MEDIUM LEVEL PROGRAMMING LANGUAGES FOR MICRO PROCESSORS

JAN-ERIC ASPERNÄS

RE-177 May 1976
Department of Automatic Control
Lund Institute of Technology

MEDIUM LEVEL PROGRAMMING LANGUAGES

FOR MICRO PROCESSORS.

Author Jan-Eric Aspernäs
Supervisors Leif Andersson
Johan Wieslander

ABSTRACT.

This report of master thesis describes two programming languages for micro processors namely:
MLP for Intel 8008 developed at C.E.R.L., Great Britain and HILP 80 for Intel 8080 developed as master thesis at Lund Institute of Technology.
The compilers generate assembly code and are intended to be implemented on a host computer. The compilers are based on the macro processor STAGE 2, and need about 15 K of core on a PDP-15. The report is written in two parts, which could be read separately.

Detta examensarbete beskriver två programmeringsspråk för mikro-processorer nämligen:
MLP för Intel 8008 utvecklat vid C.E.R.L., Great Britain och HILP 80 för Intel 8080 utvecklat som ett examensarbete vid Lunds Tekniska Högskola.
Kompilatorerna genererar assembler kod och är avsedda att implementeras på en värd-dator. Kompilatorerna är baserade på makro processorn STAGE 2, och kräver ungefär 15 K minne på en PDP-15. Rapporten är skriven i två delar, vilka kan läsas separat.

THE MEDIUM LEVEL PROGRAMMING LANGUAGE MLP FOR THE MICRO PROCESSOR INTEL 8008.

Author Jan-Eric Aspernäs
Supervisors Leif Andersson
Johan Wieslander

CONTENTS.

This report describes a medium level language for the micro computer Intel 8008. The language is called MLP and is developed at C.E.R.L Great Britain.

This report is based on a part of master thesis at Lund Institute of Technology (LTH), Sweden. The aim of the work at LTH was to test MLP and make it fit a PDP-15, which is the host machine used at LTH. The aim was also to try to improve MLP and increase the efficiency of the generated assembly code.

The next page shows the way from MLP source program to a loaded Intel 8008 Computer.

Chapter B.1 contains a discussion of the work at LTH. The semantics of MLP is described in part B.2. Appendix C contains an MLP program, which shows the generated code from some of the MLP statements.

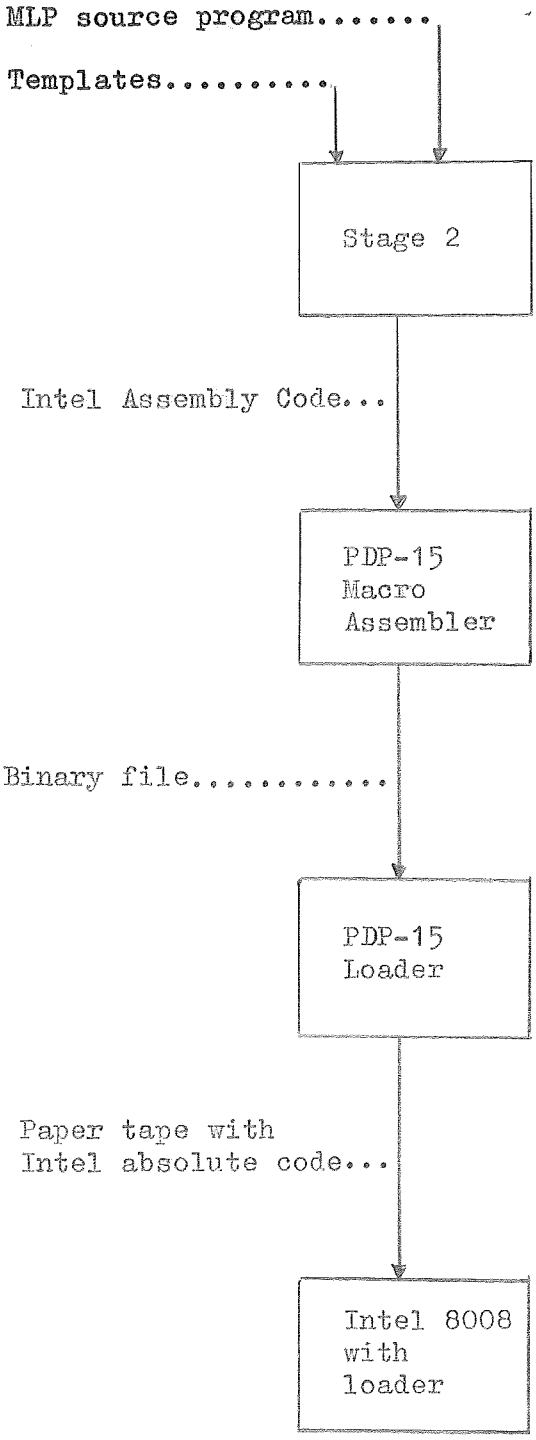


fig 1. From MLP to Intel 8008.

B.1.1. GENERAL

The original MLP from C.E.R.L used PROM banks for the data. As the usage of RAM banks for the data would increase the efficiency of the bank address manipulation, the whole part of the data declaration has been rewritten.

For optimization of the bank address setting during the translation, MLP has an internal variable LASTH, which holds the bank value from the last memory reference. When a new memory reference is made, MLP could check LASTH against the bank address of the variable, and if necessary generate a "LHI instruction". Because of this technique MLP got the opportunity to test for undeclared variables.

B.1.2. DATA DECLARATIONS.

All the variables used must be declared and their bank addresses specified. The above mentioned optimization could be further increased if all the variables are declared in the same bank. In this case LASTH will not be destroyed after a label, as when the jump to the label takes place, the last bank address must be equal to the bank address at the label. For this purpose a SAME BANK statement should be given and could be used provided that, when all statements generating labels are reached, the H register holds the desired bank address. Used subroutines must not change the H register when SAME BANK is valid.

B.1.3. INDIRECT REGISTER CONTROL.

Experience has shown that many applications require very much address manipulation. The indirect register control allows the user to work with addresses in an efficient way. This extension of the original MLP has proved to be very useful.

B.1.4. LOOPS

The programming language PASCAL has given the idea to the three loops implemented.

B.1.5. SUBROUTINES WITH ARGUMENTS.

The usage of arguments gives a technique to give entry values to registers before calling a subroutine. The arguments are register operations, which are carried out before the subroutine is called. The subroutine could be called with any number of arguments.

B.1.6. OTHER CHANGES OF MLP.

In some statements of the original MLP the optimization has been improved, why the efficiency of the generated code has increased.

Every digit in the source program is converted to octal by MLP. This is because of the PDP-15 assembler.

Some minor errors in the original MLP templates have also been corrected.

B.2.1 GENERAL FORMAT.

The general format of the language is:

Declarations
Statements
Subroutine declarations
FINISH

Main rules for statements in MLP:

1 Only one MLP statement is allowed per line, except for IF statements.

2 Leading tabs and spaces are allowed.
MLP words should be separated with one space.
Logical operators,+,-,= should not be separated by spaces.

3 Two types of comments may be used:
a) comment string
b) statement;comment string

4 Names may consist of up to six characters, starting with a letter.

5 Empty lines may be used.

6 Octal digits are defined by starting with ø.

7 Integers in MLP should be within the ranges:
-128 to 255 or -ø2øø to ø377.
Note: Integers >= 128 will set the sign bit, and therefore be equal to a negative number.

B.2.2 ABBREVATIONS.

r = registers (A,B,C,D,E,H,L or M)
c = numeric constant
id = identifier
a = address reference: id, id(id), id(rc)
rac = either of r, a, c
ra = either of r, a
mc = machine code
M = memory location addressed by H and L

Variables used in the program must be declared and their bank addresses specified:

BANK <constant>
WORDS <list of id or id(c) separated by commas>

The user may define addresses for the words by setting id↑c or id(c)↑c freely in the list. The words are defined consecutively. The first word is defined ∅ if the user does not declare a start address.

GLOBAL <list of id separated by commas>

If you make references to names in other programs, the loader program has to know these names as global, otherwise the loader may not be able to set up a common address for these names.

SAME BANK

This statement is used when all the words are specified in the same bank.

BANK IS <constant>

This statement must be given if SAME BANK is used, and generates a LHI instruction.

Note; Array index runs from ∅ to i-1.
Only one-dimensional arrays are allowed.
No variables may be preset.

Datadeclarations of separately compiled subroutines.

What is said above about main programs is valid. In special cases when the H-reg is set in the main program and the subroutine does not change the H-reg, the data declaration should be:

Words statement as above
SAME BANK

B.2.4 REGISTER CONTROL.

REG r_1=$r_2$ac

Loads r_1 with the contents of $r_2$ac.

REG r=ra+1

Loads r with the contents of ra and increments r.

REG r=ra-1

Loads r with the contents of ra and decrements r.

a=REG r

Loads a memory location with the contents of r.

M=a

Loads the address of a into H and L.

MASK ⟨expression⟩ WITH rac

The expression is evaluated and loaded into A-reg., which then is masked with rac.

B.2.5 INDIRECT REGISTER CONTROL. (r ≠ M)

REG r↑ac
REG r↑a+c
REG r↑a-c

r will be loaded with an address

REG ↑r_1=r_2c
REG ↑r_1+1=r_2c
REG ↑r_1-1=r_2c

The address specified by r_1, r_1+1, r_1-1 will be loaded with r_2c.

REG r_1=↑r_2
REG r_1=↑r_2+1
REG r_1=↑r_2-1

The contents specified by the address specified by r_2, r_2+1, r_2-1 will be loaded into r_1.

PUSH ↑r+1=list of rc separated by comma
PUSH ↑r-1=list of rc separated by comma

Takes each element in the list and loads them into the address specified by r and then increments or decrements r.

POP list of r=↑r_2+1
POP list of r=↑r_2-1

Decrements or increments r_2 and then loads r from the address specified by r_2. This is carried out for each element in the list.

B.2.6 ASSIGNMENTS FOR EXPRESSIONS.

```
SET ra=expression
ra=expression
```

The expression may start with an INPUT-statement or an unconditional CALL-statement.

Allowed operators in expressions:

```
+
-
.OR
.XR
.AND
. Rotate Instruction in machine code
```

No brackets are allowed in expressions.

