INTERNATIONAL ATOMIC ENERGY AGENCY
UNITED NATIONS EDUCATIONAL, SCIENTIFIC AND CULTURAL ORGANIZATION

INTERNATIONAL CENTRE FOR THEORETICAL PHYSICS
34100 TRIESTE (ITALY) - P.O.B. 586 - MIRAMARE - STRADA COSTIERA 11 - TELEPHONES: 224281/2/3/4/5/6
CABLE: CENTRATOM - TELEX 460392-I

SMR/101 - 5

SECOND COLLEGE ON MICROPROCESSORS: TECHNOLOGY AND APPLICATIONS IN PHYSICS

(18 April - 13 May 1983)

ASSEMBLER LANGUAGE PROGRAMMING FOR THE MOTOROLA 6809

H. von EICKEN
DD Division
CH 1211 Geneva 23
Switzerland

These are preliminary lecture notes, intended only for distribution to participants.
Missing or extra copies are available from Room 230.

HARDWARE and SOFTWARE ENVIRONMENT

Hardware:

Computer Terminal consisting of:

- a keyboard to enter programs, data, text, ... , etc
- a CRT display screen to visualize the data we entered and the messages we receive from the computer system we connect to
- an interface cable to connect the terminal to the computer system

Computer System consisting of:

- a matching interface to connect our terminal
- a high speed random access memory to store programs and their data
- a processing unit to control the hardware and to execute the program(s) stored in the random access memory
- a lower speed mass storage system (usually a disk) to more permanently store programs and their data
- a printer to obtain listings
- a "data transport" medium
  _ magnetic tape
  _ floppy disk or disk cartridge
  _ network connection

HARDWARE and SOFTWARE ENVIRONMENT (continued)

Resident - Software:

Operating System providing:

- Input / Output drivers to handle the peripheral equipment like keyboard, printer, disk, ... , etc
- Command interpreter and program scheduler
- File manager to retrieve, update and store information on the mass storage device
- Text Editor and utilities for file copy, transfer, etc.

Language system providing:

- Language processors like:
  _ Macro assembler
  _ Compiler(s) (Fortran, Pascal, Modula-2, Ada, ... , etc)
  _ Interpreters (Basic, intermediate languages)
- Library of run time support routines (Fortran library, Pascal library, etc)
- Linkage editor and loader
- Debugging aids

HARDWARE and SOFTWARE ENVIRONMENT (continued)

Cross - Software:

Software package that executes on a computer, usually called a **Host Computer** preparing code to execute on another computer, usually called the **Target Computer**

At CERN this technique is widely used to provide support on different hosts for a variety of different target processors:

- Hosts are:

  CDC Cyber series;
  DEC VAX (VMS or Unix);
  IBM 370 series;
  NORD computers;
  Siemens 7800 series;

- Targets are:

  Intel 8080, 8085;
  Motorola 6800, 6801, 6809, 68000;
  Texas Instruments 9900, 99000;

The following language processors are provided:

- Assemblers for all targets.
- Modula-2 for M 6809, M 68000, TMS 9900, TMS 99000
- Pascal for M 68000

All language processors produce CUFOM, the Cern Universal FOrmat for Object Modules. Consequence:

- one linkage editor can handle code for all targets
- one librarian can handle code for all targets
- each target format however needs its own pusher

HARDWARE and SOFTWARE ENVIRONMENT (continued)

Cross - Software (continued):

Why does one use cross software?

- Host computer is usually a time shared system providing simultaneous access for many computer users (cost effective)
- familiarity with existing tools (command language, text editor, filing system, etc. , **no new learning effect**)
- the filing system usually provides automatic backup and allows sharing of libraries for target computers
- host offers high speed printing facility
- cross software tools are usually written in a higher level language and may therefore be more easily "transported" from one host system to another one

Anything against cross software?

