To be presented at the Eighth Texas Instruments- Members Information Exchange Conference, Atlanta, Georgia, March 27-30, 1979

LBL-8755

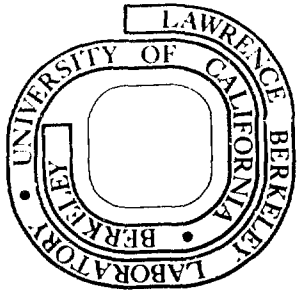
CONF-790333--1

COMPUTER SYSTEM CROSS-FERTILIZATION: MAKING YOUR TI 980 PLAY YOUR TMS 9900

John D. Meng

February 1979

Prepared for the U. S. Department of Energy
under Contract W-7405-ENG-48



COMPUTER SYSTEM CROSS-FERTILIZATION: MAKING YOUR TI 980 PLAY YOUR TMS 9900*

John D. Meng
Lawrence Berkeley Laboratory
University of California
Berkeley, California 94720

ABSTRACT

Being faced with the considerable problem of wanting to use TMS 9900 devices in several small control and data acquisition applications, but not wanting to pay for a development system to do it, we developed several simple, effective techniques for doing TMS 9900 programming and debugging on our TI 980 system. The 980 assembler lends itself easily to the redefinition of operation codes required to assemble programs for the 9900. Also, a simple interconnection between the 980 and the 9900 allows us to operate the 9900 and to monitor the operation on the 980. Finally, we have developed special operation codes within the 980 assembler which allow us to program hardware control on the 9900 system via a macro-language tailored to a particular 9900 hardware configuration.

INTRODUCTION

Back in the early days of the TMS 9900 (the spring of 1976 when the price was $99.32), it became obvious that this particular device was going to become very useful to us. We deal in one-of-a-kind applications, many requiring fairly high rates of transfer of 16-bit data. For example, we designed and built a special-purpose disc controller for our TI 980 system. The TMS 9900 appeared on the scene just in time to do the job for us, and is still performing admirably. In another application, we needed a preprocessor for data from an array of sodium iodide crystals in a series of tomographic experiments with heavy ions. Again, the TMS 9900 had just appeared and is today preprocessing data to be shipped to a 16-bit minicomputer system. Operating speed, the on-chip multiply, the 16-bit word length and the CRU input-output made the device peculiarly useful to us.

---

*This work was supported by the Division of Physical Research of the Department of Energy under Contract No. W-7405-ENG-48.

However, several age-old problems existed back in 1976. There were no development systems available for the TMS 9900 and even if there were, our budget was not ready to accept what would have been a relatively expensive item for developing a system around an as-yet untried product. Als , although assembly language is fine for some simple jobs (and req red for some fast jobs), it can be a very trying and expensive diet if taken too regularly.

G tting started, we attacked these various problems in good engineering fa hion; one at a time, and using our available resources, mainly a TI 980 A system which we inherited from a defunct series of experiments. Our initial thought was to develop a simple cross-assembler so we could at leas use the editing capabilities of the 980. From the depths of despair or thinking about this idea, however, came what proved proved to be an spiration. The 980 assembler allows the definition of new op-codes (OPD, Operation Define) using existing instruction formats. It also allows the user to define fields for new instructions (FRM, Format a New Instruction). Using these two features allowed us to produce an assembler for the TMS 9900 which executed on our TI 980-A system, the entire project (after the initial inspiration) requiring less than an hour to implement. The TMS 9900 assembler, including comments, consists of exactly 130 lines of code.

Of course, there are some small irreconcilable differences between the 980 and the 9900, but like the Wright Brothers we were off the ground at last.

