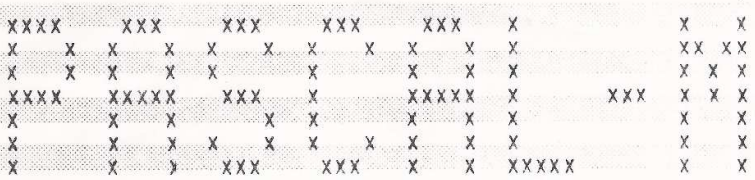

PRELIMINARY M-CODE DOCUMENTATION

BY MARK D. RUSTAD

AND

G.J.V.D.GRINTEN

<

THE OBJECT CODE PRODUCED BY THE PASCAL-M COMPILER IS COMPOSED OF LINES OF STANDARD ASCII CHARACTERS. IN MOST CASES, THE CHARACTERS SHOULD BE INTERPRETED AS HEXADECIMAL DIGITS. THOUGH THERE ARE SEVERAL TYPES OF RECORDS USED IN M-CODE, EACH TYPE OF RECORD STARTS WITH AN ASCII UPPER-CASE ≠P≠ AND ENDS WITH THE CHECKSUM OF THE RECORD CODED AS TWO HEXADECIMAL DIGITS.

THE TYPE OF P-RECORD IS DETERMINED BY THE CHARACTER IMMEDIATELY FOLLOWING THE P. THE EXACT FORMAT OF EACH OF THE DIFFERENT RECORD TYPES IS GIVEN BELOW%

THE P1 RECORD

THE P1 RECORD IS USED TO LOAD OBJECT CODE INTO THE INTERPRETER≠S PROGRAM AREA. THE TWO HEX DIGITS FOLLOWING THE P1 INDICATE HOW MANY BYTES OF OBJECT CODE ARE IN THIS RECORD (TWO HEX DIGITS PER BYTE). AFTER THE LAST OBJECT CODE BYTE IS THIS RECORD≠S CHECKSUM BYTE, AGAIN, TWO HEX DIGITS. NO LOAD ADDRESS IS SPECIFIED IN THIS RECORD SINCE THE M-CODE LOADER MAINTAINS ITS OWN CURRENT LOADING ADDRESS WHICH IS AUTOMATICALLY INCREMENTED AS P1 RECORDS ARE LOADED. A TYPICAL P1 RECORD COULD APPEAR AS%

P10B09600F9E100902BD0CA159

THE P2 RECORD

THE P2 RECORD IS USED FOR ≠SATISFYING≠ FORWARD REFERENCES BY GOING BACK TO A PREVIOUSLY-LOADED AREA IN THE PROGRAM AND INSERTING AN ADDRESS. IN THIS CASE, THE FOUR HEX DIGITS APPEARING AFTER THE P2 SPECIFY THE BYTE ADDRESS TO BE MODIFIED (RELATIVE TO THE BEGINNING OF THE CURRENT PROCEDURE) AND THE NEXT FOUR HEX DIGITS SPECIFY THE VALUE TO BE INSERTED INTO THAT LOCATION AND THE FOLLOWING ONE (SINCE THE ADDRESSES PLANTED ARE ALWAYS TWO BYTES IN LENGTH). A TYPICAL P2 RECORD COULD APPEAR AS%

P20001000AFA

THE P4 RECORD

THE P4 RECORD IS USED TO DECLARE PROCEDURE/FUNCTION ENTRY ADDRESSES AND SPECIFY THE NAME AND NUMBER OF THE PROCEDURE. THE TWO HEX DIGITS IMMEDIATELY FOLLOWING THE P4 SPECIFY THIS PROCEDURE≠S NUMBER AND THE NEXT 8 ASCII CHARACTERS ARE THE FIRST 8 CHARACTERS OF THE PROCEDURE≠S NAME (SPACE-FILLED IF NECESSARY). THE PROCEDURE NAME SPECIFICATION IN THIS CASE IS AN EXCEPTION TO THE GENERAL RULE THAT M-CODE CONSISTS OF HEX DIGITS. THE CHECKSUM OF THIS RECORD DOES INCLUDE THE ASCII CHARACTERS ADDED IN ACCORDING TO THEIR ASCII VALUES. A P4 RECORD COULD APPEAR AS%

