# Small Pocket Guide

## Minimum Character Set (48 required)

```
0 1 2 3 4 5 6 7 8 9
A B C D E F G H I J K L M
N O P Q R S T U V W X Y Z
+ - * / = . , ; ( ) '
space
```

**Extended Character Set** – All additional characters available on host system, e.g. 80 additional ASCII characters

**Comments** – Begin with asterisk * and extend to end-of-line, may begin anywhere a statement can begin, not allowed in expressions or lists

## Statements

```
IF condition;
  THEN statements
  ENDIF
IF condition;
  THEN statements
  ELSE statements
  ENDIF

DO WHILE condition;
  statements
  ENDDO
REPEAT statements
  UNTIL condition;

CASE expression;
  OF case-label-list;
  set-of-cases
  ENDCASE

LABEL label
GO TO label

EXIT  * exit from surrounding
      * control structure
```

## Procedures

```
PROCEDURE name[(arg-list)]
  statements
  ENDPROC
REC PROC name[(arg-list)]
  statements
  ENDPROC

RETURN       * return no value
RETURN expression;

DCL REC RETURN(stacksize);
```

(This is deprecated and only used on hosts that do not support stack frames in hardware)

## Procedure Calls

```
CALL name[(arg-list)]
```

Procedures with no return value, actual arg-list optional

## Function Calls

```
name(arg-list)
```

Call to procedures that return an integer value can appear in expressions, must be declared before calls

## Declarations

```
SET name=pexp;
```

A compile-time expression can be evaluated by Stage 2 and all terms must be defined when encountered

```
DCL name;          a scalar variable
DCL name=pexp;     initialize to value
DCL name='char';   init to char code
```

## One Dimensional Arrays

```
DCL name(pexp);    reserve pexp+1
DCL name=(datalist); initialize
```

Arrays can be 0-indexed 0 through pexp-1, or 1-indexed 1 through pexp

## Messages

```
MSG name='any string';
```

A message is a specialized array initialization to a sequence of character codes addressed by 1-indexing. Element (0) contains the number of characters in the message

## Assignment Statements

```
variable=expression;
```

A variable can be a declared scalar or array element. An expression may consist of terms, primaries, or parenthetical subexpressions preceded by unary operators or combined by multiplicative or additive operators.

## Operators

| | |
|---|---|
| Unary: | + - NOT |
| Multiplicative: | * / MOD AND SHL SHR |
| Additive: | + - OR XOR |
| Relational: | EQ NE LE LT GE GT |

## Conditional Expression

Consists of a single arithmetic expression, false if it evaluates to zero, otherwise true. Or two arithmetic expressions separated by a relational operator

## Modular Directives

```
BEGIN [modulename]
END
```

Every Small module (file) begins with BEGIN and ends with END. The optional modulename should correspond to the file name.

```
START label
STOP
```

A Small module that implements a main program should use START to skip over any data declarations, and use STOP to return to the OS

## Operating System Interface

```
ENTRY ent-list;
EXTERN ext-list;
```

Declare global entry points in this module, or global to be found in other modules at load time

## Keywords and Abbreviations

| Keyword | Abbreviation |
|---|---|
| DECLARE | DCL |
| ENTRY | ENT |
| EXTERNAL | EXT |
| MESSAGE | MSG |
| PROCEDURE | PROC |
| RECURSIVE | REC |
| IF | |
| THEN | |
| ELSE | |
| DO | |
| WHILE | |
| UNTIL | |
| EXIT | |
| LABEL | |
| GO | |
| TO | |

Keywords and their abbreviations are reserved words and may not be used for identifiers. In addition the compiler written in Small compresses all keywords and identifiers to six leading characters, so only the first six characters are significant in distinguishing them.

## Mill Instructions

| | | |
|---|---|---|
| L | opnd | load opnd into AC |
| + | opnd | add opnd to AC |
| - | opnd | subtract from AC |
| * | opnd | multiply AC by opnd |
| / | opnd | divide AC by opnd |
| MOD | opnd | remainder of divide |
| AND | opnd | bitwise AND |
| OR | opnd | bitwise OR |
| XOR | opnd | bitwise XOR |
| SHL | opnd | shift AC left |
| SHR | opnd | shift AC right |
| - | | negate AC |
| NOT | | logical invert AC |
| ST | opnd | store AC in opnd |
| L | *.AC | load indirect through AC |
| ST | *label | store indirect through label |
| J | label | jump to label |
| JEQ | label | jump if zero |
| JNE | label | jump if not zero |
| JLE | label | jump if <= zero |
| JGE | label | jump if >= zero |
| JLT | label | jump if < zero |
| JGT | label | jump if > zero |
| JX | n | indexed jump for case-label-list |
| JC | label | case label pointer |

## Operand Forms

| | |
|---|---|
| label | operand is at label |
| =n | immediate integer n |
| =label | operand is address of label |
| D name | operand is formal parameter |

## Mill Pseudo-Operations

| | | |
|---|---|---|
| BEGIN | | initialize module |
| STRT | label | start execution at label |
| LABEL | label | define label here |
| SPACE | n | reserve data words |
| CONST | n | initialize word |
| ENT | label | define global entry |
| EXT | label | define external |
| SUBR | label | define procedure entry point |
| RSUBR | label | define recursive procedure entry |
| NPARS | n | define number of formal parameters |
| PAR | label | define name of formal parameter |
| DEND | | end of formal parameter list |
| ARGT | name,n | transfer formal parameter name to corresponding actual parameter n |
| SCALL | label | procedure call |
| NARGS | n | number of actual parameters |
| ARG | label | address of actual parameter |
| CEND | | end of actual parameter list |
| RETN | label | return from procedure |
| RRETN | label | return from recursive procedure |
| END | | end of module |