# Assembly Language Programming

- Aims of this Module:
  - To introduce the usage of assembler and to begin assembly language programming
- Contents:
  - Types of Assemblers
  - Assembly Process
  - Assembly Instruction Format
  - Basic Assembler Directives
  - Using a Simulator to Run Assembler Programs

# Machine & Assembly Language

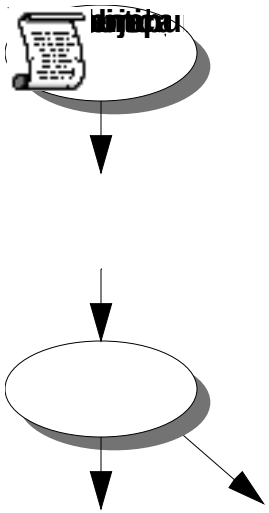
- Machine language instruction:
  - Binary number for processor consumption
  - Extremely hard to read or write even for very simple programs
- Assembly language instruction:
  - Mnemonic (easy to remember code) representing machine language
- Solution:
  - Programmer uses assembly language
  - Processor uses machine language
  - Use assembler to translate from assembly to machine
- Assembly language is a form of the native language of a computer in which
  - machine code instructions are represented by mnemonics
    - e.g., MOVE, ADD, SUB
  - addresses and constants are usually written in symbolic form
    - e.g., NEXT, BACK_SP

# Assemblers

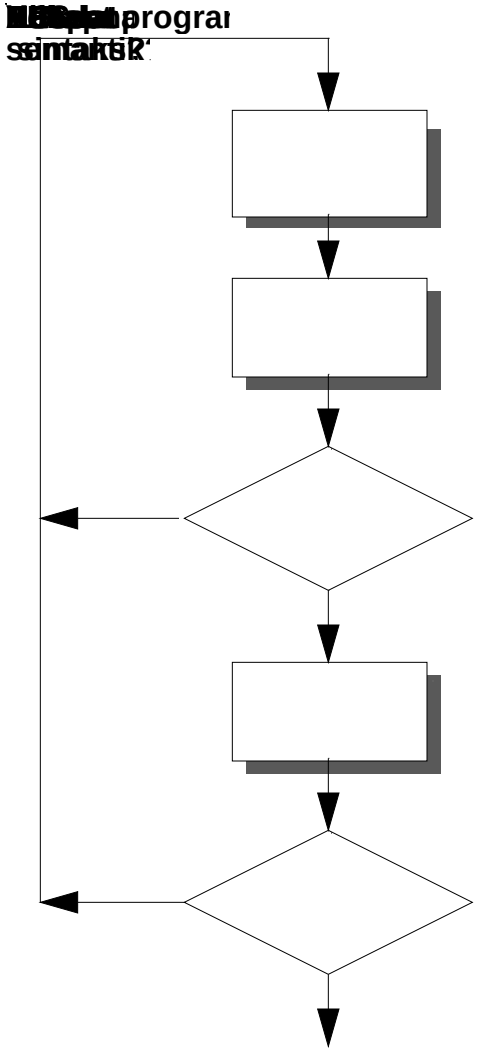
- **Assembler** — software that translates from assembly language to machine language
- Source program / **source code** — program written by humans, as input to the assembler
- Object program / **object code** — machine language program generated by the assembler
- **Cross Assembler** — assembler that generates machine code for a different processor
  - Example: ASM68K generates code for Motorola processor but runs on PC with Intel processor
- **Integrated Development Environment (IDE)** — all-in-one package that contains editor, assembler and simulator

# Assembly Process

*Aliran Data*



*Aliran Kerja*



# Files Created by Assembler


- Binary file or object file is recognized by machine.
- Listing file contains the information of program assembling.
- If a program written in more than one files, LINKER is needed to link the object files together before execution.

# Source File Example

```
*    ATURCARA BAGI MENGIRA JUMLAH 10 KATA
                        ORG     $1000
                        CLR.W   D0              ;JUMLAH=0
                        MOVEQ   #10,D1          ;PEMBILANG=JUMLAH UNSUR
                        LEA     DATA,A0         ;PENUNJUK=UNSUR
                                                PERTAMA
                ULANG   ADD.W   (A0)+,D0
                        SUBQ    #1,D1
                        BNE     ULANG
                        MOVE.W  D0,JUMLAH
                        MOVE.B  #227,D7
                        TRAP    #14

*   TATASUSUNAN DATA DI SINI
                        ORG     $1020
                DATA    DC.W    13,17,14,68,-3,20,85,30,1,19
                JUMLAH  DS.W    1
                        END
```

# Listing File Example

```
00001000 Starting Address
Assembler used: EASy68K Editor/Assembler V1.1 April 19, 2005
Assembled on: 23-Jul-07 9:27:15 PM

                                    1          *ATURCARA BAGI MENGIRA JUMLAH 10 KATA
00001000                            2          ORG       $1000
00001000  4240                      3          CLR.W     D0              ;   JUMLAH=0
00001002  720A                      4          MOVEQ     #10,D1     ;   PEMBILANG=JUMLAH UNSUR
00001004  41F9 0000101C       5     LEA       TATA,A0    ;   PENUNJUK=UNSUR PERTAMA
0000100A  D058                      6 ULANG    ADD.W     (A0)+,D0
0000100C  5341                      7          SUBQ      #1,D1
0000100E  66FA                      8          BNE       ULANG
00001010  33C1 00001030       9     MOVE.W    D0,JUMLAH
00001016  1E3C 00E3                 10         MOVE.B    #227,D7
0000101A  4E4E                      11         TRAP      #14
                                    12         * TATASUSUNAN DATA DI SINI
00001020                            13         ORG       $1020
00001020  000D0011000E        14 TATA  DC.W      13,17,14,68,-3,20,85,30,1,19
00001034                            15 JUMLAH DS.W    1
00001036                            16                END

No errors detected
No warnings generated

SYMBOL TABLE INFORMATION
. . .
```

# Listing File with Errors

```
00001000 Starting Address
Assembler used: EASy68K Editor/Assembler V1.1 April 19, 2005
Assembled on: 23-Jul-07 9:27:15 PM

                                                    1           * ATURCARA MENUNJUKKAN RALAT
                                                    2           *
00000400                                            3           ORG     $400
Line    4: Error in expression: label 'DATB' not defined
00000400                                            4           MOVE    DATB,D5
Line    5: Illegal operand(s)
                                                    5           ADD     NEXT,D8
Line    6: Unknown opcode
                                                    6           MOV     D5,HASIL
00000406  5345                                      7           SUBQ    #1,D5
00000408  60FE                                      8           BRA     *
00000500                                            9           ORG     $500
00000500  1234                                      10        DATA    DC      $1234
00000502  ABCD                                      11        LAGI    DC      $ABCD
00000504                                            12        HASIL   DS      1
00000506                                            13        END

3 error(s) detected.
```

# Object File Example

- Object file is also known as S-Record file because each line (record) starts with the letter S.
- Contains memory values in hex format.

```
S0030000FC
S2140010004240720A41F900001020D058534166FA57
S21000101033C1000010341E3C00E34E4EBE
S2140010200000D0011000E0044FFFD00140055001EC8
S208001030000010013A3
S804000000FB
```

- Details of the S-Record format can be found at http://www.cs.net/lucid/moto.htm

