INTERNATIONAL ATOMIC ENERGY AGENCY
UNITED NATIONS EDUCATIONAL, SCIENTIFIC AND CULTURAL ORGANIZATION
INTERNATIONAL CENTRE FOR THEORETICAL PHYSICS
I.C.T.P., P.O. BOX 586, 34100 TRIESTE, ITALY, CABLE CENTRATOM TRIESTE

H4.SMR. 403/16

FIFTH COLLEGE ON MICROPROCESSORS: TECHNOLOGY AND APPLICATIONS IN PHYSICS

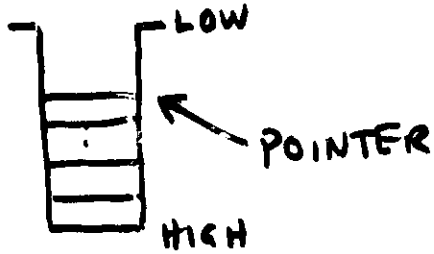
2 - 27 October 1989

Assembly Language Programming

Nii N. QUAYNOR
Digital Equipment Corporation
Marlboro, U.S.A.

STACKS

- DATA STRUCTURE IN MEMORY
- LAST COME FIRST SERVE
  - ACCESS RESTRICTED TO TOP
- POINTER TO TOP
- S, U, X, Y, MEMORY AS POINTERS



- USED FOR SUBROUTINES
  TEMPORARIES
  SCOPE VARIABLE ETC.

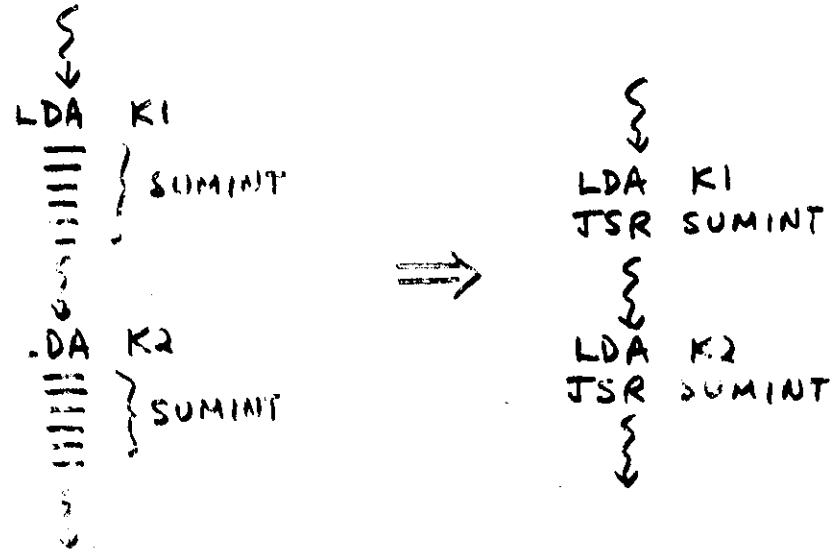
X, Y - STACKS

```
          ORG   $1000
STACK     RMB   $100      STACK DEFINITION
STACKBS   EQU   *
          :
          LDX   #STACKBS   INITIALIZATION
          :
PUSHAX    EQU   *
          LEAX  -1,X       PUT ON X-STACK
          STA    0,X
          :
PULLAX    EQU   *
          LDA   0,X        TAKE FROM STACK
          LEAX  1,X
          :
```

