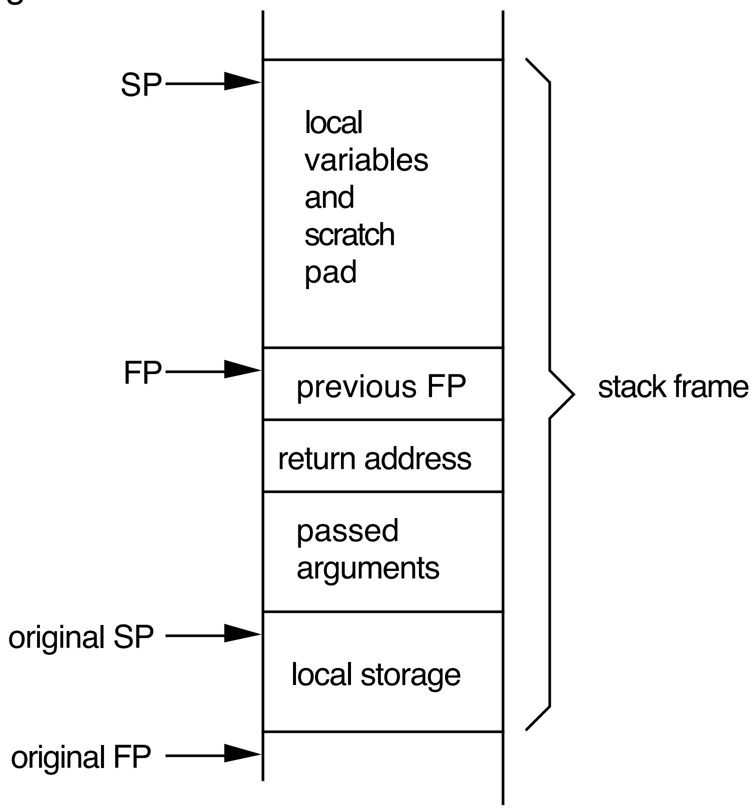
STACK FRAMES

The MC68000 provides two special instructions to allocate and deallocate a data structure called a <u>frame</u> in the stack to make subroutines easier to code.

general structure of a frame:

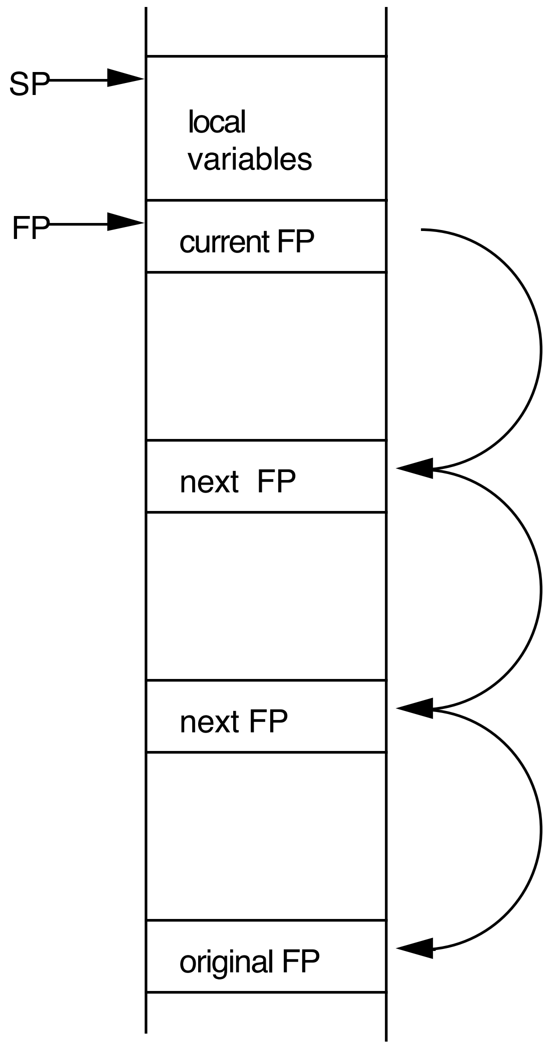


where register An is used as the argument pointer.