P401OUT 66

THE P9 RECORD

THE P9 RECORD SIGNIFIES THE END OF LOADING. AT SOME

POINT, THE P9 RECORD SHOULD INCLUDE THE NUMBER OF OBJECT RECORDS LOADED (FOR ERROR DETECTION PURPOSES), BUT DOES NOT AT THIS TIME. A P9 RECORD APPEARS AS%

```
P9
```

M-CODE OPERATORS

THE M-CODE INTERPRETER ACTUALLY EXECUTES THE INSTRUCTION SET OF A HYPOTHETICAL STACK COMPUTER. ALTHOUGH THE DESIGN OF M-CODE WAS BASED ON THE P-CODE PRODUCED BY THE P2 COMPILER, NUMEROUS CHANGES HAVE BEEN MADE. FOLLOWING IS A DESCRIPTION OF EACH OF THE M-CODE OPERATORS.

0X, LDCIS - LOAD SMALL INTEGER CONSTANT.
THE LDCIS INSTRUCTION PUSHES A SMALL INTEGER CONSTANT IN THE RANGE [0..15] ONTO THE STACK IN STANDARD INTEGER FORM (16-BIT REPRESENTATION). THE CONSTANT IS IN THE LOWER 4 BITS OF THE INSTRUCTION BYTE ITSELF ALLOWING THE ENTIRE OPERATION TO ONLY REQUIRE A SINGLE BYTE OF M-CODE.

1X YY, LDAS - LOAD SHORT ADDRESS.
THE LDAS LOADS THE ABSOLUTE ADDRESS OF THE YYTH BYTE AT THE XTH LEVEL ONTO THE STACK. THIS INSTRUCTION IS ONLY GENERATED FOR OFFSETS LESS THAN 256 BYTES FROM A BASE ADDRESS.

2X YY YY, LDA - LOAD ADDRESS.
THE LDA INSTRUCTION PUSHES THE ABSOLUTE ADDRESS OF THE YYYYTH BYTE AT THE XTH LEVEL ONTO THE STACK.

3X, MSTO - MARK STACK WITHOUT RETURN BYTES.
THE MSTO INSTRUCTION MARKS THE STACK IN PREPARATION FOR A PROCEDURE CALL. THE MSTN INSTRUCTION IS USED FOR FUNCTION CALLS.

4X YY, MSTN - MARK STACK WITH RETURN BYTES.
THE MSTN INSTRUCTION MARKS THE STACK IN PREPARATION FOR A FUNCTION CALL.

5X YY, LOD1 - LOAD 1-BYTE DATA ITEM ONTO STACK.
THE LOD1 INSTRUCTION LOADS THE BYTE YYTH BYTE AT THE XTH LEVEL ONTO THE STACK. BEFORE PUSHING THE ITEM ONTO THE STACK, IT IS EXPANDED TO 16 BITS.

6X YY, LOD2 - LOAD 2-BYTE DATA ITEM ONTO STACK.
THE LOD2 INSTRUCTION LOADS THE YYTH AND YY+1TH BYTES AT THE XTH LEVEL ONTO THE STACK.

7X YY, STR1 - STORE 1-BYTE DATA ITEM INTO MEMORY.
THE STR1 INSTRUCTION PULLS ONE 16-BIT ITEM OFF THE STACK AND STORES THE LEAST SIGNIFICANT BYTE INTO THE YYTH BYTE AT THE XTH LEVEL.

8X YY, STR2 - STORE 2-BYTE DATA ITEM INTO MEMORY.
THE STR2 INSTRUCTION PULLS ONE 16-BIT ITEM OFF THE STACK AND STORES IT INTO THE YYTH AND YY+1TH BYTES AT THE XTH LEVEL.

90, LEQ2 - 2-BYTE LESS THAN OR EQUAL TEST.
THE LEQ2 INSTRUCTION PULLS TWO 2-BYTE ITEMS OFF THE STACK AND PUSHES A BOOLEAN ITEM ONTO THE STACK. THE BOOLEAN ITEM WILL BE A ONE (TRUE) WHENEVER THE ITEM NEXT TO THE TOP OF THE STACK IS LESS THAN OR EQUAL TO THE ITEM AT THE TOP OF THE STACK. OTHERWISE, THE BOOLEAN ITEM WILL BE ZERO (FALSE).

