University Institute of Engineering

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Bachelor of Engineering

Subject Name: System Programming

Subject Code: CST-315

Compilers

Chapter-1.2
Assembler

- Variants of Assemblers
- Design of two pass assembler

Variants of Assemblers

- There is a list of assemblers ,computer programs that translate assembly language source code into binary programs. Some assemblers are components of a compiler system for a high level language and may have limited or no usable functionality outside of the compiler system.
- - Some assemblers are hosted on the target processor and operating system, while other assemblers (cross-assemblers) may run under an unrelated operating system or processor.

Variants of Assemblers

- **As part of a compiler suite**
- **GNU Assembler** (gas): GPL: many target instruction sets including ARM architecture, Atmel AVR, x86, x86-64, Freescale 68HC11, Freescale v4e, Motorola 680x0, MIPS, PowerPC, IBM System z, TI MSP430.
- **ASxxxx Cross Assembler** (part of the Small Device C Compiler project): GPL: several target instruction sets including Intel 8051, Freescale 68HC08, PIC microcontroller.
- **The Amsterdam Compiler Kit (ACK**) targets many architectures of the 1980s, including 6502, 6800, 680x0, ARM, x86 and Z8000.
- LLVM targets many platforms, however emits no per-target assembly language, instead more high-level typed intermediate representation assembly-like language used.
- Some others self-hosted native-targeted language implementations (like Go, Free Pascal, SBCL) have their own assemblers with multiple targets. They may be used for inline assembly inside language, or even included as a library, but not always suitable for standalone application - no command-line tool exists, or only intermediate representation used as a source, or support for targets very limited.

Single Target assembler

An assembler may have a single target processor or may have options to support multiple processor types. Very simple assemblers may lack features, such as macros, present in more powerful versions.

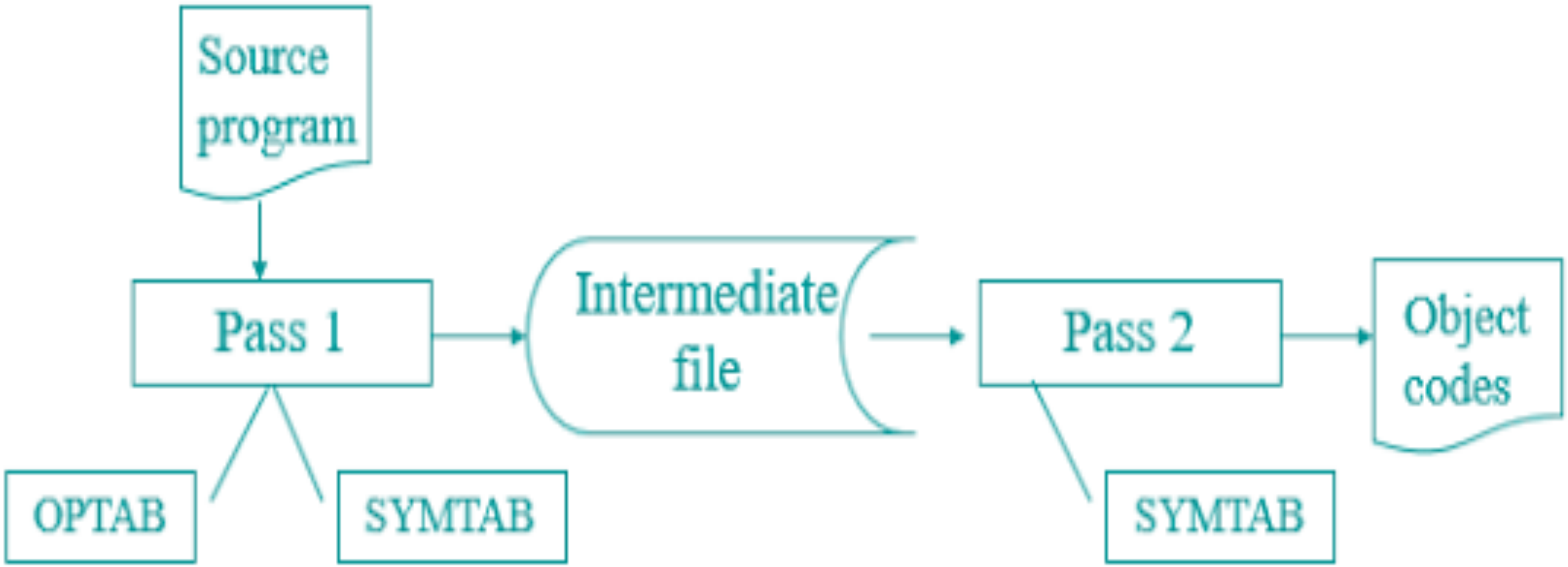
- Various types are:
- 6502 assemblers
- 680x0 assemblers
- ARM assemblers
- Mainframe Assemblers
- POWER, PowerPC, and Power ISA assemblers
- x86 assemblers
- Z80 assemblers
- Other single target assemblers

Design of two pass Assembler

One-pass assembler cannot resolve forward references of data symbols. It requires all data symbols to be defined prior to being used. A two-pass assembler solves this dilemma by devoting one pass to exclusively resolve all (data/label) forward references and then generate object code with no hassles in the next pass. If a data symbol depends on another and this another depends on yet another, the assembler resolved this recursively.

