1080,2040,2060 ENGINEERING FUNCTIONAL SPEC - CHAP 0.1

TO: KL10 LIST

TITLE: TABLE OF CONTENTS - REV 4

STATUS: DESCRIBES ALL PLANNED CHAPTERS.

FILE: CH0S01.SPC

PDM #: 200-200-008-04

DATE: 5 DEC 75

SUPERSEDED MEMOS: NONE

ENGINEER: T. HASTINGS

APPROVED: T. HASTINGS

EDITOR: T. HASTINGS

TYPIST: T. HASTINGS

REVIEWED:

ABSTRACT

THE TABLE OF CONTENTS LISTS EACH OF THE PLANNED CHAPTERS OF THE ENGINEERING FUNCTIONAL SPECIFICATION FOR THE 1080, 2040, AND 2060 SYSTEMS. EACH CHAPTER IS A SEPARATE, MACHINE-READABLE FILE AND DESCRIBES ONE FUNCTIONAL COMPONENT OF THE SYSTEM.

* THESE CHAPTERS ARE NOT AVAILABLE

REVISION HISTORY

REV	DESCRIPTION	CHG NO	ORIG	DATE	APPD BY	DATE
0	ORIGINAL			18 FEB	TNH	
1	INPUT FROM ENGINEERS			11 MAR		
3	ADDED 1.6, 2.15, 2.16			1 APR	TNH	
4	CLEANUP			5 DEC		

TABLE OF CONTENTS FOR 1080, 2040, AND 2060:

[END OF CH0S01,SPC]

1080,2040,2060 ENGINEERING FUNCIONAL SPEC - CHAP 0.2

TO: KL10 LIST, J. PARSLOW

TITLE: PREFACE - REV 1

STATUS: CURRENT PLANS FOR ENGINEERING FUNCTIONAL SPECIFICATION. TO BE REVIEWED WED, 13 MAR 74, 1:00 P.M., KL10 STEERING COMMITTEE.

FILE: [EFS]CH0S02.SPC

PDM #: 200-200-010-01

DATE: 11 MAR 74

SUPERSEDED MEMOS: NONE

ENGINEER: T. HASTINGS

APPROVED: T. HASTINGS

EDITOR: T. HASTINGS

TYPIST: M. PROUTY

REVIEWED:

ABSTRACT

THE PREFACE CONTAINS ALL OF THE INFORMATION ABOUT HOW THE ENGINEERING FUNCTIONAL SPECIFICATION FOR THE 1080, 2040, AND 2060 SYSTEMS WILL BE TYPED, FORMATTED, DISTRIBUTED, AND REVIEWED. AN ALGOL-LIKE ENGLISH FLOW LANGUAGE IS DESCRIBED. THIS LANGUAGE IS USED THROUGHOUT THE FUNCTIONAL SPECIFICATION. THE PREFACE ALSO DESCRIBES THE GOALS AND INTENDED AUDIENCES OF THE SPECIFICATIONS THEMSELVES.

REVISION HISTORY

REV	DESCRIPTION	CHG NO	ORIG	DATE	APPD BY	DATE
0	ORIGINAL			13 FEB	TNH	
1	DEVELOPED			11 MAR		

CONTENTS OF PREFACE:

1. GOALS OF THE ENGINEERING FUNCTIONAL SPECIFICATIONS

THE MAJOR GOAL OF THE ENGINEERING FUNCTIONAL SPECIFICATIONS IS TO PROVIDE A CURRENT, ACCURATE , AND EASILY UPDATABLE DESCRIPTION OF THE FUNCTIONAL CHARACTERISTICS OF THE 1080, 2040, 2060 SYSTEMS. THE SPECIFICATIONS WILL DESCRIBE THE GOALS AND FUNCTIONS OF THE SYSTEMS AS A WHOLE AS WELL AS THE FUNCTIONS OF THE INDIVIDUAL FUNCTIONAL COMPONENTS. THE SPECIFICATIONS WILL ALSO DESCRIBE THE MOTIVATIONS AND THE "WHYS" BEHIND THE FUNCTIONAL DESIGN. A FUNCTIONAL COMPONENT IS A LOGICALLY DISTINCT PART OF THE SYSTEM, SUCH AS THE EBOX OR THE CACHE, RATHER THAN A PHYSICAL PART OF THE SYSTEM SUCH AS A PARTICULAR BOARD. EACH CHAPTER WILL DESCRIBE A DIFFERENT FUNCTIONAL COMPONENT. NO ATTEMPT WILL BE MADE TO DESCRIBE THE INTERNALS OF A COMPONENT, EXCEPT WHERE THIS IS NECESSARY IN ORDER TO UNDERSTAND THE FUNCTIONS. IN OTHER WORDS THESE FUNCTIONAL SPECIFICATIONS ARE NOT INTENDED TO SERVE AS DESIGN SPECIFICATIONS AS WELL. HOWEVER SOME OF THE INTERFACES BETWEEN FUNCTIONAL COMPONENTS WILL BE DESCRIBED EVEN WHEN THEY ARE NOT VISIBLE TO THE PROGRAMMER. THEREFORE ANOTHER GOAL OF THE FUNCTIONAL SPECIFICATIONS IS TO ENSURE THAT THE FUNCTIONAL COMPONENTS WORK TOGETHER PROPERLY AS A SYSTEM.

FINALLY EACH CHAPTER WILL CONTAIN AN ERROR HANDLING AND MAINTAINABILITY SECTION. THIS SECTION WILL DESCRIBE ALL OF THE ERRORS WHICH ARE AND ARE NOT DETECTED BY THE HARDWARE AND SOFTWARE. IT WILL DESCRIBE THE RECOMMENDED SOFTWARE RECOVERY ACTIONS. IT WILL DESCRIBE THE ERROR INFORMATION AVAILABLE FOR ERROR REPORTING. IT WILL DESCRIBE THE STRATEGY FOR PERFORMING CORRECTIVE AND PREVENTATIVE MAINTENANCE FOR THE FUNCTIONAL COMPONENT. SEE CHAPTER 1,5, SYSTEM ERROR HANDLING AND MAINTAINABILITY FOR DETAILS ABOUT WHAT WILL BE IN EACH CHAPTER.

IT IS RECOGNIZED THAT THESE SPECIFICATIONS SHOULD HAVE BEEN PRODUCED EARLIER THAN THIS. IN FACT THEY SHOULD HAVE BEEN DONE BEFORE THE DESIGN SPECIFICATIONS WHICH SHOULD HAVE BEEN DONE BEFORE THE FIRST BOARD WAS SUBMITTED TO LAYOUT. HOWEVER, MOST OF THE COMPONENTS HAVE HAD ONE OR MORE MEMOS PUBLISHED TO THE KL10 LIST AND HAVE BEEN REVIEWED. THESE MEMOS WILL SERVE AS THE BASIS FOR THE ENGINEERING FUNCTIONAL SPECIFICATIONS WHICH WILL THEN SUPERSEDE THESE MEMOS. FROM NOW ON THE ENGINEERS HAVE A CHOICE OF (1) SENDING FUNCTIONAL SPECIFICATIONS MEMOS AND CHANGES TO THE KL10 LIST OR (2) CONTACTING TOM HASTINGS TO HAVE THEM INCORPORATED INTO A NEW OR EXISTING CHAPTER OF THE ENGINEERING FUNCTIONAL SPECIFICATIONS.