91, FOR - FOR LOOP PROCESSING INSTRUCTION.
THE FOR INSTRUCTION IS PLACED AT THE END OF EVERY FOR LOOP. THE INSTRUCTION BOTH INITIALIZES THE FOR LOOP AND DOES THE END CHECK DEPENDENT UPON DATA ON THE STACK. FOR THIS REASON, IT IS NECESSARY THAT THERE BE NO GOTO IN THE LANGUAGE AND THAT NO EXTRANEOUS DATA BE LEFT ON THE STACK AS A RESULT OF GOING THROUGH THE LOOP.

92 XX XX, LEQM - LESS THAN OR EQUAL TEST FOR ARRAYS AND RECORDS.
SEE 95 - LESM FOR DESCRIPTION.

93, LES2 - LESS THAN TEST FOR 2-BYTE DATA ITEMS.
THE LES2 INSTRUCTION IS SIMILAR TO THE LEQ2 INSTRUCTION EXCEPT IT DOES A ≡LESS THAN≡ CHECK.

94, LEQ8 - LESS OR EQUALS TEST FOR SETS.
THIS IS THE IS CONTAINED IN TEST FOR SETS. IT PULLS 2 8 BYTE SETS OFF THE STACK AND COMPARES THEM FOR AT LEAST THE COMMON BITS.

95 XX XX, LESM - LESS THAN TEST FOR ARRAYS AND RECORDS.
THE LESM INSTRUCTION PULLS TWO ADDRESSES OFF THE STACK AND COMPARES XXXX BYTES. IF THE FIRST AREA IS LESS IN VALUE THAN THE AREA POINTED BY THE SECOND ADDRESS THEN A 1 (TRUE) IS PUSHED ONTO THE STACK, OTHERWISE A 0 (FALSE) IS PUSHED.

96, EQU2 - EQUAL TEST FOR 2-BYTE DATA ITEMS.
THE EQU2 INSTRUCTION PULLS TWO 2-BYTE DATA ITEMS OFF THE STACK AND PUSHES A BOOLEAN ITEM ONTO THE STACK. THE BOOLEAN ITEM WILL BE A 1 (TRUE) WHEN THE ITEMS COMPARED ARE EQUAL AND A 0 (FALSE) OTHERWISE.

97, GEQ8 - GREATER THAN OR EQUAL TEST FOR SETS.
THIS IS THE CONTAINS TEST FOR SETS. SEE 94 INSTRUCTION. IF TRUE IT SENDS A FALSE ON THE STACK AND VISA VERSA DUE TO THE GENERATION OF A AC - NOT INSTRUCTION FOLLOWING THIS ONE BY THE PASCAL-M COMPILER.

98 XX XX, EQUM - EQUAL TEST FOR ARRAYS AND RECORDS.
THE EQUM INSTRUCTION PULLS TWO ADDRESSES OFF THE STACK AND COMPARES XXXX BYTES. IF THE TWO AREAS ARE EQUAL, A 1 (TRUE) IS PUSHED ON THE STACK, OTHERWISE A 0 (FALSE) IS PUSHED ON THE STACK.

99, EQU8 - EQUAL TEST FOR SETS.
THE EQU8 INSTRUCTION PULLS TWO 8-BYTE SETS OFF THE STACK AND COMPARES THEM. IF THEY ARE EQUAL, A 1 (TRUE) IS PUSHED ON THE STACK, OTHERWISE A 0 (FALSE) IS PUSHED ON THE STACK.

9A, IND1 - INDIRECTLY LOAD 1-BYTE DATA ITEM.
THE IND1 INSTRUCTION PULLS AN ADDRESS OFF THE STACK AND PUSHES THE BYTE AT THAT LOCATION ONTO THE STACK AFTER EXPANDING IT TO 2 BYTES (AS WITH THE LOD1 INSTRUCTION).

9B, IND2 - INDIRECTLY LOAD 2-BYTE DATA ITEM.
THE IND2 INSTRUCTION PULLS AN ADDRESS OFF THE STACK AND PUSHES THE BYTE AT THAT LOCATION AND THE FOLLOWING ONE ONTO THE STACK.

9C, IND8 - INDIRECTLY LOAD 8-BYTE DATA ITEM.
THE IND8 INSTRUCTION PULLS AN ADDRESS OFF THE STACK AND PUSHES THAT BYTE AND THE FOLLOWING 7 ONTO THE STACK.