LINK	An,d	1. put An at -(SP) Example: decrement stack pointer and put A0 on the stack. 2. put SP into An Example: set A0 to point to this value. 3. change SP-d to SP, i. e. decrement the SP
UNLK	An	1. An $\rightarrow$ SP, change the value of the SP to that contained in An

2. (SP)+ $\rightarrow$ An, put that value on the stack into An and deallocate that stack space.

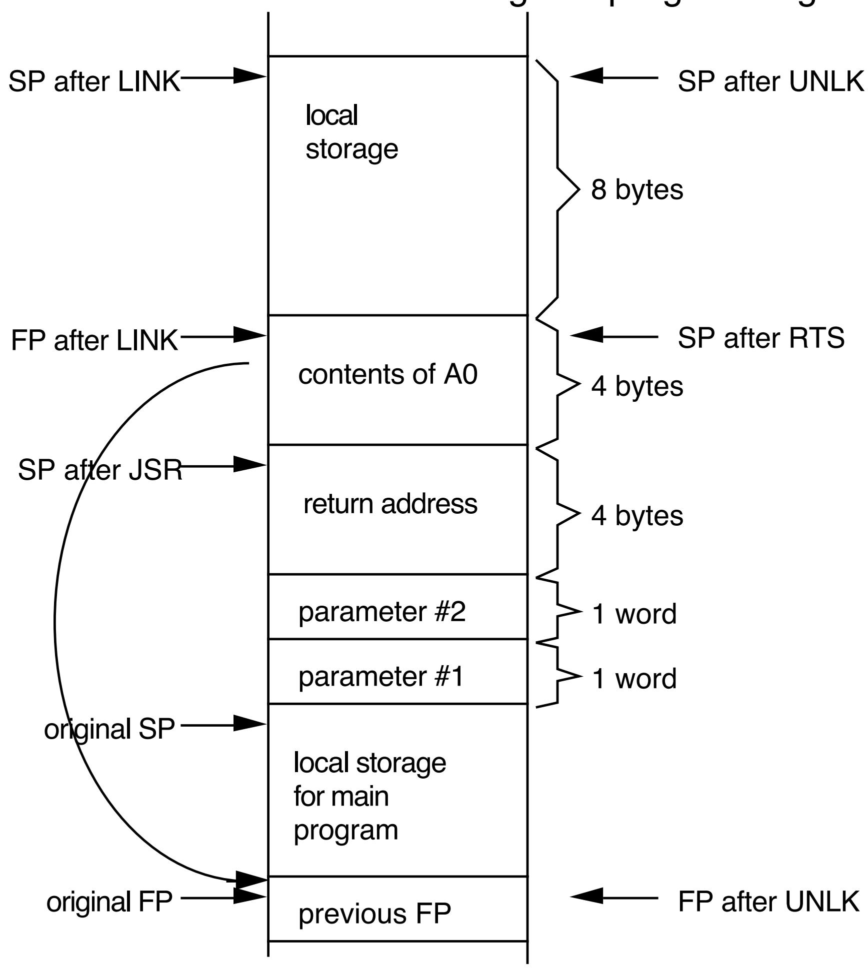
Return addresses <u>and</u> passed arguments are always <u>positive</u> relative to the frame pointer (FP).



Example:

```
        MOVE.W   D0,-(SP)     ;push parameter #1 onto stack
        MOVE.W   D1,-(SP)     ;push parameter #2 onto stack
        JSR      SBRT         ;jump to subroutine SBRT

SBRT    LINK     A0,-#$8      ;establish FP and local storage
        .
        .
        .
        MOVE.W   10(A0),D5    ;retrieve parameter #1
        .
        .
        .
        UNLK     A0           ;FP for the calling routine re-established.
                              Deallocate stack frame
        RTS                   ;return
```