# How to Edit

- Be a hacker!
  - Go with MS-DOS and use EDIT command
  - Faster if you can touch-type
  - Command-line assembler also uses MS-DOS
- Be a wimp (windows icon mice pointer)
  - Go with Windows default and use NotePad - not recommended
  - Alternatively, get a programmer's editor like Emacs or SCiTE
- Use an IDE
  - Integrated Development Environment
  - All-in-one software with editor, assembler and simulator
  - Examples are IDE68k and EASy68k

# How to Assemble

- Assemblers are included in IDE
- If you use the DOS window, you can use DOS assemblers
  - ASM68K & A68K are free DOS assemblers
  - Good enough for us
- Paid products have some advantages
  - can optimize the code
  - assembler faster
  - have “macro” features
  - Support really large programs
  - One example of commercial assembler is XASM68K

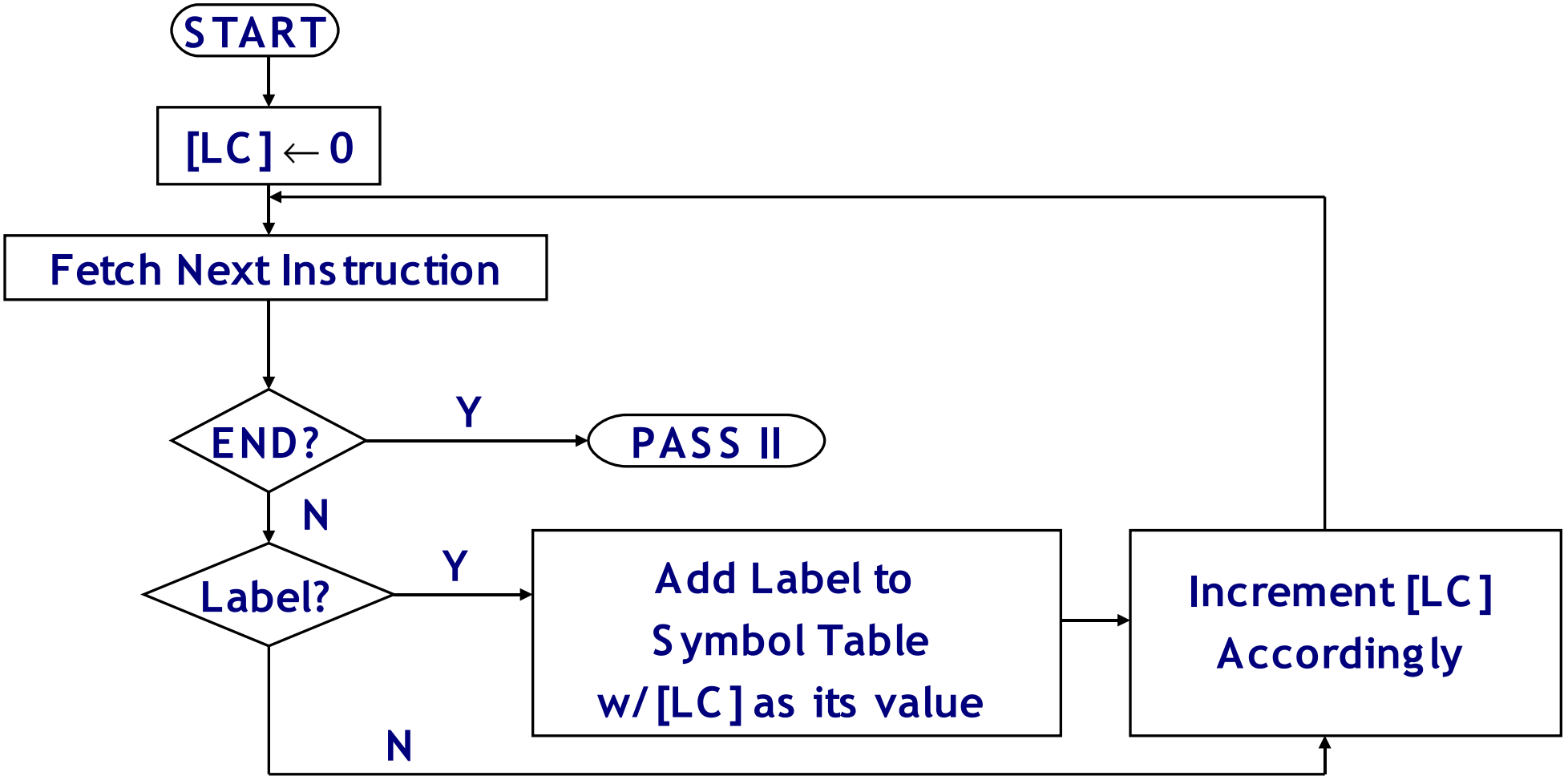
# How Assembler Works

- The assembler is responsible for translating the assembly language program into machine code
- The translation process is essentially one of reading each instruction and looking up its equivalent machine code value
- LC: Assembler's simulation of PC
  - When an assembly program is assembled, LC is used to keep track of the "memory location" at which an instruction would be should that instruction be executed.
  - So that machine code can be generated correctly from assembly code.
- As labels or variable names are encountered, addresses must be filled in, branch offsets calculated etc
- Labels in assembly language can be used before they are defined