- **Two Pass Assembler**
- *Read from input line*
    - LABEL, OPCODE, OPERAND

Design of 2 pass Assembler



Design of 2 – Pass Assembler

PASS 1*:*

- Separate the Symbol, Mnemonic opcode, and operand fields
- Build the symbol table
- Perform LC Processing
- Construct Intermediate Representation
- **PASS 2:**
- *SYNTHESIZE THE TARGET PROGRAM*
- Advanced Assembler Directives
- ORIGIN
- EQU

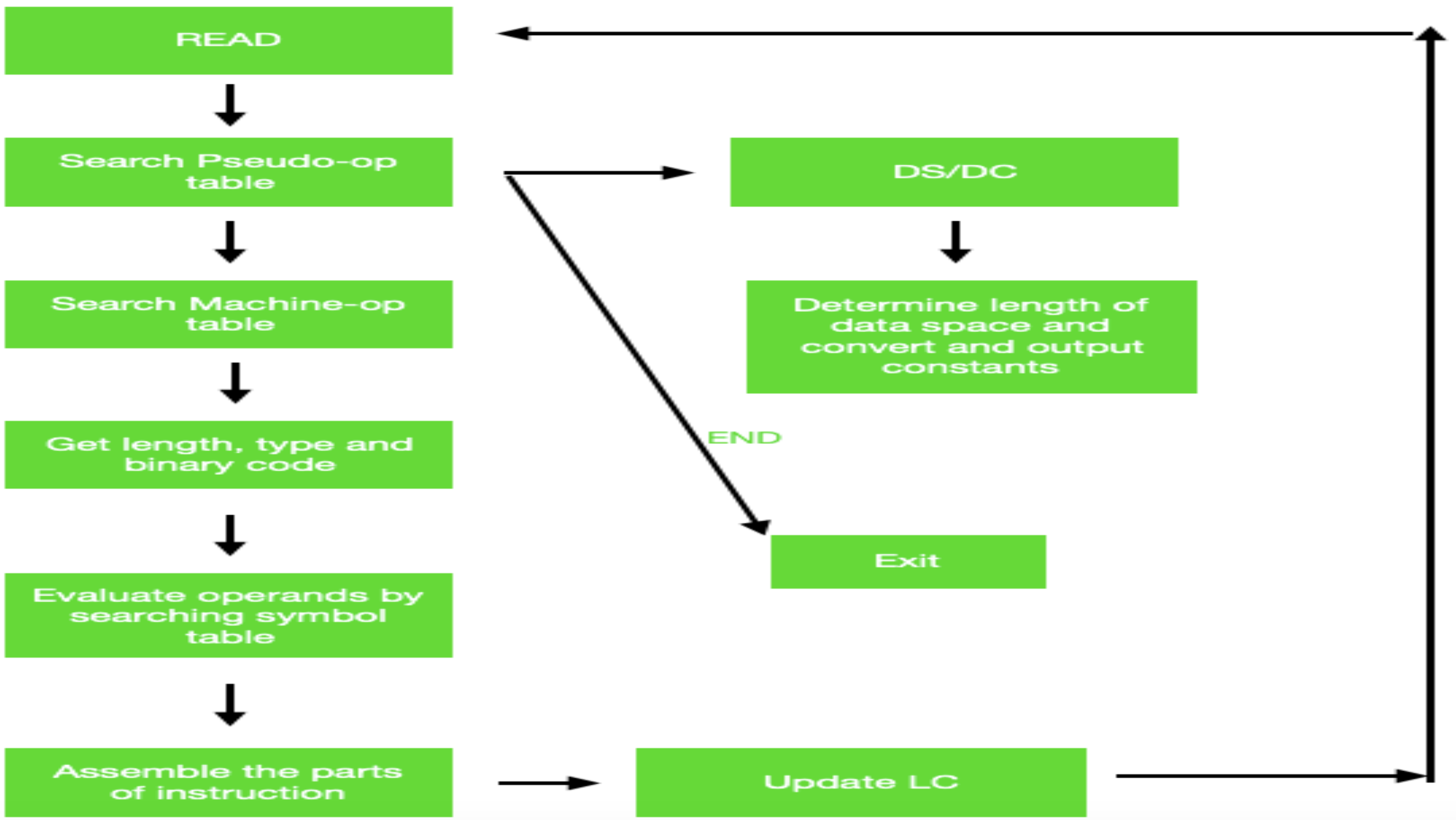
Design of 2 – Pass Assembler

- **EQU**Syntax:
  <Symbol> EQU <Address Specification>E.g. MAXLEN EQU 4096 Pass I of Assembler
- Pass I Use following Data Structures
  - OPTAB
  - SYMTAB
  - LITTAB
  - POOLTAB

Design of 2 – Pass Assembler

- 2-pass system is to address the problem of forwarding references — references to variables or subroutines that have not yet been encountered when parsing the source code. A strict 1-pass scanner cannot assemble source code which contains forward references. Pass 1 of the assembler scans the source, determining the size and address of all data and instructions; then pass 2 scans the source again, outputting the binary object code.

Design of 2 – Pass Assembler


READ
Search Pseudo-op table
Search Machine-op table
Get length, type and binary code
Evaluate operands by searching symbol table
Assemble the parts of instruction
DS/DC
Determine length of data space and convert and output constants
END
Exit
Update LC

Working of Pass-2

- Pass-2 of assembler generates machine code by converting symbolic machine-opcodes into their respective bit configuration(machine understandable form). It stores all machine-opcodes in MOT table (op-code table) with symbolic code, their length and their bit configuration. It will also process pseudo-ops and will store them in POT table(pseudo-op table).
- Various Data bases required by pass-2:

1. MOT table(machine opcode table)
2. POT table(pseudo opcode table)
3. Base table(storing value of base register)
4. LC (location counter) Take a look at flowchart to understand:

References

- [PDF] Systems Programming and Operating Systems by Dhamdhere - Free Download PDF (dlscrib.com)
- [PDF] Principles of Compiler Design By Alfred V. Aho & J.D.Ullman Free Download – Learnengineering.in

THANK YOU