B.2.7 IF-STATEMENTS.

```
IF condition THEN
             statements on separate lines
             ELSE
             statements on separate lines
             END

IF condition THEN
             statements on separate lines
             END

IF condition THEN statement ELSE statement END
IF condition THEN statement END
```

B.2.8 LOOPS.

```
REPEAT
     statements on separate lines
UNTIL condition

WHILE condition DO
     statements on separate lines
ENDWHILE

LOOP
     statements on separate lines
EXIT IF condition
     statements on separate lines
ENDLOOP
```

B.2.9 LABLES, BRANCHES, SUBROUTINE DECLARATIONS.

Label:statement
Label:
The label is set at the beginning of a line.

GOTO label
Unconditional branch to a label.

GOTO label IF condition
Conditional branch to a label.

PROC label
Subroutine heading

ENDPROC
Subroutine end.

RETURN
Unconditional exit from a subroutine.

RETURN IF condition
Conditional exit from a subroutine.

B.2.10 SUBROUTINE CALLS.

a) With Arguments:

Arguments are in fact register operations, which are carried out before calling the subroutine.
List of allowed arguments:

REG r=rac

REG r=ra$\pm$1

REG r↑ac

REG r↑a$\pm$c

REG ↑r=rc

REG ↑r$\pm$1=rc

REG r=↑r

REG r=↑r$\pm$1

Note: In this version of MLP "REG" must be omitted.

CALL label<list of arguments>
Unconditional branch to a subroutine

CALL label<list of arguments> IF condition
Conditional branch to a subroutine.

b) With expression.

The expression is first evaluated and then loaded into the A-reg.

CALL label(expression)
Unconditional branch to a subroutine.

CALL label(expression) IF condition
Conditional branch to a subroutine.

c) Without expression or arguments.

CALL label
Unconditional branch to a subroutine.

CALL label IF condition
Conditional branch to a subroutine.

B.2.11 OTHER INSTRUCTIONS IN MLP.

ra=INPUT(ch)
ra is loaded from input-chanel ch.
Ch is a digit between Ø and Ø7

OUTPUT(ch)=expression
The value of the expression is sent to chanel ch.
Ch is a digit between Ø and Ø27.

HALT
Halts the computer.

CODE list of machine code separated by commas.
The instructions are directly transfered to the generated code.

FINISH
This statement must be given at the end of the source-program.

TITLE character string
This statement makes newpage and the string is printed on top of every new page.

B.2.12 CONDITIONS IN MLP.

The conditions refere to the state of the four flip-flops ZERO, CARRY, SIGN, PARITY. These flip-flops may be affected by:

COMPARE expression WITH rac
MASK expression WITH rac
SET ra=expression
Machine code instructions
Increment or decrement instructions

In the COMPARE statement the flip-flops sets as follows:

ZERO	If expression is equal to rac
CARRY	If expression is less than rac
SIGN	If the result of an expression has bit 8 set
PARITY	If the result of an expression has even parity

a) ‹condition› ::=‹flip-flop› TRUE/‹flip-flop› FALSE

b) ‹condition› ::=PARITY OF expression IS ODD/
PARITY OF expression IS EVEN

c) condition ::= expression comparator rac
comparator ::= = / ≠ / ⩾ / < .

When the comparators ⩾ and < is used, the numbers are considered to be positive (∅ to 255), with the exception of < ∅ where the numbers are considered to be a signed integer (-128 to +127).

B.2.13 ERROR REPORTS.

1 ILLEGAL SYNTAX
2 ILLEGAL CONDITION
3 IF/END DIFF.= ....
4 LOOP/ENDLOOP DIFF.= ....
5 REPEAT/UNTIL DIFF.= ....
6 WHILE/ENDWHILE DIFF.= ....
7 ILLEGAL USE OF REG ....
8 MISSING DECLARATION OF ....

Note: MLP does not check if any array references exceeds its permitted size.
MLP does not check that the declared data type is used in the program, i.e. both Arr and Arr(3) will be accepted.

B.2.14 HARDWARE REGISTER USAGE.

A-reg
The A-reg is used to evaluate expressions and is therefore overwritten when a statement containing an expression is used, except when expression is A.

Note: The SET instruction does not use the A-reg when the expression is a constant ≠ ∅.
The INPUT instruction loads the A-reg with the input signal.

E-reg
The E-reg will be overwritten when an array reference is made, except for id(c).

M-reg
The M-reg will be overwritten when a memory reference is made. It will also be used in indirect register control except for:
REG r↑c
REG r↑a (-c)(+c), when a is id or id(c)

B,C,D-reg
Will never be implicity used by MLP.

Alphabet (A-Z)
Numeral (Ø-9)
Spaces
Integer or Octal Arithmetic
Operators: +, -, ., .OR, .XR, .AND
Comparators: =, ≠, ≥, <
Flip-flops: ZERO/CARRY/SIGN/PARITY
States: ODD/EVEN/TRUE/FALSE
Registers: A/B/C/D/E/H/L/M
PARITY OF IS
CODE
COMPARE WITH
REG
OUTPUT
INPUT
SET
IF THEN ELSE END
CALL (IF)
RETURN (IF)
GOTO (IF)
REPEAT UNTIL
WHILE DO ENDWHILE
LOOP EXIT IF ENDLOOP
HALT
PROC ENDPROC
WORDS
BANK
SAME BANK
BANK IS
TITLE
GLOBAL
FINISH

```
<PROGRAM>::=
          <DATA DEC LIST><STATEMENT LIST><SUBROUTINE LIST><END STATEMENT>

<STATEMENT LIST>::=
          <STATEMENT><NL><STATEMENT LIST>/<NULL>

<SUBROUTINE LIST>::=
          <SUBROUTINE><SUBROUTINE LIST>/<NULL>

<END STATEMENT>::=
          FINISH

<SUBROUTINE>::=
          PROC <IDENTIFIER><NL><SUBROUTINE STATEMENT LIST>ENDPROC

<SUBROUTINE STATEMENT LIST>::=
          <STATEMENT><NL><SUBROUTINE STATEMENT LIST>/<RETURN STATEMENT>
          <NL><SUBROUTINE STATEMENT LIST>/<NULL>

<DATA DEC LIST>::=
          <BANK DEC><WORDS DEC LIST><SAME BANK STATEMENT><BANK STATEMENT>

<SAME BANK STATEMENT>::=
          SAME BANK<NL>/<NULL>

<BANK DEC>::=
          BANK <INTEGER><NL>/<NULL>

<BANK STATEMENT>::=
          BANK IS <INTEGER><NL>/<NULL>

<WORDS DEC LIST>::=
          WORDS <ADDRESS DEC LIST><NL>/<NULL>

<ADDRESS DEC LIST>::=
          <IDENTIFIER>↑<INTEGER>,<ADDRESS DEC LIST>
          /<IDENTIFIER>[<INTEGER>]↑<INTEGER>,<ADDRESS DEC LIST>
          /<IDENTIFIER>,<ADDRESS DEC LIST>
          /<IDENTIFIER>[<INTEGER>],<ADDRESS DEC LIST>
          /<IDENTIFIER>[<INTEGER>]/<IDENTIFIER>

<STATEMENT>::=
          <SIMPLE STATEMENT>/<BASIC STATEMENT>
          /<INDIRECT STATEMENT>/*<COMMENT STRING>/HALT

<SIMPLE STATEMENT>::=
          <LABEL>:<SIMPLE STATEMENT>/<SIMPLE STATEMENT>;<COMMENT STRING>
          /<GOTO STATEMENT>/<CALL STATEMENT>/<ASSIGN STATEMENT>
          /<LOOP STATEMENT>/<CONDITIONAL STATEMENT>/<INPUT STATEMENT>
          /<OUTPUT STATEMENT>/<NULL>

<BASIC STATEMENT>::=
          <LABEL>:<BASIC STATEMENT>/<REGISTER LOAD>/<REGISTER CHANGE>
          /<MEMORY REGISTER LOAD>/<MEMORY LOCATION LOAD>/<MASK STATEMENT>
          /<COMPARE STATEMENT>/<MASHINE CODE LIST>/<NULL>

<INDIRECT STATEMENT>::=
          LABEL:<INDIRECT STATEMENT>/<INDIRECT REGISTER LOAD>
```

```
        /<INDIRECT MEMORY LOCATION LOAD>/<STACK OPERATION>

<COMMENT STRING>::=
        <CHAR><COMMENT STRING>/<NULL>

<LOOP STATEMENT>::=
        REPEAT<NL><STATEMENT LIST>UNTIL <CONDITION>
        /WHILE <CONDITION> DO<NL><STATEMENT LIST>ENDWHILE
        /LOOP<NL><STATEMENT LIST>EXIT IF <CONDITION><NL>
        <STATEMENT LIST>ENDLOOP

<GOTO STATEMENT>::=
        GOTO <LABEL>/GOTO <LABEL> IF <CONDITION>

<CALL STATEMENT>::=
        CALL <IDENTIFIER>/CALL <IDENTIFIER> IF <CONDITION>
        /CALL <IDENTIFIER>[<ARGUMENT LIST>]
        /CALL <IDENTIFIER>[<ARGUMENT LIST>]IF <CONDITION>

<CONDITIONAL STATEMENT>::=
        IF <CONDITION> THEN <STATEMENT LIST> ELSE <STATEMENT LIST>END
        /IF <CONDITION> THEN <STATEMENT LIST>END

<ASSIGN STATEMENT>::=
        SET <STORE>=<EXPRESSION>/<STORE>=<EXPRESSION>

<INPUT STATEMENT>::=
        <STORE>=INPUT(<INPUT CHANNEL>)

<OUTPUT STATEMENT>::=
        OUTPUT(<OUTPUT CHANNEL>)=<EXPRESSION>

<REGISTER LOAD>::=
        REG <USER REGISTER>=<PRIMARY>

<MEMORY LOCATION LOAD>::=
        <STORE>=REG <USER REGISTER>

<MEMORY REGISTER LOAD>::=
        M=<ADDRESS REFERENCE>

<REGISTER CHANGE>::=
        REG <USER REGISTER>=<USER REGISTER><ARITH OPERATOR>1

<COMPARE STATEMENT>::=
        COMPARE <EXPRESSION> WITH <PRIMARY>

<MASK STATEMENT>::=
        MASK <EXPRESSION> WITH <PRIMARY>

<INDIRECT REGEISTER LOAD>::=
        REG <REGISTER>↑<ADDRESS REFERENCE>
        /REG <REGISTER>↑<ADDRESS REFERENCE><ARITH OPERATOR>1
        /REG <REGISTER>↑<CONSTANT>
        /REG <REGISTER>=↑<REGISTER>
        /REG <REGISTER>=↑<REGISTER><ARITH OPERATOR>1

<INDIRECT MEMORY LOCATION LOAD>::=
        REG ↑<REGISTER>=<CONSTANT>
        /REG ↑<REGISTER><ARITH OPERATOR>1=<CONSTANT>
        /REG ↑<REGISTER>=<REGISTER>
        /REG ↑<REGISTER><ARITH OPERATOR>1=<REGISTER>

<STACK OPERATION>::=
```

```
        PUSH ↑<REGISTER><ARITH OPERATOR>1=<LIST OF RC>
        /POP <LIST OF REGISTER>=↑<REGISTER><ARITH OPERATOR>1

<LIST OF ARGUMENT>::=
        <ARGUMENT>,<LIST OF ARGUMENT>/<ARGUMENT>

<ARGUMENT>::=
        <REGISTER LOAD>/<REGISTER CHANGE>
        /<INDIRECT REGISTER LOAD>
        /<INDIRECT MEMORY LOACTION LOAD>

<LIST OF RC>::=
        <REGISTER>,<LIST OF RC>/<REGISTER>
        /<CONSTANT>,<LIST OF RC>/<CONSTANT>

<LIST OF REGISTER>::=
        <REGISTER>,<LIST OF REGISTER>/<REGISTER>

<CONDITION>::=
        <STATUS FLAG><TRUTH STATE>
        /<EXPRESSION><COMPARATOR><PRIMARY>
        /PARITY OF <EXPRESSION> IS <PARITY STATE>

<STORE>::=
        <USER REGISTER>/<ADDRESS REFERENCE>

<EXPRESSION>::=
        <CALL STATEMENT><ARITH OPERATOR><ORDINARY EXPRESSION>
        /<CALL STATEMENT>.<CODE STATEMENT><ORDINARY EXPRESSION>
        /<INPUT STATEMENT><ARITH OPERATOR><ORDINARY EXPRESSION>
        /<INPUT STATEMENT>.<CODE STATEMENT><ORDINARY EXPRESSION>
        /<ORDINARY EXPRESSION>

<ORDINARY EXPRESSION>::=
        <TERM>/<ARITH OPERATOR><TERM>
        /<ORDINARY EXPRESSION><ARITH OPERATOR><TERM>

<PRIMARY>::=
        <USER REGISTER>/<ADDRESS REFERENCE>/<CONSTANT>

<CODE STATEMENT STRING>::=
        <CODE STATEMENT>,<CODE STATEMENT STRING>
        /<CODE STATEMENT>

<TERM>::=
        <PRIMARY>/<PRIMARY>.<CODE STATEMENT>

<LABEL>::=
        <LETTER DIGIT STRING>

<ADDRESS REFERENCE>::=
        <IDENTIFIER>/<IDENTIFIER>[<INDEX>]

<INDEX>::=
        <CONSTANT>/<REGISTER>/<IDENTIFIER>

<IDENTIFIER>::=
        <LETTER><LETTER DIGIT STRING>/<NON-REGISTER LETTER>

<CONSTANT>::=
        <INTEGER>/<ARITH OPERATOR><INTEGER>/"<CHAR>"

<POSITIVE CONSTANT>::=
        <INTEGER>/+<INTEGER>/"<CHAR>"
```

```
<LETTER DIGIT STRING>::=
        <LETTER><LETTER DIGIT STRING>
       /<DIGIT><LETTER DIGIT STRING>

<USER REGISTER>::=
        <REGISTER>/M

<REGISTER>::=
        A/B/C/D/E/H/L

<INPUT CHANNEL>::=
        <INTEGER>  (RANGE: 0-07)

<OUTPUT CHANNEL>::=
        <INTEGER>  (RANGE: 0-07, 10-017, 20-027)

<STATUS FLAG>::=
        ZERO/CARRY/SIGN/PARITY

<TRUTH STATE>::=
        TRUE/FALSE

<PARITY STATE>::=
        ODD/EVEN

<ARITH OPERATOR>::=
        +/-

<COMPARATOR>::=
        %>/</%=/=

<CODE STATEMENT>::=
        (DEFINED IN PARTICULAR IMPLEMENTATION)

<INTEGER>::=
        <DIGIT STRING>/0<OCTAL STRING>

<DIGIT STRING>::=
        <DIGIT><DIGIT STRING>/<DIGIT>

<OCTAL STRING>::=
        <OCTAL DIGIT><OCIAL STRING>/<OCTAL DIGIT>

<LETTER>::=
        A/B/C...../Z

<NON-REGISTER LETTER>::=
        F/G/I/K/O../Z

<DIGIT>::=
        0/1/2/3/4/5/6/7/8/9

<OCTAL DIGIT>::=
        0/1/2/3/4/5/6/7

<CHAR>::=
        (ANY OF THE SET OF RECOGNISED CHARACTERS IN A
        PARTICULAR IMPLEMENTATION EXCEPT FOR <NL>)

<NL>::=
        START NEW LINE

<NULL>::=
```