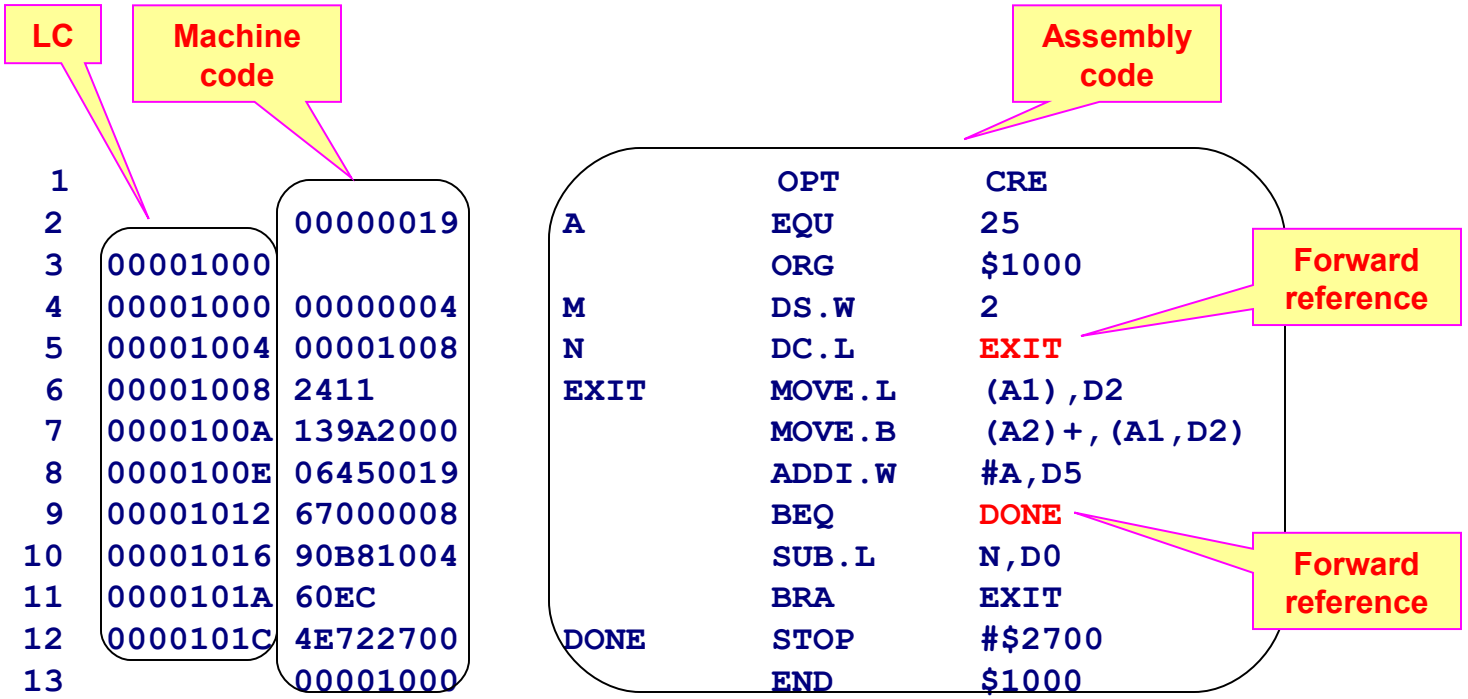
# Two-Pass Assembler

- When a forward reference is encountered, the assembler does not know what value to replace it with
- This is solved by reading the source code twice — the **two-pass assembler**
- Pass I:
  - Search source program for symbol definitions and enter these into symbol table.
- Pass II:
  - Use symbol table constructed in Pass I and op-code table to generate machine code equivalent to source

# Pass I



# Symbol Table



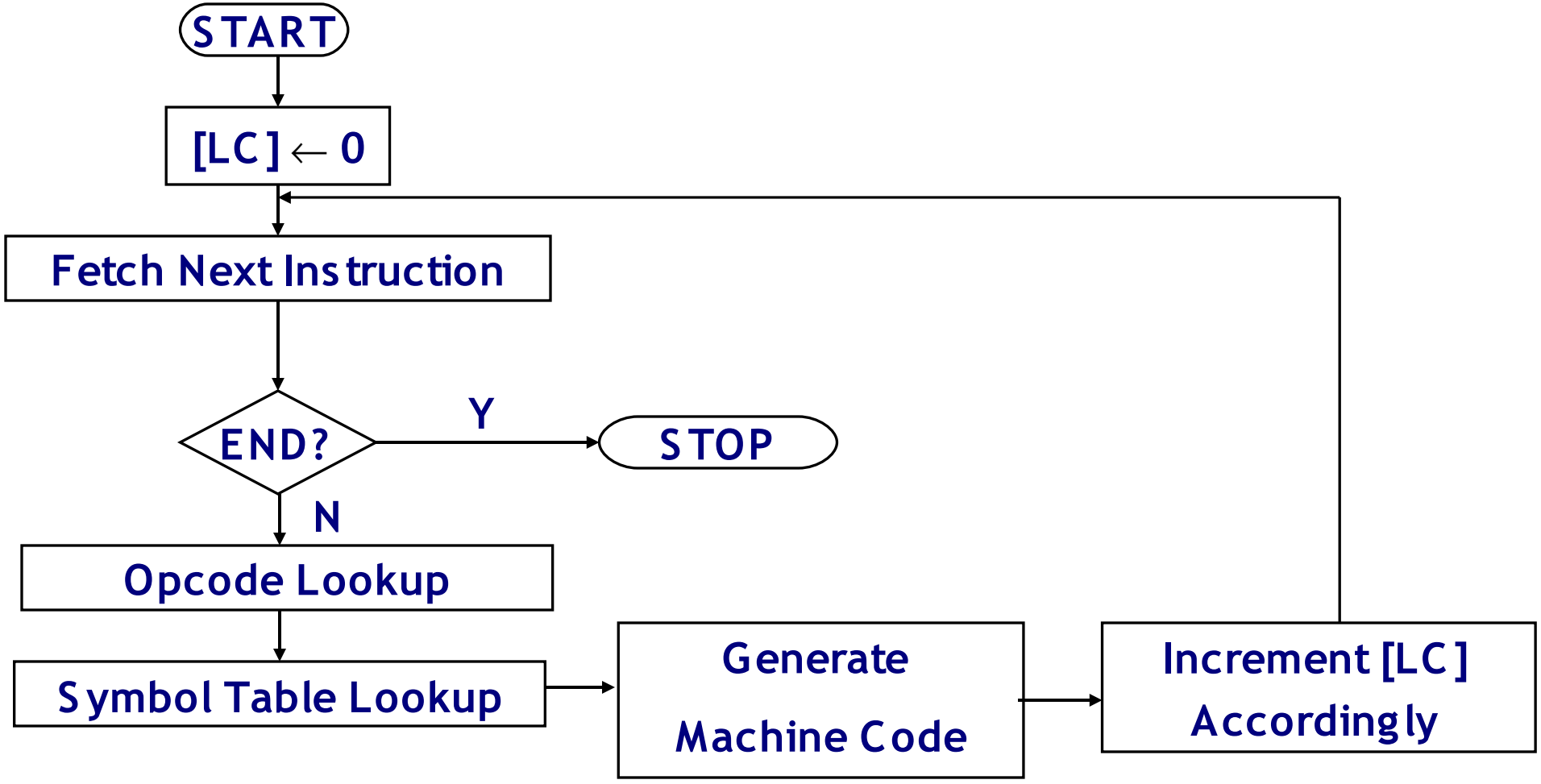

```
 1                      00000019          OPT       CRE
 2                      00000019  A       EQU       25
 3  00001000                      M       ORG       $1000
 4  00001000  00000004  N       DS.W      2
 5  00001004  00001008  EXIT    DC.L      EXIT
 6  00001008  2411      EXIT    MOVE.L    (A1),D2
 7  0000100A  139A2000          MOVE.B    (A2)+,(A1,D2)
 8  0000100E  06450019          ADDI.W    #A,D5
 9  00001012  67000008          BEQ       DONE
10  00001016  90B81004          SUB.L     N,D0
11  0000101A  60EC              BRA       EXIT
12  0000101C  4E722700  DONE    STOP      #$2700
13  00001000                    END       $1000
Lines: 13, Errors: 0, Warnings: 0.
```

What we care in the symbol table

```
SYMBOL  TABLE  INFORMATION

Symbol-name    Type     Value      Decl  Cross reference line numbers
A              EQU      00000019      2     8.
DONE           LABEL    0000101C     12     9.
EXIT           LABEL    00001008      6     5,   11.
M              LABEL    00001000      4   * * NOT USED * *
N              LABEL    00001004      5    10.
```

# Pass II

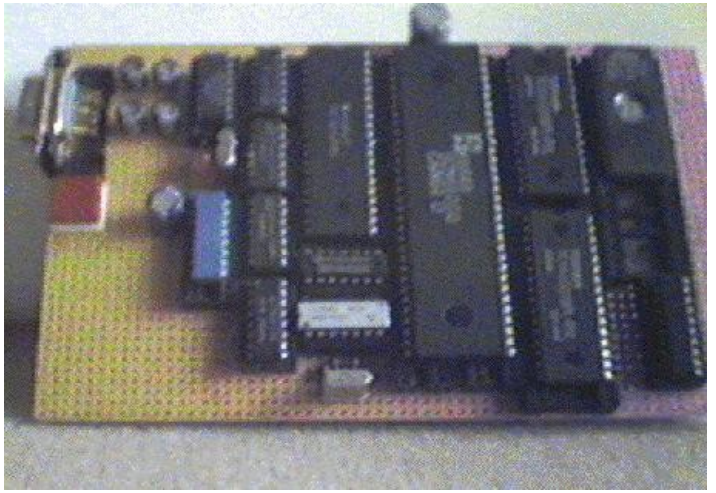


# How to Run a Program

- Use a simulator
  - SIM68K and E68K: free, MS-DOS based
  - Simulator in included in IDE68k and EASy68k
  - Commercial products are also available
- Download & run on a target board
  - Our lab has the Flight 68K
- Burn into EPROM & run on a real board
  - Must build a board first
- Use an emulator
  - Expensive

