SEE 3223 Microprocessor Systems

3: Assembly Language

Muhammad Mun'im Ahmad Zabidi (munim@utm.my)

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

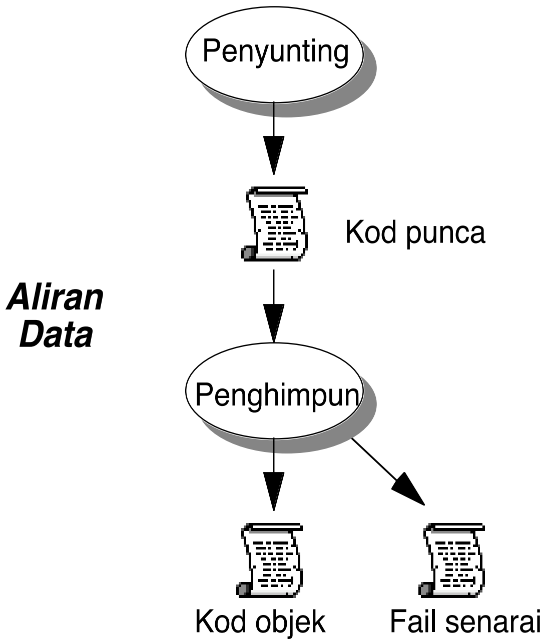
Machine & Assembly Language

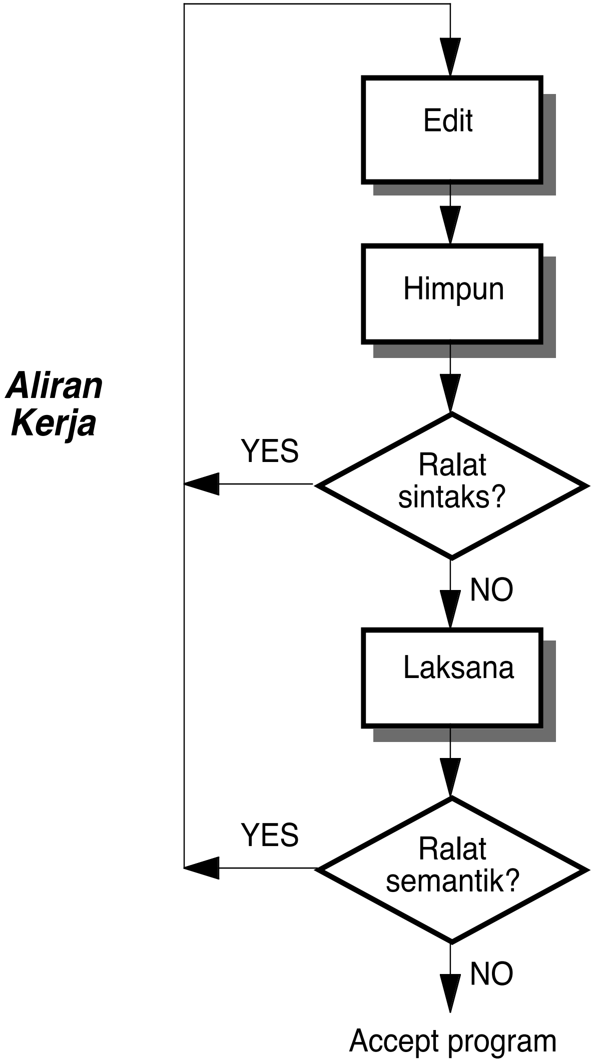
- Machine language instruction:
  - Binary number for processor consumption
  - Extremely hard to read or write even for very simple programs
- Assembly language instruction:
  - Mnemonic (easy to remember code) representing machine language
- Solution:
  - Programmer uses assembly language
  - Processor 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

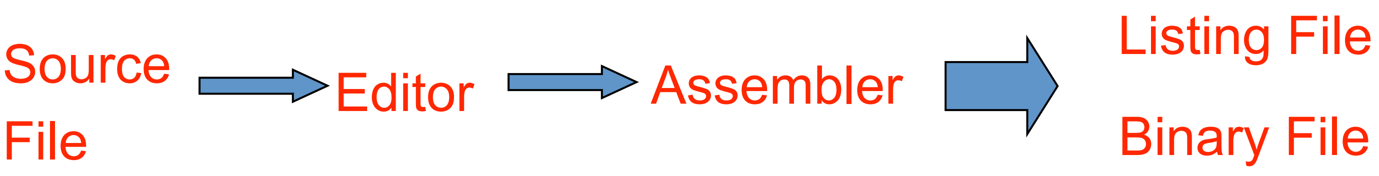
- Assembler — software that translates from assembly language to machine language
- Source program / source code — program written by humans, as input to the assembler
- Object program / object code — machine language program generated by the assembler
- Cross Assembler — assembler that generates machine 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





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

```
*    PROGRAM TO ADD TEN WORDS
        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

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

Listing File Example

```
MC68000 Cross Assembler
Copyright (C) Stephen Croll, 1991.  Author: Stephen Croll
Version 2.00 beta 1.02

                             1 * PROGRAM TO ADD TEN WORDS
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 * DATA ARRAY STARTS HERE
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.
```

Listing File with Errors

```
MC68000 Cross Assembler
Copyright (C) Stephen Croll, 1991.  Author: Stephen Croll
Version 2.00 beta 1.02

                                      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
S2140010004240720A41F900001020D058534166
FA57
S21000101033C1000010341E3C00E34E4EBE
S2140010200000D0011000E0044FFFD0014005500
1EC8
S2080010300001 0013A3
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 code
  - Assemble faster
  - Have "macro" features
  - Support really large programs
  - One example 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