9D, STO1 - INDIRECTLY STORE 1-BYTE DATA ITEM.
THE STO1 INSTRUCTION PULLS 2-BYTES OF DATA OFF THE STACK AND AN ADDRESS. THE LEAST SIGNIFICANT BYTE OF DATA IS STORED AT THE ADDRESS THAT WAS PULLED OFF THE STACK.

9E, STO2 - INDIRECTLY STORE 2-BYTE DATA ITEM.
THE STO2 INSTRUCTION PULLS 2-BYTES OF DATA OFF THE STACK AND AN ADDRESS. THE 2 BYTES OF DATA ARE STORED INTO MEMORY STARTING AT THE ADDRESS THAT WAS PULLED OFF THE STACK.

9F, STO8 - INDIRECTLY STORE 8-BYTE DATA ITEM.
THE STO8 INSTRUCTION PULLS 8-BYTES OF DATA OFF THE STACK AND AN ADDRESS. THE 8 BYTES OF DATA ARE STORED INTO MEMORY STARTING AT THE ADDRESS THAT WAS PULLED OFF THE STACK.

A0 XX XX, LDC - LOAD 2-BYTE CONSTANT.
THE LDC INSTRUCTION PUSHES THE 2-BYTE CONSTANT, XXXX ONTO THE STACK.

A1, RETP - RETURN FROM PROCEDURE (OR FUNCTION).
THE RETP INSTRUCTION RETURNS FROM THE CURRENT PROCEDURE, CLEANING UP THE STACK APPROPRIATELY.

A2, ADI - ADD INTEGER.
THE ADI INSTRUCTION PULLS TWO 2-BYTE INTEGERS OFF THE STACK, ADDS THEM TOGETHER AND PUSHES THE RESULT ONTO THE STACK.

A3, AND - BOOLEAN AND.
THE AND INSTRUCTION PULLS TWO BOOLEAN ITEMS OFF THE STACK, LOGICALLY AND-S THEM TOGETHER AND PUSHES THE RESULT ONTO THE STACK.

A4, DIF - SET DIFFERENCE.
THE DIF INSTRUCTION PULLS TWO 8-BYTE SETS OFF THE STACK, FINDS THE DIFFERENCE BETWEEN THE TWO BY LOGICALLY AND-ING AND EXCLUSIVE OR-ING EACH BYTE AND PUSHES THE RESULT ONTO THE STACK.

A5, DVI - DIVIDE INTEGER.
THE DVI INSTRUCTION PULLS TWO 2-BYTE INTEGERS OFF THE STACK AND DIVIDES THE TOP OF THE STACK INTO THE SECOND AND PUSHES THE RESULT ONTO THE STACK.

A6, INN - TEST IF ELEMENT IN SET.
THE INN INSTRUCTION PULLS AN 8-BYTE SET OFF THE STACK AND A 2-BYTE INTEGER. IF THE SET ELEMENT INDICATED BY THE INTEGER IS IN THE SET, A 1 (TRUE) WILL BE PUSHED ON THE STACK, OTHERWISE A 0 (FALSE) WILL BE PUSHED ON THE STACK.

A7, INT - SET INTERSECTION.
THE INT INSTRUCTION PULLS TWO 8-BYTE SETS OFF THE STACK AND FINDS THE INTERSECTION BY LOGICALLY AND-ING EACH BYTE TOGETHER. THE INTERSECTION IS THEN PUSHED ONTO THE STACK.

A8, IOR - INCLUSIVE OR.
THE IOR INSTRUCTION PULLS TWO BOOLEAN ITEMS OFF THE STACK, LOGICALLY OR-S THEM TOGETHER AND PUSHES THE RESULT ONTO THE STACK.

A9, MOD - MODULUS FUNCTION.
THE MOD INSTRUCTION PULLS TWO INTEGERS OFF THE STACK AND FINDS THE MODULUS BY TAKING THE REMAINDER OF DIVIDING THE TOP OF THE STACK INTO THE SECOND. THE RESULTING REMAINDER IS THEN PUSHED ONTO THE STACK.

AA, MPI - MULTIPLY INTEGER.
THE MPI INSTRUCTION PULLS TWO INTEGERS OFF THE STACK, MULTIPLIES THEM AND PUSHES THE RESULT ONTO THE STACK.