# Running Your Program in Hardware

Flight 68k Educational Board



Prototype board


HMI-200-6800 In-Circuit Emulator
(http://www.avocet.com)

# Assembly Language Statement

- Generic instruction format

```
<label> opcode<.field> <operands> <;comments>
```

  - `<label>` pointer to the instruction's memory location
  - `opcode` operation code (MOVE, ADD, etc)
  - `<.field>` width of operand (B,W,L)
  - `<operands>` data used in the operation
  - `<;comments>` for program documentation

- Examples:

| Instruction | | | RTL |
|---|---|---|---|
| | MOVE.W | #100,D0 | [D0] ← 100 |
| | MOVE.W | 100,D0 | [D0] ← [M(100)] |
| | ADD.W | D1,D0 | [D0] ← [D0] + [D1] |
| | MOVE.W | D1,100 | [M(100)] ← D1 |
| DATA | DC.B | 20 | [DATA] ← 20 |
| | BRA | LABEL | [PC] ← label |

# Label Field

- Optional.
- Required if the statement is referred by another instruction.
  - Target of Bcc, BRA, JMP, JSR or BSR instructions
  - Data structure
- Basic rules:
  - If used, label must start at column 1.
  - 1st character must be a letter (A-Z, a-z).
  - Subsequent characters must be letter or digit.
  - If 1st character is ; or *, the whole line is a comment.
- Labels must be unique.
- The symbols A0-A7, D0-D7, CCR, SR, SP & USP are reserved for identifying processor registers.

# Label Field

- Valid symbolic name contains 8 letters or numbers.
- Name starts with letter.
- Only 8 letters are significant:
  - TempVa123, TempVa127 are recognized as TempVa12 by assembler
- Use meaningful labels!

| Valid labels | Valid but meaningless | Invalid labels |
|---|---|---|
| ALPHA | CONFIUS | 123 |
| First | ENCIK | 1st |
| Second | TOLONG | 2nd |
| NUMBER3 | LULUSKAN | AK-47 |
| MIX3TEN | SAYA | DIV/2 |

# Opcode Field

- Two types of statements
  - Executable instructions
  - Assembler directives
- Executable instructions
  - Must exist in instruction set
  - translated into executable machine code
  - tells the machine what to do at execution
  - e.g. MOVE, ADD, CLR
- Assembler directives
  - Controls the assembly process
  - non-executable -> not translated into machine code
  - Varies by assembler
  - e.g., EQU, DC, DS, ORG, END
- May have size specifier (Byte, Word or Longword)

# Operands

- Operands can be
  - Registers
  - Constants
  - Memory addresses (variables)
- Operands specify addressing modes such as
  - Dn: data register direct `MOVE.W D0, D1`
  - An: address register indirect `MOVE.W (A0),D1`
  - #n: immediate `MOVE.W #10,D1`
  - N: absolute `MOVE.W $1000,D1`
- Operands can be specified in several formats
  - Decimal: default
  - Hexadecimal: prefixed by $
  - Octal: prefixed by @
  - Binary: prefixed by %
  - ASCII: within single quotes ‘ABC’

# Operand Field

- Number of operands (0/1/2) depends on instruction
- For two-operand instruction, separate by a comma
  - First operand = source
  - Second operand = destination
- Examples:

```
RESET                              ;zero-operand
 CLR            D2                 ;one-operand
 MOVE           D0,D1              ;two-operand
```

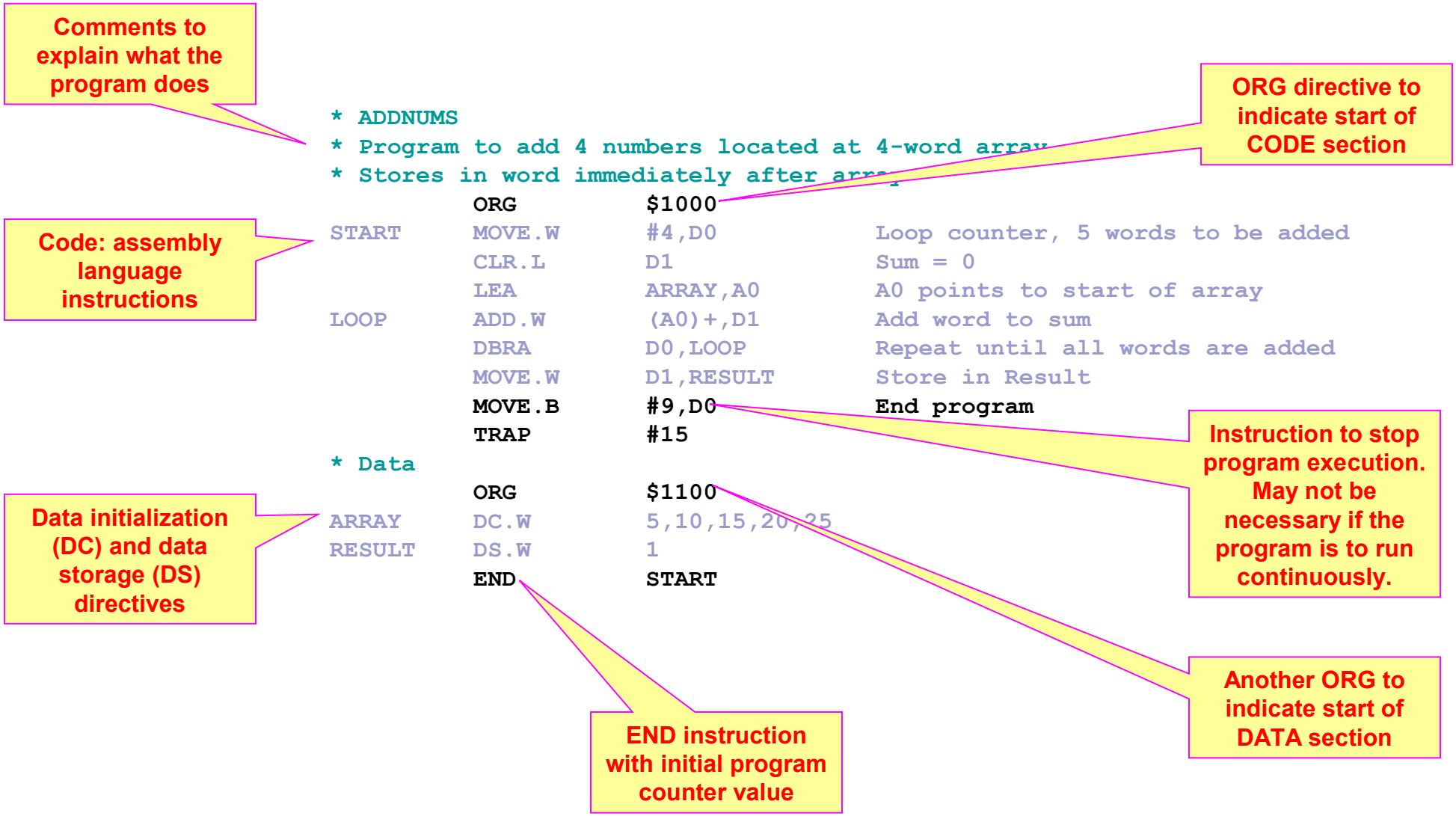
# Operand Field

- Operand field can also contain expressions (“formulas”)
- Allowed expressions include
  - Symbols
    - Follows the rules for creating labels
  - Constants
    - Numbers or ASCII strings
  - Algebraic operations
    - + (add) - (subtract)
    - * (multiply) / (divide)
    - % (remainder) ~ (NOT)
    - & (AND) | (OR)
    - ^ (XOR)
    - << (shift left/multiply by 2) >> (shift right/ divide by 2)
  - Location counter (* symbol)
    - Keeps tracks of which line is being assembled