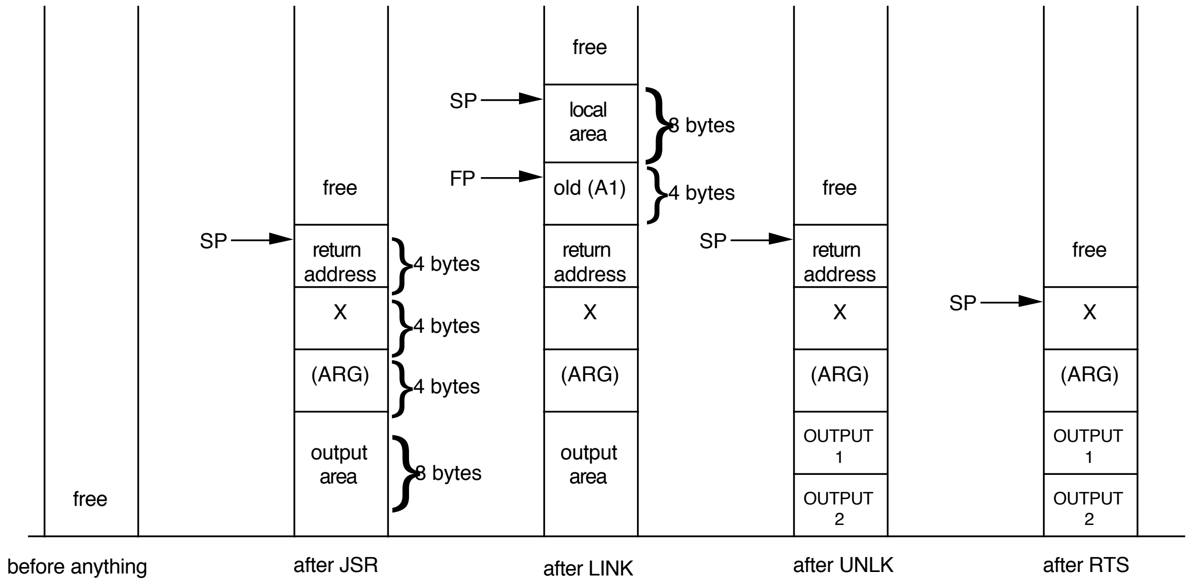
What the stack looks like during this program segment:



Note that the FP is stored in A0.

EXAMPLE:

```
ARG     DC.L                        ;number
N       EQU       8                 ;8 bytes for output
M       EQU       8                 ;8 bytes for local variables

        ADD.L     #-N,SP            ;put output area on stack
        MOVE.L    ARG,-(SP)         ;put argument on stack
        PEA       X                 ;put address of data table
                                    on stack
        JSR       SUBR              ;goto subroutine
        ADDA      #8,SP
        MOVE.L    (SP)+,D1          ;read outputs
        MOVE.L    (SP)+,D2
        .
        .
        .

SUBR    LINK      A1,#-M            ;save old SP
        .
        .
        .
        MOVE.L    LOCAL1,-4(A1)     ;save old variables
        MOVE.L    LOCAL2,-8(A1)     ;
        .
        .
        .
        ADD.L     #1,-4(A1)         ;change a local variable
        MOVEA.L   8(A1),A2          ;get X
        .
        .
        .
        MOVE.L    OUTPUT,16(A1)     ;push an output
        .
        .
        .
        UNLK      A1
        RTS

LOCAL1  DC.L      $98765432         ;local variables
LOCAL2  DC.L      $87654321
OUTPUT  DC.L      'ADCB'            output value
```


free
before anything
SP
free
return address
X
(ARG)
output area
4 bytes
4 bytes
4 bytes
8 bytes
after JSR
free
SP
local area
FP
old (A1)
return address
X
(ARG)
output area
8 bytes
4 bytes
after LINK
free
SP
return address
X
(ARG)
OUTPUT 1
OUTPUT 2
after UNLK
free
SP
X
(ARG)
OUTPUT 1
OUTPUT 2
after RTS

Program to compute the power of a number using a subroutine. Power MUST be an integer. A and B are signed numbers. Parameter passing using LINK and UNLK storage space on the stack.