MOTIVATION FOR SUBROUTINES

- USE WHEN HAVE CODE USED REPEATEDLY
- EXAMPLE

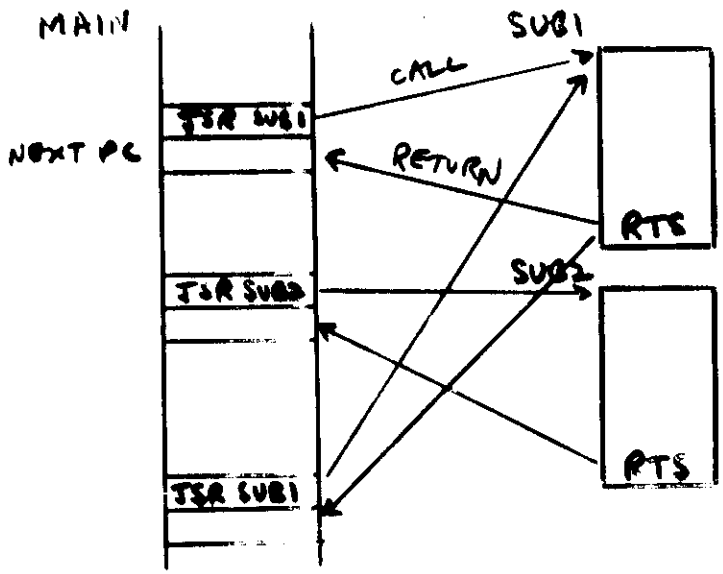
$$\text{SUMINT} = \sum_{i=K}^{K+15} i$$

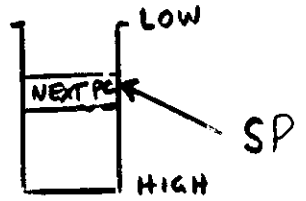


- ADVANTAGE
  - SAVE PROGRAM SPACE
  - LESS CHANCE OF ERROR (MODULA)
  - CAN SHARE SUBROUTINES

JSR EA : SP ← SP−1
(SP) ← PC
PC ← EA

RTS : PC ← (SP)
SP ← SP+1





S-STACK

=> 24,000 inst!

Type of subroutine parameters:

--> VALUE parameter

"value" of the parameter is passed

Example:

numbers, characters, value of expression

restricted to be an "input only" parameter for the subroutine

--> VARIABLE parameter

"address" of the parameter is passed

Example:

address of a variable, start address of an array

"input" and / or "output" parameter for the subroutine

Call a subroutine to initialize an array of a given size to zero:

```
CALL  Init ( ArrayAddress, SizeValue )
```

Type of subroutine parameters:

--> Subroutine of Function parameter

the address of a subroutine or a function is passed to the subroutine

Example:

Assume we want to write a subroutine integrating a function over interval [A, B]

The subroutine therefore must obtain function values f(x) for values of x in the interval [A, B]

```
CALL Integrate ( FunctionAddress,
                 ValueA, ValueB,
                 ResultAddress     )
```

* only the address of the function is passed

* the parameter definition for the function is unknown !

Ways to pass parameters to subroutines:

--> processor registers

* fast and simple

* limited by the number of registers

(rewrite fourth programming problem using registers X and Y for input of the interval boundaries and register X to return the result)

--> dedicated memory locations (mail box)

* associated with the subroutine

* calling program places input in and retrieves output from the mailbox

* address has to be known by caller

* mailbox must be in RAM (coordination problems)

This method was used in our fourth programming problem !

Ways to pass parameters to subroutines:

--> parameter areas

* associated with the calling program

* base address is passed to subroutine using a register

* possible to use it for different subroutines, hence its size must be big enough to hold the largest number of parameters

* parameter area must reside in RAM (coodination problems)

* "in-line" parameter area is a special case (i.e. return address is address of first parameter)

Example:

```
        BSR     Init    initialise an array
        FDB     Bin     address of array Bin
        FCB     5       size of array Bin
        ---             next instruction
```

Note: Should only be used, if the parameters are "constant"

Ways to pass parameters to subroutines:

--> stack-oriented parameter passing

* parameters are "pushed" on the system stack prior to calling the subroutine
* dynamic allocation (stack overflow !)
* no worry about RAM coordination

PROBLEMS:

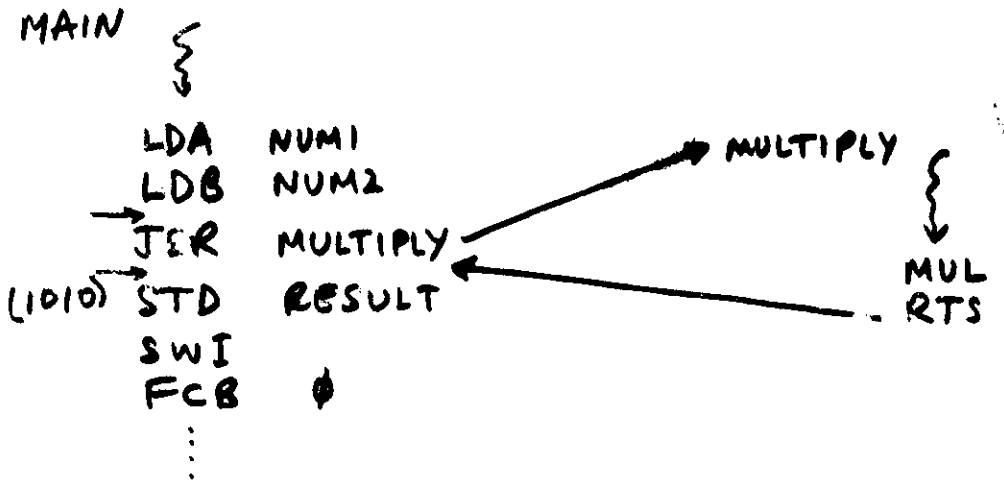
* order of pushing must be defined
* space for output parameters
* addressing of parameters
* who "cleans" the stack ?