AB, NGI - NEGATE INTEGER.
THE NGI INSTRUCTION PULLS ONE INTEGER OFF THE STACK, CHANGES ITS SIGN AND PUSHES IT BACK ONTO THE STACK.

AC, NOT - NEGATE BOOLEAN.
THE NOT INSTRUCTION PULLS ONE BOOLEAN ITEM OFF THE STACK, LOGICALLY COMPLEMENTS IT AND PUSHES IT BACK ONTO THE STACK.

AD, SBI - SUBTRACT INTEGER.
THE SBI INSTRUCTION PULLS TWO INTEGERS OFF THE STACK, SUBTRACTS THE TOP OF STACK FROM THE NEXT AND PUSHES THE DIFFERENCE ONTO THE STACK.

AE, SGS - GENERATE SINGLETON SET.
THE SGS INSTRUCTION PULLS AN 16 bit INTEGER OFF THE STACK, GENERATES A SET WITH THAT INTEGER AS ITS ONLY ELEMENT AND PUSHES THE GENERATED SET ONTO THE STACK.

AF, UNI - SET UNION.
THE UNI INSTRUCTION PULLS TWO SETS OFF THE STACK, LOGICALLY OR-S THEM TOGETHER AND PUSHES THE RESULTING SET ONTO THE STACK.

B0 XX XX, LNC - LOAD NEGATIVE CONSTANT.
THE LNC INSTRUCTION LOADS THE NEGATIVE OF XXXX ONTO THE STACK.

B1 X X, FJP - FALSE JUMP.
THE FJP INSTRUCTION PULLS ONE BOOLEAN ITEM OFF THE STACK. IF THE ITEM IS ZERO (FALSE), CONTROL WILL TRANSFER TO THE XXXXTH BYTE OF THE CURRENT PROCEDURE, OTHERWISE CONTROL PASSES TO THE NEXT INSTRUCTION IN SEQUENCE.

B2 X X, UJP - UNCONDITIONAL JUMP.
THE UJP INSTRUCTION ALWAYS TRANSFERS CONTROL TO THE XXXXTH BYTE OF THE CURRENT PROCEDURE.

B3 X X, DEC - DECREMENT.
THE DEC INSTRUCTION PULLS ONE 2-BYTE INTEGER OFF THE STACK, DACTS XXXX FROM IT AND PUSHES THE RESULT BACK ONTO THE STACK.

B4 X X, INC - INCREMENT.
THE INC INSTRUCTION PULLS ONE 2-BYTE INTEGER OFF THE STACK, ADDS XXXX TO IT AND PUSHES THE RESULT BACK ONTO THE STACK.

B5 X X, ENT - ENTER BLOCK.
THE ENT INSTRUCTION IS ALWAYS GENERATED AS THE FIRST INSTRUCTION OF A PROCEDURE. IT RESERVES STACK SPACE FOR ALL VARIABLES LOCAL TO THE PROCEDURE.

B6 ----, CAS - CASE STATEMENT PROCESSOR.
THE CAS INSTRUCTION DOES MOST OF THE PROCESSING FOR THE CASE STATEMENT. IT OCCUPIES A VARIABLE AMOUNT OF MEMORY BECAUSE THE INSTRUCTION INCLUDES THE COMPLETE JUMP TABLE FOR THE CASE STATEMENT.

B7 XXXX, MOV - MOVE STORAGE.
THE MOV INSTRUCTION IS USED FOR MOVING ARRAYS AND RECORDS AROUND IN MEMORY. THE ADDRESS OF THE SENDING FIELD IS PULLED OFF THE TOP OF THE STACK AND THE ADDRESS OF THE RECEIVING FIELD IS PULLED OFF NEXT. XXXX BYTES ARE THEN TRANSFERED FROM THE SENDING TO THE RECEIVING FIELD.

B8, DEC1 - DECREMENT BY 1.
THE DEC1 INSTRUCTION PULLS ONE 2-BYTE INTEGER OFF THE STACK, SUBTRACTS 1 FROM IT AND PUSHES THE RESULT BACK ONTO THE STACK.

B9, INC1 - INCREMENT BY 1.
THE INC1 INSTRUCTION PULLS ONE 2-BYTE INTEGER OFF THE STACK, ADDS 1 TO IT AND PUSHES THE RESULT BACK ONTO THE STACK.