Certain 9900 instructions correspond in form with 980 instructions. (See Figure 1) These, of course, fit directly into the the 980 assembler (with an OPD directive for each.) These are 9900 formats 2, 6, 7, and 8 which correspond in form to 980 formats, respectively, 1, 3, 5, and 6 as illustrated in Figure 1. The remaining 9900 instructions are implemented in the 980 assembler via the FRM directive. One directive is used per class of instruction, followed by EQU (equivalent) statements to define the appropriate 9900 mnemonics. Figure 2 is a listing of "SAL 99003." This 130-line block must preceed each assembly. Notice that a JMP instruction goes in as:

```
JMP     ARG
```

whereas a MOV instruction becomes:

```
ARI     MOV,D,0,I,13.
```

ADD becomes:

```
ARI     A,X,0,X,0
DATA    SOURCE,DESTINATION
```

The system is not ideal. Routines written this way cannot be run verbatim into any assembler existing on any 9900-based system. Also, since the 980 assembler only recognizes 8 registers (but fortunately will accept 16-register codes), we get meaningless error flags for some 9900 format 6 instructions. However, as a group which never had the opportunity to become accustomed to 9900-based assembly language, we quickly became fluent in our own version.

A fundamental difference between the 9900 and the 980 is memory addressing. The 9900 uses byte addressing, except for JMP instructions, JMP instructions use word-relative addressing, making them compatible with 980 field mnemonics. For example, 9900 Format 1 instructions (add, subtract, move, compare, and, or) require:

```
ARI     FRM     4,2,4,2,4,
```

the first field corresponding to an operation code and the subsequent fields setting up addressing. Next, to allow mnemonic references to the five fields, the following equivalences are defined (using MOV as an example):

```
MOV     EQU     >C
D       EQU     0
I       EQU     1
X       EQU     2
XINC    EQU     3
```

The D,I,X and XINC equivalences define mnemonics for use in the two 2-bit fields (fields 2 and 4) which specify addressing type. D is for register direct, I for indirect, X for indexed and XINC for indirect/auto incrementing. A move instruction which is designed to move a word of data into register 0 from a location pointed to by register 2, for example, is mnemonically written as:

```
ARI     MOV,D,0,I,2.
```

To move a word of data from some arbitrary memory location:

```
          ARI     MOV,D,0,X,0
          DATA    SOURCE+SOURCE
           .
           .
           .
SOURCE    DATA    VALUE
```

Where SOURCE is a pointer to the data word containing VALUE. In 980 language, memory addresses are assumed to be 16-bit word addresses. In 9900 language, memory is addressed by bytes. Consequently, an address which the 980 assembler defines as a memory address must be doubled to produce the correct 9900 memory address. This is the reason for representing the location of SOURCE in the above statement as SOURCE+SOURCE. The statement source*2 could be used if the result of the multipication is less than 32,768 (most significant bit reset). Otherwise, the hardware multiply instruction used by the 980 assembler may reset this bit, producing an incorrect value (980 hardware assumes the most significant bit of each half of a multiply result to be a sign bit.) Our one-hour assembler is displayed in Figure 2. Since we have never had the legitimate 9900 assembler to learn on, the peculiarities of our own version have become conventions to us, no longer seeming particularly clumsy or illogical.

CARRYING ON

Now we were assembling programs for our embryonic systems. However, as anybody who has tried it knows, debugging computer programs with just an oscilloscope and selected test points (without the benefit of a control panel) is, at best, tedious.

One possible solution to this dilemma would have been to build a control panel complete with lights, switches and debugging features. Our solution was to connect a 9900 chip to our 980 via a standard 16-in/16-out data module. This connection allowed us to program the 980 to use a reserved block of its memory as the memory space of the 9900. Enough logic was added to the connection to allow the 980 to micro-step the 9900, to generate 9900 interrupts and to implement the CRU channel (see Figure 3).

Next we wrote a 980 program to drive the 9900, allowing the 980 to intervene in selected memory accesses. For example, the 980 can record and display every memory access. Or it can select only Instruction Acquisition (IAQ) accesses to record and display. The 980 can also display only-write or only-read accesses; or accesses only to selected memory locations, or only to selected IAQ locations. The 980 can set breakpoints at arbitary points and then operate the 9900 chip until it reaches a breakpoint.

Now we could write a program and run through its execution in enough detail to be sure it would not suffer software hangups. Of course, the 9900 was not being operated at top speed, and many 9900 input/output operations were not practical to emulate with this scheme. However, our economy bootstrap routine was running.