# Comment Field

- Comments are important!
  - Explains how program works
  - Explains how to use the program
  - Makes modifications easier in future
- Comments are ignored by the assembler
- Comment field starts with ; or *
- Tips:
  - Not easy to have “just the right amount” of comments
  - Don’t comment the obvious
  - A line begins with an ‘*’ in its first column is a comment line → ignored by the assembler

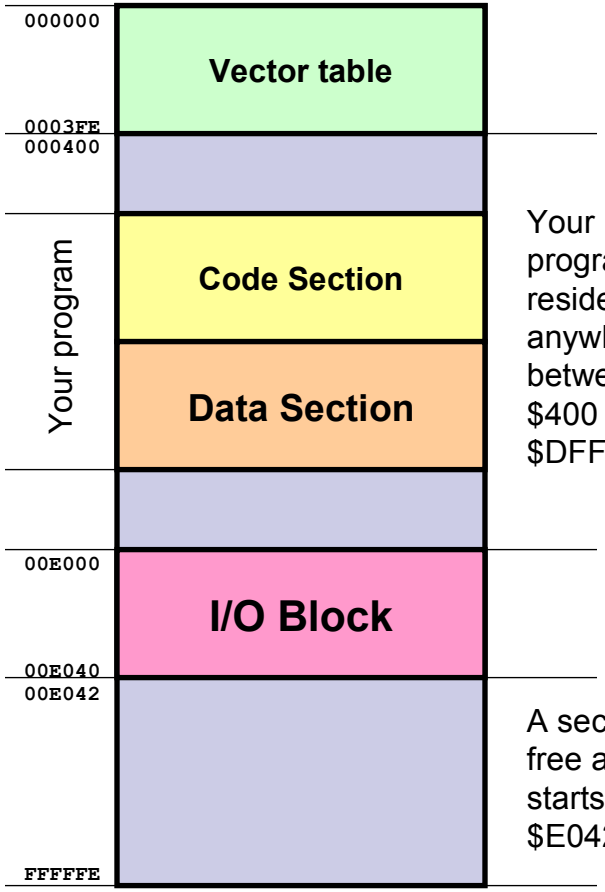
# Program Template




```
* ADDNUMS
* Program to add 4 numbers located at 4-word array
* Stores in word immediately after array
          ORG       $1000
START     MOVE.W    #4,D0          Loop counter, 5 words to be added
          CLR.L     D1             Sum = 0
          LEA       ARRAY,A0       A0 points to start of array
LOOP      ADD.W     (A0)+,D1       Add word to sum
          DBRA      D0,LOOP        Repeat until all words are added
          MOVE.W    D1,RESULT      Store in Result
          MOVE.B    #9,D0          End program
          TRAP      #15
* Data
          ORG       $1100
ARRAY     DC.W      5,10,15,20,25
RESULT    DS.W      1
          END       START
```

# Where to Put Your Program

**Running a program in IDE68K**

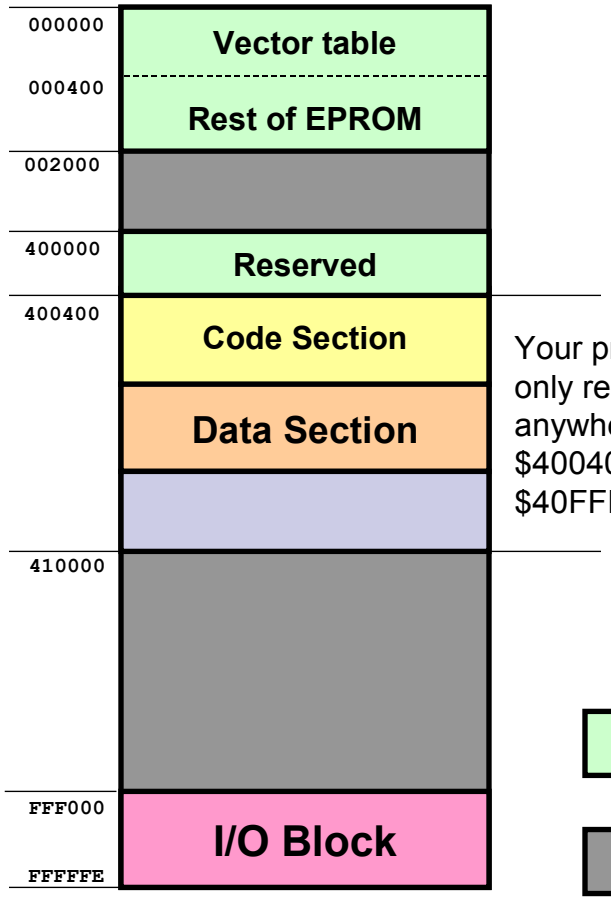



Your program can reside anywhere between $400 and $DFFE.

A second free area starts from $E042.

* IDE68K doesn't have ROM!

**Running a program in Flight 68K board**




Your program can only reside anywhere between $400400 and $40FFFE.

For system use. Not for user programs.

Ambiguous areas - do not contain any devices.

Free area for programs.

# ORG Directive

- Sets the address for following instruction or data
  - Example:

```
ORG         $400
MOVE        D0,D1
```

    - Puts the MOVE instruction at location $40.

- ORG actually reset the value of location counter (LC)
- LC: Assembler's simulation of PC

# END Directive

- Tells the assembler to stop assembling
- Usually the last statement in a file
- If not the last statement, the following statement will be ignored
- Some assemblers don't need the instruction
- Some assembler make you supply the starting address of the program
  - Example: END $2000 means set the program counter to $2000 when running this program

# EQU Directive

- Link a name to a value
- ex 1

```
SAIZ        EQU     20
            ORG     $400
            MOVE    #SAIZ,D0
```

MOVE #SAIZ,D0 has the same effect as MOVE #20,D0

- ex 2

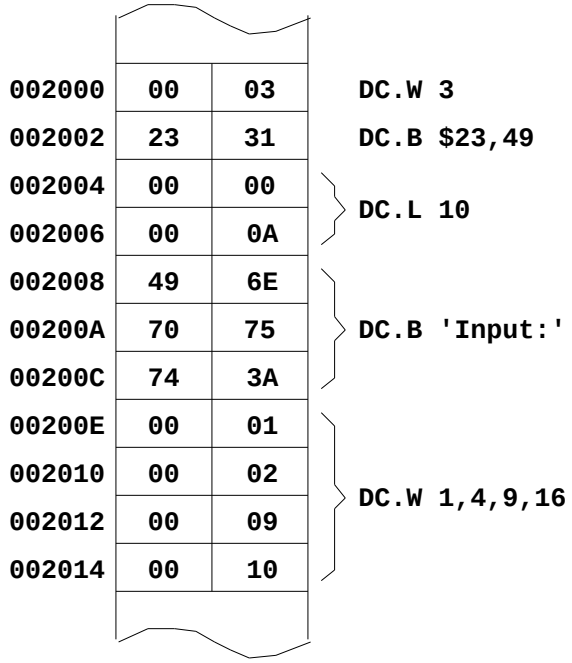
```
ROW         EQU     5
COLUMN      EQU     ROW+1
SQUARE      EQU     ROW*COLUMN
```

SQUARE will be replaced by 30 every time it is used.