- a large host computer is expensive
- if the central host computer is overloaded, one might have to wait very long to have even small assemblies done
- everything must be down-line loaded from the host into the target
- some people like to hide and this is much easier, if they have there own little pet system

First programming problem: Add the first 15 integers

Constraints:

- the integer 15 with which to begin the calculation is kept in a memory location
- the result should be stored in another memory location
- the program should start at location 0400 hex and return to the system monitor
- accumulator A is free for use

Possible solutions: (expressed in a Pascal like syntax)

```
VAR
  Count  : INTEGER;          {to count the repetitions}
  NVal   : INTEGER;          {the initial value 15}
  Sum    : INTEGER;          {to calculate the sum}
```

```
⇒                            { WHILE <condition> DO <body> }
  Sum:= 0; Count:= 0;
  WHILE Count < NVal DO BEGIN
    Count:= Count + 1;
    Sum:= Sum + Count;
  END;
```

```
⇒                            { REPEAT <body> UNTIL <condition> }
  Sum:= 0; Count:= NVal;
  REPEAT
    Sum:= Sum + Count;
    Count:= Count - 1;
  UNTIL Count = 0;
```

```
⇒                            { FOR <iterative condition> DO <body> }
  Sum:= 0;
  FOR Count:= 1 TO NVal DO
    Sum:= Sum + Count;
```

First programming problem: Add the first 15 integers (continued)

Let's try to hand_code the REPEAT -- UNTIL construct:

```
⇒                            { REPEAT <body> UNTIL <condition> }
  Sum:= 0; Count:= NVal;
  REPEAT
    Sum:= Sum + Count;
    Count:= Count - 1;
  UNTIL Count = 0;
```

location	contents	opcode	address mode	comment
0400	4F	CLRA	inherent	Sum:= 0;
0401	B7 201B	STA	extended	
0404	B6 201C	LDA	extended	Count:= NVal;
0407	B7 201A	STA	extended	REPEAT
040A	BB 201B	ADDA	extended	Sum:= Sum + Count;
040D	B7 201B	STA	extended	
2010	B6 201A	LDA	extended	Count:= Count - 1;
2013	8B FF	ADDA	immediate	
2015	26 F0	BNE	relative	UNTIL Count:= 0;
2017	3F 00	MON 0	system call	return to monitor
201A			location to keep index	
201B			location to keep sum	
201C	0F		location containing initial value	

Are there any problems with this approach?

- easy to make mistakes and nobody checks!
- address calculation and allocation is tedious
- just imagine you had made an error!

Assembler Language:

The assembler language provides a means to create a computer program. The goals of such a language are programs that are:

- easier to create
- easier to modify
- easier to read and understand
- translated into a machine readable load module

What are the features of such a language?

- symbolic machine operation codes (mnemonics)
- symbolic address assignments and references
  - instruction addresses
  - operands
  - operand addresses
- comments and remarks for program documentation
- relative addressing
- storage reservation and data creation
- expression handling
- assembler directives

What else does an assembler provide?

- listing of the source code including:
  - addresses and generated code
  - optional titles and subtitles
  - optional formatting
  - optional cross reference of all symbols
- detailed error diagnostics
- parameterized macro facility
- conditional assembly facility
- absolute and relocatable code in a format suitable for a linkage editor

M6800 M6801 M6805 M6809 MACRO ASSEMBLERS REFERENCE MANUAL
Motorola, M68MASR(D2), Second Edition, September 1979

Assembler Language Elements:

Identifiers:

- consists of 1 to 6 characters
- valid characters in an identifier are:
  "A" through "Z", "0" through "9", "." and "$"
- first character must alphabetic or "."

Note: Reserved Identifiers

_ "A", "B", "D"	- accumulators A, B and D (A, B concatenated)
_ "X", "Y "	- index registers X and Y
_ "U", "S"	- user and system stack pointer
_ "PC", "PCR"	- program counter
_ "CC"	- condition code register
_ "DP"	- direct page register

Constants:

- decimal constant, range 0 - 65535 inclusive, digits (0-9)

- hexadecimal constant, range 0000 - $FFFF inclusive, either:
  - prefixed by "$", followed by digits (0-9) and letters (A-F), or
  - postfixed with "H" and preceded by digits (0-9) and letters (A-F)
    first digit must be 0-9