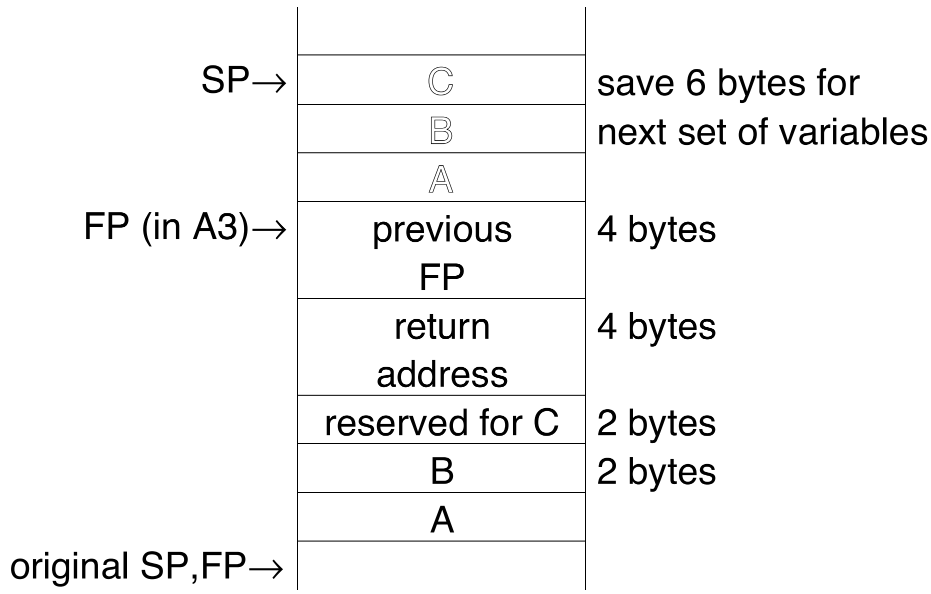
```
MAIN    LINK     A3,#-6        ;sets up SP
        MOVE     A,-2(A3)
        MOVE     B,-4(A3)
        JSR      POWR          ;call subroutine POWR
        LEA      C,A5
        MOVE     -6(A3),(A5)
        UNLK     A3

ARG     EQU      *
A       DC.W     4
B       DC.W     2
C       DS.W     1

POWR    EQU      *
        MOVE     -2(A3),D1     ;put A into D1
        MOVE     -4(A3),D2     ;put B into D2
        MOVE.L   #1,D3         ;put starting 1 into D3
LOOP    EQU      *
        SUBQ     #1,D2         ;decrement power
        BMI      EXIT          ;if D2-1<0 then quit  NOTE: this
                               gives us A**0=1
        MULS     D1,D3         ;multiply out power
        BRA      LOOP          ;and repeat as necessary
EXIT    EQU      *
        MOVE     D2,-6(A3)     ;C=(D3)
        RTS

        END      MAIN
```

	(D2)	2 bytes
	B	2 bytes
*FP→	A	2 bytes
SP→	value of A3	

*fixed while the SP changes

Better way.