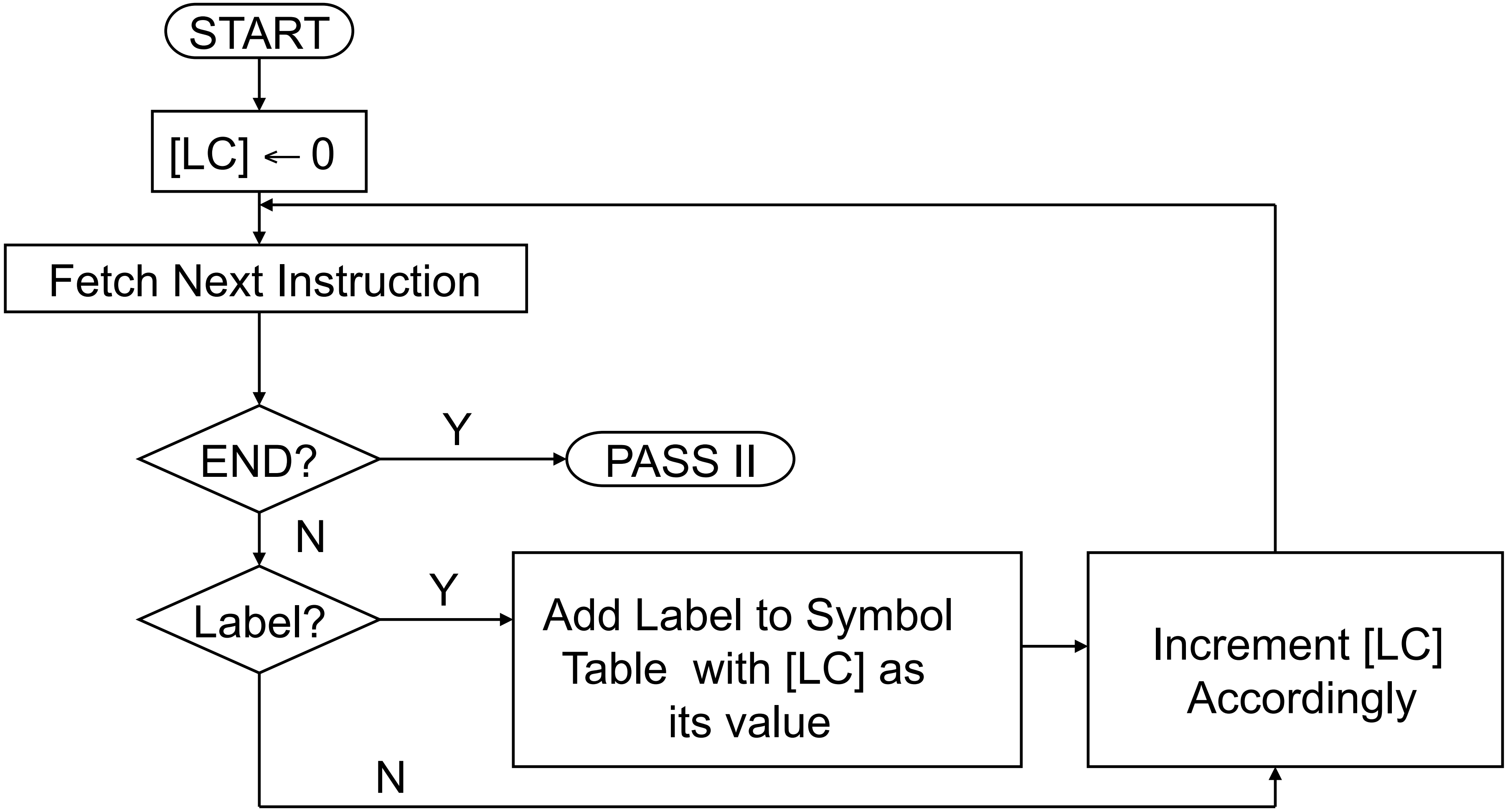
Two-Pass Assembler

- 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


START
[LC] ← 0
Fetch Next Instruction
END?
Y
PASS II
N
Label?
Y
Add Label to Symbol Table with [LC] as its value
Increment [LC] Accordingly
N

Symbol Table

LC

Machine code

Assembly code

```
 1                                OPT       CRE
 2           00000019   A         EQU       25
 3  00001000                      ORG       $1000
 4  00001000 00000004   M         DS.W      2
 5  00001004 00001008   N         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.
```

