I/O

- ULTIMATELY, INTERACTION W/ I/O DEVICES INVOLVES READING & WRITING REGS.

  (NOWADAYS MEM TOO)

- 2 MECHANISMS FOR TRANSFER

  - SPECIAL I/O INSTRS.

    e.g. | IN | DR | DEV # |

    | OUT | SR | DEV # |

  - "MEMORY MAPPED I/O" -> (MMIO)! JUST USE REGULAR LOADS/STORES, BUT FROM/TO SPECIAL ADDRESSES [WHY?]
    1) PIN LIMITATIONS
    2) OPCODE BLOAT
    3) FLEXIBILITY

    x86 has both.

  - THESE ADDRS DO NOT GO OFF TO DRAM CHIPS, BUT TO DEVICE REGS

- ON LC-3, xOOOO -> x FDFF = MEM

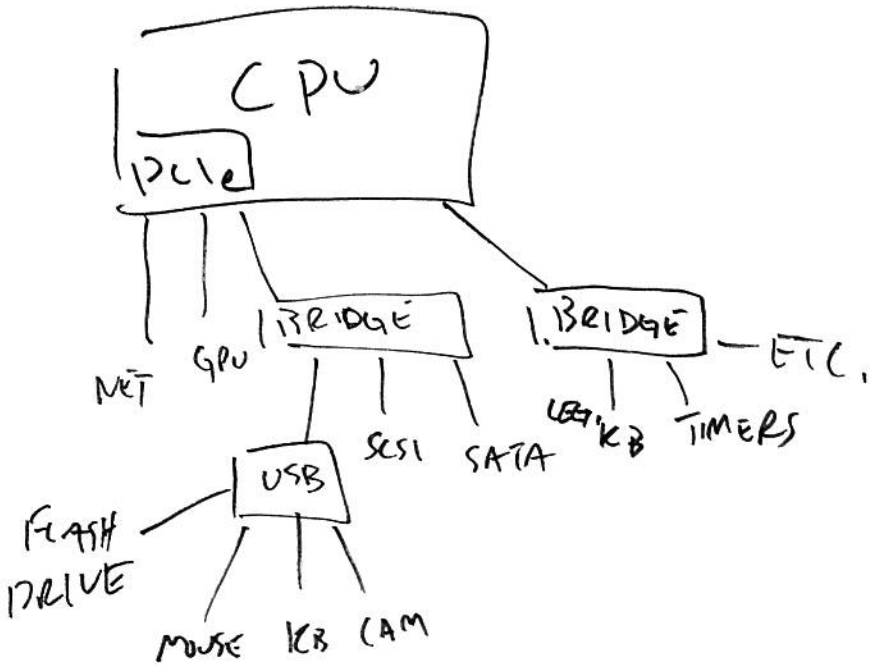
  x FE00 -> xFFFF = MMIO ADDRS

  ↳ 512 DEV REGS AT MOST!

- IN REALITY THINGS ARE MUCH MORE FLEXIBLE. PCI ALLOWS US TO ADD/REMOVE MMIO DEVICES AND CHANGE WHERE THEY LIVE

- ON LINUX, WE CAN LOOK AT /proc/iomem

- THERE'S A PIECE OF SOFTWARE (BIOS) THAT DETECTS ALL DEVICES, AND ASSIGNS ADDRESSES TO THEM. ALMOST ALL SYSTEMS USE PCI BUS FOR THIS (PCIe THESE DAYS)



- WITH LC3 WE CAN'T PLUG IN NEW DEVICES.

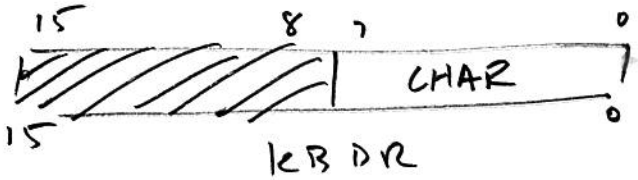
- MUST SYNCHRONIZE WITH I/O DEVS SOMEHOW
  [HANDSHAKES / READY / STATUS BITS]
  ↳ JUST MORE REGISTERS ON DEVICE
  (SYNCH. VS. ASYNCH. DISCUSSION)

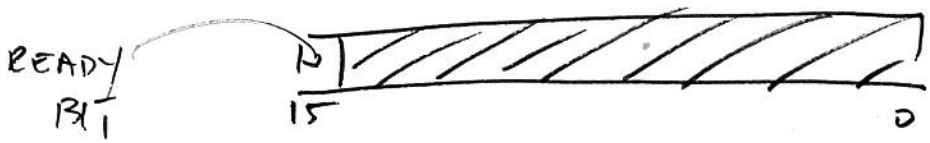
- POLLING VS. INTERRUPT-DRIVEN I/O

LC-3 KEYBOARD

2 REGS: - KBDR (DATA)
- KBSR (STATUS)

THIS IS PRETTY COMMON FOR SIMPLE I/O DEVS.





- KB H/W PUTS ASCII CODE IN KBDR WHEN USR TYPES KEY, SETS READY BIT IN KBSR TO 1

- CPU READS KBSR, IF DATA AVAIL., CAN GET CODE FROM KBDR.

- READING KBDR AUTOMATICALLY CLEARS KBSR[15]

KBSR IS @ xFE00

KBDR IS @ xFE02