LANGAUAGE DEVELOPMENTS

As mentioned earlier, we deal in one-of-a kind hardware projects. Our latest has been an X-ray fluorescence trace element analysis system which counts secondary X-radiation from a series of samples mounted in carriers and moved through the counting station by a mechanical transport mechanism. The 9900 subsystem in this is responsible for monitoring and moving the mechanical pieces as well as for reading and recording the raw data in a large attached paged memory. The heavy analysis for the system is done on an attached desk-top programmable calculator. It is desirable for us to produce a system which is easy to change and for which simple changes do not require delving into the details of an assembly-language program. Consequently, we have done our 9900 assembly language programming in small packets which do specific jobs for specific hardware. Each small packet is affiliated with a driver routine which simultaneously services several of the packets. By passing appropriate arguments to this driver routine, the appropriate packet or sequence of packets is called to perform the necessary job.

In order to simply and efficiently make these routines accessable to a user in a flexible way, we have utilized the unique context-switching capability of the 9900-based subsystem. The 9900 executes a short loop, which controls a pseudo program-counter stepping through a list of pseudo-instructions. The pseudo-instructions form the body of a language tailor-made to operate the attached hardware.

Each pseudo-instruction is defined by a workspace pointer/program counter pair which form the calling parameters for a 9900 BLWP instruction (Figure 4). The op-code for the pseudo-instruction is defined as the pointer to the appropriate workspace pointer/program counter parameter pair. To execute a program in pseudo-language requires the 9900 loop:

```
START   LI    0,PROGM     POINTER TO FIRST STEP
                          (PSEUDO-PC)
RUN     MOV   I,0,D,2     OPCODE IS PARAMETER POINTER
        JEQ   START       ZERO OP CODE MEANS RESTART
        INCT  D,0         STEP PSEUDO PC
        BLWP  I,2         EXECUTE THE PSEUDO-INSTRUCTION
        JMP   RUN         LOOP
```

Each instruction execution is a context switch. A user program example would be:

```
PROGM       @SCAL    RESET     RESET SCALER
            @TIMR    RESET     RESET TIMER
            @TIMR    START     START TIMER
            @ADC     ON        TURN ON ADC
             FIN               END.  RESTART.
```

The operations (such as SCAL, TIMR) are defined in 980 assembly language directives, and the @ preceeding the mnemonic (@SCAL, @TIMR) forces the 980 assembler to reserve an extra instruction word. Location START in the execution loop resets the pseudo program-counter (register 0) to point to the first statement of the user program (PROGM). Op code 0 is the FIN statement which signals the end of the user program (and forces a restart). Next, the pseudo program-counter is stepped to point to the next location in the user program. This is the location of the parameter RESET. The SCAL routine will pick up this parameter and use it to direct resetting the scaler. The SCAL routine steps the pseudo program-counter after getting the parameter (INCT I,13), thus preparing for an exit via RTWP after its job is finished.

EPILOG

What we have described is not a tutorial on what to do. Nearly everything we have done has been superceded in economic and efficient fashion by material now available from Texas Instruments. We still use our cross assembler simply because we have it and we are very familiar with it. However, TIBUG achieves much of what we were attempting with our cross-connection between the 980 and a 9900 chip, and the recent introduction of POWER BASIC supercedes our own pseudo-language developments.

What we have described is, first of all, history. It is a story of challenges successfully met when a new and apparently useful device appeared without much manufacturer support. It is also a story of how to learn in great depth about a new device. Finally it is a story about the immeasurable value of ingenuity in the face of crucial challenges coupled with a perenial budget crunch.

ACKNOWLEDGMENTS

References to a company or product name does not imply approval or recommendation of the product by the University of California or the United States Department of Energy to the exclusion of others that may be suitable.