EMPHASIS WILL BE PLACED ON CLARITY OF DESCRIPTION AND SPEED OF DISTRIBUTION. ENGLISH, STYLE, AND APPEARANCE WILL BE CONSIDERED ONLY OF SECONDARY IMPORTANCE. EACH CHAPTER WILL BE A SEPARATE MACHINE READABLE FILE USING VTED, RTRANS, AND RUNOFF PROGRAMS. UPDATE PAGES WILL BE ISSUED FREQUENTLY WITH COMPUTER GENERATED CHANGE BARS USING FILCOM. UPPER-LOWER CASE INPUT VT05 TERMINALS AND UPPER-LOWER CASE LINE PRINTERS WILL BE USED

TO PRINT EACH CHAPTER ON 8-1/2 BY 11 INCH TELETYPE PAPER,

THE FUNCTIONAL SPECIFICATIONS DESCRIBE THE HARDWARE AND FIRMWARE, THEY WILL BE THE DOCUMENTS FROM WHICH THE SOFTWARE ENGINEERS AND DIAGNOSTIC ENGINEERS WRITE THEIR PROGRAMS, THE FUNCTIONAL SPECIFICATIONS WILL SERVE AS INPUT TO THE TECHNICAL WRITERS WHO ARE WRITING HARDWARE, PROGRAMMING AND MAINTENANCE MANUALS, THESE MANUALS WILL CONTAIN ALL OF THE INFORMATION IN THE FUNCTIONAL SPECIFICATIONS AND SO WILL SUPERSEDE THEM, HOWEVER, THE FUNCTIONAL SPECIFICATIONS WILL CONTINUE AS THE SOURCE DOCUMENT FOR INTERNAL USE ONLY,

2. MACHINE VERSIONS

FOUR VERSIONS OF THE MACHINE ARE PLANNED:

1. 3 BREADBOARD MACHINES
2. 5 PROTOTYPE MACHINES
3. 7 PRE-PRODUCTION MACHINES
4. N PRODUCTION MACHINES

THE 1080, 2040, AND 2060 WILL USE THE SAME KL10 CPU WITH DIFFERENT ARRANGEMENTS OF OTHER FUNCTIONAL COMPONENTS. THE FUNCTIONAL SPECIFICATIONS WILL DESCRIBE THE PRODUCTION MACHINE AND FIRST RELEASE MICRO-CODE. EDITORIAL NOTES IN SQUARE BRACKETS WILL BE USED TO FLAG OMISSIONS, CHANGES, OR DIFFERENCES BETWEEN THE PRODUCTION MACHINE AND ITS PREDECESSORS: THE BREADBOARD, PROTOTYPE, AND PILOT VERSIONS AND THE KI10. THESE COMMENTS WILL ALSO INDICATE ANY FUNCTIONAL ECOS DONE TO THE PREDECESSORS DURING CHECKOUT OR FUNCTIONAL CHANGES IN THE MICRO-CODE. THUS THE FUNCTIONAL SPECIFICATIONS WILL SERVE AS A LIVING DOCUMENT DURING CHECKOUT. AN APPENDIX WILL ALSO SUMMARIZE THESE DIFFERENCES.

3. REVIEWS OF FUNCTIONAL SPECIFICATIONS

EACH CHAPTER WILL BE ISSUED SEPARATELY AS SOON AS IT IS READY. MOST CHAPTERS WILL THEN HAVE A REVIEW. SINCE MOST OF THE COMPONENTS HAVE HAD REVIEWS BASED ON EARLIER MEMOS, THESE REVIEWS SHOULD BE VERY BRIEF AND SHOULD COVER WHAT HAS CHANGED SINCE THE MEMOS WERE WRITTEN. FUNCTIONAL SPEC REVIEWS WILL BE HELD BEFORE DESIGN REVIEWS OF BOARD LAYOUT AND RELAYOUTS. TOM HASTINGS (X6512) WILL KEEP EXTRA COPIES OF THE FUNCTIONAL SPECIFICATIONS. IF YOU HAVE QUESTIONS, COMMENTS, CHANGES, YOU CAN SIMPLY MARK UP YOUR COPY AND SEND IT TO HIM. HE WILL RETURN IT IMMEDIATELY OR SEND YOU A FRESH COPY. SUCH QUESTIONS AND COMMENTS WILL BE CONSIDERED AT THE DESIGN REVIEW OR WILL BE TAKEN UP WITH THE ENGINEER INDIVIDUALLY. THEREFORE YOU DO NOT NEED TO ATTEND THE REVIEW UNLESS YOU WOULD LIKE TO.