BA X X X X X X X X, LDCS - LOAD SET CONSTANT.
THE LDCS INSTRUCTION LOADS THE 8-BYTE SET, XXXXXXXXXXXXXXXX, ONTO THE STACK.

BB X X, CAP - CALL ASSEMBLY PROCEDURE.
THE CAP INSTRUCTION IS USED TO CALL AN ASSEMBLY-LANGUAGE ROUTINE PREVIOUSLY LOADED INTO A SPECIFIED ADDRESS. THE ACTUAL ACTION OF THIS INSTRUCTION MAY BE CONSIDERED TO BE SYSTEM-DEPENDENT.

BC X ----, LCA - LOAD CONSTANT ADDRESS.
THE LCA INSTRUCTION PUSHES THE ADDRESS OF THE STRING STARTING 2 BYTES AFTER THE INSTRUCTION CODE ONTO THE STACK. THE LENGTH OF THE STRING IS SPECIFIED BY XX. FOR THIS REASON, ALL ADDRESSES USED WITHIN THE PASCAL-M SYSTEM, WHETHER POINTERS OR ARRAY BASES, MUST REFER TO THE ACTUAL HARDWARE ADDRESS OF THE ITEM.

BD X, CSP - CALL STANDARD PROCEDURE.
THE CSP INSTRUCTION IS USED FOR CALLING STANDARD PROCEDUBES THAT EXIST WITHIN THE INTERPRETER ITSELF. XX IS A NUMERIC INDEX THAT DETERMINES WHICH PROCEDURE IS TO BE EXECUTED. SINCE A LARGE NUMBER OF POSSIBLE INSTRUCTION CODES HAVE NOT YET BEEN USED, IT MAY BE DESIRABLE AT SOME POINT TO ELIMINATE THIS INSTRUCTION AND INCORPORATE ALL STANDARD PROCEDURES AS INSTRUCTIONS.

BE X, CUP1 - SIMPLE CALL USER PROCEDURE.
THE CUP1 INSTRUCTION IS USED TO CALL PROCEDURES WHENEVER THE PARAMETER LIST OF THE CALL DOES NOT ITSELF INCLUDE A FUNCTION CALL. XX IS A NUMERIC INDEX THAT DETERMINES THE PROCEDURE TO BE CALLED. THE COMPILER ASSIGNS A NUMBER TO EACH PROCEDURE AT COMPILE TIME. THE NUMBER IS THEN ASSOCIATED WITH AN ADDRESS AT LOAD TIME.

BF X, CUP2 - COMPLEX CALL USER PROCEDURE.
THE CUP2 INSTRUCTION IS USED TO CALL PROCEDURES WHENEVER THE PARAMETER LIST OF THE CALL DOES INCLUDE A CALL TO A FUNCTION. ONE 2-BYTE INTEGER IS PULLED OFF THE STACK, WHICH INDICATES HOW MANY BYTES OF PARAMETERS ARE BEING PASSED. IN ALL OTHER RESPECTS, CUP2 FUNCTIONS THE SAME AS CUP1.

C0, FIX21 - CLEAN UP STACK AFTER CALL TO SINGLE-BYTE FUNCTION.
THE FIX21 INSTRUCTION PUSHES A SINGLE BYTE OF ZERO ONTO THE STACK. THIS OPERATION IS USED TO STANDARDIZE THE RETURN VALUE OF SINGLE-BYTE-VALUED FUNCTIONS (SUCH AS BOOLEAN FUNCTIONS). THIS ACTION THEN MAKES THE SINGLE BYTE VALUE OCCUPY TWO BYTES ON THE STACK AS IS USUAL.

C1, LNS - LOAD NULL SET.
THE LNS INSTRUCTION PUSHES 8 BYTES OF ZERO ONTO THE STACK FOR USE AS A NULL SET.

C2, SFA - SET FILE ADDRESS (NEW VERSION ONLY).
THE SFA INSTRUCTION PULLS ONE 2-BYTE ADDRESS OFF THE STACK AND PLACES IT IN A LOCATION INTERNAL TO THE INTERPRETER FOR USE BY ALL SUBSEQUENT I/O OPERATIONS. THE ADDRESS PULLED OFF THE STACK IS THE ADDRESS OF A POINTER TO THE FILE BUFFER.

