# ASSEMBLERS AND ASSEMBLY LANGUAGE

- Creating machine code programs is difficult
- Hard to remember numbers for instructions
- Calculating addresses for jumps

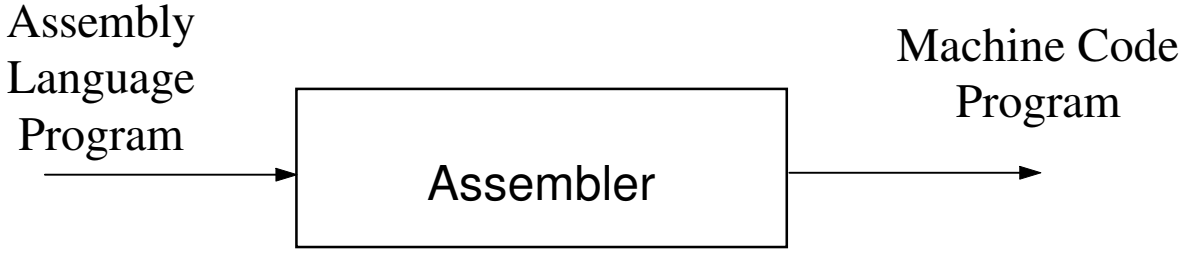
.. assembly language was developed to overcome these problems

**Assembly Language is**

- Symbolic
  Operations and operands can be written using words

- Low-Level
  Each assembler instruction corresponds to one machine code instruction

- Translated
  By an assembler, into machine code instructions



# ANATOMY OF AN ASSEMBLY LANGUAGE PROGRAM

Written as Text, with 4 fields

**Label Operation Operands Comments**

```
; Program to calculate sum of set of numbers
; start address in BX
; end address in DX
; result stored in AX

      SUB AX,AX     ;Set AX to 0
LOOP: ADD AX,[BX]   ;add data to AX
      ADD BX,2      ;Add 2 to BX (--> next word)
      CMP DX,BX     ;Compare DX to BX
      JNS LOOP      ;repeat if DX >= BX
```

# FIELDS OF ASSEMBLY PROGRAM

- Label
  Allows branching/jumps to be specified without having to know the actual address

- Operation
  Specifies instruction type without having to know the instruction code

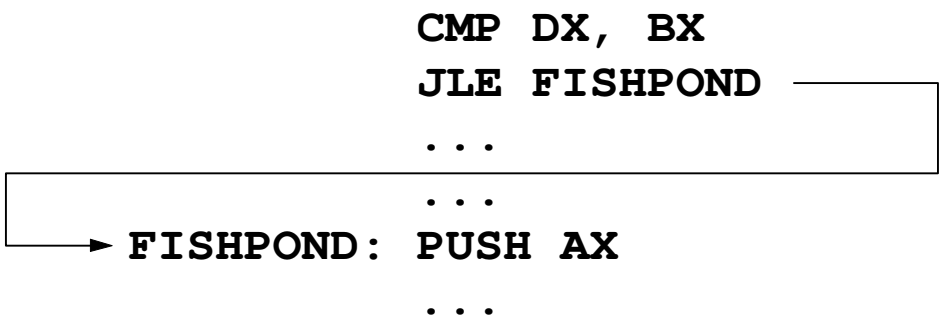
  * Machine Instructions - from the CPU's instruction set
  * Pseudo-instructions (pseudo-ops) - commands to the assembler

- Operands
  On which the operation operates

- Comments
  Allow humans to understand the program
  Comments make programs maintainable

# TWO-PASS ASSEMBLY

- Label may be used before it is defined
- Address which label refers to is unknown




- Forwards references are a problem
- "Two-Pass" Assemblers read input file twice to overcome this problem

# TWO-PASS ASSEMBLY

Two main data structures required

## 1. Opcode Table

- Contains entry for each valid combination of mnemonic and operand(s)

- Used to translate mnemonic into machine code

| *Opcode* | *Operand1* | *Operand2* | *m/c* | *length* |
|---|---|---|---|---|
| ADD | *reg* | *reg* | 1B | 2 |
| ADD | *mem* | *reg* | 1C | 3 |
| PUSH | - | - | 2F | 1 |
| ... | ... | ... | ... | ... |

- This is a “look-up table” - fast way of translating

# TWO-PASS ASSEMBLY

## 2. Symbol Table

- Holds values of symbols as they are defined

| *Symbol* | *Value* |
|---|---|
| FISHPOND | 1C00 |
| LOOP | - |
| TEMP | 2FF0 |