4. ORGANIZATION OF CHAPTERS

THE CHAPTERS OF THE ENGINEERING FUNCTIONAL SPECIFICTION HAVE BEEN DEVIDED INTO 7 GROUPS:

0. INTRODUCTION

1. SYSTEM

2. CPU

3. MEMORY

4. CONTROLLERS AND PERIPHERALS

5. INTERFACES BETWEEN FUNCTIONAL COMPONENTS

A. APPENDICES

CHAPTER NUMBERS ARE TWO NUMBERS, M.N. THE FIRST NUMBER SPECIFIES ONE OF THE 7 GROUPS. THE SECOND NUMBER SPECIFIES A CHAPTER WITHIN THE GROUP. IN THIS WAY NEW CHAPTERS CAN BE ADDED AND PUT AT THE END OF THE APPROPRIATE GROUP. SEE CHAPTER 0,1, TABLE OF CONTENTS FOR A LISTING OF ALL OF THE CHAPTERS.

5. CHAPTER FORMAT

EACH CHAPTER WILL BEGIN WITH A TITLE PAGE OF A STANDARD FORMAT. THE TITLE PAGE ON THE FRONT OF THIS CHAPTER SERVES AS AN EXAMPLE. EACH LINE IS EXPLAINED AS FOLLOWS:

TITLE OF SPECS AND CHAPTER NUMBER AS CHAP M.N.

TO: LISTS THE DISTRIBUTION FOR THE CHAPTER AND ALL UPDATES. THIS WILL BE THE KL10 LIST INITIALLY. CHAPTERS COVERING USER-MODE INSTRUCTIONS WILL HAVE A WIDER CIRCULATION, INCLUDING THE DEC SYSTEM 10 SOFTWARE ENGINEERING GROUP.

SINCE TITLE: IS THE TITLE OF THE CHAPTER. IT IS USUALLY THE NAME OF THE FUNCTIONAL COMPONENT. FOLLOWING THE TITLE IS THE REVISION NUMBER. THE FIRST DISTRIBUTION IS CONSIDERED REVISION 0. A DEC OPTION DESIGNATION IS INCLUDED IN PARENTHESES IF THERE IS ONE.

STATUS: IS A BRIEF DESCRIPTION OF THE STATUS OF THE CHAPTER, INCLUDING A REVIEW ANNOUNCEMENT. THE REVIEW STATUS OF THE SUPERSEDED MEMOS WILL ALSO BE MENTIONED.

FILE: IS THE FILE NAME, EXTENSION, AND GENERATION NUMBER OF THE SOURCE FILE. USUALLY IT WILL BE MPN.SPC WHERE M AND N ARE THE CHAPTER NUMBERS.

PDM #: IS THE PROGRAMMING DEPT. MEMO # IN THE 200 SERIES.

DATE: IS THE DATE OF THE LATEST REVISION AND HENCE DISTRIBUTION.

SUPERSEDED MEMO: LISTS THE TITLES, AUTHORS, AND DATES OF THE MEMOS TO THE KL10 LIST WHICH THIS CHAPTER SUPERSEDES. ALL OF THE INFORMATION IN SUCH MEMOS WILL BE INCORPORATED INTO THE CHAPTERS AS IS OR WITH SOME EDITING AND ADDED EXPLANATION, CODING EXAMPLES, ETC. NO INFORMATION WILL BE LEFT OUT.

ENGINEER: IS THE NAME OF THE ENGINEER RESPONSIBLE FOR THE FUNCTIONAL COMPONENT.

APPROVED: IS THE INDICATION THAT THE RESPONSIBLE ENGINEER OR A SUPERIOR HAS APPROVED THE SPECIFICATION. IT IS EQUIVALENT TO A SIGNATURE AND WILL BE ADDED AT THE LAST MOMENT BEFORE DISTRIBUTION.

EDITOR: IS THE TECHNICAL PERSON PREPARING THE CHAPTER.

TYPIST: IS THE PERSON WHO TYPED REV 0.