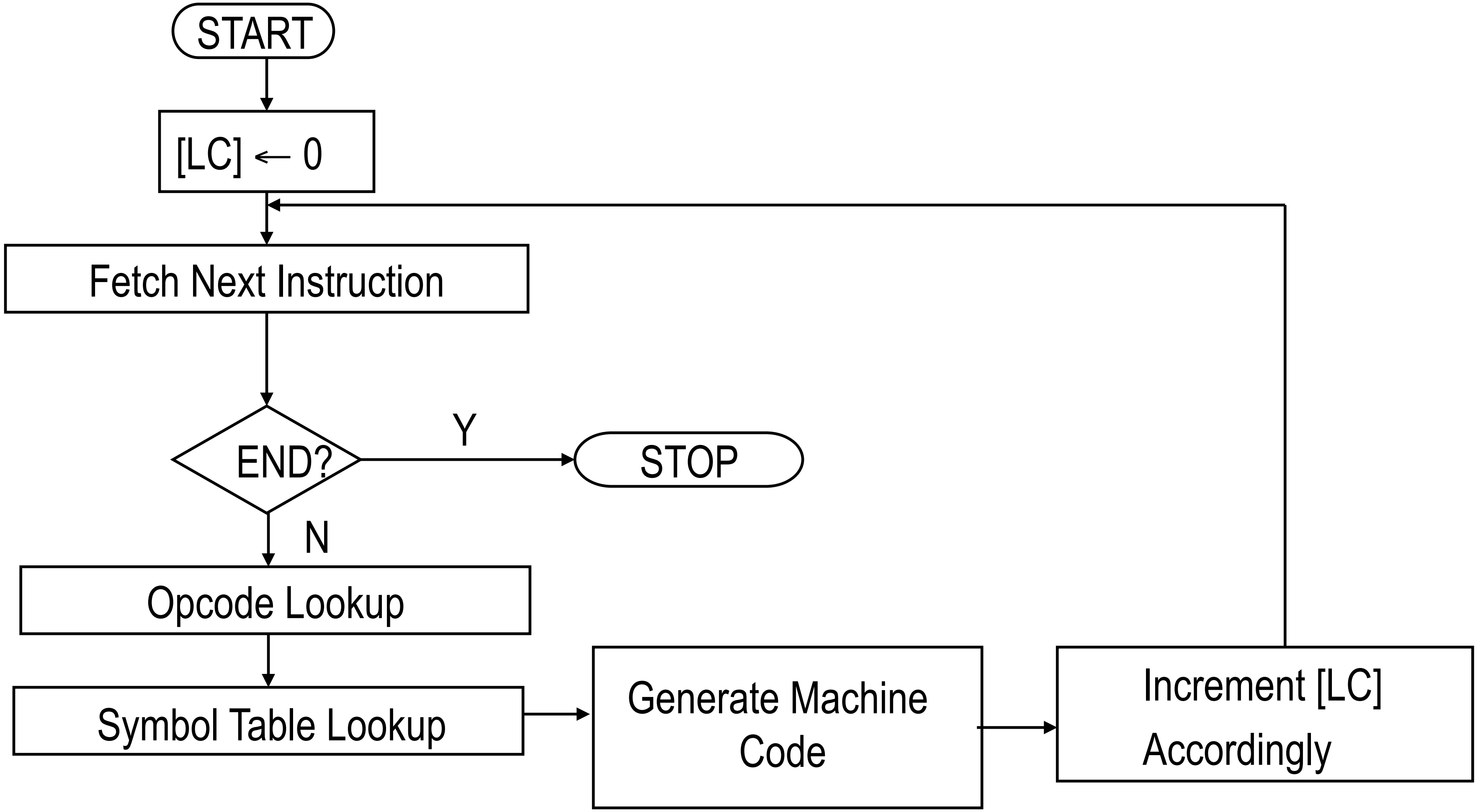
Forward reference

Forward reference

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


START
[LC] ← 0
Fetch Next Instruction
END?
Y
STOP
N
Opcode Lookup
Symbol Table Lookup
Generate Machine Code
Increment [LC] Accordingly

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

Assembly Language Statement

- Generic instruction format

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

- □ <label> pointer to the instruction's memory location
- □ opcode operation code (MOVE, ADD, etc)
- □ <.size> size of operand (B,W,L). If omitted, usually defaults to .W
- □ <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:

MOVE is equivalent to MOVE.W. If a size specifier applies to an instruction, the default data size is Word.

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

The RESET instruction is one of several instructions that do not have a size specifier.

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
    - → line is ignored by the assembler

Program Template

Comments to explain what the program does

ORG directive to indicate start of CODE section

Code: assembly language instructions

```
* ADDNUMS
* Program to add 4 numbers located at 4-word array
* Stores in word immediately after array
                    $1000
             W      #4,D0          Loop counter, 5 words to be added
                    D1             Sum = 0
                    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
                    $1100
                    5,10,15,20,25
                    1
                    START
```

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

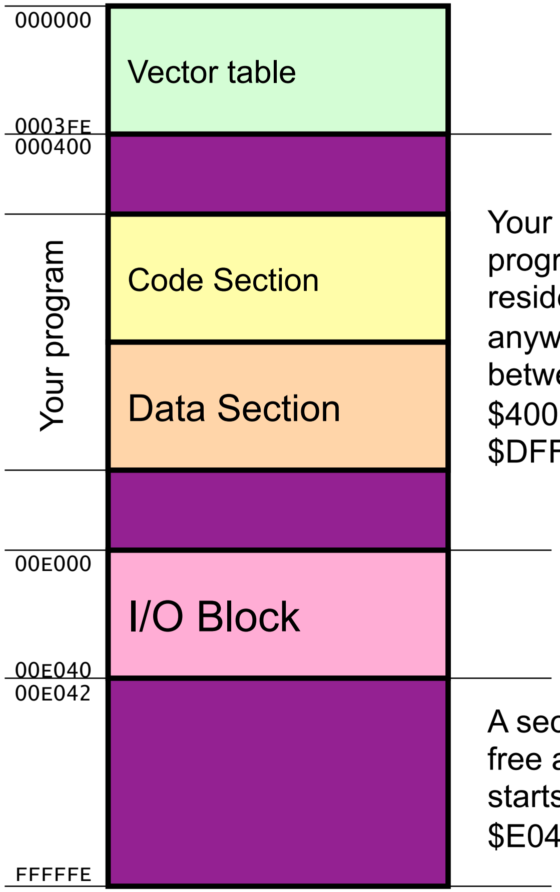
Data initialization (DC) and data storage (DS) directives

Another ORG to indicate start of DATA section

END instruction with initial program counter value

Where to Put Your Program

Running a program in IDE68K



000000

Vector table

0003FE

000400

Your program

Code Section

Data Section

00E000

I/O Block

00E040

00E042

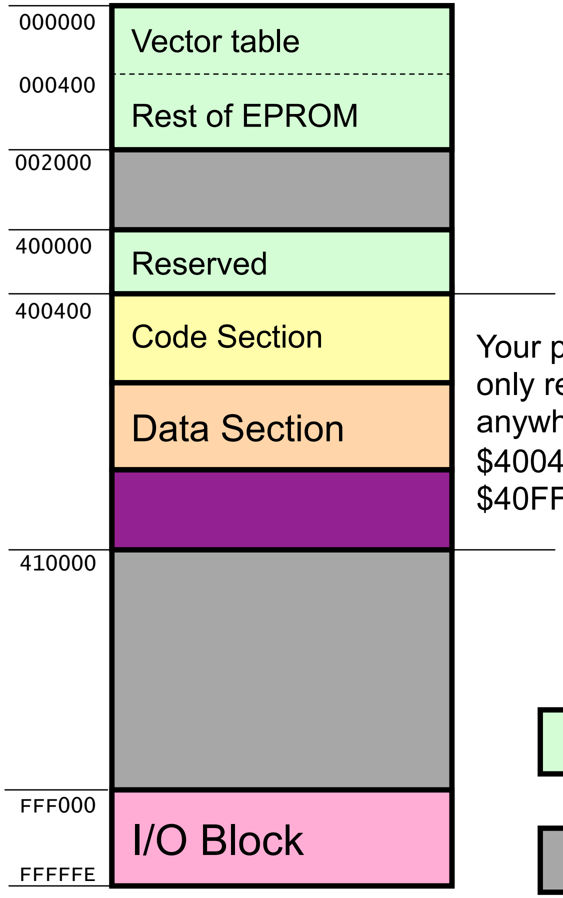
FFFFFE

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



000000

Vector table

000400

Rest of EPROM

002000

400000

Reserved

400400

Code Section

Data Section

410000

FFF000

I/O Block

FFFFFE

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 assemblers 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