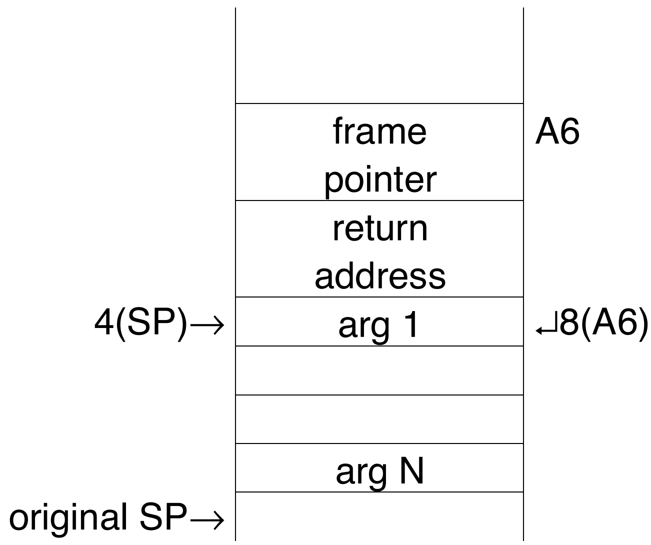
```
MAIN    MOVEA.L   SP,A3
        MOVE      A,-(SP)
        MOVE      B,-(SP)
        ADD.L     #2,SP           ;save output area
        JSR       POWR            ;call subroutine POWR
        LEA       C,A5
        MOVE      -6(A3), (A5)    ;put answer somewhere

ARG     EQU       *
A       DC.W      4
B       DC.W      2
C       DS.W      1

POWR    EQU       *
        LINK      A3,#-6
        MOVE      10(A3),D1       ;put A into D1
        MOVE      12(A3),D2       ;put B into D2
        •
        •
        •
        MOVE      D2,8(A3)        ;C=(D3)
        UNLK      A3
        RTS

        END       MAIN
```


SP→
C
B
A
save 6 bytes for
next set of variables
FP (in A3)→
previous
FP
4 bytes
return
address
4 bytes
reserved for C
2 bytes
B
2 bytes
A
original SP,FP→

Calling conventions for C or Pascal

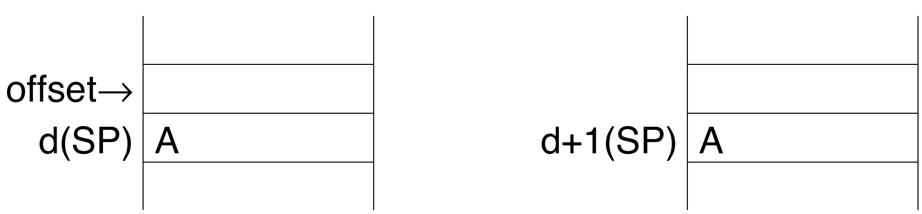
Arguments are pushed onto the stack in the reverse order of their appearance in the parameter list.

Just after a subroutine call:



If the function begins with a
LINK A6,#
High level language always generates LINK A6,# instructions

All arguments occupying just a byte in C are converted to a word and put in the low byte of the word, i.e.



Result, if any, is returned in D0 for function calls.

IT IS THE PROGRAMMER'S RESPONSIBILITY TO REMOVE THE ARGUMENTS FROM THE STACK.

The C calling sequence looks like this:

```
MOVE    ___,-(SP)   ;last argument
•
•
•
MOVE    ___,-(SP)   ;first argument
JSR     FUNCT
ADD     #N,SP       ;total size of arguments
```

Subroutine functions:

```
LINK    A6,#N
•
•
•
MOVE    ...,D0
UNLK    A6
RTS
```

The Pascal calling sequence pushes arguments in left to right order, then calls the function. The result if any is left on the stack. An example looks like this:

```
SUB     #N,SP       ;save space for result
MOVE    ...,-(SP)   ;push first argument onto stack
•
•
•
MOVE    ...,-(SP)   ;last argument
JSR     FUNCT
MOVE    (SP)+,...   ;store result
```

Subroutine code:

```
LINK    A6,#N
•
<code>
•
UNLK    A6
MOVE    (SP)+,A0    ;return address
ADD     #N,SP       ;total size of arguments
MOVE    ...,(SP)    ;store return result
JMP     (A0)
```

Symbols defined in assembly routines with the DS directive and exported using XDEF and XREF can be accessed from C as external variables. Conversely, C global variables can be imported and accessed from assembly using the XREF directive.

Miscellaneous comments about subroutines.

Parameter passing via MOVEM (move multiple registers)

If you have a small assembly language program this instruction allows you to save the values of registers NOT used to pass parameters.

