Interrupt module notes:

1/18/2021  
  
It seems that some instructions entail interrupts as part of their normal operation, as of now the operating state does not distinguish between instruction interrupts or run of the mill interrupts. This could be problematic if a processor is operating in a non-interruptible mode, and wishes to use an instruction that requires usage of the interrupt module. I haven’t figured out what modes should be non-interruptible yet, but I assume that maybe when the processor is in root mode or when it is first starting up it should be non-interruptible. To counter the issue outlined above, I will be supplying the hierarchy/interrupt generator control logic a copy of the operating state so that it can restrict the types of interrupts that can request processor interruption to interrupts used by standard instructions.

As for the ram module, I think that there should be an interrupt hierarchy chip for each microprocessor.  
( each hierarchy chip is supplied with the same set of signals )   
  
In the hypothetical scenario where there are no other interrupts aside from a processor A interrupt request and a processor B interrupt request, Hierarchy chip A should generate an address and request processor access, and Hierarchy chip B should not generate a request.  
  
  
Like my initial design for the ram access system, I believe that the interrupt module should have a now servicing input so that an incoming higher ranking interrupt signal cannot steal the interrupt by a lower ranking interrupt signal.

> although the control unit is to be much larger, I only use one chip as to symbolize the whole unit.  
All eeproms will receive the same input signal, so only one is needed to symbolize the cu  
  
  
  
Description:  
  
Typically the control unit starts off a new instruction by performing the fetch cycle and then said instructions microinstructions. However, if the interrupt request line is active and the current operating state allows for interrupts, the control unit will instead perform an interrupt instruction. Because an interrupt instruction must be able to occur during any form of instruction, every instruction must have interrupt code programmed into the cu.  
  
To prevent an interrupt from only partially activating, i.e. request occurs during micro instruction 011,  
there must be some sort of initial activation which acts as a prerequisite for an interrupt instruction.   
  
Part of an interrupt instructions first micro is to change the operating state into “interrupting”.   
If the interrupt occurs after it is able to enable this state, then the current operation will continue.  
  
To rephrase, the only operating state that an interrupt is able to occur during is the “interrupting” state.  
The only chance the operating state gets to change into the interrupting state, is when the current operating state is interruptible, the first micro is occurring, and an interrupt request is present.  
  
  
  
  
  
  
hierarchy chip:  
  
  
Interrupt requests from every module are sent to an eeprom’s inputs. From here, the eeprom will make a decision on which interrupt has the highest priority. (i.e a shutdown interrupt would have a higher priority than keyboard interrupt). After deciding on an appropriate signal, the eeprom outputs a binary value representing the selected interrupt.  
  
   
  
  
  
This value is fed into an address generator, which holds pointers to software based interrupt resolution.  
  
during an interrupt, this address is sent to the instruction module, via the cu.

The resolution lines lead to the module in question who prompted an interrupt. Upon resolving an interrupt, the CU will set the respective resolution line to one, which resets the interrupt sr latch