- Equates 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

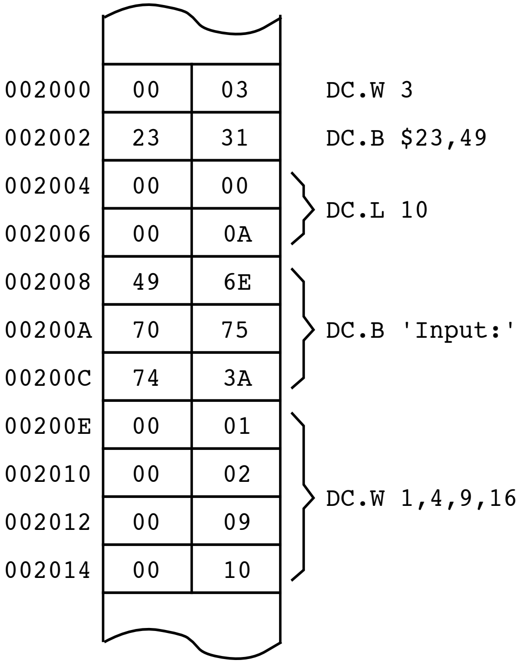
```
ROW         EQU     5
COLUMN      EQU     ROW+1
SQUARE      EQU     ROW*COLUMN
```

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

DC Directive

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

- 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



002000	00	03	DC.W 3
002002	23	31	DC.B $23,49
002004	00	00	DC.L 10
002006	00	0A	
002008	49	6E	DC.B 'Input:'
00200A	70	75	
00200C	74	3A	
00200E	00	01	DC.W 1,4,9,16
002010	00	02	
002012	00	09	
002014	00	10	

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 array for calculation results
- All values in the array are set to zero (cleared)

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.

- Example 1:
  - Here, * = 2004 because $2000 + 4 bytes of data = $2004.
    When we save the value of * into MYSTERY, we will have MYSTERY = $2004. We can use this to calculate length of the data array.

```
        ORG $2000
DATA    DC.B 1,2,3,4
MYSTERY EQU *
```

The Location Counter

- Example 2:
  - Here, LENGTH will get the value 4. So the MOVE instruction will put 4 into D0. What if you add more items in the array?

```
         ORG      $1000
         MOVE     #LENGTH,D0
         ORG      $2000
DATA     DC.B     1,2,3,4
LENGTH   EQU      *-DATA
```

- Example 3:
  - Here, LENGTH will get the value 9. You don't have to count how many items in the array. Did you notice number 7 is missing in the BYTE array?

```
         ORG      $1000
         MOVE     #LENGTH,D0
... process the array in 9 loops ...
         ORG      $2000
DATA     DC.B     1,2,3,4,5,6,8,9,10
LENGTH   EQU      *-DATA
```

the Location Counter

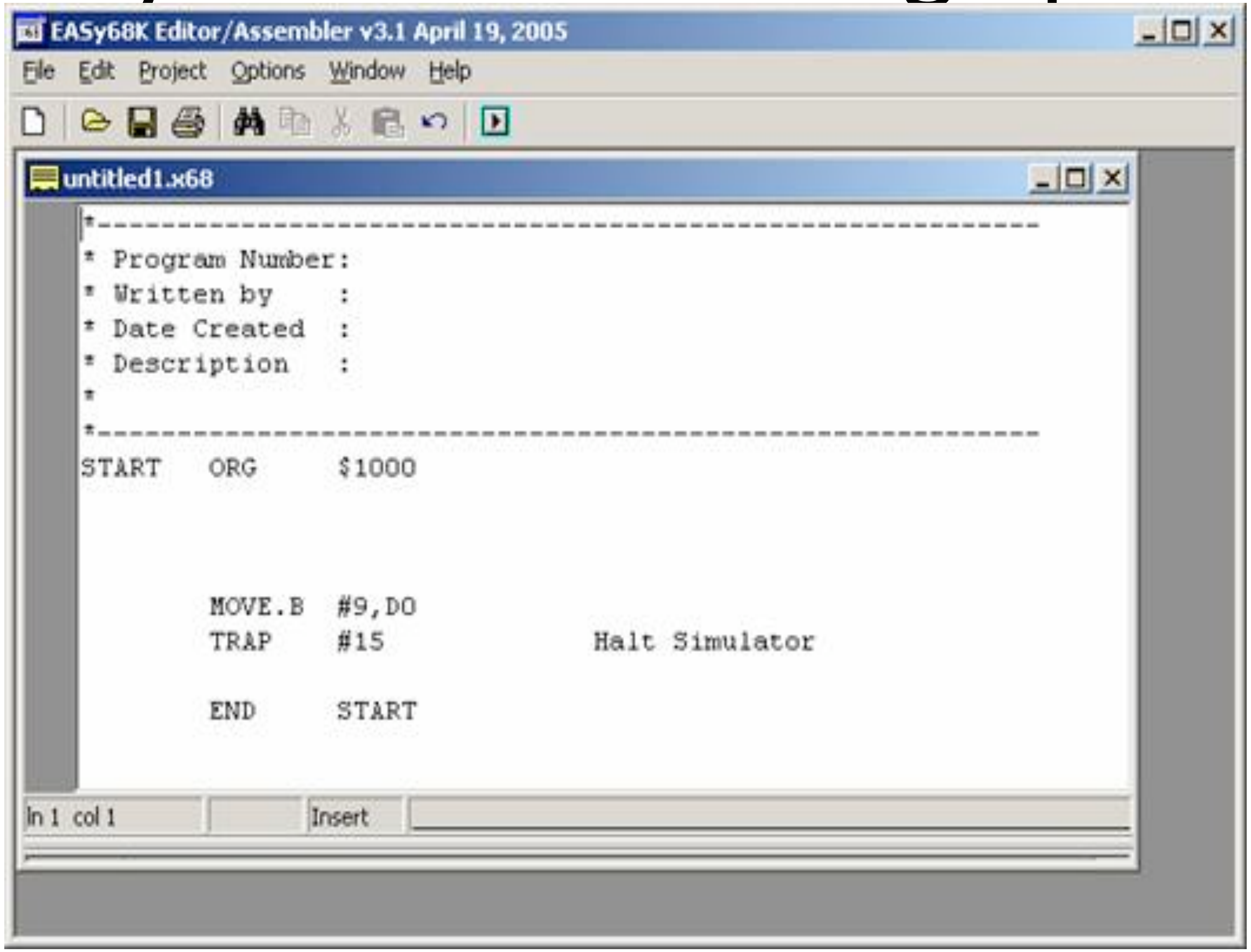
- Example 4 (Wrong way):
  - Here, LENGTH will get the value 18. IT'S NOT THE NUMBER OF ITEMS IN THE ARRAY, but the number of bytes. Normally you want to know the number of items.

```
                ORG       $1000
                MOVE      #LENGTH,D0
... process the array in 18 loops ...
                ORG       $2000
DATA            DC.W      1,2,3,4,5,6,8,9,10
LENGTH          EQU       *-DATA
```

