Core Loading Automaton

A tool I originally developed at the request of Peter Vaughan of The National Museum of Computing, this minor modification to an IBM 1130 will allow a PC based file of memory words to be loaded automatically into the core of an 1130.

A significant speedup was accomplished over previous versions by redesigning to use the cycle steal mechanism of the IBM 1130, analogous to Direct Memory Access (DMA) of microprocessor systems. This supported higher speed peripherals which used cycle stealing to fetch or store words into core memory without involving programs running on the CPU.

The device controller requests use of cycle stealing by raising the request line for the CS level it was assigned. When the CPU has finished a memory cycle, it grants the highest priority request a cycle steal and signals it was granted by raising the CS Level line.

The controller sets the bits of the data word and the memory address by pulling down the bits of the IO Data and Address bus to zero for each bit that has a 1 value. When the cycle steal clock reaches state X6, the request is dropped and the IO Data and Address lines are turned off by the controller. The end of the cycle steal is indicated by the dropping of the CS Level line.

My device is implemented as a shield sitting on an Arduino Mega 2560 to handle the conversation with a terminal or other program outside the IBM 1130. It watches the key CPU signals and sets the data, address and cycle steal request lines just as a device controller would do. Thus, it will trigger a cycle steal for a specific core address and store a given word there.

To simplify the design, it requires that the CPU be stopped before it will work. This ensures that no device controller is attempting cycle steals and that no other requests for memory access are taking place. It uses the highest priority cycle steal level (CS Level 0).

Sending a line over the serial line starting with # indicates the start of a session. The loader will ignore input until activated. Sending another # will end the session and inform the operator to resume normal 1130 operations. If core contents are saved from the IBM 1130 simulator, that file needs to have a line with # added at the front and bottom in order to use it with the loader.

Core Loading Automaton

When an address (hex word prefixed by @) is received, it sets up that address in the Arduino as the location for the next data word received When it receives a hex word without the prefix, it uses the saved address in the Arduino to load the data and increments the address afterwards so that the next data word goes into the next sequential location.

When an address prefixed by = is received, the code saves that address and then when the session is deactivated, it will remind the user to load that into the IAR so that the 1130 will fetch instructions from that address when Prog Start is pushed.

If a hex word is prefixed by Z then the hex value is a count for how many words of 0000 should be loaded into contiguous locations in 1130 core.

The loader communicates over a serial link on a USB cable, interpreting each line of text sent to it as four hex characters with some simple error checking. The line is converted to the 16 bit values for the data word to load.

The device uses open collector gates to connect selected pins on the 1130 to ground when that signal is asserted by the logic. Gates are connected to pins where we need to read CPU signals, passing them to the logic. Wire wrap is used to link the pins to a header on the shield.

We connect to the +12V power supply from the main 1130 distribution terminal block and use it to provide power to our loader device. The components on the shield regulates the raw supply down to 9V for powering the Arduino and to 5V to power the logic on the shield.

Core Loading Automaton

Software

The Arduino program loops listening to the serial port (USB link to PC) at 9600-N-8-1 setting. Command characters (#, =, @, or Z) are pulled off the line and handled first. It builds up a line of four characters, if longer it is an error and if shorter the input is ignored. The code skips any NL (x0A) or CR (x0D). It looks for exactly four characters that are valid hexadecimal - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, or F. The letters can be upper or lower case ASCII.

Once it has a good word, it sets the data value and address on its output pins and raises the request line to the shield via another output pin. It waits for the incoming Request Done signal on an input pin, then drops the data, address and request lines to end this cycle. Once it is done, we go back to give a prompt and accept the next word of entry from the terminal hooked to our USB cable. We also bump the saved memory address at the end of each cycle.

If the user enters an @ in the line with the four hex characters, we capture that as the new memory address we will output for the next data request.

This code is written for an Arduino Mega 2560 board. We need 32 output pins and 1 input pin, which excludes most of the smaller Arduino boards. The hardware that interfaces this to the IBM 1130 is contained on a shield, a small PCB that is mounted atop the Arduino.

The code and other design files is shared on Github to be available to any 1130 restorer wishing to use or modify this device.