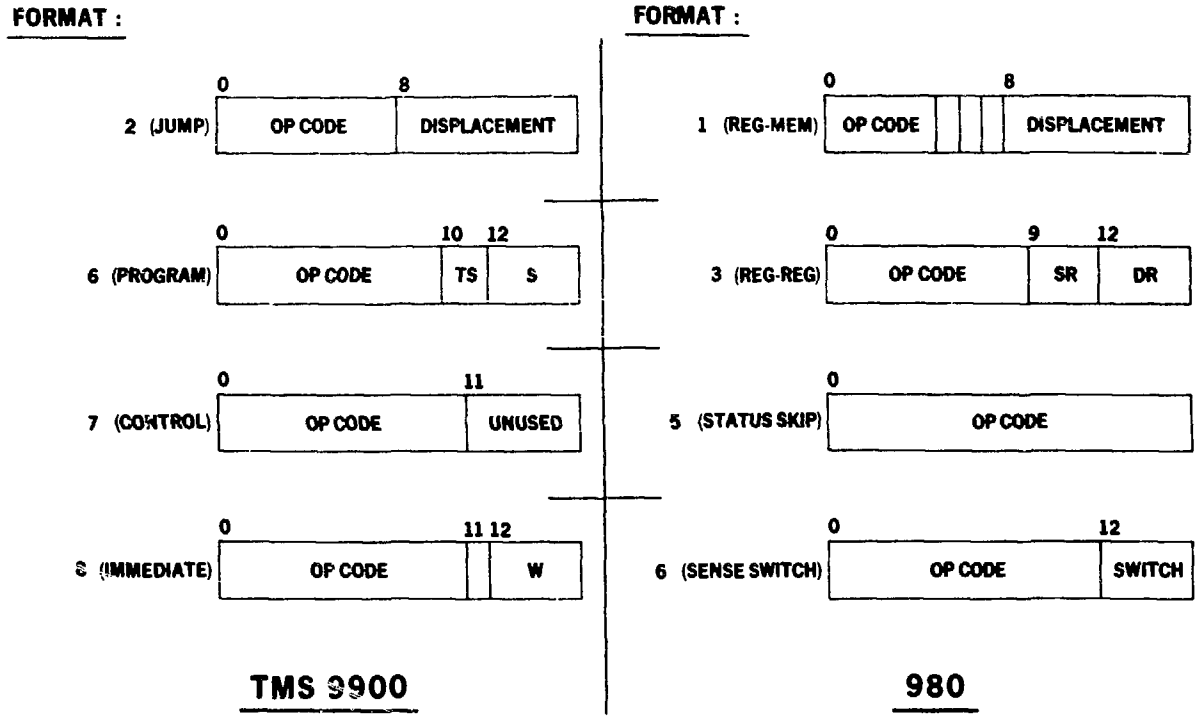
REFERENCES

1. Texas Instruments Engineering Staff: TMS 9900 Microprocessor Data Manual. Texas Instruments, Inc. Dec., 1976.

2. Texas Instruments Digital Systems Division: Model 980 Computer Assembly Language Programmers Reference Manual. Texas Instruments, Inc. March 1976.