- Example 5 (Correct way):
  - Here, LENGTH will get the correct value 9.

```
                ORG       $1000
                MOVE      #LENGTH,D0
... process the array in 9 loops ...
                ORG       $2000
DATA            DC.W      1,2,3,4,5,6,8,9,10
LENGTH          EQU       (*-DATA)/2
```

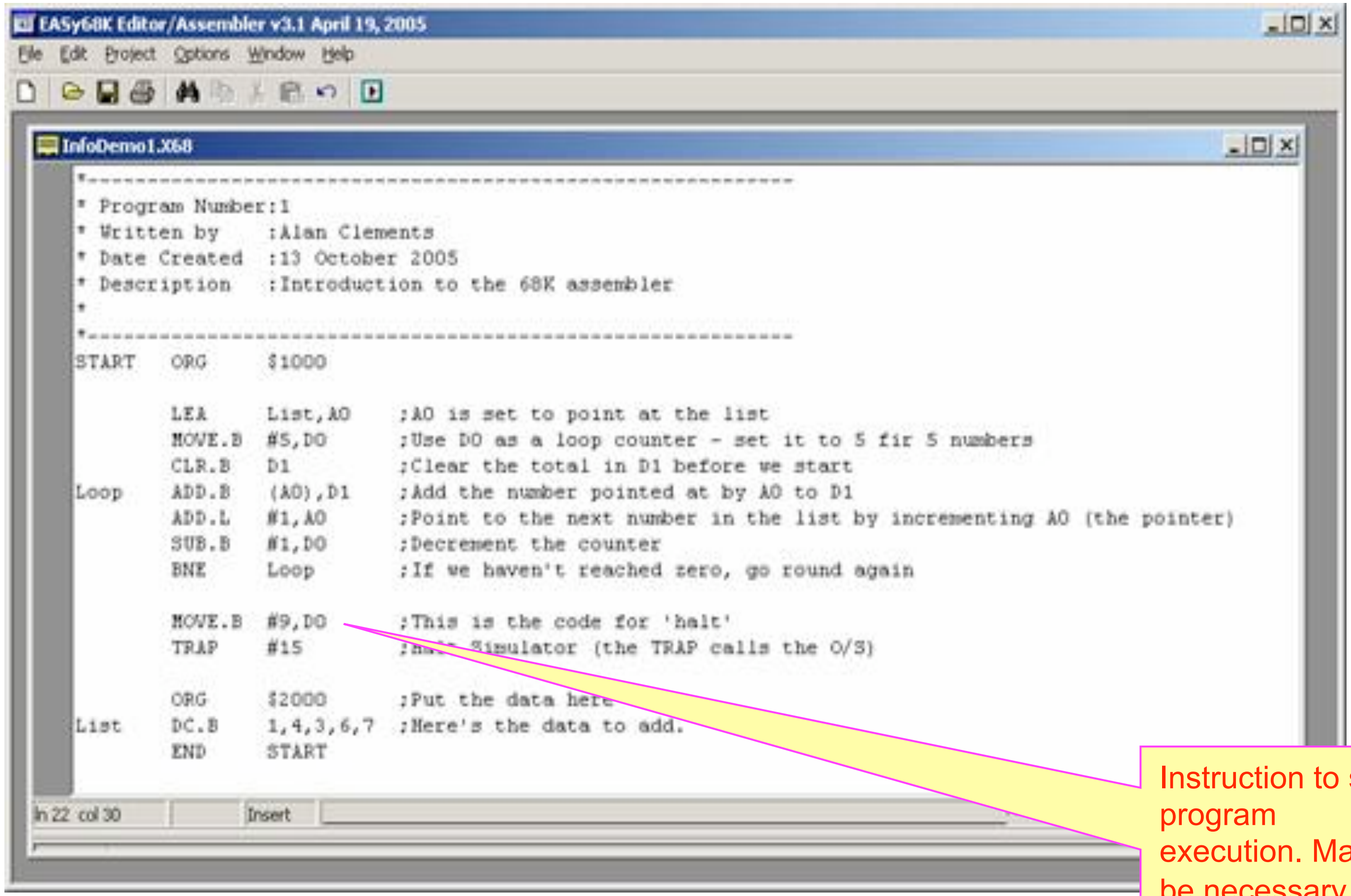
EASy68K: Just starting up



EASy68K Editor/Assembler v3.1 April 19, 2005
File Edit Project Options Window Help
untitled1.x68

```
*-----------------------------------------------------------
* Program Number:
* Written by    :
* Date Created  :
* Description   :
*
*-----------------------------------------------------------
START    ORG     $1000



         MOVE.B  #9,D0
         TRAP    #15              Halt Simulator

         END     START
```

