            USING A
       LDA
       LAL            EXCHANGE L AND E
       LLE
       LEA
       INP  STPOP     RESTORE A FROM STACK
       RET            AND RETURN
```

Reprinted by permission from The Computer Hobbyist, *May 1975.*

that uses it and so on without loss of data as long as the stack's capacity is not exceeded. The only requirement is that all of the data saved on the stack by a subroutine be popped back off before it returns.

Fig. 2 shows a completely general interrupt service routine that uses the data stack to save all registers and the state of the conditions (C, Z, S, P). When entered, register A is first pushed onto the stack. Then the remaining six registers are saved one at a time by first loading them into A and then pushing A onto the stack. None of the instructions necessary to do this affect the condition flags. Finally a chain of conditional jumps is executed to create a "magic number" in A that reflects the state of the conditions. After A is pushed onto the stack, the interrupt may be serviced without restrictions on register usage.

In order to return to the interrupted program, first the magic number is popped off and added to itself with an ADA instruction. The number is such that the proper conditions are set when it is added to itself. Next the six index registers are popped off and restored in reverse order from which they were saved. Finally A is restored and a RET instruction is executed. This method of complete status saving may be modified for use by a debug program. Debug can be entered by a console interrupt and the user may examine things. Then program execution may be resumed with no loss of data. These two programming examples should serve to illustrate the use of the stack.

The stack is implemented with an up-down counter and

Hal Chamberlin and his associates at *The Computer Hobbyist* put out excellent small systems technology . . . designs include a high reliability audio cassette recording method, an inexpensive high resolution graphics display – and this article's stack design among others. Several of their more general purpose designs (e.g., tape interface, CRT display) are soon to be available in kit or assembled versions. This article describes a custom modification of an 8008 