3. Texas Instruments, Inc.; Model 990 Computer Programming Card, November, 1975.

4. Texas Instruments, Inc.; Programmer Reference Card for Model 980-A Computer, September, 1973.



XBL 791-7833

Fig. 1: The correspondence between 9900 and 980 instruction format, allowing some simple op-code redefinitions. The 980 OPD directive defines a 16-bit op code independent of the size determined by the above field definitions.

```
0001          UNL
0002          IDT     'SAL990'
0003 *
0004          HED     SAL9900 3
0005          HED     DEFINE TMS9900 FORMAT 1 INSTRUCTIONS (ARITHMETIC)
0006          DEF     BFBOT,BFER
0007 *
0008 ARI      FRM     4,2,4,2,4
0009 *
0010 D        EQU     0
0011 I        EQU     1
0012 X        EQU     2
0013 XINC     EQU     3               TD/TS OPTIONS
0014 *
0015 A        EQU     >A              ADD
0016 AB       EQU     >B              ADD BYTES
0017 C        EQU     >8              COMPARE
0018 CB       EQU     >9              COMPARE BYTES
0019 MOV      EQU     >C              MOVE
0020 MOVB     EQU     >D              MOVE BYTES
0021 S        EQU     >6              SUBTRACT
0022 SB       EQU     >7              SUBTRACT BYTES
0023 SZC      EQU     4               AND
0024 SZCB     EQU     5               AND BYTES
0025 SOC      EQU     >E              OR
0026 SOCB     EQU     >F              OR BYTES
0027 *
0028 *
0029          HED     DEFINE FORMAT 2 INSTRUCTIONS (JMP)
0030 *
0031 JEQ      OPD     >1300,1         JMP EQUAL TO
0032 JGT      OPD     >1500,1         JMP GREATER THAN
0033 JH       OPD     >1B00,1         JMP HIGH
0034 JHE      OPD     >1400,1         JMP HIGH OR EQUAL
0035 JL       OPD     >1A00,1         JMP LOW
0036 JLE      OPD     >1200,1         JMP LOW OR EQUAL
0037 JLT      OPD     >1100,1         JMP LESS THAN
0038 JMP      OPD     >1000,1         JUMP
0039 JNC      OPD     >1700,1         JMP NO CARRY
0040 JNE      OPD     >1600,1         JMP NOT EQUAL
0041 JNO      OPD     >1900,1         JMP NO OVERFLOW
0042 JOC      OPD     >1800,1         JMP ON CARRY
0043 JOP      OPD     >1C00,1         JMP ODD PARITY
0044 *
0045 BIT      FRM     8,8
0046 *
0047 TB       EQU     >1F
0048 SBO      EQU     >1D
0049 SBZ      EQU     >1E
0050 *
0051 *
0052          HED     FORMAT 3,9,4 INSTRUCTIONS (LOG., MPY/DIV, XOP, CR
0053 *
0054 EXT      FRM     6,4,2,4
0055 *
0056 MPY      EQU     >E
0057 DIV      EQU     >F
0058 COC      EQU     8               COMPARE ONES CORRESPONDING
0059 CZC      EQU     9               COMPARE ZEROS CORRESPONDING
0060 LDCR     EQU     >C              LOAD CRU REG
0061 STCR     EQU     >D              STORE CRU REG
