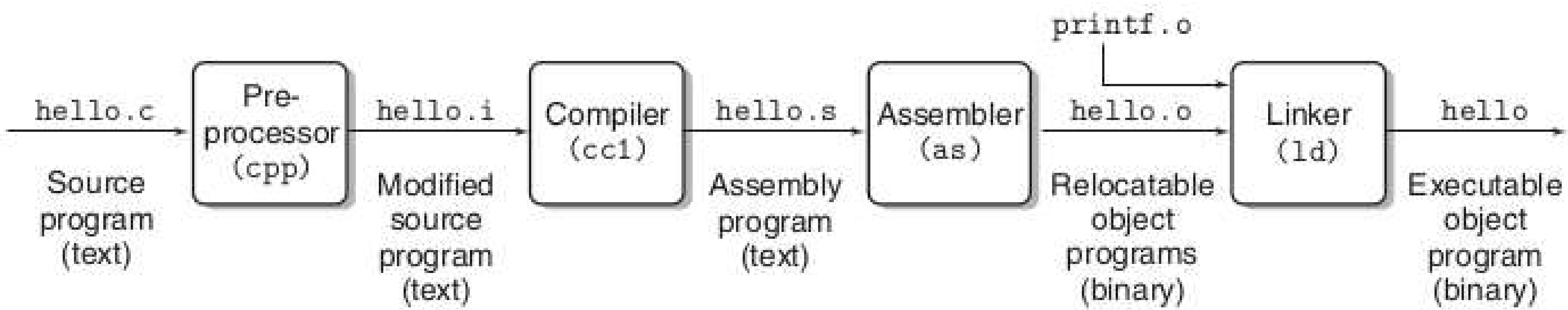
Computer Organization and Assembly Language

Fall 2023

Dr Jameel Malik
muhammad.jameel@seecs.edu.pk
Office: A-311 SEECS

The Compilation System

- Pre-processor removes comments and expands the header files.
- Compiler converts the C/C++ code into assembly language.
- Assembler converts the assembly code into the machine code.
- Linker will link the machine code with associated libraries to generate the executable file.



Translating Languages

English: Display the sum of A times B plus C.

C++: cout << (A * B + C);

Assembly Language:

```
mov eax,A
mul B
add eax,C
call WriteInt
```

Intel Machine Language:

```
A1 00000000
F7 25 00000004
03 05 00000008
E8 00500000
```

Assembly Language(AL)

- Popular Assemblers
  - MASM (Microsoft Assembler)
  - NASM (Netwide Assembler)

Assembly Language Applications

- Direct hardware manipulation i.e., device drivers
  - A software program that controls a specific hardware device attached to a computer

- Embedded systems
  - A system that is designed to perform a single task e.g. monitoring a temperature of a device

Basic Elements of Assembly Language

- Classes of Instructions
- Reserved words and identifiers
- Directives and instructions
- Labels
- Mnemonics and Operands
- Comments

Classes of Instructions

- Data Transfer
  - Ex: Load (LD) , Store (ST)
- ALU
  - Ex: ADD, SUB, AND, OR
- Control Flow
  - Ex: jump, branch
- Floating point
  - Ex: ADD.D , SUB.S
- Multimedia
  - Ex: ADD.PS , SUB.PS
- String
  - Ex: MOVSB

Basic Elements of Assembly Language

- Reserved words
  - Reserved words cannot be used as identifiers
  - Instruction mnemonics, directives, predefined symbols
  - Examples: BYTE, MOV, ADD

Basic Elements of Assembly Language

- Directives
  - Used to declare code, data areas, select memory model, declare procedures, etc.
  - not case sensitive
  - Example: .data,   .code

- Different assemblers have different directives

Basic Elements of Assembly Language

- Instructions
  - Assembled into machine code by assembler
  - Executed at runtime by the CPU
  - An instruction contains:
    - Label (optional)
    - Mnemonic (required)
    - Operand (depends on the instruction)
    - Comment (optional)