based system which you can add to an input/output port to achieve a stack mechanism. The method is that old standby of minicomputer instruction set escape mechanisms – use I/O commands to implement "new instructions." With a stack of sufficient size and suitable save/restore subroutines accessed by RST instructions of the 8008, you can eliminate conflicts in register usage between multiple levels of subroutines. The overhead penalty is a single RST or CAL instruction in the linkage code, the register save and restore routines, and the time required to execute the save/restore subroutines.

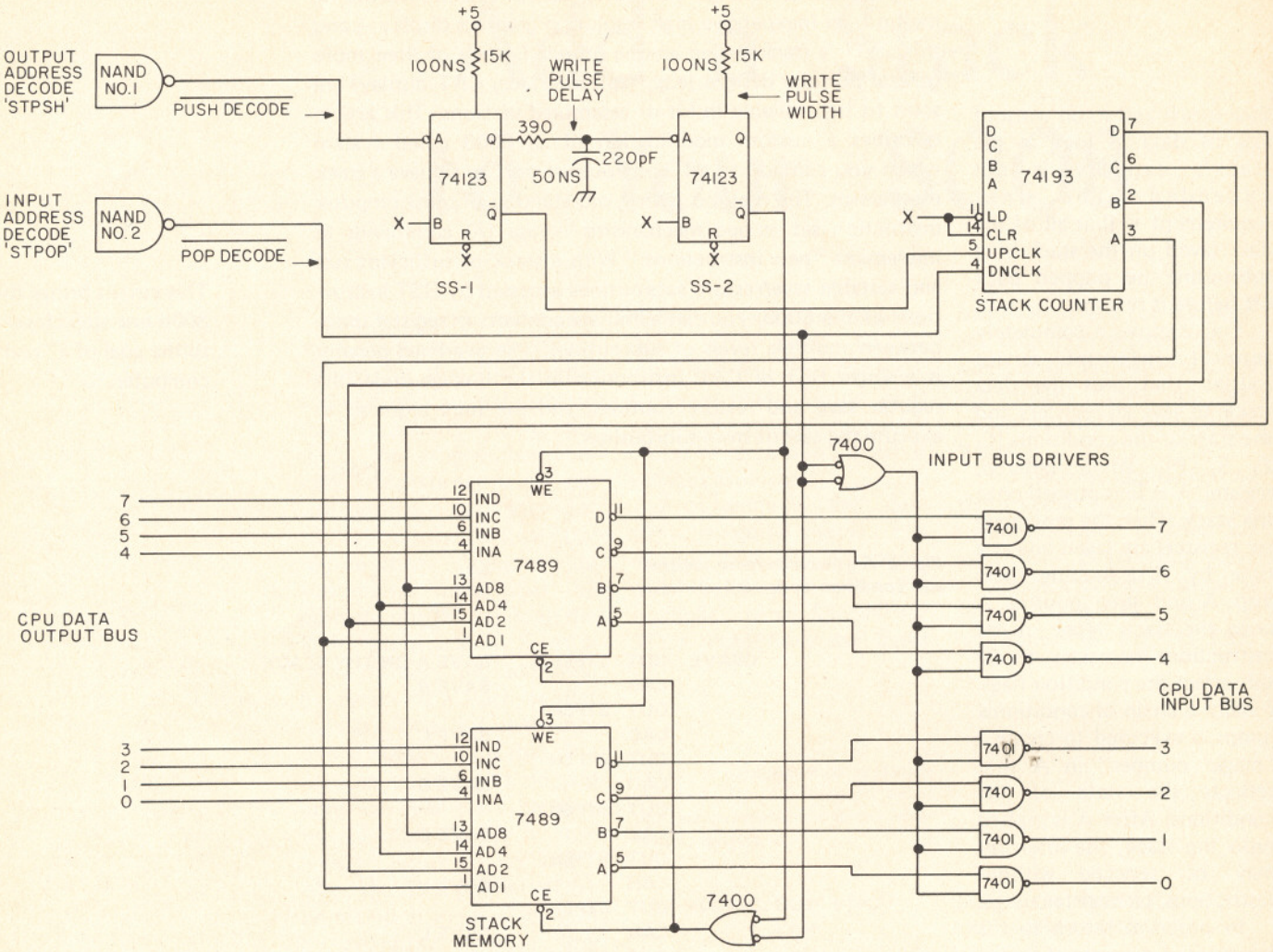
. . . CARL

This circuit brings the 8008 one step closer to the goal of a "real" computer.

Fig. 2. A general purpose register and condition code save routine.

```
GSAVE   OUT   STPSH     SAVE A ON THE STACK
        LAB             SAVE B
        OUT   STPSH
        LAC             SAVE C
        OUT   STPSH
        LAD             SAVE D
        OUT   STPSH
        LAE             SAVE E
        OUT   STPSH
        LAH             SAVE H
        OUT   STPSH
        LAL             SAVE L
        OUT   STPSH
        LAI   0         CLEAR A
        RAR             PUT CARRY IN HIGH ORDER
        JTZ   GSAV3     JUMP IF ZERO FLAG IS ON
        LBI   170B      PUT INTO B THE BIT MASK TO
        JTS   GSAV1       TURN OFF THE ZERO FLAG AND
        LBI   030B        RESTORE THE SIGN FLAG
GSAV1   JTP   GSAV2     OR IN A 004B IF PARITY
        ORI   004B        INDICATOR IS OFF
GSAV2   ORB             COMBINE B AND A
GSAV3   OUT   STPSH     SAVE MAGIC NUMBER ON STACK
*           REGISTER AND CONDITION RESTORE ROUTINE
GRSTR   INP   STPOP     RESTORE MAGIC NUMBER FROM STACK
        ADA             ADD IT TO ITSELF TO RESTORE CONDITIONS
        INP   STPOP     RESTORE L
        LLA
        INP   STPOP     RESTORE H
        LHA
        INP   STPOP     RESTORE E
        LEA
        INP   STPOP     RESTORE D
        LDA
        INP   STPOP     RESTORE C
        LCA
        INP   STPOP     RESTORE B
        LBA
        INP   STPOP     RESTORE A
        RET             RETURN WITH STATUS RESTORED
```

Fig. 3. Logic Diagram of a 16 element data stack. NOTE: "X" refers to a source of logical one, usually a 1k resistor to +5.