APPENDIX C


```
/                               BANK 1
/                               WORDS P1↑25,Q1,R1↑050,S1↑100,TEMP[50],T1
P1=031
Q1=032
R1=050
S1=0144
TEMP=0145
T1=0227
/                               GLOBAL SUB,SUB4
        .GLOBL SUB,SUB4
/                               SAME BANK
/
/                               BANK IS 1
        LHI     01
/
/                               ***** REGISTER CONTROL *****
/                               REG B=25
        LBI     031
/                               REG C=D+1
        LCD
        INC
/                               REG D=TEMP[24]+1
        LLI     TEMP+030
        LDM
        IND
/                               REG D=TEMP[T1]-1
        LLI     T1
        LAM
        ADI     TEMP
        LLA
        LDM
        DCD
/                               TEMP[012]=REG A
        LLI     TEMP+012
        LMA
/                               M=TEMP[P1]
        LLI     P1
        LAM
        ADI     TEMP
        LLA
/                               MASK 24+Q1 WITH 255
        LAI     030
        LLI     Q1
        ADM
        NDI     0377
/
/                               ***** INDIRECT REGISTER CONTROL *****
/                               REG B↑TEMP[200]+8
        LBI     TEMP+0310+010
/                               REG B↑TEMP[Q1]-077
        LAM
        ADI     TEMP-077
        LBA
/                               REG ↑D=B
        LLD
        LMB
/                               REG ↑L+1=9
        INL
        LMI     011
/                               REG ↑B-1=C
        LLB
        DCL
```

```
        LMC
/                           REG B=↑C
        LLC
        LBM
/                           REG C=↑L+1
        INL
        LCM
/                           REG D=↑B-1
        LLB
        DCL
        LDM
/                           PUSH ↑B+1=0,25,C
        LLB
        LMI     0
        INB
        LLB
        LMI     031
        INB
        LLB
        LMC
        INB
/                           POP A,B,C=↑L-1
        DCL
        LAM
        DCL
        LBM
        DCL
        LCM
/
/                           ***** ASSIGNMENTS *****
/                           SET A,B,C=0
        XRA
        LBA
        LCA
/                           SET T1=-P1+7
        LLI     P1
        SUM
        ADI     07
        LLI     T1
        LMA
/                           SET TEMP[7]=-8
        LLI     TEMP+07
        LMI     -010
/                           SET TEMP[Q1]=TEMP[P1]-TEMP[T1]
        LLI     P1
        LAM
        ADI     TEMP
        LLA
        LAM
        LEA
        LLI     T1
        LAM
        ADI     TEMP
        LLA
        LAE
        SUM
        LEA
        LLI     Q1
        LAM
        ADI     TEMP
        LLA
        LAE
        LMA
/                           SET B=INPUT(5)+S1
```

```
        5*2+101
        LLI     S1
        ADM
        LBA
/                       SET C=CALL SUB+19
        CAL     SUB
        ADI     023
        LCA
/
/                       ***** LABLES,GOTO-STM.,IF-STM. *****
/               LAB1:
LAB1=.
/               LAB2:   GOTO LAB1
LAB2=.
        JMP     LAB1
/                       GOTO LAB1 IF PARITY OF Q1 IS ODD
        LLI     Q1
        LAM
        CPI     0
        JTP     LAB1
/                       IF P1<0 THEN Q1=0
        LLI     P1
        LAM
        CPI     0
        JFS     X1
        XRA
        LLI     Q1
        LMA
/                               ELSE S1=0 END
        JMP     X2
X1=.
        XRA
        LLI     S1
        LMA
X2=.
/                       IF P1<0 THEN
        LLI     P1
        LAM
        CPI     0
        JFS     X3
/                               IF Q1=0 THEN R1=0
        LLI     Q1
        LAM
        CPI     0
        JFZ     X4
        XRA
        LLI     R1
        LMA
/                                       S1=0
        LLI     S1
        LMA
/                                       ELSE T1=0
        JMP     X5
X4=.
        XRA
        LLI     T1
        LMA
/                                       END
X5=.
/                               ELSE
        JMP     X6
X3=.
/                               P1=0
        XRA
```

```
        LMA
/                                   END
X6=.
/
/                                   ***** DO-LOOPS *****
/                                   LOOP
XL7=.
/                                   EXIT IF ZERO TRUE
        JTZ     XL8
/                                   ENDLOOP
        JMP     XL7
XL8=.
/
/                                   REPEAT
XR9=.
/                                   UNTIL 255=L
        LAI     0377
        CPL
        JFZ     XR9
/
/                                   WHILE A<10 DO.
XW10=.
        CPI     012
        JFC     X11
/                                   ENDWHILE
        JMP     XW10
X11=.
/
/                                   ***** SUBROUTINE CALLS *****
/                                   CALL SUB
        CAL     SUB
/                                   CALL SUB IF B=7
        LAB
        CPI     07
        CTZ     SUB
/                                   CALL SUB4<A,B,C=7,↑D=C,L↑TEMP[0]>
        LCI     07
        LLD
        LMC
        LLI     TEMP
        CAL     SUB4
/
/                                   FINISH
        .END
```

HILP 80
A HIGH LEVEL PROGRAMMING LANGUAGE FOR INTEL 8080

J. E. ASPERNÄS

Report 7626 (C) May 1976
Department of Automatic Control
Lund Institute of Technology

H I L P 8 0.