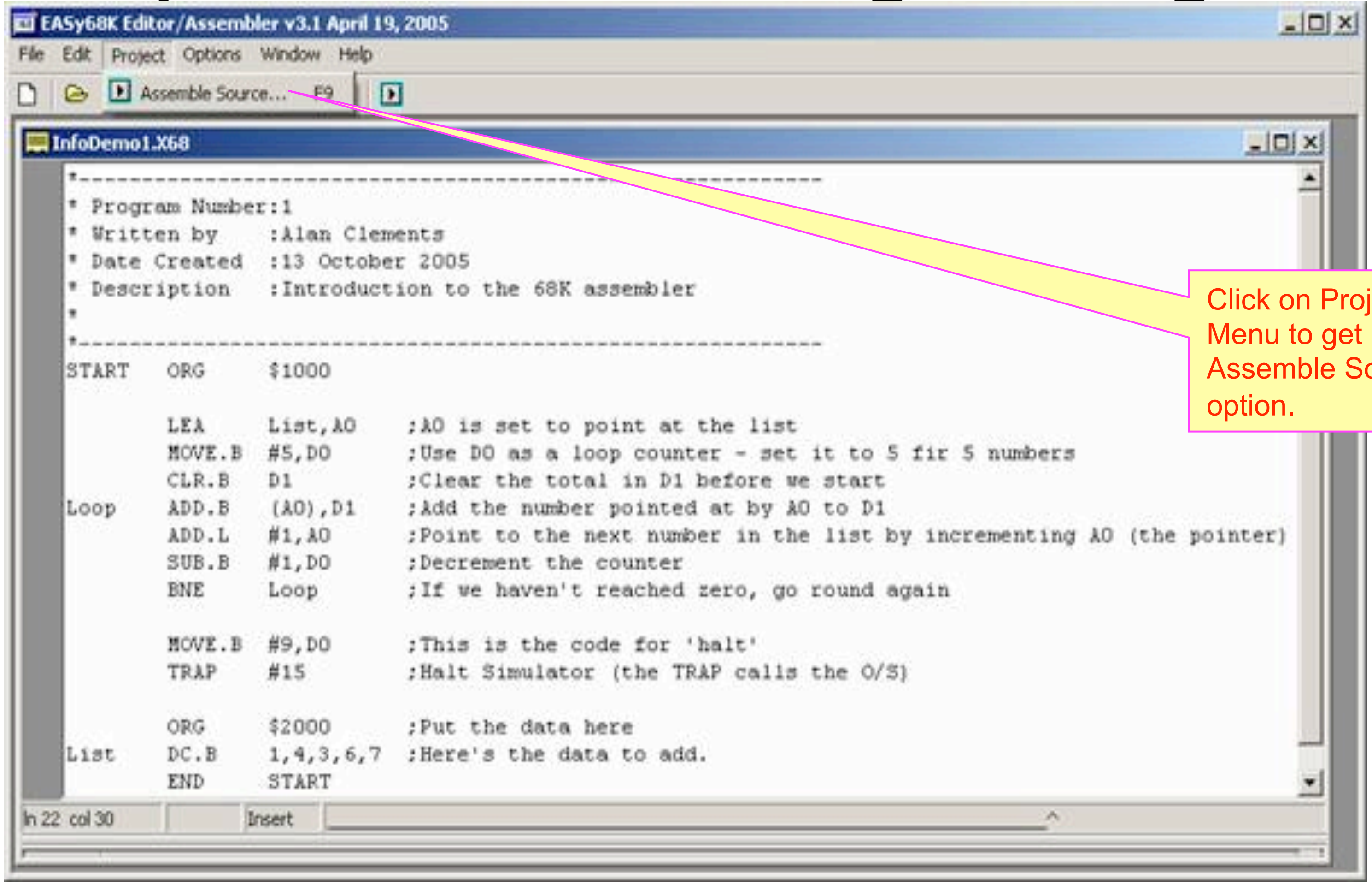
EASy68K: Entering a sample program


EASy68K Editor/Assembler v3.1 April 19, 2005
File Edit Project Options Window Help
InfoDemo1.X68
ln 22 col 30
Insert

```
*-----------------------------------------------------------
* Program Number:1
* Written by    :Alan Clements
* Date Created  :13 October 2005
* Description   :Introduction to the 68K assembler
*
*-----------------------------------------------------------
START   ORG     $1000

        LEA     List,A0     ;A0 is set to point at the list
        MOVE.B  #5,D0       ;Use D0 as a loop counter - set it to 5 fir 5 numbers
        CLR.B   D1          ;Clear the total in D1 before we start
Loop    ADD.B   (A0),D1     ;Add the number pointed at by A0 to D1
        ADD.L   #1,A0       ;Point to the next number in the list by incrementing A0 (the pointer)
        SUB.B   #1,D0       ;Decrement the counter
        BNE     Loop        ;If we haven't reached zero, go round again

        MOVE.B  #9,D0       ;This is the code for 'halt'
        TRAP    #15         ;Halt Simulator (the TRAP calls the O/S)

        ORG     $2000       ;Put the data here
List    DC.B    1,4,3,6,7   ;Here's the data to add.
        END     START
```