# DC Directive

- define constant
- Reserves memory location and initialize (put in initial value)
- Can initialize many data in a time
- The sizes will be considered in B,W or .L
- Take care: A 16-bit word should not be stored across the even boundary, e.g. at $1001

```
ORG     $2000
DC.W    3
DC.B    $23,49
DC.L    10
DC.B    'Input:'
DC.W    1,2,9,16
```



# DS Directive

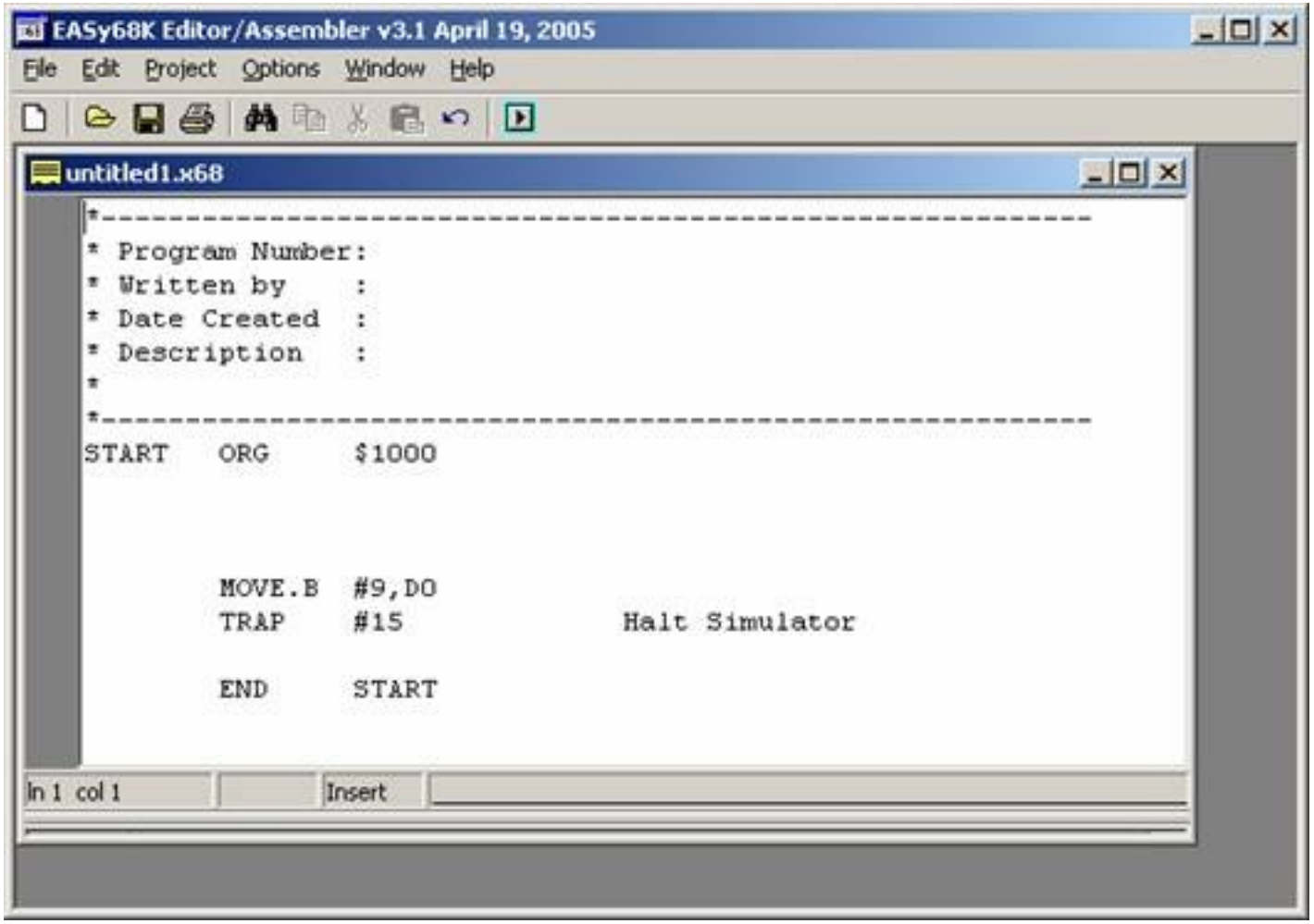
- define storage
- Reserves (allocates) storage location in memory
- Similar to DC,  but no values stored
  - DC: set up values in memory locations
  - DS: reserve memory space for variables
- Useful to define storage for calculation results