Basic Elements of Assembly Language

- **Labels**
  - Follow identifier rules
  - Data label
    - must be unique
    - example: **myArray** (not followed by colon)
  - Code label
    - target of jump and loop instructions
    - example: **L1:** (followed by colon)

Basic Elements of Assembly Language

- Instruction Mnemonics
  - Identify the operation carried out by an instruction
  - examples: MOV, ADD, SUB, MUL, INC, DEC
- Operands
  - constant
  - register
  - memory (data label)
  - Example: MOV count, bx : INC ax

Basic Elements of Assembly Language

- Comments
  - Single-line comments
    - begin with semicolon (;)
    - Example: ; my code goes here
  - Multi-line comments
    - begin with COMMENT directive and a programmer-chosen character
    - Example: COMMENT &

      These lines are commented

      &

An Assembly Program Template

```
; Program Description:
; Author:
; Creation Date:
; Revisions:
; Date:                Modified by:

INCLUDE Irvine32.inc
.data
   ; (insert variables here)
.code
main PROC
   ; (insert executable instructions here)
   exit
main ENDP
   ; (insert additional procedures here)
END main
```

Basic Elements of Assembly Language

- Basic Elements of Assembly Language
- Defining Data

Data Types

- BYTE, SBYTE
  - 8-bit unsigned integer; 8-bit signed integer
- WORD, SWORD
  - 16-bit unsigned & signed integer
- DWORD, SDWORD
  - 32-bit unsigned & signed integer
- QWORD
  - 64-bit integer
- TBYTE
  - 80-bit integer

Defining BYTE and SBYTE Data

```
value1 BYTE 'A'              ; character constant
value2 BYTE 0                ; smallest unsigned byte
value3 BYTE 255              ; largest unsigned byte
value4 SBYTE -128            ; smallest signed byte
value5 SBYTE +127            ; largest signed byte
value6 BYTE ?                ; uninitialized byte
```

Defining Byte Arrays

```
list1 BYTE 10,20,30,40
list2 BYTE 10,20,30,40
      BYTE 50,60,70,80
      BYTE 81,82,83,84
list3 BYTE ?,32,41h,00100010b
list4 BYTE 0Ah,20h,‘A’,22h
```

Defining Strings

```
str1 BYTE "Enter your name",0
str2 BYTE 'Error: halting program',0
str3 BYTE 'A','E','I','O','U'
greeting  BYTE "Welcome to the Encryption Demo program "
          BYTE "created by Kip Irvine.",0
```

Defining Strings

```
menu BYTE "Checking Account",0dh,0ah,
    "1. Create a new account",0dh,0ah,
    "2. Open an existing account",0dh,0ah,
    "3. Credit the account",0dh,0ah,
    "4. Debit the account",0dh,0ah,
    "5. Exit",0ah,0ah,
    "Choice> ",0
```

Defining Strings

```
menu BYTE "Checking Account",0dh,0ah,
    "1. Create a new account",0dh,0ah,
    "2. Open an existing account",0dh,0ah,
    "3. Credit the account",0dh,0ah,
    "4. Debit the account",0dh,0ah,
    "5. Exit",0ah,0ah,
    "Choice> ",0
```

End-of-line character sequence:

- 0dh = carriage return (return to the beginning of current line)
- 0ah = line feed (move to next output line)

Using the DUP Operator (Duplicate)

```
var1 BYTE 20 DUP(0)          ; 20 bytes, all equal to zero
var2 BYTE 20 DUP(?)          ; 20 bytes, uninitialized
var3 BYTE 4 DUP("STACK")     ; 20 bytes: "STACKSTACKSTACKSTACK”
```

Defining WORD and SWORD Data

```
word1   WORD   65535          ; largest unsigned value
word2   SWORD -32768          ; smallest signed value
word3   WORD   ?              ; uninitialized, unsigned
word4   WORD   "AB"           ; double characters
myList WORD   1,2,3,4,5       ; array of words
array   WORD   5 DUP(?)       ; uninitialized array
```