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

EASy68K: Assembling a Program

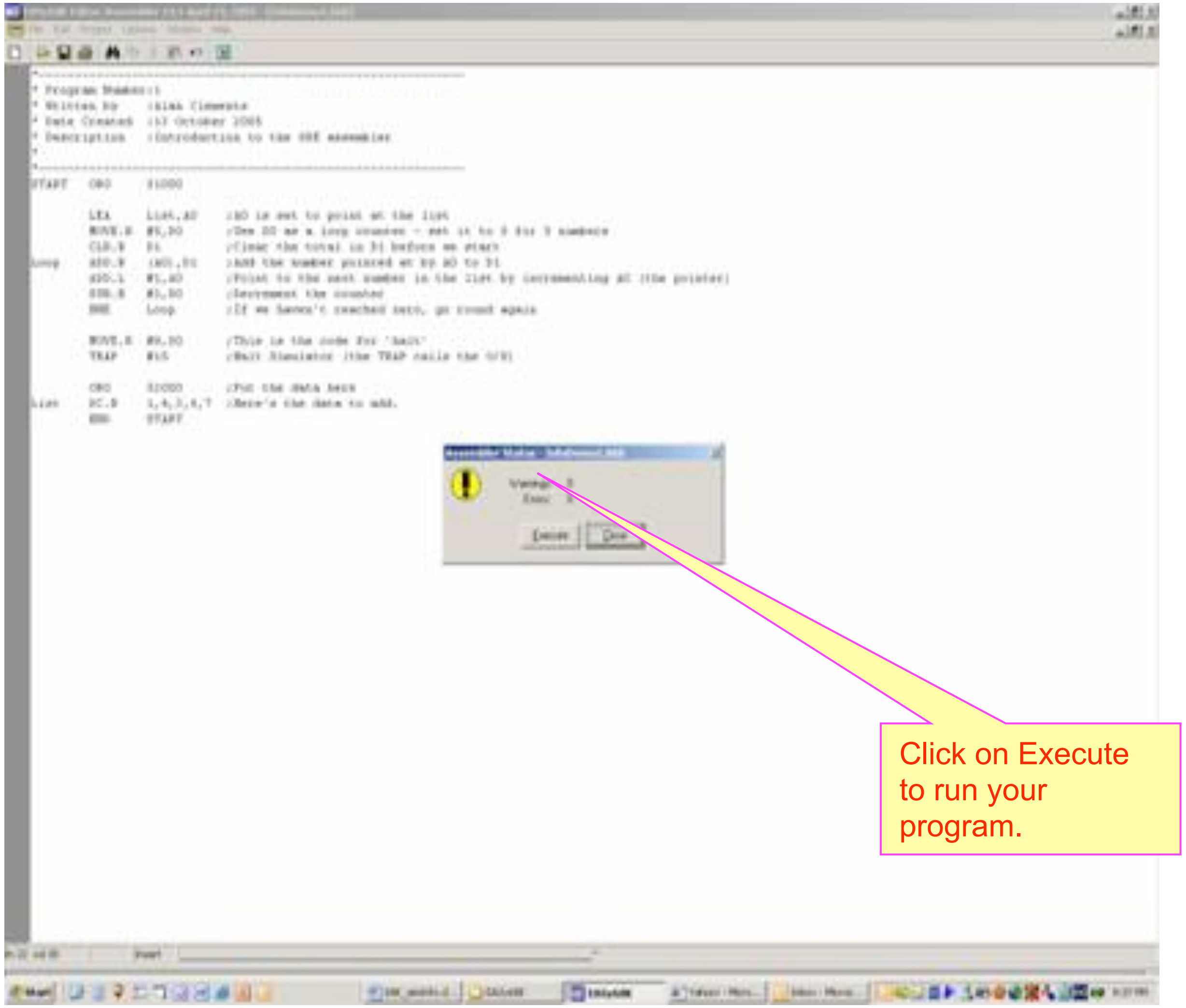


EASy68K Editor/Assembler v3.1 April 19, 2005
File Edit Project Options Window Help
Assemble Source... F9
InfoDemo1.X68
ln 22 col 30
Insert

```
*-----------------------------------------------------------
* Program Number:1
* Written by    :Alan Clements
* Date Created  :13 October 2005
* Description   :Introduction to the 68K assembler
*
*-----------------------------------------------------------
START   ORG     $1000

        LEA     List,A0     ;A0 is set to point at the list
        MOVE.B  #5,D0       ;Use D0 as a loop counter - set it to 5 fir 5 numbers
        CLR.B   D1          ;Clear the total in D1 before we start
Loop    ADD.B   (A0),D1     ;Add the number pointed at by A0 to D1
        ADD.L   #1,A0       ;Point to the next number in the list by incrementing A0 (the pointer)
        SUB.B   #1,D0       ;Decrement the counter
        BNE     Loop        ;If we haven't reached zero, go round again

        MOVE.B  #9,D0       ;This is the code for 'halt'
        TRAP    #15         ;Halt Simulator (the TRAP calls the O/S)

        ORG     $2000       ;Put the data here
List    DC.B    1,4,3,6,7   ;Here's the data to add.
        END     START
```

Click on Project Menu to get Assemble Source option.