A HIGH LEVEL PROGRAMMING LANGUAGE FOR INTEL 8080.

Author Jan-Eric Aspernäs
Supervisors Leif Andersson
Johan Wieslander

CONTENTS.

1. INTRODUCTION.

This report describes a High Level Programming Language, HILP 80, for micro processors. This version of HILP 80 is made for INTEL 8080, but could be changed to fit any simular micro processor.

The compiler is based on STAGE 2 (Waite, 1973). The compiler generates assembly code. A flow diagram (fig 1) shows the way from HILP 80 source program to a running Intel 8080.

The compilation is intended to take place on a host machine. At Lund Institute of Technology (LTH), a PDP-15 is used as a host machine. The compiler is almost machine independent, and could be implemented on other computers. The macro processor STAGE 2 requires about 3 K of core, and the templates need about 12 K with high degree of packing. The translation time for the in appendix given exemple was about 65 seconds.

The language HILP 80 could be used at different levels down to assembly code. The medium and low level statements are valuable to increase the efficiency e.g. in loops.

Improvals of the limited data structures and arithmetic facilities are under discussion.

EFFICIENCY OF HILP 80.

The main responsibility for the efficiency of the generated code lies in the hands of the user. Thus it is up to the user to decide e.g. whether registers should be used instead of address references in order to increase the efficiency.

It is hard to measure the efficiency of HILP 80 depending on the various levels of the statements. However it is always possible to generate a code with very high degree of efficiency.

When addressing the data, INTEL 8080 uses register pair H&L, where H contains the high address (= bank) and L contains the low address. However, the A register could be loaded and stored directly using a two word address or loaded and stored indirectly using register pairs B&C or D&E. The different addressing facilities and the hardware usage of register pair H&L as an accumulator in double precision makes it hard to control the addressing in an efficient way. However, HILP 80 mostly solves this problem for the user, but in some cases the user could act in order to increase the efficiency. For this purpose HILP 80 has two "Bank manipulation statements".

REFERENCE.

Waite, W.M., 1973, Implementing Software for Non-Numeric Applications, Prentice-Hall.

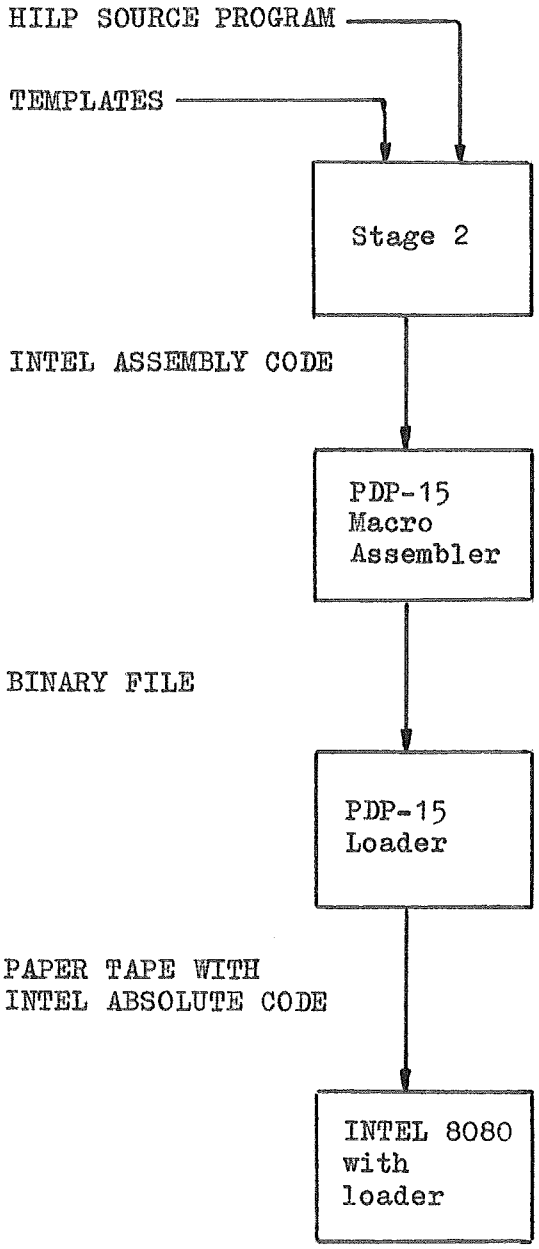


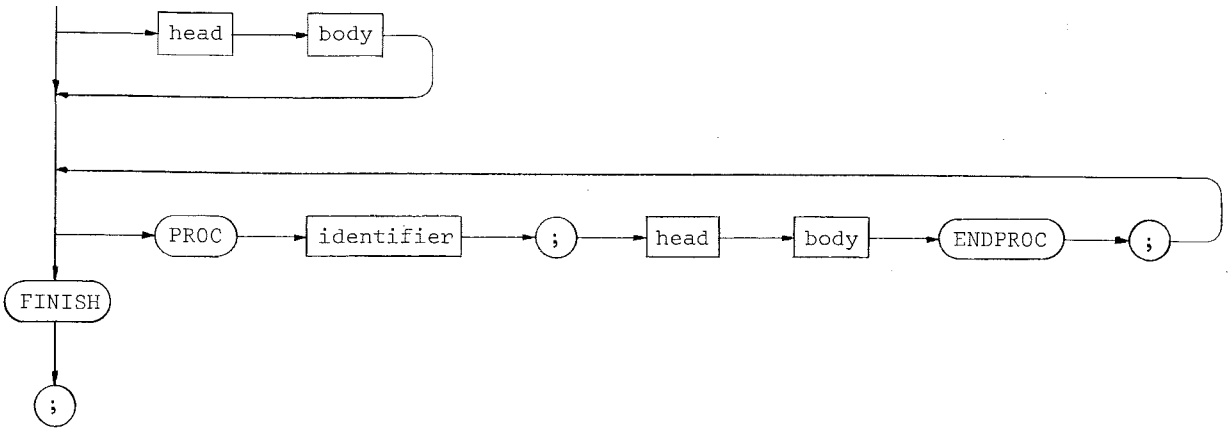
fig 1. From HILP 80 source to loaded INTEL 8080.

The syntax of HILP 80 is described in flow diagram form. The terminal symbols of the language are enclosed by circles or by ovals e.g. (REPEAT) . The nonterminal symbols are written with small letters and enclosed by rectangles e.g. [identifier] .

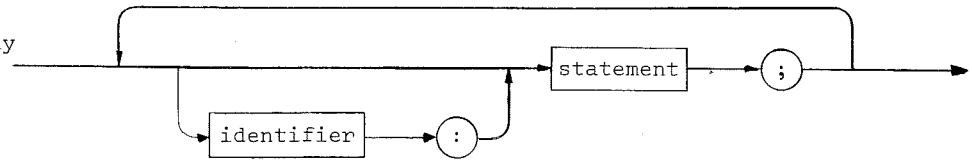
The semicolon is the "end of statement" marker. In the actual source program it has the following equivalent forms (↵ denotes carriage return):