- octal constant, range 0 - @177777 inclusive, either:
  - prefixed by "@", followed by digits (0-7), or
  - postfixed with "O" and preceded by digits (0-7)

- binary constant, range 0 - %1111111111111111 inclusive, either:
  - prefixed by "%", followed by digits (0-1), or
  - postfixed with "B" and preceded by digits (0-1)

- character constant, ASCII character prefixed with "'" (apostrophe)

Assembler Language Elements (continued):

Opcodes:

- an opcode is a mnemonic for a machine instruction (CLR, DEC, etc.)
- there is a one to one correspondence between opcodes and machine instructions
- the assembler verifies the opcode, its operands and generates the correct binary code for the load module

Assembler directives:

Assembler directives are instructions to the assembler. More commonly they are called: **pseudo opcodes.** According to their function they may be grouped as follows:

- module identification (NAM, END)
- section control (ASCT, BSCT, COMM, CSCT, DSCT, ORG, PSCT)
- symbol definition (EQU, REG, SET)
- module linkage (XDEF, XREF)
- data generation, storage reservation (BSZ, FCB, FCC, FDB, RMB)
- object code control (OPT, SETDP)
- macro definition (MACRO, NARG, ENDM)
- conditional assembly (ENDC, FAIL, IFC, IFNC, IFEQ, IFGE, IFGT, IFLE, IFLT, IFNE)
- listing control (OPT, PAGE, SPC, TTL)

Assembler Language Elements (continued):

Label symbols: A label symbol is an identifier that specifies a value and its associated attributes. The assembler has four types of label symbols:

Absolute Symbol:

- the symbol is equated (EQU) or SET to an absolute value
- the symbol is defined in the absolute section of the program
- its value is unaffected by any possible future applications of the link-editor to the module

Relative Symbol:

- the symbol is equated (EQU) or SET to a relative symbol
- the symbol is defined in a relative section of the program
- its value is affected by future applications of the link-editor to the module

External Symbol:

- the symbol is listed as parameter in an XREF pseudo instruction
- the symbol is not defined in the current assembly module
- its value is set to zero and must be defined during a subsequent link-editor run

Undefined Symbol:

- the symbol is not defined in the current assembly and not listed as parameter in an XREF pseudo instruction
- the occurrence of such a symbol is indicated as an error

Assembler Language Elements (continued):

Expressions:

- an expression is a combination of symbols, constants, algebraic operators, and parentheses
- an expression specifies a value which is to be used as an operand
- expressions follow the conventional rules of algebra
- operators are:
  - multiplication "*", division "/"
  - addition "+", subtraction "-"
  - exponentiation "!-"
  - logical AND "!.", inclusive OR "!+", exclusive OR "!X"
  - shift left "!<", shift right "!>"
  - rotate left "!L", rotate right "!R"
- expressions may contain relocatable or externally defined symbols but:
  - relative symbols cannot be multiplied, divided or operated on with the special two-character operators
  - a relative or external symbol may have an absolute value added to or subtracted from it, the result is relative
  - a relative symbol may be subtracted from another relative symbol provided they are both defined in the same section, the result is absolute

Assembler Source Statement Format:

Each source statement for the assembler may include up to four fields:

Label field:

The label field starts in column one of the line. If the line starts with:

- a "*" (star), then the line is treated as a comment line
- a " " (space), then the label field is empty
- with an identifier, then this identifier is called a label symbol
  - an identifier may occur only once in the label field (except with SET directive)
  - a label symbol is assigned the value of the current program counter (except for some directives like EQU, MACR, REG AND SET)
- for any line starting differently an error indication is given

Operation field:

The operation field occurs after the label field preceded by at least one space and it must contain a symbol of one of the following three types:

- opcode (machine instruction mnemonic)
- pseudo (assembler directive)
- macro call (evaluated macro body inserted in place of call)

Assembler Source Statement Format (continued):

Operand field:

