

**Problem**

A Motorola 68000( external **EIGHT-BIT DATA BUS VERSION**) based data acquisition system incorporates 8K bytes of SRAM, 8K bytes of ROM, a simple generic UART, an Analog-to-Digital converter, a Centronics parallel printer, and a separate stand-alone latch which will serve as an interrupt vector register. **SRAM starts at address zero( YES, THIS IS NOT A TYPO)** and therefore occupies the lowest 8K byte addresses\*\*. ROM immediately follows SRAM. Next come the A/D, the Printer, the UART, and finally, the single 8-bit latch that will serve as the interrupt vector register. The A/D will use its end-of-conversion (EOC) output to issue interrupt using level two . Although the A/D, by itself, **does not support vectored interrupt**, it will still work in conjunction with an externally derived vectored interrupt scheme to be described below. The printer and the UART will work with polling.

**The whole system works as follows:**

- a) Microprocessor polls UART( with a remote terminal attached), looking for any key to be struck by an operator.
  - b) When keystroke is detected, microprocessor sends the number 64 decimal to the stand-alone external latch to serve as the vector number for the level two interrupt by the A/D.
  - c) The microprocessor then commands A/D to do an initial conversion.
  - d) Microprocessor sits in a dummy loop waiting for interrupt from A/D at end of conversion.
  - e) When interrupt is detected, microprocessor goes to execute the interrupt subroutine( ISR), which simply reads the byte from the A/D, polls the printer, sends the byte to the printer, triggers the A/D to convert again, and returns to the dummy loop of the main program to wait for another interrupt from the A/D at the end of the current conversion.
- 
- A. Generate a memory map for the system. Use a block decoding scheme to enable simple partial decoding to be done. Clearly label beginning and ending addresses of all devices. Label all devices.
  - B. Provide an address decoding table using partial decoding. Design the address decoder.
  - C. Sketch the **detailed** interconnection circuitry showing how the microprocessor buses and address decoder outputs connect to SRAM, ROM, A/D, UART, Printer, Interrupt generation and interrupt acknowledgment circuitry, DTACK and all appropriate glue logic. **Include complete detailed circuitry to show how the interrupt vector in the stand-alone latch will be read by the CPU during interrupt acknowledgement cycle.** Assume that the UART's addressable internal registers have **only two addresses**, namely, Data and Control. The Data address is for both Transmit and Receive. The Control address is for both control and status. Specify your own simple minded UART control and status bit patterns for a very simple transmit and receive mode using polling.
  - D. Write the assembly language statements that represent initialization of the exception vector table, making sure to take care of the reset exception as well as the vectored interrupt with vector number 64 decimal used for the A/D's interrupt subroutine. Choose appropriate starting addresses in **ROM** for your main program and the ISR.
  - E. Write the assembly language statements for the main program, i.e, the initialization of the UART, the polling of the UART for any received character, the writing of vector number 64 decimal into the external latch, the initial triggering of A/D conversion, and the dummy loop while waiting for interrupt from A/D.
  - F. Write the assembly language routine that represents the interrupt subroutine for the A/D. This entails... Reading from A/D, polling the printer, writing converted byte to printer, and return to dummy loop in main program with the 68000 instruction : RTI.

**\*\* NOTE: Even though problem one requires you to let the SRAM occupy the lowest 8K bytes of the CPU's address space( zero thru the first 8K byte addresses), the exception vector table must still occupy the lowest 1K bytes, which IS IN RAM IN THE SYSTEM DESCRIBED ABOVE!**

**However, your application programs in parts E and F of the problem , must still be in ROM.**