1. ;↵
2. ↵
3. ; Any sequence of characters ↵

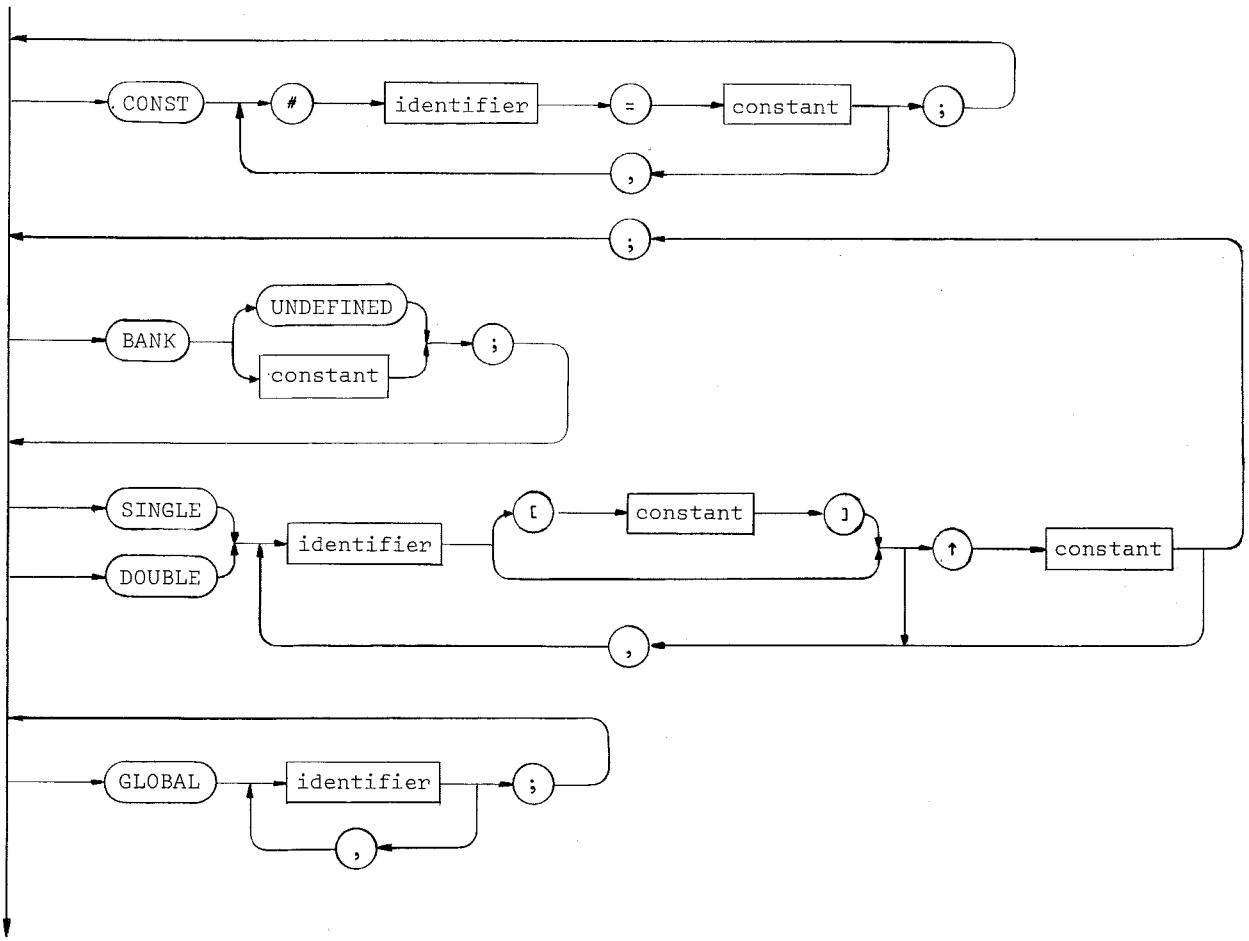
program

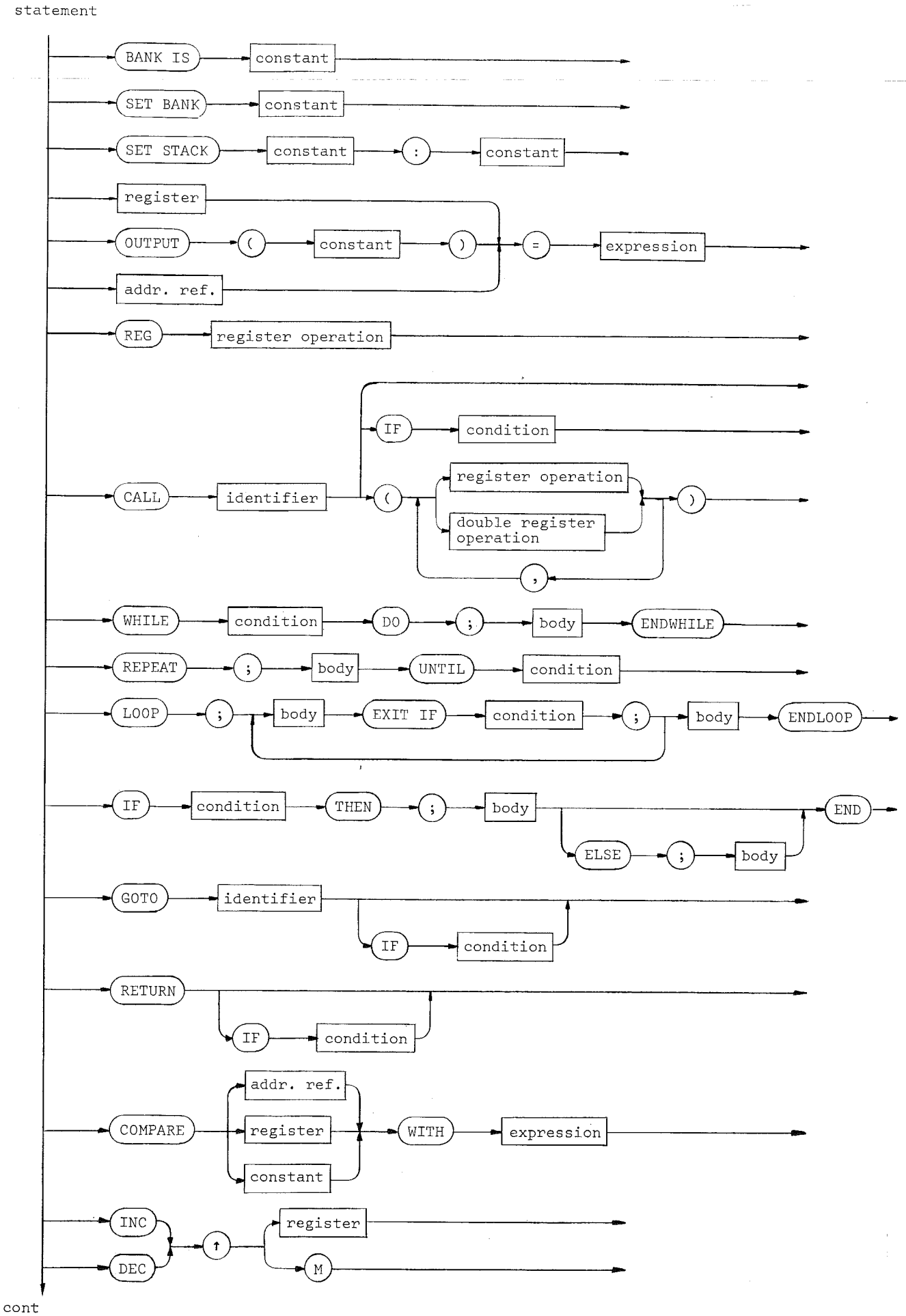


body

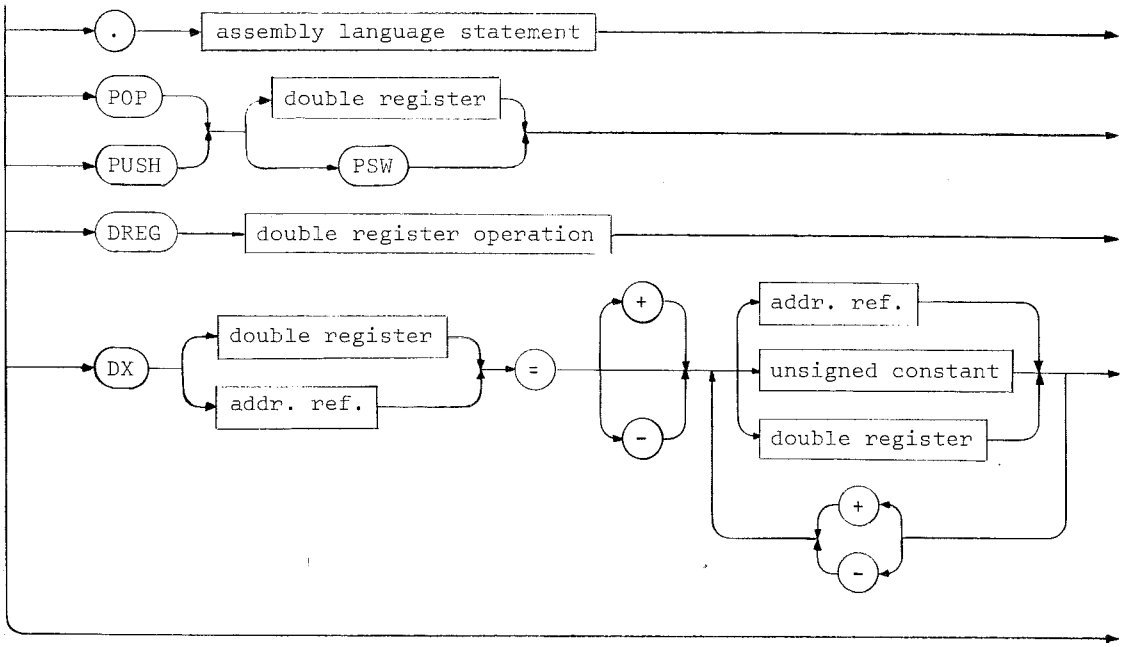


head

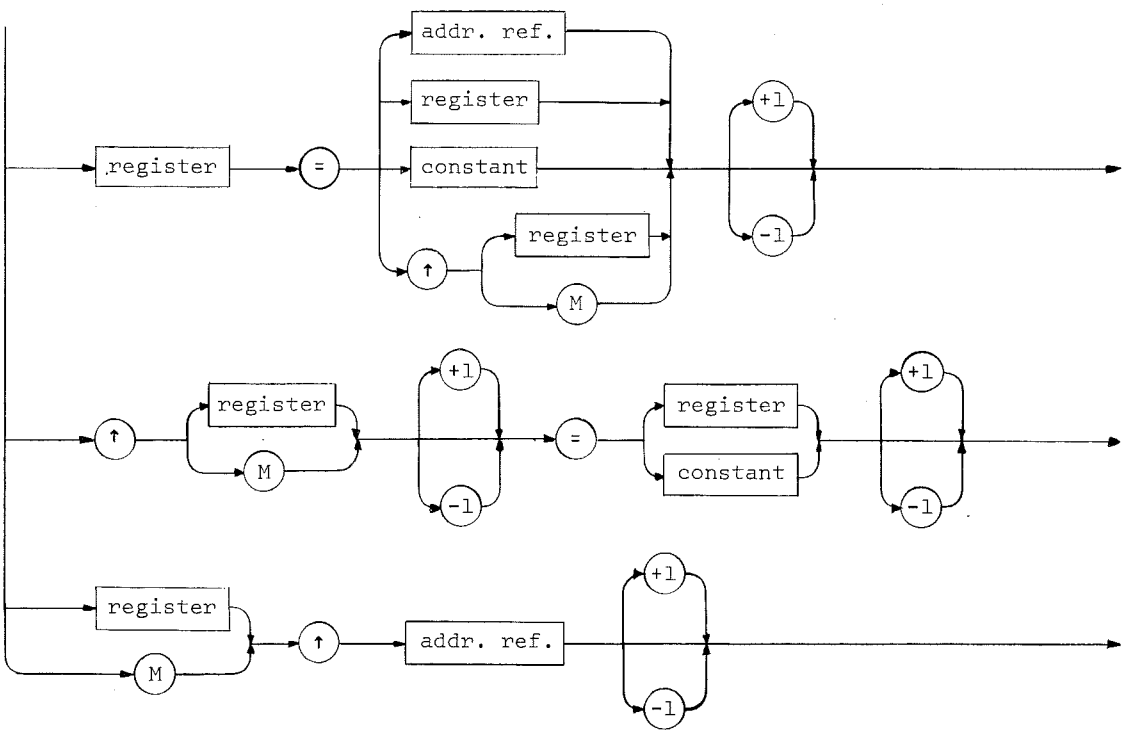



statement
BANK IS
constant
SET BANK
constant
SET STACK
constant
:
constant
register
OUTPUT
(
constant
)
=
expression
addr. ref.
REG
register operation
IF
condition
CALL
identifier
(
register operation
double register operation
,
)
WHILE
condition
DO
;
body
ENDWHILE
REPEAT
;
body
UNTIL
condition
LOOP
;
body
EXIT IF
condition
;
body
ENDLOOP
IF
condition
THEN
;
body
ELSE
;
body
END
GOTO
identifier
IF
condition
RETURN
IF
condition
COMPARE
addr. ref.
register
constant
WITH
expression
INC
DEC
↑
register
M
cont