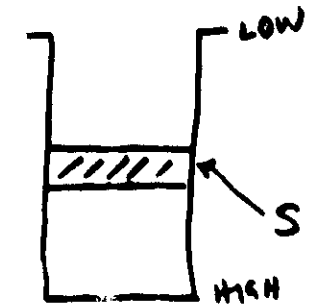
Example:

```
LDA     #5      push size of Bin
STA     -S
LDX     #Bin    push address of Bin
STX     --S
BSR     Init    get it initialized
LEAS    3,S     remove parameters
```

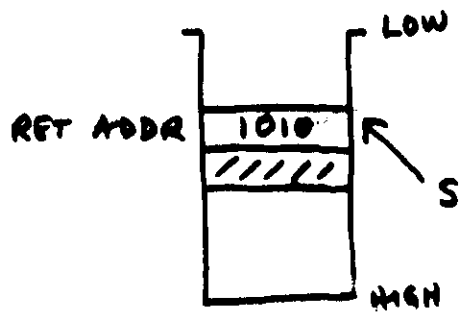

SUBROUTINES SUMMARY

PARAMETER PASSING IN REGISTERS



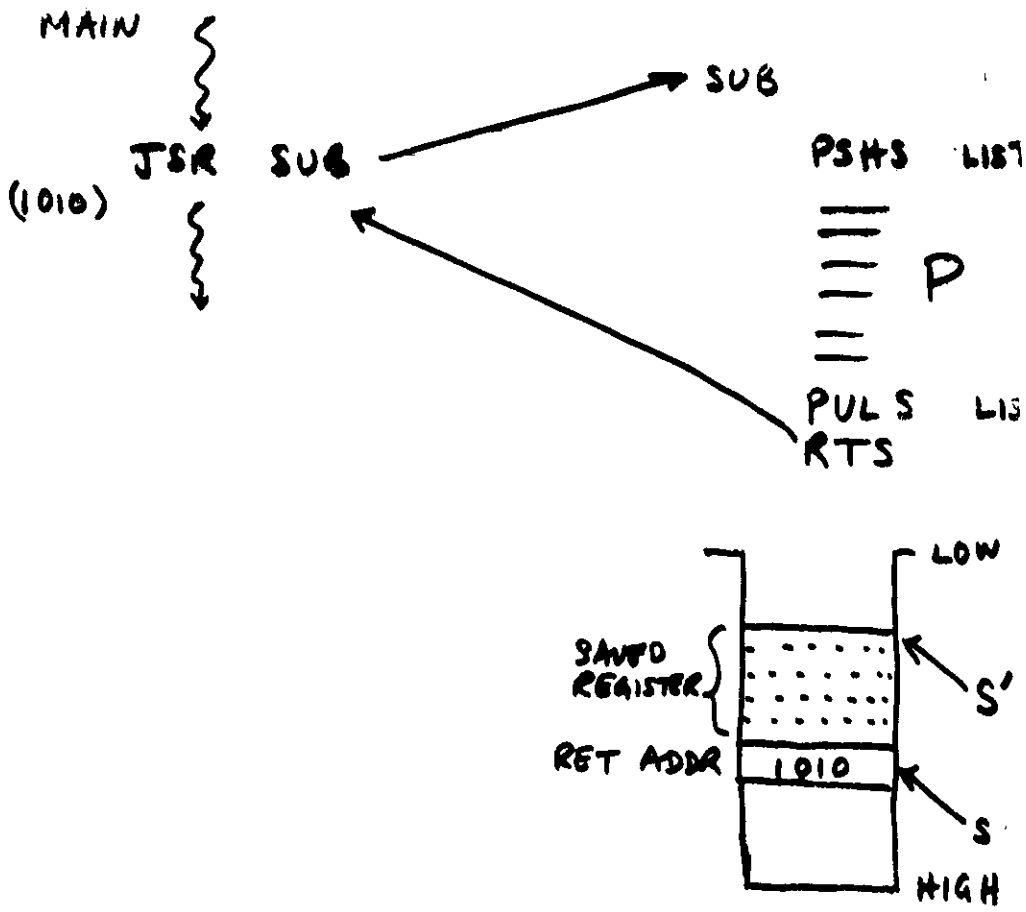


BEFORE JSR/
AFTER RTS

