Today’s Topic

- Beck’s Section 2.4: Assembler Design Options
  - One-pass assemblers
  - Multi-pass assemblers
  - Two-pass assembler with overlay structure
Two-Pass Assembler with Overlay Structure (1/3)

- For small memory
  - Pass 1 and pass 2 are never required at the same time
  - Three segments
    - Root: driver program and shared tables and subroutines
    - Pass 1
    - Pass 2
  - Tree structure
  - Overlay program

Two-Pass Assembler with Overlay Structure (2/3)

- Tree structure
Two-Pass Assembler with Overlay Structure (3/3)

- Overlay programs

One-Pass Assemblers (1/2)

- Main problem
  - Forward references
    - Data items
    - Labels on instructions
  
- Solution
  - Data items: require all such areas be defined before they are referenced
  - Labels on instructions: no good solution
One-Pass Assemblers (2/2)

- Two types of one-pass assembler
  - Load-and-go
    - Produces object code directly in memory for immediate execution
  - The other
    - Produces usual kind of object code for later execution

Load-and-go Assembler (1/3)

- Characteristics
  - Useful for program development and testing
  - Avoids the overhead of writing the object program out and reading it back
  - Both one-pass and two-pass assemblers can be designed as load-and-go.
  - However one-pass also avoids the overhead of an additional pass over the source program
  - For a load-and-go assembler, the actual address must be known at assembly time, we can use an absolute program
Load-and-go Assembler (2/3)

• Forward references handling
  1. Omit the address translation
  2. Insert the symbol into SYMTAB, and mark this symbol undefined
  3. The address that refers to the undefined symbol is added to a list of forward references associated with the symbol table entry
  4. When the definition for a symbol is encountered, the proper address for the symbol is then inserted into any instructions previous generated according to the forward reference list

Load-and-go Assembler (3/3)

• At the end of the program
  – Any SYMTAB entries that are still marked with * indicate undefined symbols
  – Search SYMTAB for the symbol named in the END statement and jump to this location to begin execution

• The actual starting address must be specified at assembly time
• Example
  – Figure 2.18, 2.19
Figure 2.18

Sample program for a one-pass assembler.

Figure 2.19(a)

Object code in memory and symbol table entries for the program in Fig. 2.18 after scanning line 40.
Producing Object Code

- When external working-storage devices are not available or too slow (for the intermediate file between the two passes
- Solution:
  - When definition of a symbol is encountered, the assembler must generate another Tex record with the correct operand address
  - The loader is used to complete forward references that could not be handled by the assembler
  - The object program records must be kept in their original order when they are presented to the loader
- Example: Figure 2.20
Figure 2.20

Object program from one-pass assembler for program in Fig. 2.18.

Multi-Pass Assemblers

- Restriction on EQU and ORG
  - No forward reference, since symbols’ value can’t be defined during the first pass
- Example
  - Use link list to keep track of whose value depend on an undefined symbol
- Figure 2.21
Figure 2.21 (1/5)

1. HALFSZ EQU MAXLEN/2
2. MAXLEN EQU BUFFER-BUFFER
3. PRIVMT EQU BUFFER-1
4. BUFFER RESB 4696
5. BUFEND EQU *

Figure 2.21 (2/5)

1. HALFSZ EQU MAXLEN/2
2. MAXLEN EQU BUFFER-BUFFER
3. PRIVMT EQU BUFFER-1
4. BUFFER RESB 4696
5. BUFEND EQU *

Figure 2.21 Example of multi-pass assembler operation.
Figure 2.21 (5/5)

1. HALFSZ EQU MAXLEN/2
2. MAXLEN EQU BUFEND-BUFFER
3. PRVBT EQU BUFFER-1

(a)

(f)

Figure 2.21 (cont'd)