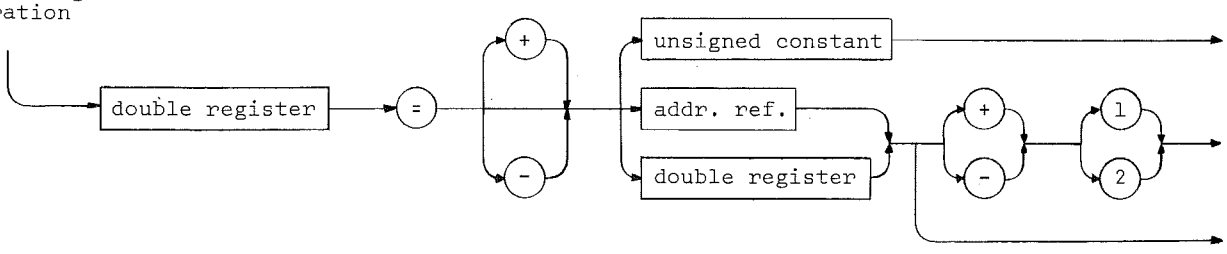
statement continued



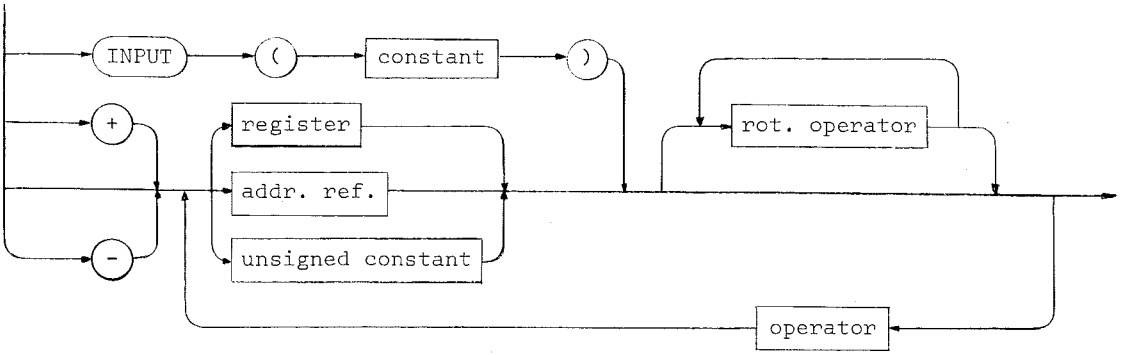
register operation



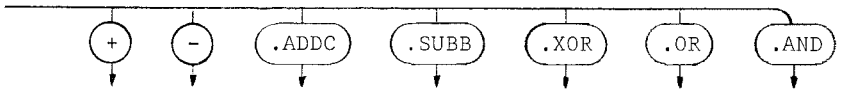
double register operation



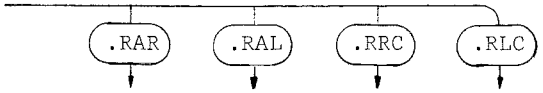
expression



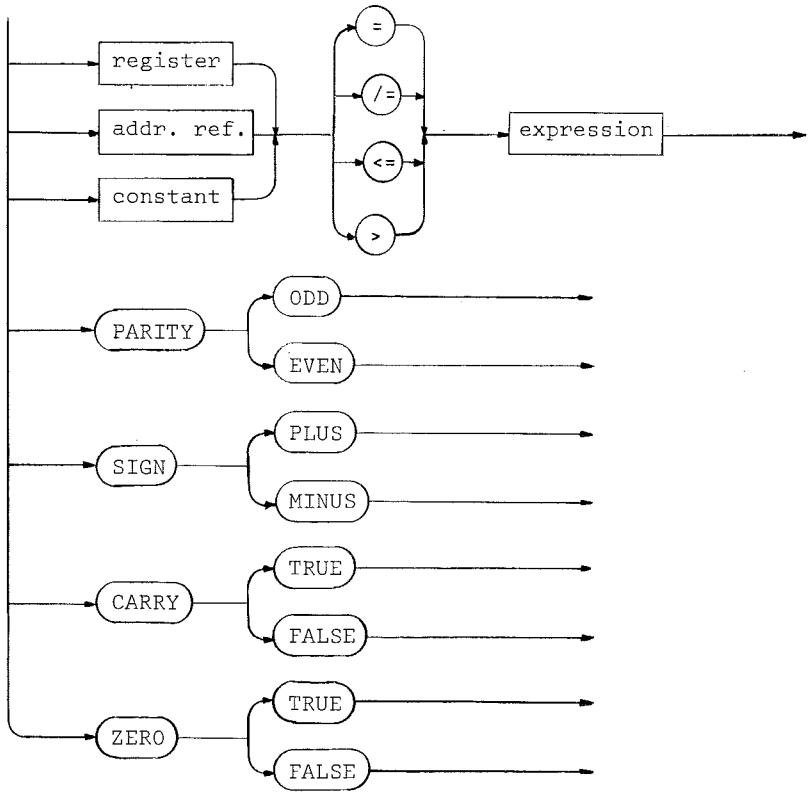
operator



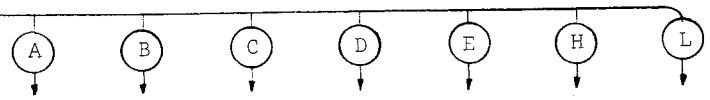
rot. operator



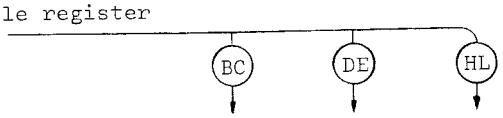
condition



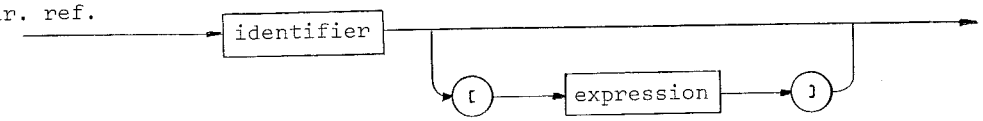
register



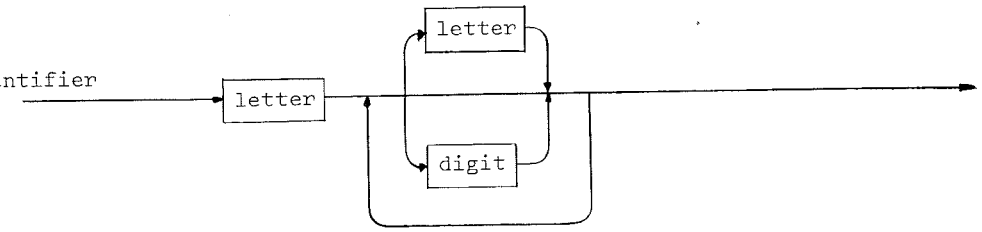
double register



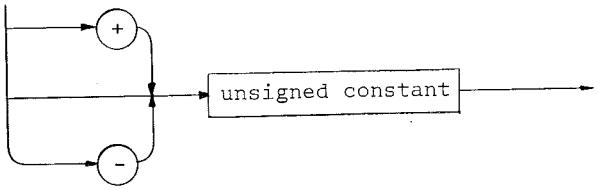
addr. ref.



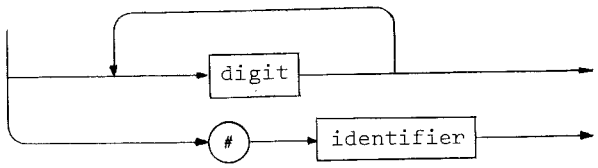
identifier



constant



unsigned constant



3.1 GENERAL RULES.

Main rules for statements in HILP:

1 Only one statement is allowed per line.

2 Leading tabs and spaces are allowed.
HILP words should be separated by one space.
Logical operators, +, -, ., = should not be separated by spaces.

3 Comments may be used in HILP:
a) ;comment string
b) statement;comment string

4 Names may consist of up to six characters, starting with a letter, and for constants starting with #.

5 Empty lines may be used.

6 Numbers are interpreted as octal if they start with ∅ and decimal otherwise.

7 Integers in HILP should be within the ranges:
SINGLE : -128 to +127
-0200 to +0177
DOUBLE : -32768 to +32767
-0100000 to +077777

3.2 ABBREVIATIONS.

r	register (A,B,C,D,E,H,L)
c	constant
id	identifier
a	address reference (=id, id[expression])
rac	either r or a or c above
ra	either r or a above
rc	either r or c above
code	Intel assembly code
M	memory location addressed by H and L
dr	double register (BC,DE,HL)

a) CONST declaration.

The user may define constants, which will be replaced by the assembler with their current values.
The constants could be useful e.g. in the bank declaration.

b) VARIABLE declaration.

1. BANK c

Variables used in the program must be declared and their bank addresses specified. The bank number given in the declaration is valid until another bank declaration is made.

SINGLE declaration.
DOUBLE declaration.

The user may define addresses for the variables by stating ↑c after the variables in the list, or state ↑c for the first variable and then let the others be defined consecutively until another address is stated. The first word declared in a bank is given the address Ø if the user does not give a start address. No error report will be given if the user declares addresses greater than 255 (Ø377).

2. BANK UNDEFINED

This bank declaration could be used in procedures, where the variables define different memory locations (different high addr., same low addr.) depending on the actual value in the H register.

SINGLE declaration as above.
DOUBLE declaration as above.

c) GLOBAL declaration.

If references are made to names in other programs, the loader program has to know that these names are global, otherwise the loader will not be able to set up common addresses for these names.

d) SET STACK c:c.