# the Location Counter

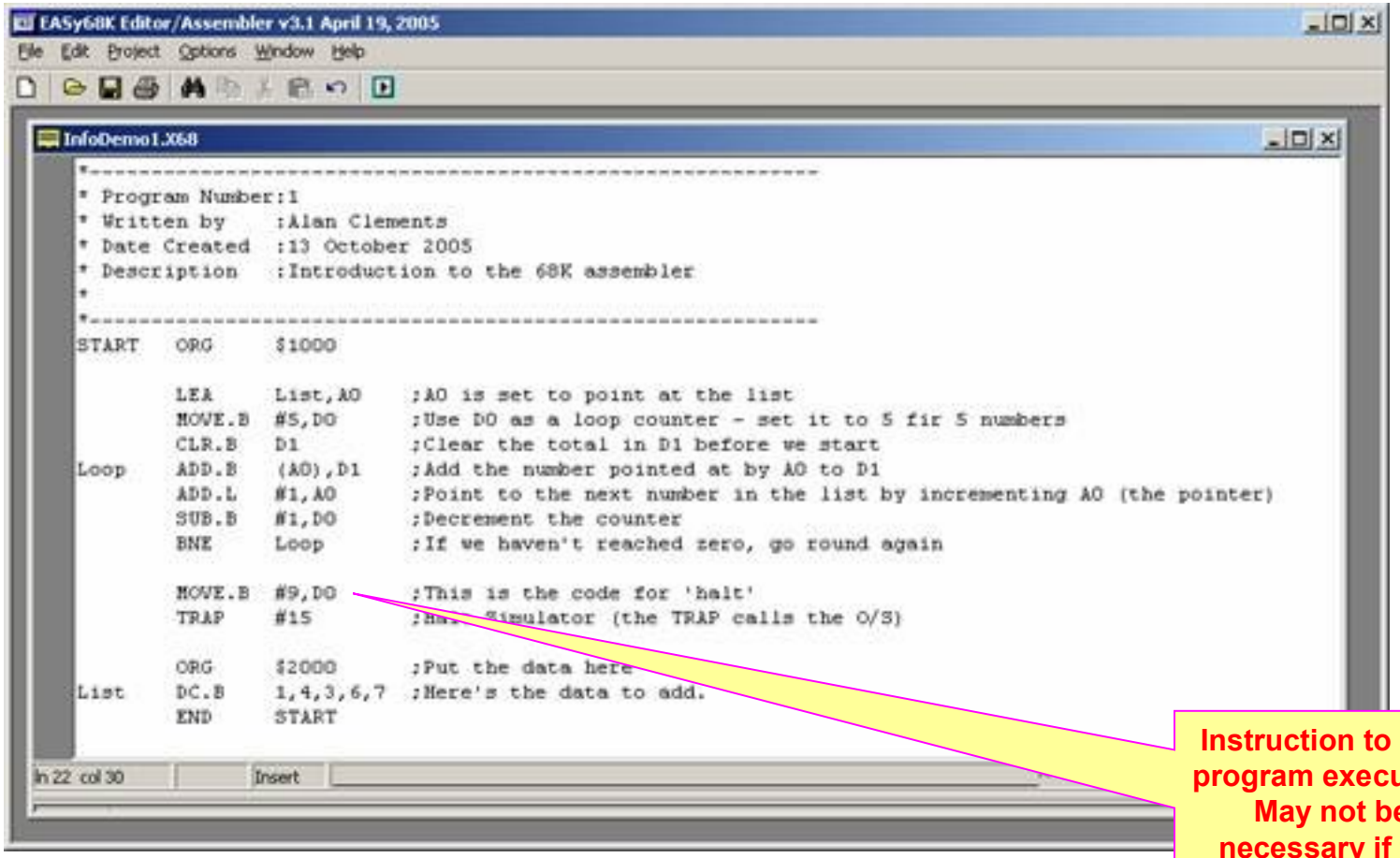
- Location Counter (LC) can be accessed by the * symbol.
  - In this example, you can change the string any time, and STRLEN will automatically be updated.

```
* Program to convert lower case to upper case
          ORG         $1000
LC2UC     MOVE.W      #STRLEN,D0        Counter = String length
          LEA         STRING,A0         Pointer = Start of string
LOOP      MOVE.B      (A0),D1           Get a byte
          CMP.B       #'a',D1           Compare against a
          BLT         NEXT              If less, skip it
          CMP.B       #'z',D1           Compare against z
          BGT         NEXT              If greater, skip it
          SUB.B       #'a'-'A',D1       Else convert to upper case
          MOVE.B      D1,(A0)           Put it back in the array
NEXT      ADDA        #1,A0             Bump pointer
          SUB.W       #1,D0             Decrement counter
          BNE         LOOP              Do it until finished
          STOP        #$2000            End of code
* Data
STRING    DC.B        'Hello, Good Morning to you!'
STRLEN    EQU         * - STRING
          END         $1000
```

