I. Design of Instruction sets based on the needs of the applications that will use it.
   A. Arrays often need to do the same series of operations on each member.
      1. If the size of array is known then could use a label for each element. This is inflexible though fast.
      2. If size of array is unknown, then could make self-modifying code to change the address of a direct mode
         instruction. How?
      3. Indexed addressing adds the value of the X register to the operand to determine the final address.
   B. By the end of the quarter, you should know what HLL operations specific instructions facilitate.

II. Index Addressing Functionality
   A. Use the + to indicate Index Addressing in mnemonics, e.g. LDA+ $102, STA+ $105.
   B. Uses unsigned addition.
      1. Can access addresses before base address. For example, if XR is $FFF then LDA+ $101
         will load the ACCUM with the value at $100.

III. X Register Instructions
   A. LDX will affect OV flag. This is true for all CUSP instructions that load an address register with a word, i.e. LDX,
      LDS, and LDF.
   B. ADX and SBX use unsigned arithmetic, so OV and EQ affected, but not LT.
   C. CMX affects LT and EQ.
   D. TAX, transfer lower 12 bits of ACCUM to XR. If any of the upper 12 bits of the ACCUM are set then OV is set.
   E. TXA, transfer XR to lower 12 bits of ACCUM, and clear the upper 12 bits of the ACCUM. No flag affects-why?
   F. Can use any addressing mode.
      1. Index addressing is allowed because address calculations are performed before the XR is modified.

IV. Array Transfers
   A. Array[I] := Array[J ];
      1. LDX J
      2. LDA+ Array
      3. LDX I
      4. STA+ Array
      5. ....
      6. Array: .BLKW 10, 0
   B. For transferring from one array to another, it would be handy to have more than one XR. Some computers do.
      What are the disadvantages of more than one XR?

V. Array Looping Instructions
   A. Common in HLL to change array index by one and then compare with some value to determine if we are done with
      a loop.
   B. AOC, Add One and Compare.
      1. AOC $100 replaces ADX# 1, CMX $100
      2. Advantages: speedier, smaller, and flexible about source of comparison value.
      3. Disadvantages: Can't move backward through array.
   C. SOJ, Subtract One and Jump if not less than zero.
      1. SOJ $100 replaces SBX# 1, JNO $100
         a) Use OV because unsigned arithmetic does change the LT
      3. Disadvantages: Limited to comparison of XR with zero.

VI. Non-zero-Based Arrays
   A. NZArray : Array[StartIndex ... LastIndex];
      1. Address of NZArray[i] = (Address of NZArray[StartIndex]) + i - StartIndex
      2. NZArray[i] := 17;
         a) LDX i
         b) SBX# StartIndex
         c) LDA# 17
         d) STA+ NZArray
      3. Or NZArray[i] := 17;
         a) LDX i
         b) LDA# 17
c) STA+ NZArray - StartIndex
d) This is more efficient.

B. Special case when StartIndex is zero.
   1. Address of NZArray[i] = (Address of NZArray[0]) + i.
   2. Can use SOJ.

C. Must use AOC for loops with Start Index != 0

VII. Matrix Operations
A. Use [row, col]
B. Usually stored in row major order in which all of the elements in the leftmost dimension are located in consecutive memory locations.
C. Numbering usually starts at 1 rather than 0
D. Given a (Rows x Cols) matrix M, and starting indices of 1
   1. Address of M[i,j] = Address of M[1,1] + (j - 1 + ((i - 1) * Cols)) * w/e, where w/e = words per element.
E. Given a (Rows x Cols) matrix M, and starting indices of [r,c]
   1. Address of M[i,j] = Address of M[r,c] + (j - c + ((i - r) * Cols)) * w/e.
F. Matrix Print Programs
   1. Double For loop Pascal
      a) for i := r to Rows + r - 1 do begin
         (1) for j := c to Cols + c - 1 do
            (a) write( M[i, j], ' ');
            (2) writeln;
         b) end;
   2. Double For loop Assembly Language.
G. Redundant computations.

```assembly
.EQU    Rows, $2
.EQU    Cols, $3
.EQU    r, $0A
.EQU    c, $08

LDS#   $E00
LDA#   r;
STA    i;
FOR1:  CMA#   Rows + r - 1
JGT    END1
LDA#   c
STA    j
LDA    j
FOR2:  CMA#   Cols + c - 1
JGT    END2
LDA    i
SBA#   r
MUL#   Cols
ADA    j
SBA#   c
TAX
LDA+   M
JSR   $E07    ; PUT_NUM2
LDA#   '
JSR   $E08    ; PUTCHR
INC    j
LDA    j
JMP    FOR2
END2:  INC    i
JSR   $E06    ; PUTNL
LDA    i
```

JMP     FOR1
END1:   HLT

I:      .WORD 0
J:      .WORD 0
M:      .WORD 7
        .WORD 6
        .WORD 12
        .WORD 53
        .WORD 193
        .WORD 19