As HILP implicitly uses the stack, this statement must always be given. The first constant is the bank address and the second is the low address.

NOTE: Array index runs from Ø to i-1.
Only one dimensional arrays are allowed.
No variables may be preset.

a) GENERAL.

HILP translates the source program statement by statement going once from top to bottom. This technigue causes some optimization problems when a label is generated by HILP e.g. in loop statements. The problems mentioned arise from the fact that when a label is jumped to during the execution, the translation path is different from the execution path.

For optimization during the translation, HILP uses internal variables:

LASTA	last value of the A register.
LASTM	last memory reference addressed by M (H&L).
LASTH	last bank value in the H register.

As for LASTH, when M is to be set to an address of a variable, HILP will be able to check LASTH against the bank address of that variable and if equal not update the H register. In case of BANK UNDEFINED, LASTH is not used since the user in this case is supposed to take full responsibility for the bank address setting (HILP will always assume that the H register is correctly set).

BANK ADDRESS MANIPULATION STATEMENTS.

SET BANK c.

The constant c is loaded into the H register if LASTH is different from c.

BANK IS c.

This statement sets LASTH to c. It is an information to the translator and will never generate any code.

b) THE BANK ADDRESSES ARE SPECIFIED IN THE HEAD.

1) GENERAL PROGRAMMING.

HILP 80 will take full responsibility for the bank address manipulation. Thus the user does not have to care about the internal usage of LASTH. Some optimization of the bank address setting could be obtained, especially if all the variables are specified in the same bank. See Appendix A.

2) THE USAGE OF POINTER OPERATIONS.

The statements:
```
REG ↑...=...
REG ...=↑...
INC ↑...
DEC ↑...
```
will assume that the H register is correctly set. To prevent errors, HILP 80 will generate an error report (NO BANK IN THE H REGISTER) if LASTH has been destroyed when one of the above mentioned statements is reached. In this case the user must give:

1. SET BANK c if the value of H is uncertain.
2. BANK IS c if the value of H is the desired bank address.

LASTH will be destroyed in the following statements:

```
REG H=...
H=expression
POP HL
DREG HL=...
DREG dr=a,+a,-a (±1,2)
DX ...=...
.code when H is changed.
When ↑M is incremented or decremented.
label:
PROC
CALL
REPEAT
LOOP
ENDLOOP    (See Appendix A for further details)
WHILE                    "
ENDWHILE                 "
END                      "
```

NOTE: Some statements containing variables will implicitly set the bank address in the H register. See Appendix B for further details.

c) BANK UNDEFINED.

This type of bank declaration could be used e.g. in procedure declarations. All variables declared in the head must refer to BANK UNDEFINED. The user must garantee that the H register is updated before using statements containing variables with their bank addresses unspecified.

No error report will be given if LASTH is destroyed in following statements:

```
REG ↑...=...
REG ...=↑...
INC ↑...
DEC ↑...
```

In double precision the DX statement is reduced to the following allowed types when variables with their bank addresses unspecified are used:

```
DX a=dr
DX dr="expression with dr and c. The expression
       may start with a, +a, -a."
```

NOTE: For internal procedures, where the main program has specified bank addresses for the variables, a reference to such a variable could cause an adjustment of the H register according to the rules for addressing given in Appendix B.

d) NO VARIABLES DECLARED.

This type of head could be used e.g. in procedure declarations. No error report will be given if LASTH is destroyed in following statements:

```
REG ↑...=...
REG ...=↑...
INC ↑...
DEC ↑...
```

NOTE: For internal procedures, where the main program has specified bank addresses for the variables, a reference to such a variable could cause an adjustment of the H register according to the rules for addressing given in Appendix B.

a) $r_1=r_2ac(\pm1)$

Register r_1 will be loaded with the value of r_2ac, then r_1 will be incremented or decremented if necessary.

b) $r\uparrow a(\pm1)$

Register r will be loaded with the low address of a, a-1, a+1.

c) $M\uparrow a(\pm1)$

Register L will be loaded with the low address of a. The bank address of a is loaded into the H register. Register H&L is incremented or decremented if necessary.

d) $\uparrow r_1(\pm1)=r_2c(\pm1)$

The value of r_1 is moved to register L. Register L is incremented or decremented if necessary. The value of $r_2c(\pm1)$ is moved to the memory.

e) $\uparrow M(\pm1)=r_2c(\pm1)$

Register H&L is incremented or decremented if necessary. The value of $r_2c(\pm1)$ is moved to the memory.

f) $r_1=\uparrow r_2(\pm1)$

The value of r_2 is moved to register L. Register L is incremented or decremented if necessary. Register r_1 will be loaded from the memory.

g) $r_1=\uparrow M(\pm1)$

Register H&L is incremented or decremented if necessary. Register r_1 will be loaded from the memory.

NOTE: Any memory reference d) to g) above will assume that the H register is correctly set.

3.6 DOUBLE REGISTER OPERATION.

a) dr_1=c

The constant will be loaded into double register dr_1.

b) dr_1=dr_2,a (±1,2)

Double register dr_1 will be loaded with the value of dr_2,a. Then if necessary, dr_1 will be incremented or decremented once or twice.

c) dr_1=-dr_2,a (±1,2)

The value of dr_2,a is complemented via the A register before loaded into dr_1. Then dr_1 will be incremented or decremented once or twice if necessary.

3.7 PROCEDURE CALLS WITH ARGUMENTS.

Arguments are register operations or double register operations, which are carried out before the procedure is called. The same procedure could be called with any number of arguments.

Example: CALL SUB(A=B,DE=07745) is equivalent to:

```
REG A=B
DREG DE=07745
CALL SUB
```

3.8 CONDITIONS IN HILP 80.

The conditions refer to the state of the four flip-flops: ZERO, CARRY, SIGN and PARITY. These flip-flops may be affected by:

COMPARE rac WITH expression
expression evaluating
Assembly code
Increment or decrement instructions

In the COMPARE statement the flip-flops are set:

ZERO	If (expr. - rac) = 0
CARRY	If (expr. - rac) < 0

3.9 INC and DEC statements.

These statements shall be interpreted as:
Set up the pointer to the memory, then increment or decrement the value addressed by M.

3.10 HARDWARE REGISTER USAGE.

a) SINGLE PRECISION.

The A register is used to evaluate expressions and is therefore overwritten when a statement containing an expression is used, except for:

```
ra=c (c≠0)
a=a+1
a=a-1
a=r
```

NOTE: The INPUT statement loads the A register. The OUTPUT statement takes the value from A.

M (H&L) will be overwritten when a memory reference is made, except for special cases. See Appendix B.

b) DOUBLE PRECISION.

Any address reference in double precision will use HL.

In the DX statement HL is used to evaluate expressions. DE is used for temporary storage when address references or constants ≠ ±1,2 are used.

Since the hardware only allows double add to HL, subtractions of address references and double registers will be carried out via complementing in the A register.

3.11 ERROR REPORTS.

```
ILLEGAL SYNTAX
ILLEGAL CONDITION
IF/END DIFF =
LOOP/ENDLOOP DIFF =
REPEAT/UNTIL DIFF =
WHILE/ENDWHILE DIFF =
NO EXIT IN LOOP/ENDLOOP
NO BANK IN H REGISTER
MISSING DECLARATION OF
MISSING BANK DECLARATION
REMOVE TRAILING SPACE FROM
```

NOTE: HILP does not check that the declared data type is used in the program, e.g. both ARR and ARR(3) will be accepted.

HILP does not check if any array references exceed their permitted size.

Alphabet (A-Z)

Integers (0-9)

Spaces

↑ #

Operators: + - .ADDC .SUBB .XOR .OR .AND

Rotate operators: .RAR .RAL .RRC .RLC

Comparators: = /= <= >

Flip-flops: ZERO CARRY SIGN PARITY

States: TRUE FALSE PLUS MINUS ODD EVEN

Register: A B C D E H L

Double register: BC DE HL

M

CONST

BANK

SINGLE

DOUBLE

GLOBAL

BANK IS

UNDEFINED

SET BANK

SET STACK

OUTPUT

INPUT

REG

CALL (IF)

GOTO (IF)

RETURN (IF)

WHILE DO ENDWHILE

REPEAT UNTIL

LOOP EXIT IF ENDLOOP

IF THEN ELSE END

COMPARE WITH

POP PUSH

PSW

INC DEC

DREG

DX

PROC ENDPROC

FINISH

1. REGISTER USAGE.

The usage of registers instead of address references will increase the efficiency, which is very valuable in loops. Consider the following example, where V and I refer to bank 1.

Source code.		Generated code.	
I=0		XRA	A
		STA	0400+I
REPEAT	XR1=.		
V[I]=0		LDA	0400+I
		ADI	V
		MOV	L,A
		MVI	H,1
		XRA	A
		MOV	M,A
I=I+1		MVI	L,I
		INR	M
UNTIL I=031		MVI	A,031
		CMP	M
		JNZ	XR1
- - - - -		- - - - -	
SET BANK 1		MVI	H,1
REG L↑V[0] -1		MVI	L,V-1
REG A↑V[030]		MVI	A,V+030
REPEAT	XR1=.		
BANK IS 1			
REG ↑L+1=0		INR	L
		MVI	M,0
UNTIL L=A		CMP	L
		JNZ	XR1

a) STATEMENTS WHICH IMPLICITLY DESTROY LASTH.

Label:
The label could be jumped to with different banks in the H register.

Incrementing or decrementing ↑M.
As H&L is incremented or decremented in this case, the H register also could be adjusted.

CALL
The procedure may have changed the H register.

PROC
The procedure could be called with different bank values in the H register.

END (from IF THEN)
LASTH will be destroyed if LASTH after the condition is evaluated, is different from LASTH just before END.

END (from IF THEN ELSE)
LASTH will be destroyed if LASTH just before ELSE is different from LASTH just before END.

REPEAT
LASTH just before REPEAT could differ from LASTH after the condition in UNTIL is evaluated.

LOOP
LASTH just before LOOP could differ from LASTH just before ENDLOOP.

ENDLOOP (Two or more EXITs in the LOOP)
LASTH after the condition evaluating in the EXITs could differ.

WHILE
LASTH just before WHILE could differ from LASTH just before ENDWHILE.