C3, GFA - GET FILE ADDRESS.
THE GFA INSTRUCTION PUSHES THE VALUE OF THE INTERNAL FILE ADDRESS AND PUSHES IT ON THE STACK.

STANDARD PROCEDURES

00, WRI - WRITE INTEGER.
THE WRI PROCEDURE WRITES ONE INTEGER ONTO OUTPUT. THIS PROCEDURE IS ELIMINATED IN THE NEW VERSION AND REPLACED WITH A PASCAL PROCEDURE WHICH IS LOADED BY THE LOADER AS NEEDED.

01, WRC - WRITE CHARACTER TO OUTPUT.
THE WRC PROCEDURE WRITES ONE CHARACTER TO OUTPUT. THIS PROCEDURE IS ELIMINATED IN THE NEW VERSION AND REPLACED BY AN EXTERNAL PASCAL PROCEDURE WHICH IS LOADED AS NEEDED.

02, WRS - WRITE STRING TO OUTPUT.
THE WRS PROCEDURE WRITES A PACKED ARRAY OF CHARACTERS ONTO OUTPUT. THIS PROCEDURE IS ELIMINATED IN THE NEW VERSION AND REPLACED BY AN EXTERNAL PASCAL PROCEDURE WHICH IS LOADED AS NEEDED.

03, RDI - READ INTEGER.
THE RDI PROCEDURE READS AN INTEGER FROM INPUT. THIS PROCEDURE IS ELIMINATED IN THE NEW VERSION AND REPLACED BY AN EXTERNAL PASCAL PROCEDURE WHICH IS LOADED AS NEEDED.

04, RLN - READ TO END OF LINE ON INPUT.
THE RLN PROCEDURE READS SKIPS TO END OF LINE ON INPUT. THIS PROCEDURE IS ELIMINATED IN THE NEW VERSION AND REPLACED BY AN EXTERNAL PASCAL PROCEDURE WHICH IS LOADED AS NEEDED.

05, RDC - READ CHARACTER FROM INPUT.
THE RDC PROCEDURE READS A SINGLE CHARACTER FROM INPUT. THIS PROCEDURE IS ELIMINATED IN THE NEW VERSION AND REPLACED BY AN EXTERNAL PASCAL PROCEDURE WHICH IS LOADED AS NEEDED.

06, WLN - WRITE END OF LINE.
THE WLN PROCEDURE WRITES AN END OF LINE.

07, NEW - ALLOCATE SPACE ON THE HEAP.
THE NEW PROCEDURE PULLS A 2-BYTE INTEGER OFF THE STACK WHICH SPECIFIES THE SIZE OF THE AREA TO BE ALLOCATED AND A 2-BYTE ADDRESS WHICH SPECIFIES THE POINTER VARIABLE TO BE SET. SPACE IS THEN ALLOCATED ON THE HEAP AND THE POINTER VARIABLE SET TO THE ADDRESS OF THAT STORAGE.

08, EOF - END OF FILE TEST.
THE EOF PROCEDURE PUSHES A BOOLEAN ITEM ON THE STACK WHICH IS TRUE IF AN END OF FILE CONDITION EXISTS.

09, RST - RESET HEAP POINTER.
THE RST PROCEDURE SETS THE HEAP POINTER TO THE VALUE PULLED OFF THE STACK.

0A, ELN - TEST FOR END OF LINE.
THE ELN PROCEDURE PUSHES A BOOLEAN ITEM ON THE STACK WHICH IS TRUE IF AN END OF LINE CONDITION EXISTS.

0B, STP - STOP.
THE STP PROCEDURE HALTS EXECUTION.

0C, ODD - TEST INTEGER FOR ODD.
THE ODD PROCEDURE PULLS A 2-BYTE INTEGER OFF THE STACK AND PUSHES A BOOLEAN ITEM ONTO THE STACK WHICH IS TRUE IF THE INTEGER IS ODD.

0D, RSET - RESET EOF INPUT FLAG.
THE RSET PROCEDURE CLEARS THE END OF FILE FLAG FOR MULTIPLE FILES INPUT FILE. THIS IS THE ONLY WAY TO CLEAR A EOF.

ADDITIONAL PROCEDURES SHOULD BE DEFINED FOR STANDARD MATHEMATICAL FUNCTIONS, ESPECIALLY IF HARDWARE IS AVAILABLE TO PERFORM THESE FUNCTIONS.