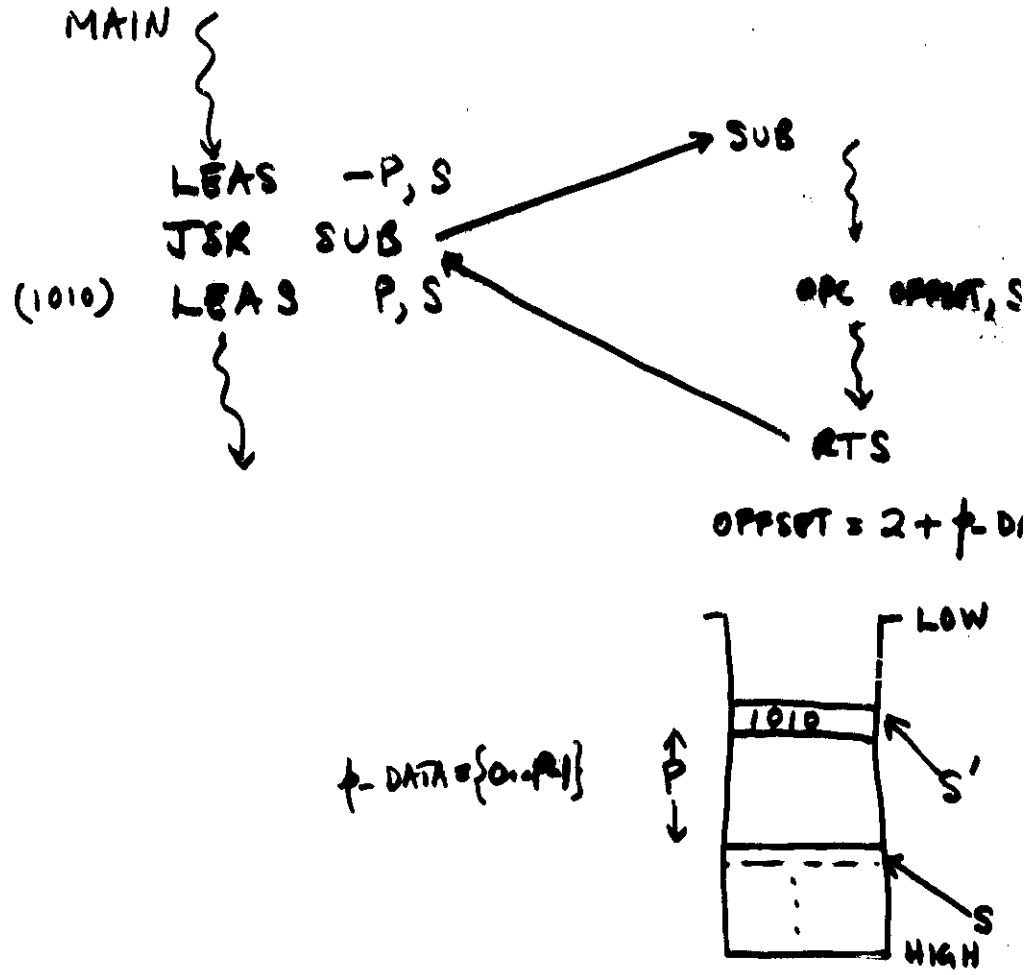


INSIDE
SUBROUTINE

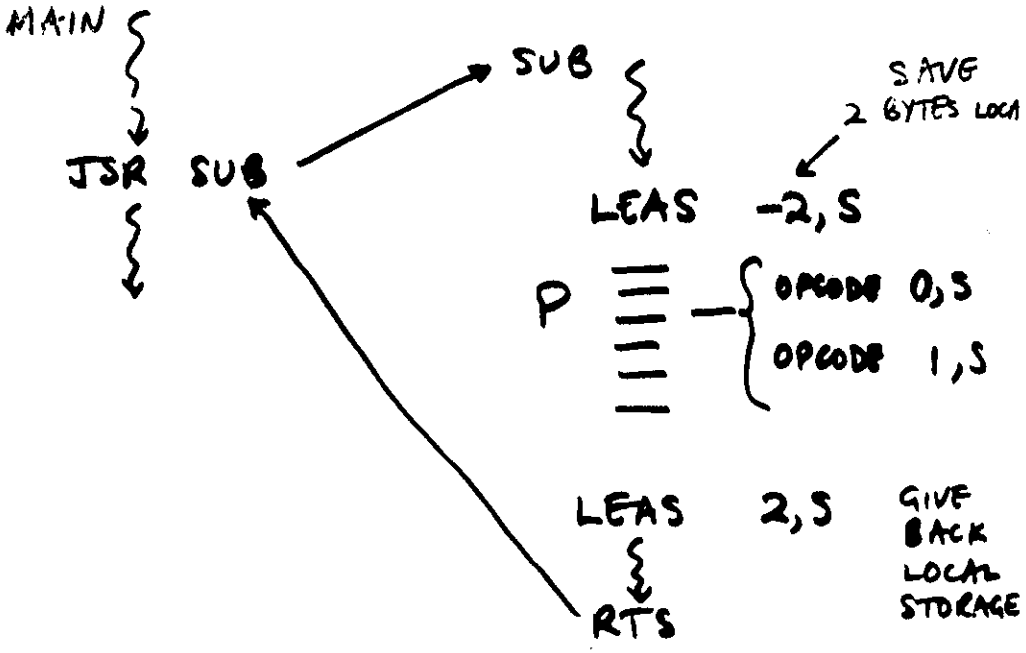
SAVING REGISTERS

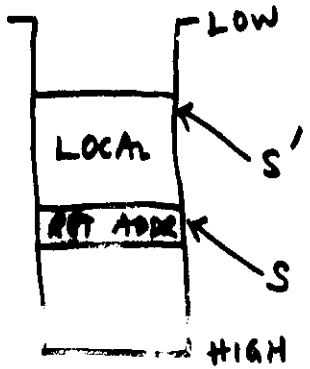


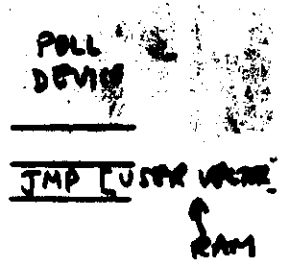
PARAMETERS ON S-STACK



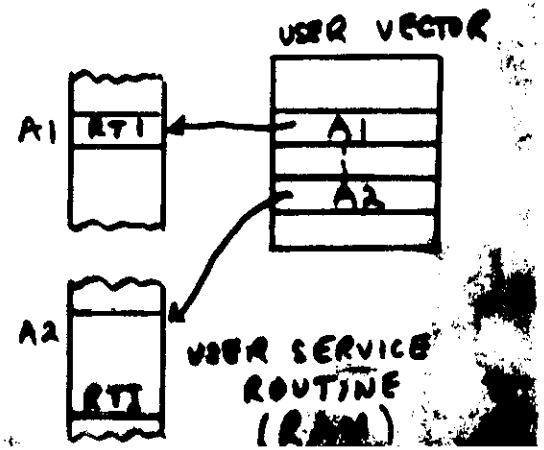
LOCAL STORAGE IN SUBROUTINE





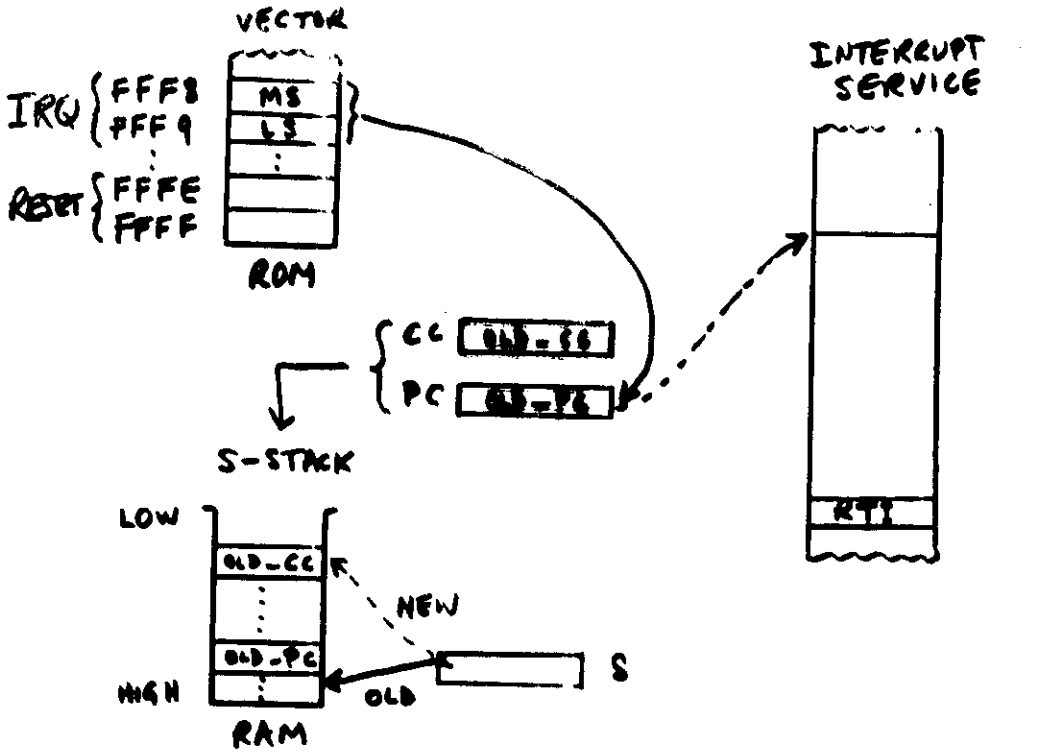


ROSY (ROM)



HW SUBROUTINE CALL – INTERRUPT