ENDWHILE
LASTH will be destroyed unless the condition evaluating in WHILE will set LASTH.

When a label is generated by HILP, LASTH will be destroyed. Thus an adjustment of the H register will take place when M is to be set to a variable. This updating of the H register could be unnecessary if the value is equal to the value of H when the jump takes place. To inhibit an unnecessary adjustment of the H register, the user may give a BANK IS c statement after the following statements:

label:
PROC
CALL
REPEAT
LOOP
ENDLOOP
WHILE
ENDWHILE

In this case the user must garantee that:

The problems discussed under a) could not occur.

The H register already holds the value c. The SET BANK c statement could be used to update the H register.

Consider the following example, which is the same as given in 1). V and I are specified in bank 1.

Source code.		Generated code.	
I=0		XRA	A
		STA	0400+I
SET BANK 1		MVI	H,1
REPEAT	XR1=.		
BANK IS 1			
V(I)=0		LDA	0400+I
		ADI	V
		MOV	L,A
		XRA	A
		MOV	M,A
I=I+1		MVI	L,I
		INR	M
UNTIL I=031		MVI	A,031
		CMP	M
		JNZ	XR1

Compared to the example given in 1) the improval is that the instruction "MVI H,1" is moved outside the loop. The technique given is most valuable when all the variables are specified in the same bank.

APPENDIX B.

DIRECT LOAD AND STORE IN SINGLE PRECISION.

Most memory references use H&L (M), when addressing variables, but the A register could be loaded and stored directly without using M. In this case the instructions LDA and STA are used instead of MOV A,M and MOV M,A.

Definition: simple addr = id or id[c].

Direct load and store of the A register will be used:

1 When the expression has no address references, except that it may start with a simple addr.

2 Simple addr=expression

3 REG A=simple addr(±1)

APPENDIX C.

GENERATED CODE. **HILP 80 STATEMENTS.**

```
/                               CONST ¤BK1=011,¤BK2=19,¤STBK=3
%BK1=011
%BK2=023
%STBK=03
/                               BANK ¤BK1
/                               SINGLE X1↑96,Y1,Z1,V1[10],TEMP1↑0377
X1=0140
Y1=X1+01
Z1=Y1+01
V1=Z1+01
TEMP1=0377
/                               BANK ¤BK2
/                               SINGLE V2[9],X2,Y2,Z2
V2=0
X2=V2+011
Y2=X2+01
Z2=Y2+01
/                               DOUBLE DV2[6],DX2,DY2↑0350,DZ2
DV2=Z2+01
DX2=DV2+06+06
DY2=0350
DZ2=DY2+01+01
/                               SINGLE TEMP2↑0377
TEMP2=0377
/                               GLOBAL SUB,LABEL
        .GLOBL SUB,LABEL
/
/                               SET STACK ¤STBK:255
        LXI     SP,%STBK*0400+0377
/                               ;*** REGISTER OPERATIONS ***
/                               REG A=0
        MVI     A,0
/                               REG A=B
        MOV     A,B
/                               REG B=B-1
        DCR     B
/                               REG C=V1[6]
        LXI     H,%BK1*0400+V1+06
        MOV     C,M
/                               REG C=V2[X1-X2]-1
        LDA     %BK1*0400+X1
        LXI     H,%BK2*0400+X2+0
        SUB     M
        ADI     V2
        MOV     L,A
        MOV     C,M
        DCR     C
/                               REG M↑Y1
        LXI     H,%BK1*0400+Y1+0
/                               REG L↑Y2
        MVI     L,Y2
/                               SET BANK ¤BK1
```

```
/                       REG C=↑C-1
        MOV     L,C
        DCR     L
        MOV     C,M
/                       REG ↑L+1=D+1
        INR     L
        MOV     M,D
        INR     M
/                       SET BANK ¤BK2
        MVI     H,%BK2
/                       DEC ↑B
        MOV     L,B
        DCR     M
/                       ;*** ASSIGNMENTS OF EXPRESSIONS ***
/                       V2[X1]=V2[X1]-1
        LDA     %BK1*0400+X1
        ADI     V2
        MOV     L,A
        DCR     M
/                       V1[0]=V2[X1]+Y1
        MOV     A,M
        LXI     H,%BK1*0400+Y1+0
        ADD     M
        STA     %BK1*0400+V1+0
/                       Z1=0
        XRA     A
        STA     %BK1*0400+Z1
/                       Z2=7
        LXI     H,%BK2*0400+Z2+0
        MVI     M,07
/                       Y1=B
        LXI     H,%BK1*0400+Y1+0
        MOV     M,B
/                       C=D
        MOV     C,D
/                       V2[2]=0
        STA     %BK2*0400+V2+02
/                       OUTPUT(6)=INPUT(15)+X1
        IN      017
        MVI     L,X1+0
        ADD     M
        OUT     06
/
/               V2[C.AND X2]=-25+Y1,ADDC Y2,RAR-V1[Y2,OR V2[V1[Y1]]]
        MVI     A,-031
        MVI     L,Y1+0
        ADD     M
        LXI     H,%BK2*0400+Y2+0
        ADC     M
        RAR
        PUSH    PSW
        MOV     A,M
        PUSH    PSW
        LDA     %BK1*0400+Y1
        ADI     V1
        MOV     L,A
        MVI     H,%BK1
        MOV     A,M
        ADI     V2
        MOV     L,A
        POP     PSW
        MVI     H,%BK2
        ORA     M
        ADI     V1
```

```
        MOV     L,A
        POP     PSW
        MVI     H,%BK1
        SUB     M
        PUSH    PSW
        MOV     A,C
        LXI     H,%BK2*0400+X2+0
        ANA     M
        ADI     V2
        MOV     L,A
        POP     PSW
        MOV     M,A
/
/                       ;*** IF STM, GOTO STM, LOOPS ***
/                       IF 0=V2[X1] THEN
        LDA     %BK1*0400+X1
        ADI     V2
        MOV     L,A
        MOV     A,M
        CPI     0
        JNZ     X1
/                                       X1=X1-1
        LXI     H,%BK1*0400+X1+0
        DCR     M
/                                       ELSE
        JMP     X2
X1=.
/                                       V2[X1]=0
        XRA     A
        MOV     M,A
/                                       END
X2=.
/                       GOTO LABEL IF Z1=0
        XRA     A
        LXI     H,%BK1*0400+Z1+0
        CMP     M
        JZ      LABEL
/                       GOTO L
        JMP     L
/               L:      LOOP
L=.
XL3=.
/                                       A=A,RAR
        RAR
/                                       EXIT IF CARRY TRUE
        JC      XL4
/                                       A=-A
        CMA
        INR     A
/                                       EXIT IF SIGN PLUS
        JP      XL4
/                       ENDLOOP
        JMP     XL3
XL4=.
/                       WHILE B<=A DO
Xw5=.
        CMP     B
        JM      X6
/                                       REG A=A-1
        DCR     A
/                                       V1[A]=A
        PUSH    PSW
        ADI     V1
        MOV     L,A
```

```
        POP     PSW
        MVI     H,%BK1
        MOV     M,A
/                       ENDWHILE
        JMP     XW5
X6=.
/                       REPEAT
XR7=.
/                               V2[D]=V2[D]+1
        MOV     A,D
        ADI     V2
        MOV     L,A
        MVI     H,%BK2
        INR     M
/                               LOOP
XL8=.
/                                       EXIT IF V2[0]=0
        XRA     A
        LXI     H,%BK2*0400+V2+0
        CMP     M
        JZ      XL9
/                                       SET BANK ¤BK1
        MVI     H,%BK1
/                                       DEC ↑B
        MOV     L,B
        DCR     M
/                                       EXIT IF C=030
        MVI     A,030
        CMP     C
        JZ      XL9
/                               ENDLOOP
        JMP     XL8
XL9=.
/                               .INR    D
        INR     D
/                       UNTIL 6=D
        MOV     A,D
        CPI     06
        JNZ     XR7
/                       ; *** SUBROUTINE CALLS ***
/                       CALL SUB
        CALL    SUB
/                       CALL SUB IF B=INPUT(6)
        IN      06
        CMP     B
        CZ      SUB
/                       CALL SUB(A=B,B=17,DE=DV2[6],M↑X1)
        MOV     A,B
        MVI     B,021
        LHLD    %BK2*0400+DV2+06+06
        MOV     D,H
        MOV     E,L
        LXI     H,%BK1*0400+X1+0
        CALL    SUB
/                       ;*** DOUBLE PRECISION***
/                       DREG DE=BC-2
        MOV     D,B
        MOV     E,C
        DCX     D
        DCX     D
/                       DREG BC=DV2[0]-1
        LHLD    %BK2*0400+DV2+0+0
        MOV     B,H
        MOV     C,L
```

```
        DCX     B
/                       DREG BC=DV2[X1]
        LDA     %BK1*0400+X1
        ADD     A
        ADI     DV2
        MOV     L,A
        MVI     H,%BK2
        MOV     B,M
        INR     L
        MOV     C,M
/                       DREG BC=-DE
        MOV     A,D
        CMA
        MOV     B,A
        MOV     A,E
        CMA
        MOV     C,A
        INX     B
/                       DX DX2=-DY2+BC-1337+DV2[Z1]
        LHLD    %BK2*0400+DY2+0+0
        MOV     A,H
        CMA
        MOV     H,A
        MOV     A,L
        CMA
        MOV     L,A
        INX     H
        DAD     B
        LXI     D,-02471
        DAD     D
        PUSH    H
        LDA     %BK1*0400+Z1
        ADD     A
        ADI     DV2
        MOV     L,A
        MVI     H,%BK2
        MOV     D,M
        INR     L
        MOV     E,M
        POP     H
        DAD     D
        SHLD    %BK2*0400+DX2+0+0
/                       DX DV2[X1]=DE-BC-DV2[Y1]
        MOV     H,D
        MOV     L,E
        MOV     A,B
        CMA
        MOV     D,A
        MOV     A,C
        CMA
        MOV     E,A
        INX     D
        DAD     D
        PUSH    H
        LDA     %BK1*0400+Y1
        ADD     A
        ADI     DV2
        MOV     L,A
        MVI     H,%BK2
        MOV     A,M
        CMA
        MOV     D,A
        INR     L
        MOV     A,M
```