Assembler Design Options

- One-pass assemblers
- Multi-pass assemblers
- Two-pass assembler with overlay structure

Two-Pass Assembler with Overlay Structure

- 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

One-Pass Assemblers

- 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

- Main Problem
  - forward reference
    - data items
    - labels on instructions
- 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

- 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 over head 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

Forward Reference in One-pass Assembler

- For any symbol that has not yet been defined
  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 (Cont.)

- 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

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

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

Implementation Examples

- Microsoft MASM Assembler
- Sun Sparc Assembler
- IBM AIX Assembler

Microsoft MASM Assembler

- SEGMENT
  - » a collection segments, each segment is defined as belonging to a particular class, CODE, DATA, CONST, STACK
  - » registers: CS (code), SS (stack), DS (data), ES, FS, GS
  - » similar to program blocks in SIC
- ASSUME
  - » e.g. ASSUME ES:DATASEG2
  - » e.g. MOVE AX, DATASEG2

    MOVE ES,AX
  - » similar to BASE in SIC

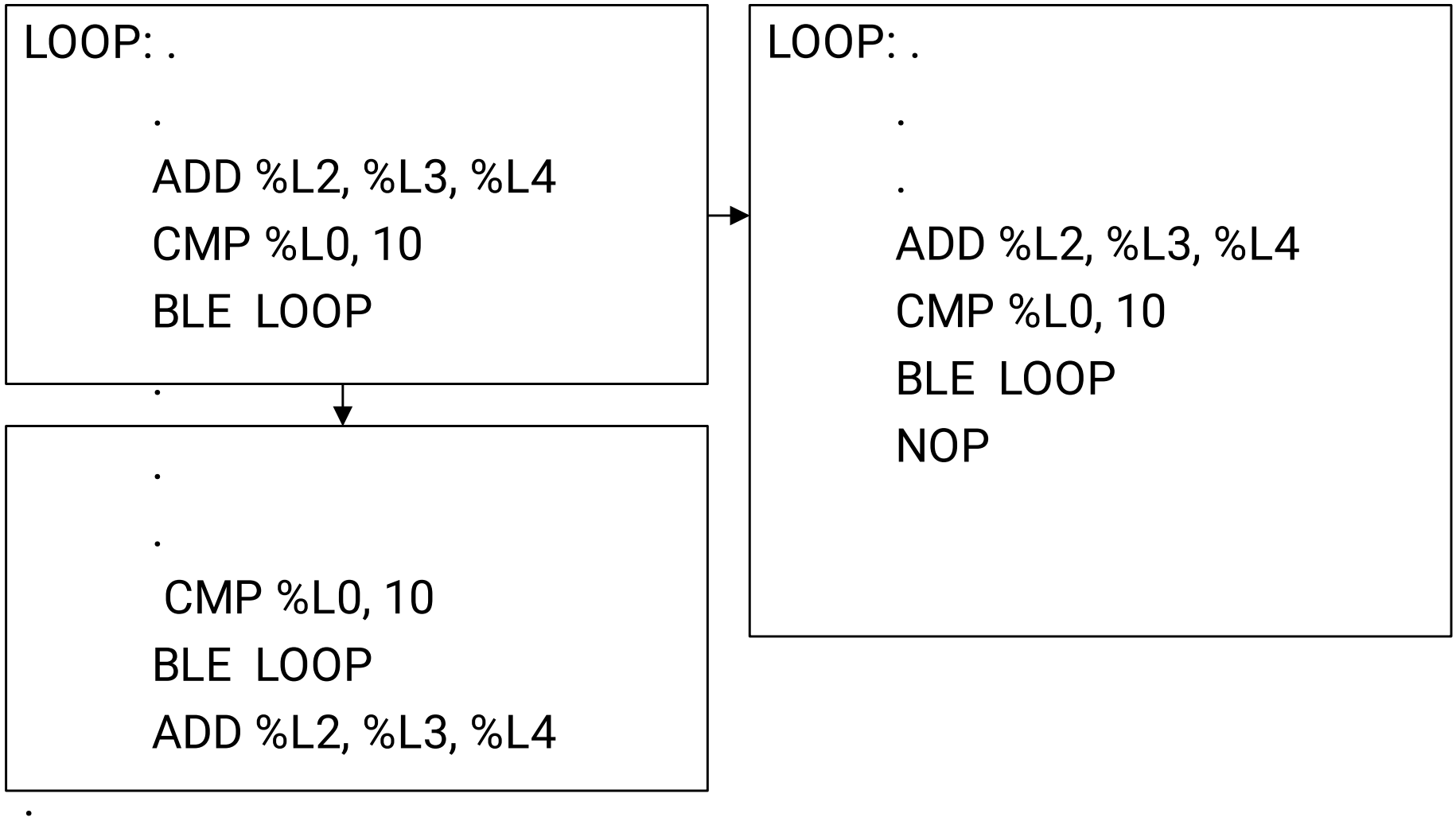
Microsoft MASM Assembler

- JUMP with forward reference
  - near jump: 2 or 3 bytes
  - far jump: 5 bytes
  - e.g. JMP TARGET
  - Warning: JMP FAR PTR TARGET
  - Warning: JMP SHORT TARGET
  - Pass 1: reserves 3 bytes for jump instruction
  - phase error
- PUBLIC, EXTRN
  - similar to EXTDEF, EXTREF in SIC

Sun Sparc Assembler

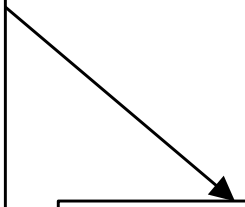
- Sections
  - .TEXT, .DATA, .RODATA, .BSS
- Symbols
  - global vs. weak
  - similar to the combination of EXTDEF and EXTREF in SIC
- Delayed branches
  - delayed slots
  - annulled branch instruction

Sun Sparc Assembler



Sun Sparc Assembler

```
LOOP: ADD    %L2, %L3, %L4
          .
          .
          CMP    %L0, 10
          BLE,A  LOOP
          .
```



```
LOOP: .
          .
          .
          CMP    %L0, 10
          BLE,A  LOOP
          ADD     %L2, %L3, %L4
```

AIX Assembler for PowerPC

- Similar to System/370
- Base relative addressing
  - » save instruction space, no absolute address
  - » base register table:
    - – general purpose registers can be used as base register
  - » easy for program relocation
    - – only data whose values are to be actual address needs to be modified
  - » e.g. USING LENGTH, 1
  - » USING BUFFER, 4
  - » Similar to BASE in SIC
  - » DROP

AIX Assembler for PowerPC

- Alignment
  - » instruction (2)
  - » data: halfword operand (2), fullword operand (4)
  - » Slack bytes
- .CSECT
  - » control sections: RO(read-only data), RW(read-write data), PR(executable instructions), BS(uninitialized read/write data)
  - » dummy section