JSR [IRQ] + PUSH REGS



NMI: (STACK) ← ALL

IRQ: (STACK) ← ALL

FIRQ: (STACK) ← PC, CC

RECURSION

PROCEDURE FACT(N)

IF N=1 THEN FACT := 1

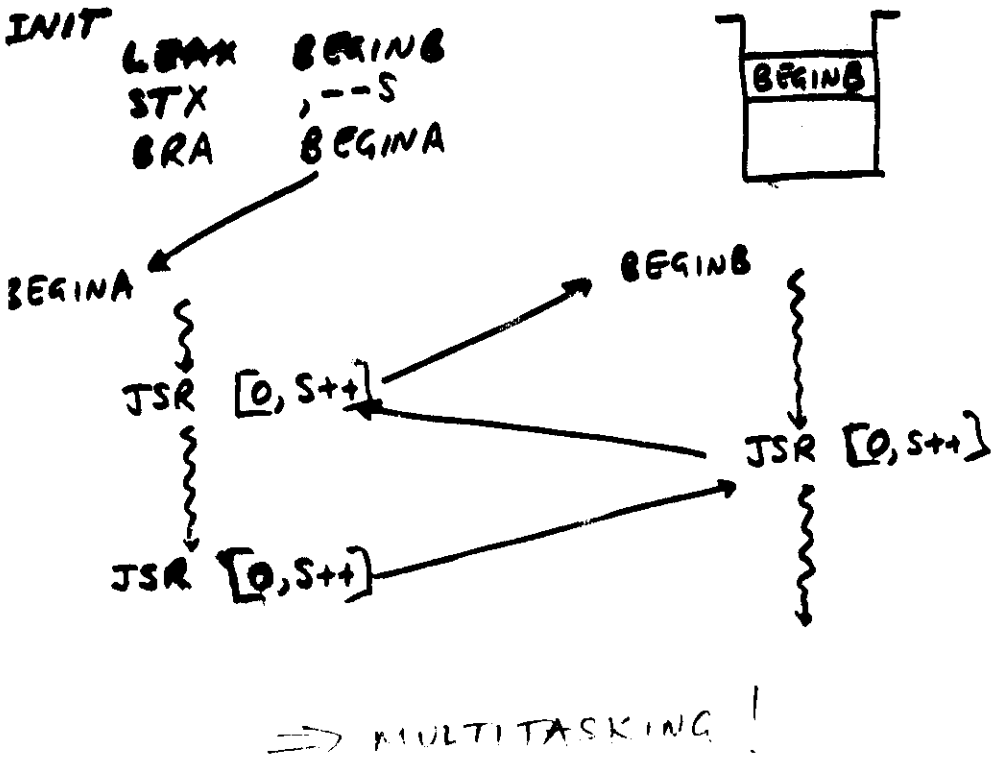
ELSE FACT := N * FACT(N-1)

* 8-bit NUMBERS

```
MAIN
        LDB  #3
        STB  0,-S          FACT(N)
        JSR  FACT
a  →    LDA  0,S+
        MUL
        STB  RESULT
        SWI
        FCB  ϕ
*
FACT
        LDB  2,S           IF N>1
        CMPB #1            THEN
        BEQ  TERMF
        DECB               FACT(N-1)
        STB  0,-S
        JSR  FACT
b  →    LDA  0,S+
        MUL
        RTS
TERMF   LDB  #1            ELSE
                           FACT = 1
```

COROUTINES

- SIMILAR TO SUBROUTINES
- BOTH ROUTINES ARE SUBROUTINES TO EACH.
- A SCHEDULING MECHANISM.



