Systems Programming

Chapter 2
Assemblers – part 2

Homework #2

(Due: Mar 24th& & 31st)

- Design and implement a SIC assembler simulator by following the software engineering and procedural / object-oriented design principles given in Chapter 8.
- Part 1: Due Mar 24th
  - Submit a document that includes
    - a system specifications document
    - object-oriented system design
    - module interfaces
    - system testing strategy.
- Part 2: Due Mar 31st
  - Implement your assembler in C++ using an object-oriented approach. Submit your full codes and a document that relates your actual work to the Part 1.

Outline

- Machine-independent features
  - handling expressions
  - program blocks
  - control sections and program linking
- Assembler design options
  - one-pass assemblers
  - multi-pass assemblers

Modification record (revised):

	Col 1	M
	Col. 2-7	Starting address of the field to be modified, relative to the beginning of the control section
	Col. 8-9	Length of the field to be modified, in half bytes.
New!	Col 10	Modification flag (+ or -)
New!	Col 11-16	External symbol whose value is to be added to or subtracted from the indicated field

See Fig. 2.17

Linking up external references

```
15  CLOOP  +JSUB    RDREC    4B100000
```

relative address 0004

M00000405+RDREC

length of the field to be modified, in half-bytes

```
190 MAXLEN   WORD   BUFEND–BUFFER   000000
```

relative address 0028

M00002806+BUFEND

M00002806-BUFFER

Doing program relocation

- Exactly the same mechanism can be used for program relocation and for program linking.
  - the modification required is adding the beginning address of the control section to certain fields in the object program.
  - The symbol used as the name of the control section has as its value the required address – control section name is automatically an external symbol.

M00000705 → M00000705+COPY

M00001405 → M00001405+COPY

M00002705 → M00002705+COPY

Handling of expressions in the existence of multiple CSs

- Def: All of the relative terms in an expr. must be paired (absolute), or that all except one must be paired (relative.)
- + Both terms in each pair must be relative within the same CS (Control Section.)
  - BUFEND–BUFFER vs. RDREC-COPY

When an expr. involves external refs, assembler cannot in general determine whether or not the expr. is legal. **Why?**

- It forms an initial expr. value, and leaves the rest to the loader.

7

Assembler Design Options

- Two-Pass Assemblers (we have covered)
- One-Pass Assemblers
- Multi-Pass Assemblers

8

One-Pass Assemblers

- Main problem: forward references
  - Instruction operands often are symbols that have not yet been defined in the source program.
- Is elimination possible?

usually applied to reduce complexity

- eliminate forward refs to data items: all should be defined before they are ref'd.
  - EASY! ACCEPTABLE!
- BUT cannot eliminate forward refs to labels on instrs
  - NOT easy. We need forward jumps frequently!

9

Two types of one-pass assemblers:

1. Load-and-go assembler: produces object code directly in memory for immediate execution.
   - no object program is written out
   - no loader needed
   - useful in heavy development/testing envs.
     - programs are re-assembled nearly every time they are run.
   - often used on systems where external working-storage devices are not available or when external storage is slow or inconvenient to use

10

Handling of forward refs in load-and-go assemblers

- less difficult since object program is produced in memory
  - assembler generates object code as it scans the program
  - if an instr operand is a not-yet-defined symbol,
    - operand address is omitted
    - symbol is entered into symbol table
    - address of operand field is added to a list of forward refs associated with the symbol table entry
  - when the definition is encountered, the forward ref list for that symbol is scanned, and the proper address is inserted into any instrs previously generated.

See Fig. 2.18 → 2.19 (a) & (b)

11

Two types of one-pass assemblers:

2. Type 2: produces usual kind of object program for later execution
   - forward refs are entered into lists as before
     - However when the def of a symbol is encountered, intrs that made forward refs to that symbol may no longer available in memory for modification
       - they have been written out as part of a Text record
     - Assembler must generate another Text record with the correct operand address
       - this is inserted into the correct place by the loader

See Fig. 2.20

12

Multi-Pass Assemblers

- Remember: any symbol used on the right-hand side be defined previously in the source program, e.g., EQU
  - The reason for this is the symbol definition process in a two-pass assembler.

```
ALPHA  EQU    BETA
BETA   EQU    DELTA
DELTA  RESQ   1
```

A 2-Pass assembler cannot resolve such sequence of defs.

Forward refs are even difficult even for human reader! We can prohibit them.

13

- But the general solution is multi-pass assemblers:
  - can make as many passes as ar needed to process the definitions of symbols
    - does not have to do > 2 passes over the entire program – Pass 1 saves portions that involve forward refs, Pass 2 re-scans them
- Store symbol defs that involve forward-refs in the symbol table
  - table also indicates which symbols are dependent on the values of others
  - See **Fig.2.21**

14