Final Remarks on Assemblers

Program Blocks:

- So far: Entire program treated as one unit; assembler produces one block of object code.
- **Blocks:** Segments of code that can be rearranged by the assembler.

Example: See Part I of Handout 5.1.

- Programmers may want to keep program and data segments together.
- May need to use 4-byte instructions.
- Instructions may be shortened by moving a block to the end.
- New assembler directive: `USE`

Example: See Part II of Handout 5.1.

- SIC/XE: Programs may have several *named* blocks and one default (unnamed) block.
- Assembler rearranges blocks so that the default block appears as a unit first; each named block follows.

Modifications to the Assembler:

- Maintain a separate location counter (LC) for each block. (LC value for each block starts at zero.)
- Save value of current LC when switching to another block; restore the LC value when switching back to the default block.
- Each entry of the Symbol Table has Symbol, Block name and LC value of the symbol within the block.
- At the end of Pass 1, length of each block is known. Assembler can assign a starting address for each block.
- Block Address Table (BAT): Each entry contains name of a block and its starting address.

- Suppose symbol X appears in block B, and the LC value of X within B is L. Then

  Address of X = Starting address of B + L

Control Sections:

- Modules that can be separately assembled; usually placed in separate files.
- Executable version of the program created by "linking" together all the separately assembled modules.
- New assembler directive: `CSECT`

Example:

```
MAIN    START   0       #Main prog.
          .
          .
        +JSUB   SORT
        +JSUB   MAX
          .
          .
```



```
SORT    CSECT           #Subroutine.
          .
          .
        RSUB
MAX     CSECT           #Subroutine.
          .
          .
        RSUB
```

Modifications to the Assembler:

- Assembler needs to know that `SORT` and `MAX` are in a *different* control section.
- New assembler directive: `EXTREF`

Example:

```
EXTREF  SORT,MAX
```

- SIC/XE statements referencing external symbols must use 4-byte format.
- Assembler cannot fill the address part of such instructions. (It will be done by the linker.)

- Assembler must produce an **External Reference Table** (ERT).
- Each entry of ERT has a symbol and the relative address where the address of the symbol is needed.
- Each control section may have an ERT.

Example: See Handout 5.2.

- New assembler directive: `EXTDEF`.

Example:

```
EXTDEF    SORT,MAX
```

- Assembler must produce an **External Definition Table** (EDT).
- Each entry of EDT has a symbol and the relative address where the symbol is defined.
- Each control section may have an EDT.

Example: See Handout 5.2.

Differences between ERT and EDT:

- A symbol may appear two or more times in an ERT. No symbol can appear more than once in an EDT.
- A symbol may appear in two or more ERTs. No symbol may appear in more than one EDT.

Algorithms for creating ERT and EDT:

– See Handout 5.3.

List of Tables used by an Assembler:

- Symbol Table (ST)
- Machine Opcode Table (MOT)
- Block Address Table (BAT)
- External Reference Table (ERT)
- External Definition Table (EDT)

Reading Assignment: Section 2.5 of [Beck].