# EASy68K: Just starting up

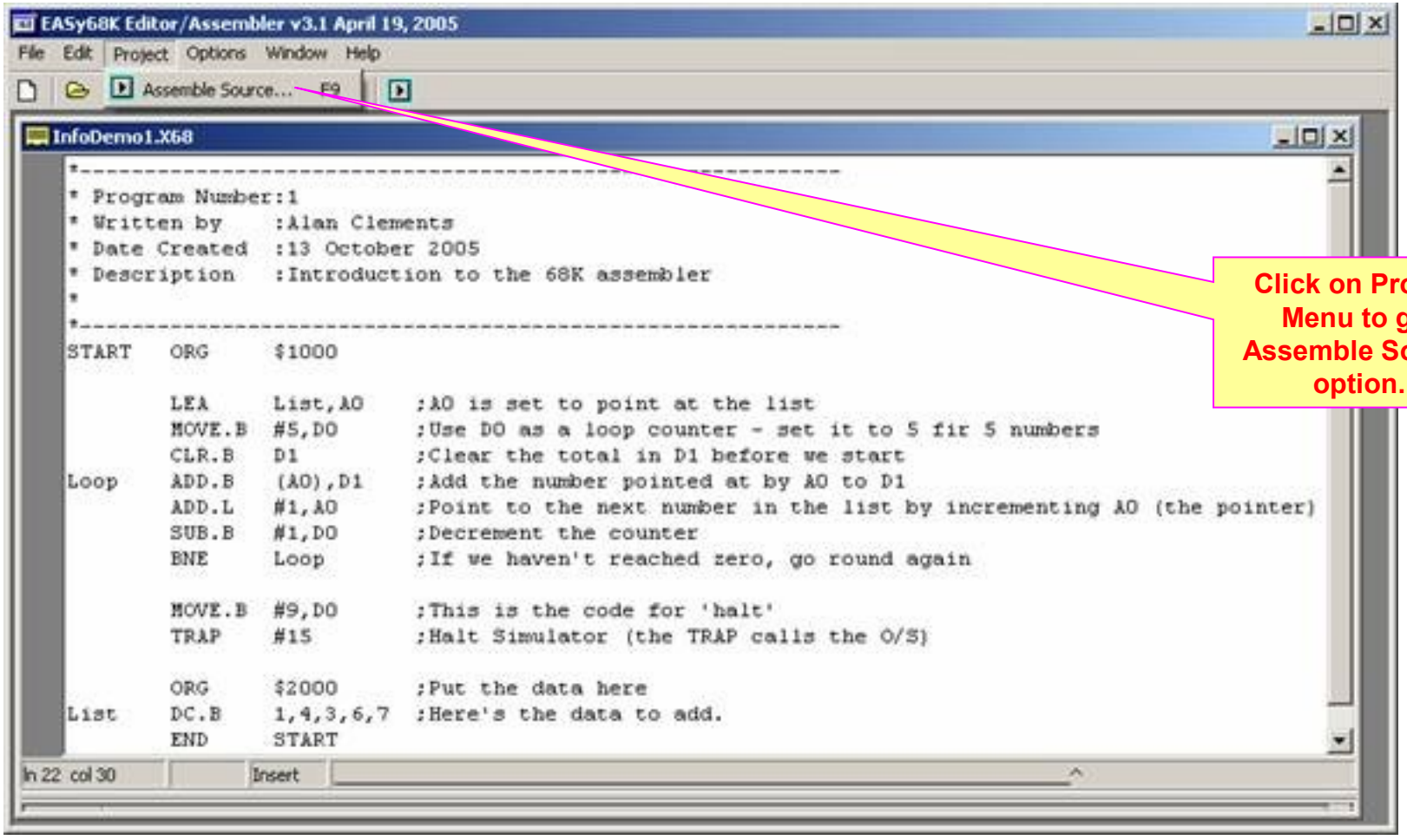


# EASy68K: Entering a sample program

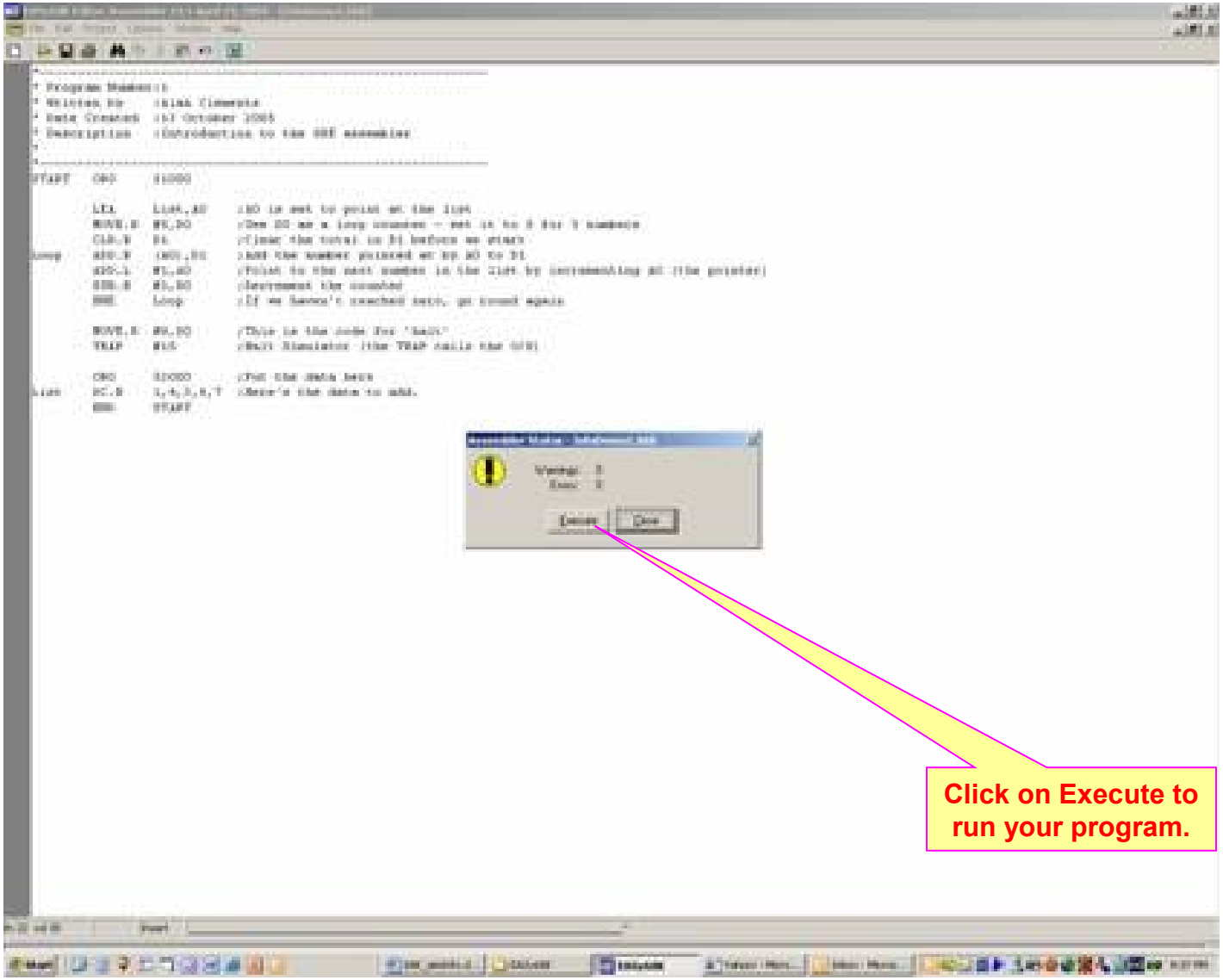



**Instruction to stop program execution. May not be necessary if the program is to run continuously.**

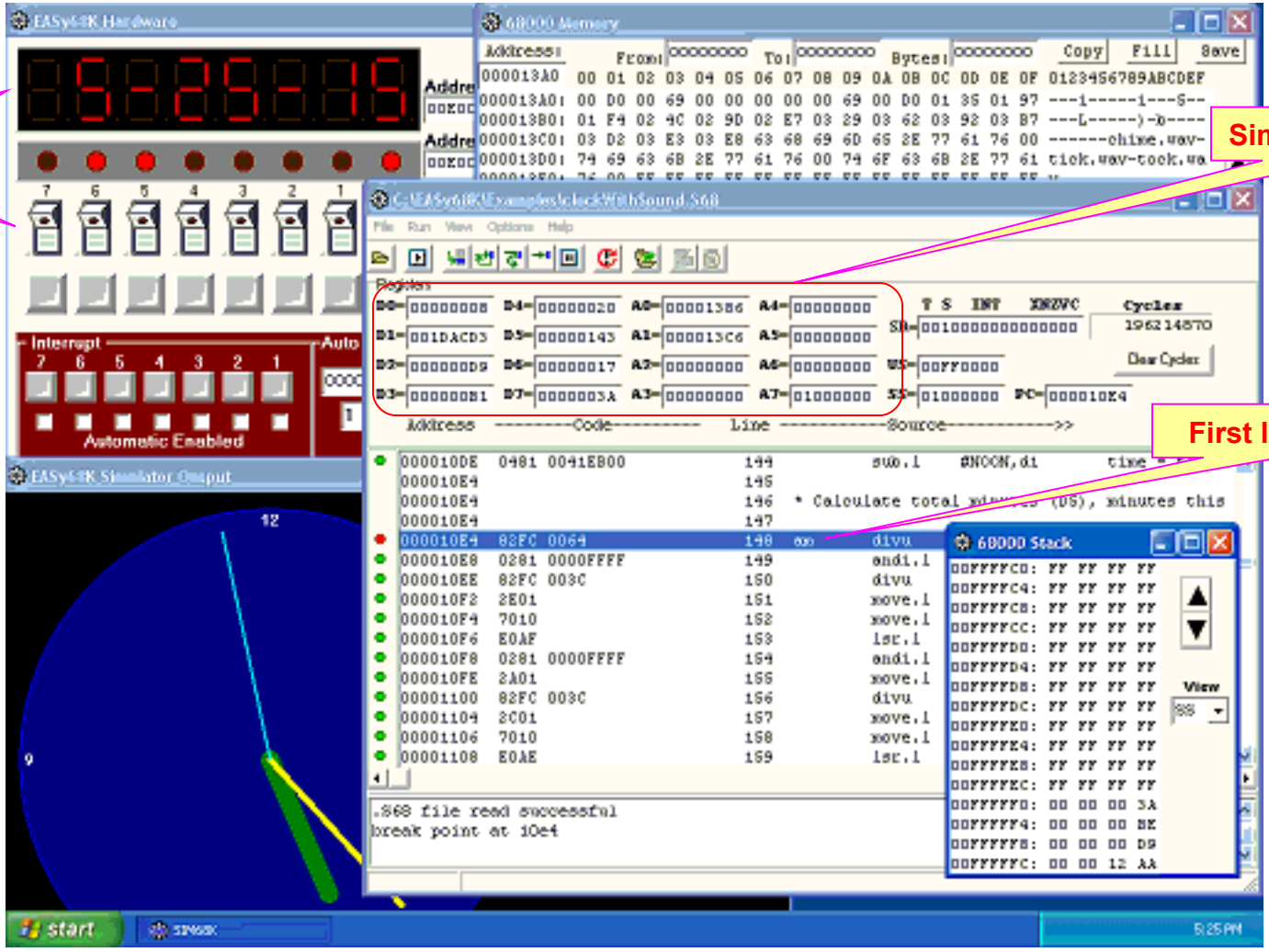
# EASy68K: Assembling a Program



# EASy68K: Successful Assembly



# Using EASy68K



# Summary

- We have seen how the assembler works
- There are two types of statements
  - Executable instructions
  - Assembler directives
- There are many assembler available such as EASy68k, IDE68K and ASM68K