Follows operation field preceded by at least one space. Interpretation is dependent on contents of operation field. For opcodes it is as follows:

operand format	M 6809 addressing mode
no operand	accumulator and inherent
<expression>	direct, extended or relative
<<expression>	forced direct
><expression>	forced extended
[<expression>]	extended indirect
<expression>,R	indexed
<<expression>,R	8-bit offset indexed
><expression>,R	16-bit offset indexed
<accumulator>,Q	accumulator offset indexed
[<expression>,R]	indexed indirect
<[<expression>,R]	8-bit offset indexed indirect
>[<expression>,R]	16-bit offset indexed indirect
[<accumulator>,Q]	accumulator offset indexed
Q+	auto increment by 1
Q++	auto increment by 2
[Q++]	auto increment indirect
-Q	auto decrement by 1
--Q	auto decrement by 2
[--Q]	auto decrement indirect
#<expression>	immediate
<register list>	immediate

with: R = S | U | X | Y | PC | PCR
and : Q = S | U | X | Y

Comment field:

Last field of an assembler source statement, separated by at least one blank from the preceding field, may contain any printable ASCII character.
Note: It is essential to comment the flow of a program!

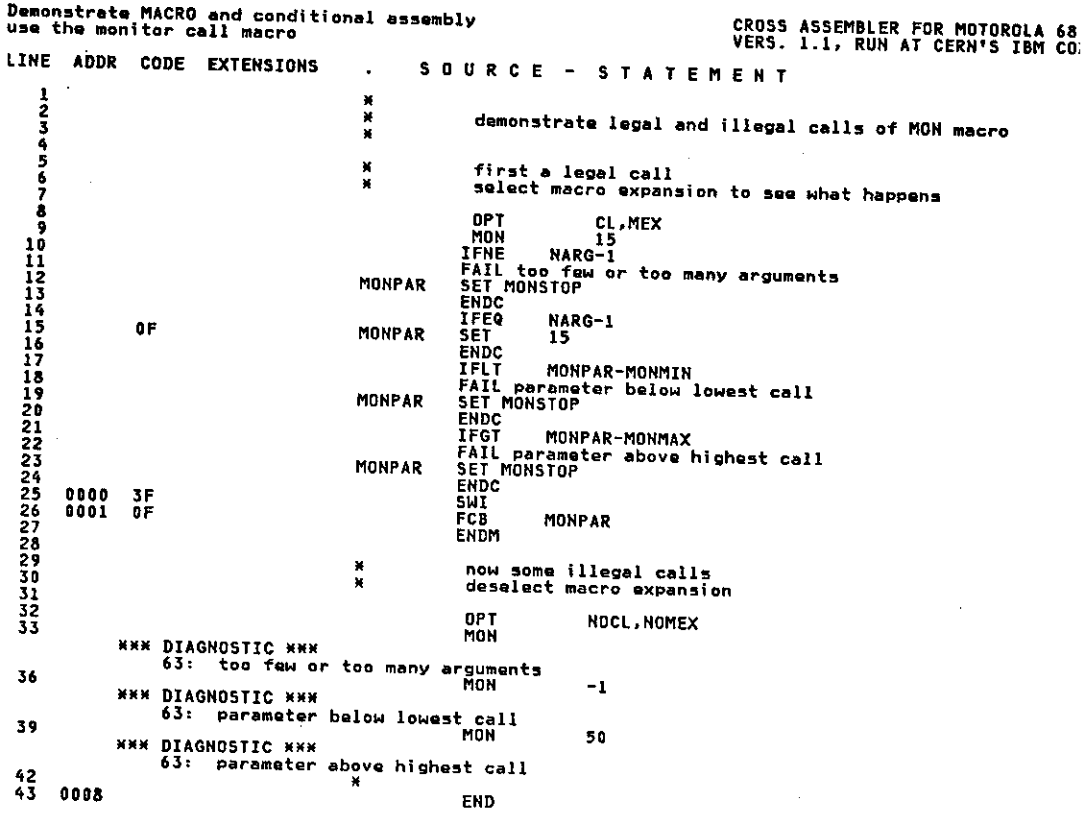
Macro definition - Conditional assembly:

MON - macro for **ROSY** monitor requests

```
label     operation  operand(s)       comment field
*
*         define range of monitor requests
*
MONMIN    EQU        0                lowest monitor request
MONMAX    EQU        46               highest monitor request
MONSTOP   EQU        1                monitor request to stop execution
          SPC        3
*
*         define macro to handle monitor requests
*
MON       MACR
*         assert we have one and only one parameter
          IFNE       NARG-1
          FAIL       too few or too many arguments
MONPAR    SET        MONSTOP
          ENDC
          IFEQ       NARG-1
MONPAR    SET                         0
          ENDC
*         assert parameter is valid
          IFLT       MONPAR-MONMIN
          FAIL       parameter below lowest call
MONPAR    SET        MONSTOP
          ENDC
          IFGT       MONPAR-MONMAX
          FAIL       parameter above highest call
MONPAR    SET        MONSTOP
          ENDC
*         generate the monitor call
          SWI
          FCB        MONPAR
          ENDM
```

Macro definition - Conditional assembly: (continued)

Demonstrate use of MON macro:



```
Demonstrate MACRO and conditional assembly
use the monitor call macro                                   CROSS ASSEMBLER FOR MOTOROLA 68
                                                             VERS. 1.1, RUN AT CERN'S IBM CO

LINE  ADDR  CODE  EXTENSIONS     .    S O U R C E  -  S T A T E M E N T

   1                             *
   2                             *       demonstrate legal and illegal calls of MON macro
   3                             *
   4
   5                             *       first a legal call
   6                             *       select macro expansion to see what happens
   7
   8                                     OPT        CL,MEX
   9                                     MON        15
  10                                    IFNE    NARG-1
  11                                    FAIL too few or too many arguments
  12                             MONPAR SET MONSTOP
  13                                    ENDC
  14                                    IFEQ    NARG-1
  15          0F                 MONPAR SET     15
  16                                    ENDC
  17                                    IFLT    MONPAR-MONMIN
  18                                    FAIL parameter below lowest call
  19                             MONPAR SET MONSTOP
  20                                    ENDC
  21                                    IFGT    MONPAR-MONMAX
  22                                    FAIL parameter above highest call
  23                             MONPAR SET MONSTOP
  24                                    ENDC
  25  0000  3F                          SWI
  26  0001  0F                          FCB     MONPAR
  27                                    ENDM
  28
  29                             *       now some illegal calls
  30                             *       deselect macro expansion
  31
  32                                     OPT        NOCL,NOMEX
  33                                     MON
        *** DIAGNOSTIC ***
            63:  too few or too many arguments
  36                                     MON        -1
        *** DIAGNOSTIC ***
            63:  parameter below lowest call
  39                                     MON        50
        *** DIAGNOSTIC ***
            63:  parameter above highest call
  42                             *
  43  0008                               END
```

First programming problem: Add the first 15 integers (revisited)

Now let's write the REPEAT -- UNTIL construct in M 6809 assembly language:

```
     label     operation   operand(s)     comment field

⇒              NAM         INTSUM         identifies the assembly unit
     *
     *         This program will add the first 15 integers.
     *
     *         On entry:
     *                     accumulator A is free for use
     *                     the integer 15 is stored in location -NVAL-
     *                     the program should start at location hex 400
     *
     *         On exit:
     *                     location -SUM- contains the calculated sum
     *
     M.RET     EQU         0              monitor return function code

⇒              ORG         $400           set program counter to $400
     START     CLRA                       clear -SUM-
               STA         SUM
               LDA         NVAL           load initial loop value
     LOOP      STA         COUNT          save value of loop count
               ADDA        SUM            add loop count to SUM
               STA         SUM            and save result
               LDA         COUNT          re-load loop count
               ADDA        #-1            decrement it
               BNE         LOOP           repeat until count is zero
⇒              MON         M.RET          return to monitor system
     *         declare variables, reserve and preset memory
⇒    IVAL      EQU         15             initial loop value is 15
⇒    COUNT     RMB         1              to contain the loop count
     SUM       RMB         1              to contain the final sum
⇒    NVAL      FCB         IVAL           create initial value
⇒              END         START          set program entry point
```