⇒ MULTITASKING !

POSITION INDEPENDENT PROGRAMMING

- RATIONAL
  - COPY PROGRAM TO ANY ADDRESS
- ADVANTAGE
  - PORTABLE
  - USEFUL FOR ROM USAGE
- CHARACTERISTICS
  - SEPARATION OF PROGRAM/DATA

⇒ 1. NO ABSOLUTE ADDRESSES
2. VARIABLES ON U-STACK
3. CONSTANTS IN PROGRAM.

POSITION INDEPENDENT PROGRAM (MODULO)

```
          ORG  ϕ
C         RMB  1          PROGRAM
DD        RMB  1          VARIABLES
E         RMB  1
SIZE      EQU  *
          ORG  $200
START     EQU  *
          LEAU -SIZE,U     GET SPACE
          LDA  C,U         ON U-STACK
          ADDA DD,U
          STA  E,U
WHL       EQU  *
          CMPA N,PCR
          BLO  EXIT
          SUBA N,PCR
          BRA  WHL
EXIT      EQU  *
          LEAU SIZE,U      RETURN
          SWI              SPACE ON
          FCB  ϕ           U-STACK
N         FCB  $8
          END
```


U-STACK (TEMPORARY VARIABLES)

```
          ORG  ϕ
CWD       RMB  2
XYZ       RMB  1
UVW       RMB  4          PROGRAM VARIABLES
          :                 ON U-STACK
SIZE      EQU  *
BUF       EQU  $E264      CONSTANTS
FP        EQU  $ϕC
          ORG  $200
START     EQU  *
          LEAU -SIZE,U
            :
          LDX  CWD,U      PROGRAM
          STA  XYZ,U
            :
EXIT      LEAU SIZE,U
          SWI
          FCB  ϕ
            :
```

ABSOLUTE ADDRESS PROGRAM

(MODULO ARITH)

```
        ORG  $200
START   LDA  C        E := C+D
        ADDA D
        STA  E

WHL     EQU  *        WHILE B DO
        CMPA N
        BLO  EXIT

        SUBA N        E := E-N
        BRA  WHL      END WHILE
EXIT    STA  E        SAVE E
        SWI
        FCB  0

C       RMB  1
DD      RMB  1
E       RMB  1
N       FCB  $8
        END
```

POSITION INDEPENDENT CODE

(RELATIVE TRANSFER VECTORS)

[CASE]

```
        ORG  $200
MAIN    EQU  *
        :
        :
        LEAX TABLE,PCR
        LDD  B,X          CALLING
        LEAX MAIN,PCR     SEQUENCE
        JMP  D,X

TABLE   EQU  *
        FDB  SUB1-MAIN    RELATIVE
        FDB  SUB2-MAIN    VECTOR
        :
        :
SUB1    EQU  *
        :                 ROUTINES
SUB2    EQU  *
        :
        :

        END
```

Macro - definition and use:

* macro assigns a name to an instruction sequence

* use of the name causes assembler to insert the instruction sequence into your program

--> TwoComp, a macro to compute two's complement of accumulator D

* macro definition:

```
TwoComp     MACRO

            COMB              complement low byte
            COMA              complement high byte
            ADDD    #1        now two's complement

            ENDM
```

* use of macro:

```
            LDD     Value1    to be complemented
            TwoComp
            STD     Value2    two's complement
```

Macro and parameters:

* macros allow parameter substitution

* parameters are indicated in the body by:

  &1, &2, ..., &9

  thus allowing up to 9 parameters

* these 'formal' parameters are replaced by the 'actual' parameters used in the macro call

```
  MACNAM    <par.1>,<par.2>, ... , <par.9>
```

NOTE:

* macros must be defined before they are used

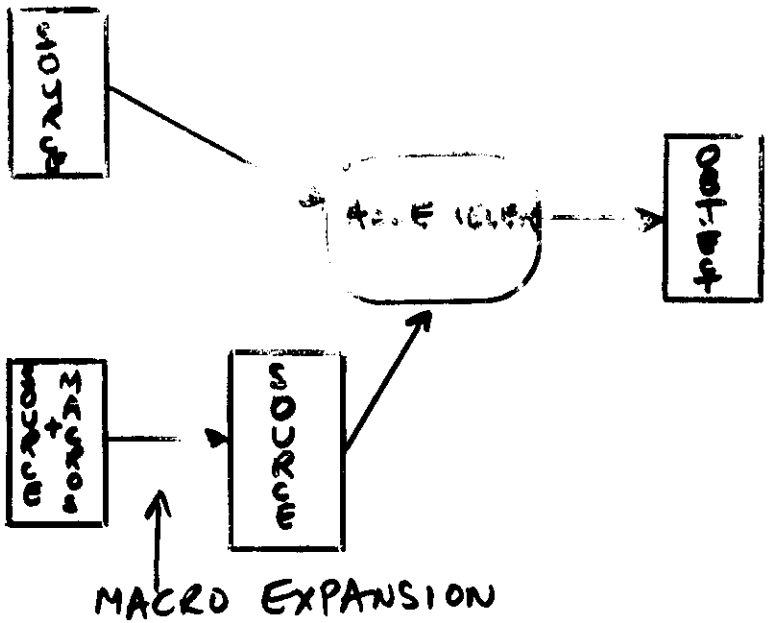
* local labels are not supported !

* number of arguments used in a macro call is not available

SEE:

chapter VII of TSC 6809 Assembler manual for more details ...

MACROS



- REPEATED SEQUENCE OF STATEMENTS (VARIANTS)
- TEXTUAL SUBSTITUTION
- EXISTS ONLY BEFORE ASSEMBLY
- NOT A SUBROUTINE (MACROS HAVE NO LIFE AT EXECUTION TIME)

Define macro with parameters:

--> TwoComp compute 2's complement of first argument and store result into second argument

```
* macro definition

TwoComp     MACRO

            LDD     &1      number to complement
            COMB            complement low byte
            COMA            complement high byte
            ADDD    #1      now two's complement
            STD     &2      store 2's complement

            ENDM

* use of macro now:

            TwoComp Value1, Value2
```

NOTE:

* we expect to obtain two memory addresses !
* there is an error in the macro !

MACRO EXAMPLE

```
TEST    MACRO
        LDA   #$ &1
        LDB   L &1
        NOP
        NOP
        NOP                    COMMENTS &2
        &3
&4      TST   M&1M
        ENDM


        TEST  1000, 'LIKE THIS', SEX, "LABEL"

        LDA   #$1000
        LDB   L1000
        NOP
        NOP
        NOP                    COMMENTS LIKE THIS
        SEX
LABEL   TST   M1000M
```

Macro definition - Conditional assembly:

```
    MON - macro for Rosy monitor requests

    * define range of monitor requests

MonMin      EQU     0       lowest request
MonMax      EQU     46      highest request
MonStop     EQU     1       stop execution
            SPC     3

    * define macro to handle monitor requests

MON         MACRO
MonPar      SET     &1      get request number

    * assert parameter is valid
            IF      MonPar < MonMin
            ERR             too small for Rosy
MonPar      SET     MonStop
            ELSE
            IF      MonPar > MonMax
            ERR             too large for Rosy
MonPar      SET     MonStop
            ENDIF
            ENDIF

    * generate the monitor call

            SWI
            FCB     MonPar
            ENDM
```

Macros - their advantage and disadvantage

Advantages of macros:

* shorter source programs
* once debugged --> code error free whenever used
* easier to implement changes
* user interface same, but macro body may change (useful in collaborations)
* extend or clarify instruction set

Disadvantages of macros:

* macro is expanded every time it is used may waste memory
* single macro may generate lot of instructions
* macro call mighthave hidden effects ... (registers, flags)

MONITOR MACROS

```
MON   MACRO
      SWI
      FCB  &1
      ENDM
```

```
START
        .
        .
        .
      MON  17        INCHW
        .
        .
        .
      MON  20        OUTCH
        .
        .
        .
      MON  26        PRINT
        .
        .
        .
      END
```

LIB MONCALLS

```
START
        .
        .
        .
      MON  INCHW
        .
      MON  PRINT
        .
        .
        .
      END
```