Example:

```
SUBRTN  EQU     *
        MOVEM   D0-D7/A0-A6,SAVBLOCK
        •
        •
        •
        MOVEM   SAVBLOCK,D0-D7/A0-A6
```

where SAVBLOCK is local memory. This is bad practice since SAVBLOCK can be overwritten by your program.

MOVEM has two forms

```
        MOVEM   register_list,<ea>
        MOVEM   <ea>,register_list
```

More common to save registers on stack

```
SUBRTN  EQU     *
        MOVEM   D0-D7/A0-A6,-(SP)
        •
        •
        •
        MOVEM   (SP)+,D0-D7/A0-A6
        RTS
```

MOVEM is often used for re-entrant (subroutines that can be interrupted and re-entered) procedures.

The MOVEM instruction always transfers contents to and from memory in a predetermined sequence, regardless of the order used to specify them in the instruction.

address register indirect with pre-decrement	transferred in the order A7→A0, then D7→D0
for all control modes and address register indirect with post-increment	transferred in reverse order D0→D7, then A0→A7

This allows you to easily build stacks and lists.

Six methods of passing parameters:

1. Put arguments in D0 thru D7 before JSR (good only for a few arguments)
2. Move the addresses of the arguments to A0-A6 before JSR
3. Put the arguments immediately after the call. The argument addresses can be computed from the return address on the stack.
4. Put the addresses of the arguments immediately after the call in the code.
5. The arguments are listed in an array. Pass the base address of the array to the subroutine via A0-A6.
6. Use LINK and UNLK instructions to create and destroy temporary storage on the stack.

JUMP TABLES

— are similar to CASE statements in Pascal
— used where the control path is dependent on the state of a specific condition

EXAMPLE:

This subroutine calls one of five user subroutines based upon a user id code in the low byte of data register D0. The subroutine effects the A0 and D0 registers.

```
        RORG     $1000          ;causes relative addressing
                                (NOTE 1)
SELUSR  EXT.W    D0             ;extend user id code to word
        CHK      #4,D0          ;invalid id code ? (NOTE 2)
        LSL      #2,D0          ;NO! Calculate index=id*4
                                since all long word
                                addresses
        LEA      UADDR,A0       ;load table addresses
        MOVEA.L  0(A0,D0.W),A0  ;compute address of user
                                specified subroutine and put
                                correct caling address into
                                A0
        JMP      (A0)           ;jump to specified routine
        •
        •
        •
UADDR   DC.L     USER0,USER1,USER2,USER3,USER4
```

NOTES:

1. The RORG is often used when mixing assembly language programs with high level programs. It causes subsequenct addresses to be relative.
2. The CHK is a new instruction. In this case it checks if the least significant word of D0 is between 0 and 4 (2's complement). If the word is outside these limits, a exception through vector address $10 is initiated. The CHK instruction checks for addresses outside assigned limits and is often used to implement subscript checking.

EXAMPLE RECURSIVE PROCEDURE USING STACK

```
DATA     EQU      $6000
PROGRM   EQU      $4000

         ORG      DATA
NUMB     DS.W     1             ;number to be factorialized
F_NUMB   DS.W     1             ;factorial of input number

         ORG      PROGRM
MAIN     MOVE.W   NUMB,D0       ;get input number
         JSR      FACTOR        ;compute factorial
         MOVE.W   D0,F_NUMB     ;save the answer

* SUBROUTINE FACTOR
* PURPOSE: Determine the factorial of a given number.
* INPUT: D0.W  = number whose factorial is to be computed
*                0 ≤ D0.W ≤ 9
* OUTPUT: D0.W = factorial of input number
* REGISTER USAGE: No registers except D0 effected
* SAMPLE CASE: INPUT: D0.W=5
*                OUTPUT: D0.W=120

FACTOR   MOVE.W   D0,-(SP)      ;push current number onto
                                stack
         SUBQ.W   #1,D0         ;decrement number
         BNE.S    F_CONT        ;not end of factorial
                                computations
         MOVE.W   (SP)+,D0      ;factorial=1
         BRA.S    RETURN
F_CONT   JSR      FACTOR
         MULU     (SP)+,D0
RETURN   RTS
```