- Two ways of defining symbols
  - Label - specifies a jump address
  - Explicit definition using pseudo-op

  *eg.* “equate” statement

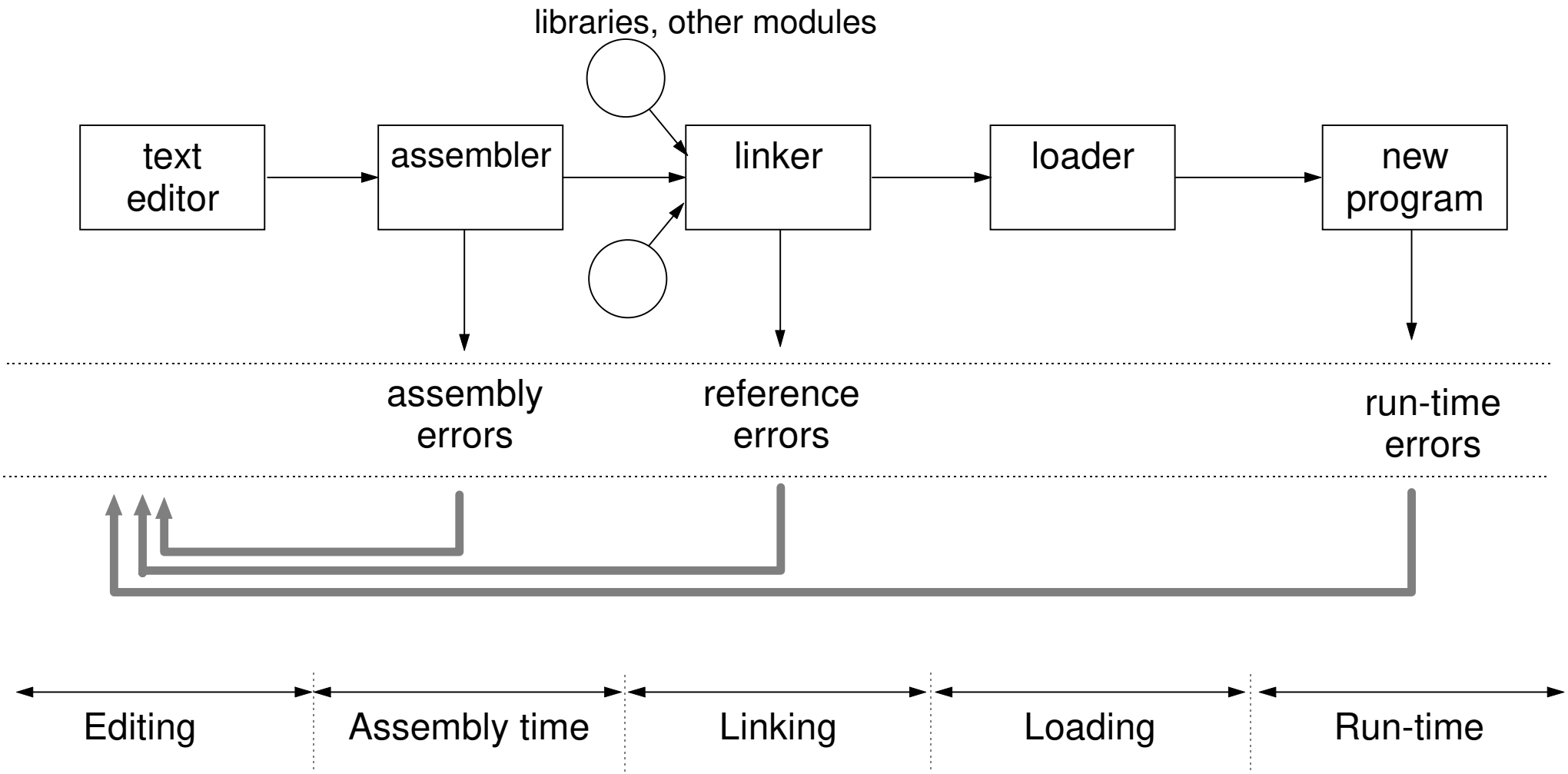
```
TEMP EQU  2FF0H
```

# TWO-PASS ASSEMBLY

## First Pass

- Main function is to build the Symbol Table
    - Scan input file
    - identify labels and pseudo-ops
    - store <label,value> pairs in Symbol Table

## Second Pass

- Generate the machine code output
    - Scan input file
    - Look-up mnemonics in Opcode Table
    - Look-up symbols in symbol table as required
    - Generate machine code output
    - Generate reports, eg. assembly listing

# ASSEMBLY PROGRAM DEVELOPMENT