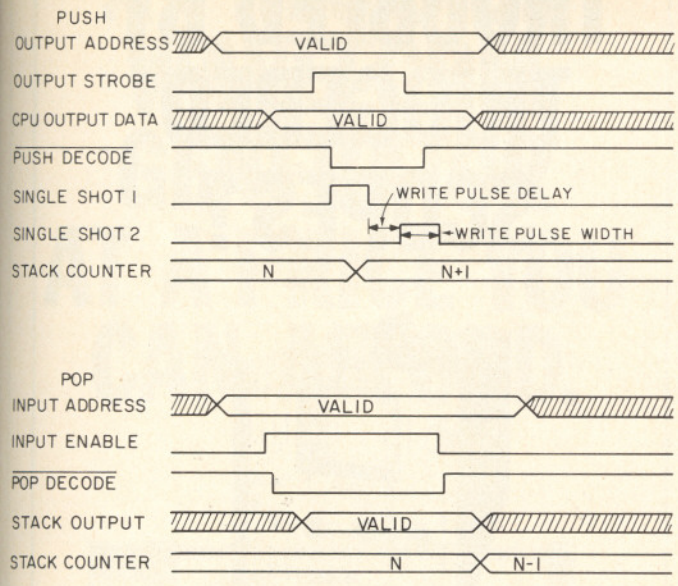
An obvious application of the stack is in writing subroutines that do their job without destroying any registers.

a random access read-write memory. Rather than the data moving when pushes and pops are executed, the up-down counter acts as a pointer to the top element on the stack and the pointer moves. The logic is set up so that when an OUT STPSH is decoded, the counter first counts up one notch and after sufficient time for the address to settle in the RAM, a write pulse is generated to write the data from A into the RAM. The write pulse delay can be fairly short (50 NS or so) in the 16 element stack but must be at least 200 NS for the slower MOS RAM used in the 256 element version. It is possible that a timing problem may arise in a system using the 8008-1 if the output data is not valid for the sum of write pulse delay and write pulse width (950 NS) required by the MOS RAM. (Timing given is for the 2101 RAM. Matters are improved if 2101-1, 2101-2, or 9101 RAM is used.) There should be no problems with the bipolar RAM in the 16 element version.

When an INP STPOP is recognized, the contents of the currently addressed location are simply gated onto the input bus. The counter counts down one notch at the end of the INP instruction thereby addressing the next lower element on the stack.

Figs. 3 and 4 show the logic diagram and timing chart respectively for a 16 element data stack. A bus type of I/O system (as opposed to a "port" type) is assumed. As shown, any system with either separate data input and output busses or a bidirectional bus may be used. Some systems may use an output bus with TRUE data and an input bus requiring FALSE data. In this case, the 7401s may be omitted and the TTL RAM outputs tied directly to the input bus. The two single-shots, SS-1 and SS-2, are used to time the sequence

Fig. 4. Stack Timing Diagram.



of events for a stack push. First, NAND gate number 1 recognizes the coincidence of the STPSH device code on the address bus and an output strobe pulse or its equivalent. The gate output triggers SS-1 which increments the stack pointer counter when its cycle is finished. An RC network between the two single-shots delays firing of SS-2 until the counter has settled down and the RAMs recognize the new address. The write enable is connected to SS-2 which allows data on the CPU output bus to be written into the newly addressed RAM location.

The occurrence of an INP STPOP is detected by NAND gate number 2. As long as the gate is satisfied, data from the RAM is placed on the CPU input bus. At the end of the INP instruction when the NAND gate output goes back to a ONE, the counter decrements to address the next lower element on the stack. A 7400 connected as an OR-NOT enables the memory when either a push or a pop is being executed and disables it otherwise.

The logic necessary for a 256 element stack is essentially the same as for the 16 element version. The major differences are that a separate single-shot should be used to time the write delay and that a buffer is absolutely necessary to drive the CPU input bus. If the polarity of the input bus is the same as that of the output bus or it is the same bus, 8093 or 74125 noninverting and quad tri-state buffers are convenient to use. Open-collector 7401 gates may be used instead if the input bus is inverted. The chip enables on the MOS RAMS should be grounded so that the chip is always enabled. The connection to the bus drivers is left as it was for the 16 element version however. The timings for the write delay and write pulse width single-shots can be set to the minimum values allowable for the standard 2101 RAM. If a 9101 is used, the timing may be speeded up considerably. An 8101 may require somewhat slower timing. In any case be sure to check the data sheet for the RAM being used.

After writing a few programs using the stack you will wonder how you got along without it. The size and speed of some routines may be improved by a factor of two if use of the stack alleviates the need to constantly reference memory. An overall improvement of 10 to 20% can be expected on large programs such as assemblers. The biggest improvement however will be in coding time since register usage will not have to be carefully planned in advance.

Use the stack to pass parameters to subroutines when you don't have enough registers.