EASy68K: Successful Assembly

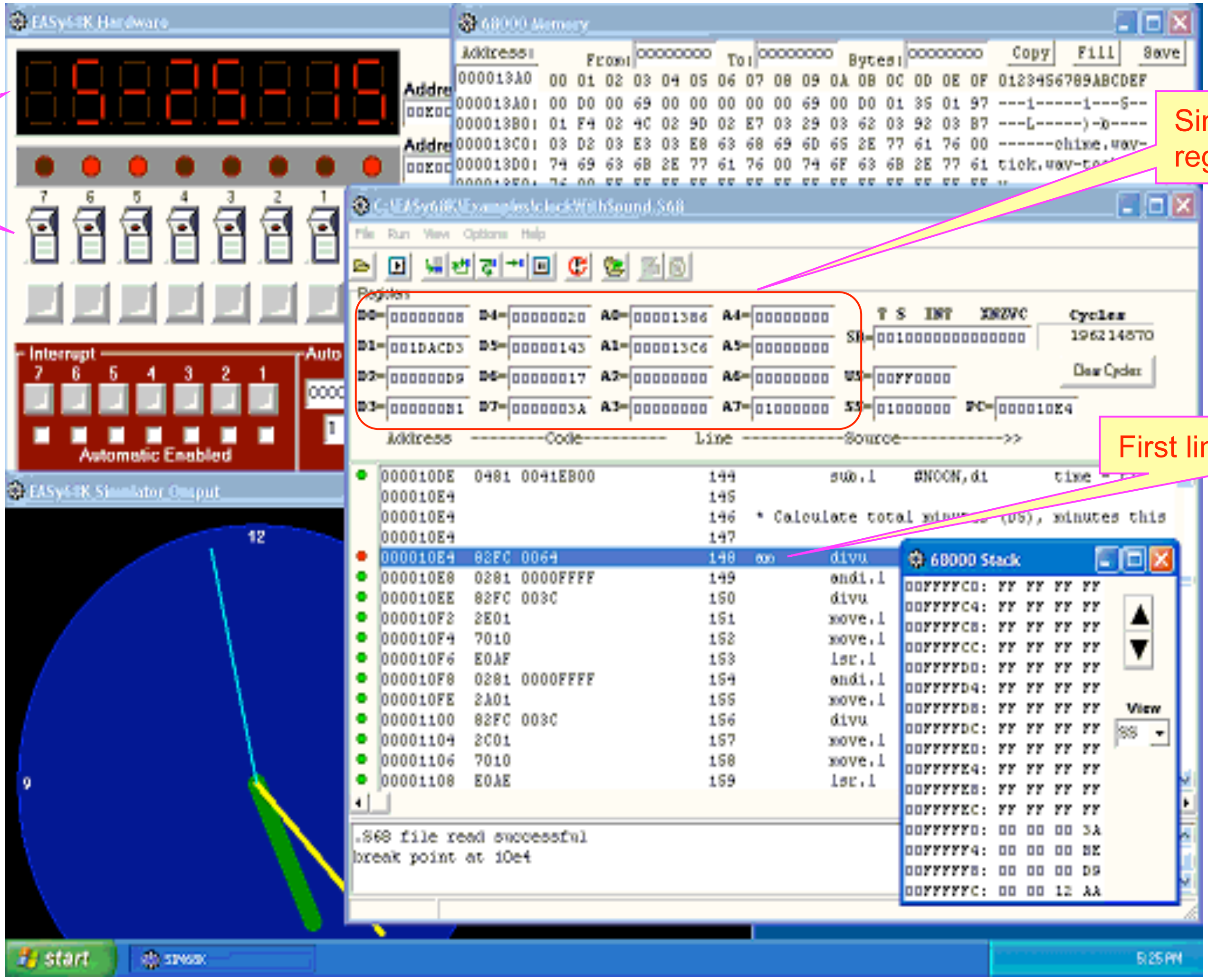


```
*-----------------------------------------------------------
* Program Number: 1
* Written by    : Alan Clements
* Date Created  : 13 October 2005
* Description   : Introduction to the 68K assembler
*
*-----------------------------------------------------------
START   ORG     $1000

        LEA     List,A0     ;A0 is set to point at the list
        MOVE.B  #5,D0       ;Use D0 as a loop counter - set it to 5 for 5 numbers
        CLR.B   D1          ;Clear the total in D1 before we start
Loop    ADD.B   (A0),D1     ;Add the number pointed at by A0 to D1
        ADD.L   #1,A0       ;Point to the next number in the list by incrementing A0 (the pointer)
        SUB.B   #1,D0       ;Decrement the counter
        BNE     Loop        ;If we haven't reached zero, go round again

        MOVE.B  #9,D0       ;This is the code for 'halt'
        TRAP    #15         ;Halt Simulator (the TRAP calls the O/S)

        ORG     $2000       ;Put the data here
List    DC.B    1,4,3,6,7   ;Here's the data to add.
        END     START
```

Warnings: 0
Errors: 0

Click on Execute to run your program.

Using EASy68K



```
Address:        From: 00000000   To: 00000000   Bytes: 00000000   Copy  Fill  Save
000013A0   00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 0123456789ABCDEF
000013A0:  00 D0 00 69 00 00 00 00 00 69 00 D0 01 35 01 97 ---i-----i---5--
000013B0:  01 F4 02 4C 02 9D 02 E7 03 29 03 62 03 92 03 B7 ---L-----)-b----
000013C0:  03 D2 03 E3 03 E8 63 68 69 6D 65 2E 77 61 76 00 ------chime.wav-
000013D0:  74 69 63 6B 2E 77 61 76 00 74 6F 63 6B 2E 77 61 tick.wav-toc
```

```
D0=00000008  D4=00000020  A0=00001386  A4=00000000       T S  INT  XNZVC     Cycles
D1=001DACD3  D5=00000143  A1=000013C6  A5=00000000  SR=0010000000000000      19621 4870
D2=000000D9  D6=00000017  A2=00000000  A6=00000000  US=00FF0000
D3=000000B1  D7=0000003A  A3=00000000  A7=01000000  SS=01000000  PC=000010E4
```

```
Address   --------Code---------   Line -----------Source------------>>
000010DE   0481 0041EB00          144          sub.l    #NOON,d1      time - 
000010E4                          145
000010E4                          146   * Calculate total minutes (D5), minutes this
000010E4                          147
000010E4   82FC 0064              148   00     divu
000010E8   0281 0000FFFF          149          andi.l
000010EE   82FC 003C              150          divu
000010F2   2E01                   151          move.l
000010F4   7010                   152          move.l
000010F6   E0AF                   153          lsr.l
000010F8   0281 0000FFFF          154          andi.l
000010FE   2A01                   155          move.l
00001100   82FC 003C              156          divu
00001104   2C01                   157          move.l
00001106   7010                   158          move.l
00001108   E0AE                   159          lsr.l
```

```
.S68 file read successful
break point at 10e4
```

```
00FFFFC0: FF FF FF FF
00FFFFC4: FF FF FF FF
00FFFFC8: FF FF FF FF
00FFFFCC: FF FF FF FF
00FFFFD0: FF FF FF FF
00FFFFD4: FF FF FF FF
00FFFFD8: FF FF FF FF
00FFFFDC: FF FF FF FF
00FFFFE0: FF FF FF FF
00FFFFE4: FF FF FF FF
00FFFFE8: FF FF FF FF
00FFFFEC: FF FF FF FF
00FFFFF0: 00 00 00 3A
00FFFFF4: 00 00 00 BE
00FFFFF8: 00 00 00 D9
00FFFFFC: 00 00 12 AA
```

Input/output devices

Simulated registers

First line of code