→	1	subtract 1, equal to zero so pop stack
	return address	
	2	
	return address	
	3	
	return address	
	4	
	return address	
	5	put D0 (current onto stack)
	return address	
original SP→		

EXAMPLE

This is a simplified version of TUTOR's "DF" command. It uses the stack to display register contents.

```
START     MOVEM.L  TESTREGS,D0-D7/A0-A6    ;assign values to
                                  registers
          MOVE,L   #-1,-(SP)      ;put something on stack
          JSR      PRINTR         ;print all registers
          MOVE.L   (SP)+,D0       ;retrieve it
          ADDQ.L   #1,D0          ;null it
          JSR      PRINTR         ;print them all again
          TRAP     #0             ;stop program

SAVESP    EQU      60

PRINTR             ;data for PRINTREGS
RMSGS:
          DC.B     ‘  D0   D1   D2   D3   D4   D5’,0
          DC.B     ‘  D6   D7   A0   A1   A2   A3’,0
          DC.B     ‘  A4   A5   A6   SP   SR   PC’,0

;                  l<---- 55 characters long ---->l
SPACES    DC.B     ‘  ‘,0         ;2 blanks
CONBUF    DS.B     10
ENDLINE   DC.B     $0D,$0A,0
; data for program
CH        DS.B     1
          DS.W     1
TSTREG    DC.L     1,2,3,4,5,6,7,8,$A,$AA,$AAA
          DC.L     $AAAA,$AAAAA,$AAAAAA,$AAAAAAA
          END
```

```
PRINTR  MOVE.W   SR,-(SP)           ;save SR on stack
        PEA      6(SP)              ;save original SP on stack
        MOVEM.L  D0-D7/A0-A6,-(SP)  ;save all regular
                                    registers
        MOVEQ    #2,D4              ;D1 counts # of rows in
                                    printout
        MOVEA.L  SP,A1              ;use A1 to point to beginning
                                    of data
        LEA      RMSGS,A2           ;use A2 to point to row
                                    headings
MLOOP                               ;output routine for heading
        MOVEA.L  A2,A0              ;set pointer to beginning of
                                    header to be printed
        JSR      PrintString        ;output heading
        MOVEQ    #5,D5              ;output six registers this line
RLOOP   TST.W    D4                 ;tests for SR to be printed
        BNE.S    NOT_SR             ;SR requires special routine
        CMP.W    #1,D5              ;as it is only word length
        BNE.S    NOT_SR             ;register
        LEA      SPACES,A0          ;load addresses of spaces
        JSR      PrintString        ;print spaces with no new
                                    line
        MOVE.W   (A1)+,D0           ;put SR word into D0
        JSR      PNT4HX             ;unimplemented routine to
                                    convert 4 hex digits in D0 to
                                    ascii code for printing
        JSR      PrintString        ;print hex contents
        LEA      SPACES,A0          ;load address of spaces
        JSR      PrintString        ;print them with no line feed
        BRA.S    ENDRPL
NOT_SR  MOVE.L   (A1)+,D0           ;put register contents into D0
        JSR      PNT8HX             ;unimplemented routine to
                                    convert 8 hex digits in D0 to
                                    ascii code for printing
```

```
ENDRPL  DBF       D5,PRLOOP         ;decrement register counter,
                                    started at 5
        LEA       ENDLINE,A0        ;print CR+LF
        JSR       PrintString
        ADDA.L    #55,A2            ;increment  heading pointer
        DBF       D4,MLOOP          ;goto another line
        MOVEM.L   (SP)+,D0-D7/A0-A6
        ADDQ.W    #4,SP             ;skip over A7 to point to SR
        RTR                         ;return and restore registers
```

SP→	D0	
	D1	
	0 0 0 0	
→	A5	
	A6	
after MOVEM→	original SP address	⇐do this since this is only way to save original value of A7
after ADD #4→	SR	⇐the RTR pops this and the return address
	return address	
→	-1	put D0 (current onto stack)