0062 XOP      EQU     >B              EXTENDED OPERATION
0063 XOR      EQU     >A              EXCLUSIVE OR
0064 *
```

Fig. 2: A listing of SAL 9900.3 - TMS 9900 assembly language defined in TI 980 terms. Formats 2,6,7, and 8 (9900 language) are defined with the OPD directive. The remainder use the FRM directive and EQUalities for complete instruction definitions.

```
0065          HED     FORMAT 5 INSTRUCTIONS (SHIFT)
0066 *
0067 SHF      FRM     8,4,4
0068 *
0069 SLA      EQU     >A              SHIFT LEFT ARITHMETIC
0070 SRA      EQU     >8              SHIFT RIGHT ARITHMETIC
0071 SRC      EQU     >B              SHIFT RIGHT CIRCULAR
0072 SRL      EQU     >9              SHIFT RIGHT LOGICAL
0073 *
0074 *
0075 *
0076          HED     FORMAT 6 INSTRUCTIONS (PROGRAM)
0077 *
0078 ABS      OPD     >740,2          ABSOLUTE VALUE
0079 B        OPD     >440,2          BRANCH
0080 BL       OPD     >680,2          BRANCH, LINK
0081 BLWP     OPD     >400,2          BRANCH; LOAD WORKSPACE POINTER
0082 CLR      OPD     >4C0,2          CLEAR
0083 DEC      OPD     >600,2          DECREMENT
0084 DECT     OPD     >640,2          DECREMENT BY 2
0085 INC      OPD     >580,2          INCREMENT
0086 INCT     OPD     >5C0,2          INCREMENT BY 2
0087 INV      OPD     >540,2          INVERT
0088 NEG      OPD     >500,2          NEGATE
0089 SETO     OPD     >700,2          SET ONES
0090 SWPB     OPD     >6C0,2          SWAP BYTES
0091 XEQ      OPD     >480,2          EXECUTE
0092 *
0093 *
0094 *
0095          HED     FORMAT 7 INSTRUCTIONS (CONTROL)
0096 *
0097 CKOF     OPD     >3C0,5
0098 CKON     OPD     >3A0,5
0099 IDLE     OPD     >340,5
0100 RTWP     OPD     >380,5
0101 *
0102 *
0103 *
0104          HED     FORMAT 8 INSTRUCTIONS (IMMEDIATE)
0105 *
0106 AI       OPD     >220,7          ADD IMMEDIATE
0107 ANDI     OPD     >240,7          AND IMMEDIATE
0108 CI       OPD     >280,7          COMPARE IMMEDIATE
0109 LI       OPD     >200,7          LOAD IMMEDIATE
0110 LIMI     OPD     >300,7          LOAD INTERRUPT MASK, IMMEDIATE
0111 LWPI     OPD     >2E0,7          LOAD WORKSPACE PTR IMMEDIATE
0112 ORI      OPD     >260,7          OR IMMEDIATE
0113 STST     OPD     >2C0,7          STORE STATUS REGISTER
0114 STWP     OPD     >2A0,7          STORE WORKSPACE POINTER
0115 *
0116 *
0117 *
0118 *  START OF TMS 9900 MEMORY
0119 *
0120          ORG     0
0121 BFBOT    EQU     $
0122 BFER     EQU     $
0123          HED     SAL9900.3
0124          LIS
0125 *
0126 * END OF PROGRAM MUST CONTAIN:
0127 * BUFTOP        EQU             $
0128 * BLENG         EQU             16384-BUFTOP
0129 *               BSS             BLENG
0130 *               END
```

FIG. 2 page 2

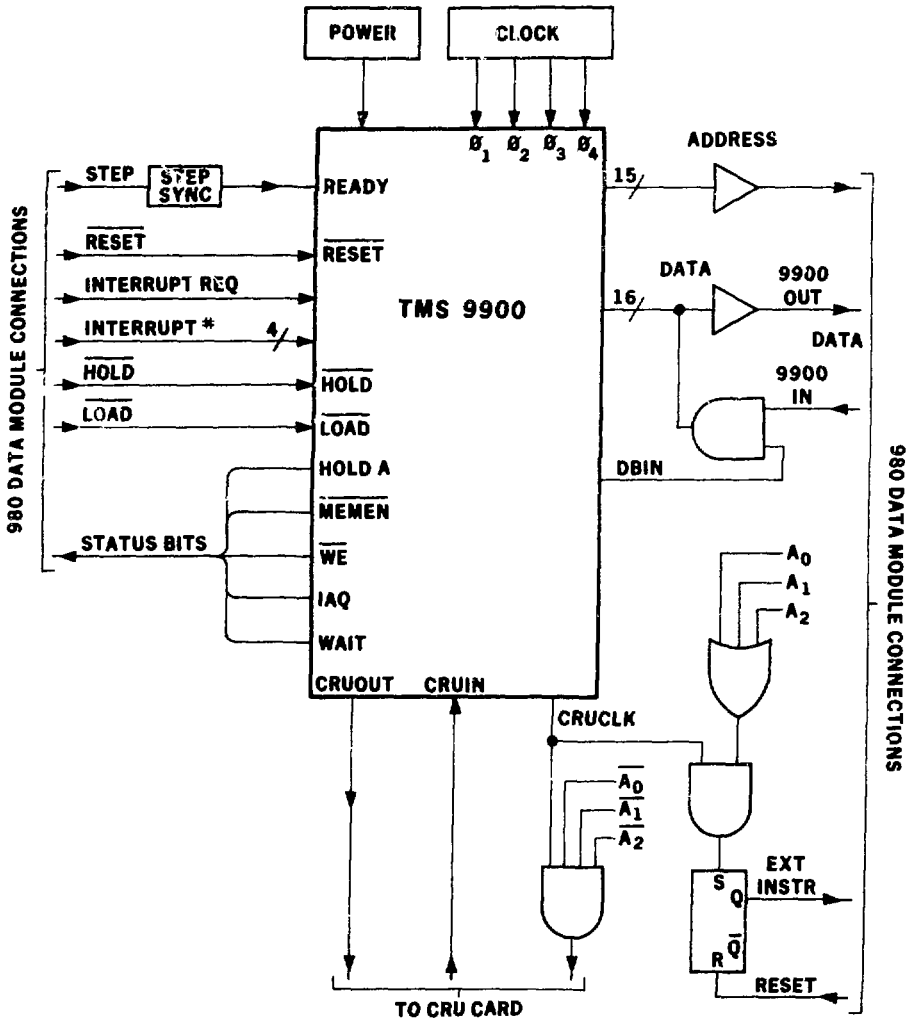


Fig. 3: Logic required to connect the 980 as 9900 memory and control.

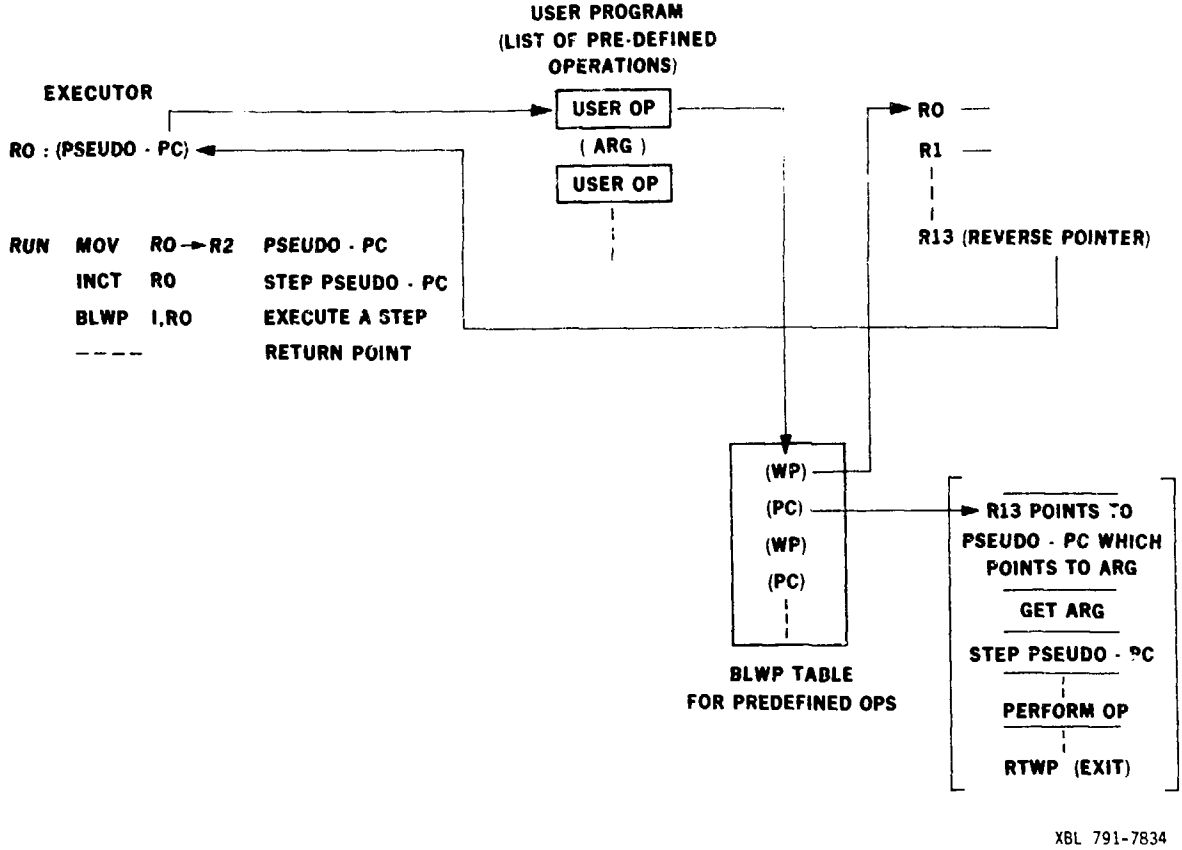


Fig. 4: Linkage required for running the user program in the user pseudo-language.