REVIEWED: IS THE DATE THAT THE CHAPTER WAS REVIEWED AT A DESIGN REVIEW CALLED BY THE ENGINEER OR EDITOR. A BLANK INDICATES THAT THE CHAPTER HAS NOT BEEN REVIEWED SINCE BEING PUT IN MACHINE READABLE FORM.

ABSTRACT - EACH CHAPTER WILL HAVE A SHORT ABSTRACT DESCRIBING THE CONTENTS OF THE CHAPTER.

REVISION HISTORY - IS A LIST OF REVISIONS TO THE FUNCTIONAL SPECIFICATIONS. IT IS IDENTICAL TO THE COMPANY STANDARD FOR ENGINEERING SPECIFICATIONS.

6. PRODUCTION OF THE SPECIFICATION

ONE OF THE GOALS IS TO MAKE IT AS EASY AND FAST AS POSSIBLE TO INPUT AND EDIT THE SPECIFICATIONS, BY SECRETARIES , SKILLED COMPUTER TYPISTS OR ENGINEERS. VTED, A SIMPLE SCOPE EDITOR, WILL BE USED WITH AN UPPER-LOWER CASE INPUT VT05 TO PRODUCE THE SOURCE FILE WITH EXTENSION .SPC. THE TYPIST TYPES AND FORMATS TEXT AS IF USING AN ORDINARY TYPEWRITER FOLLOWING A FEW SIMPLE RULES SPECIFIED BELOW. THEN THE FILE IS PASSED THROUGH RTRANS (RUNOFF TRANSLATOR) WHICH INSERTS APPROPRIATE RUNOFF COMMANDS INTO THE OUTPUT FILE GIVING IT EXTENSION .RNO. THE RNO FILE IS IN TURN PASSED THROUGH RUNOFF WHICH FILLS AND JUSTIFIES THE TEXT GIVING IT EXTENSION .LST. THE .LST FILE IS PRINTED ON AN UPPER-LOWER CASE LINE PRINTER USING 8-1/2 BY 11 TELETYPE FOLDING PAPER. THE RUNOFF DEFAULT PAGE SIZE OF 60 LINES WILL BE USED SINCE NO PHOTO REDUCTION WILL BE DONE. HOWEVER, THE RIGHT MARGIN WILL BE SET AT 63 INSTEAD OF 60. THIS WILL GIVE MORE ROOM FOR COMMENTS IN PROGRAM EXAMPLES. A LINE WIDTH OF 63 CHARACTERS WAS CHOSEN BECAUSE THE VT05 BLEEPS WHEN THE 64TH CHARACTER IS TYPED. FOR PROGRAM EXAMPLES (WHERE RUNOFF HAS BEEN TOLD NOT TO FILL AND JUSTIFY), THE TYPIST WILL KNOW A COMMENT HAS NOT OVERFLOWED IF THERE IS NO BLEEP.

SUBSEQUENT EDITS ARE MADE ON THE ORIGINAL SOURCE FILE (.SRC). EDITING IS VERY EASY SINCE THE SOURCE FILE RESEMBLES THE FORMATTED LISTING AND HAS ALMOST NO RUNOFF COMMANDS IN IT.

UPDATES WILL BE ISSUED BY PAGE WITH CHANGE BARS USING FILCOM GIVING AN EXTENSION OF .SCM. RECIPIENTS ARE URGED TO BURST THE PAPER AND PUNCH HOLES FOR INSERTION INTO 3 RING NOTEBOOKS SO THAT THEY CAN INSERT CHANGE PAGES.

7. PREDISTRIBUTION APPROVAL

BEFORE DISTRIBUTION OF THE FUNCTIONAL SPECIFICATIONS OR ANY OF THEIR REVISIONS, THE RESPONSIBLE ENGINEER MUST APPROVE A LISTING. AFTER SUCH APPROVAL EXACTLY THE FOLLOWING EDITS WILL BE MADE TO THE FILE AND NO MORE:

1. ".SUBTITLE PREFACE - REV 0-" WILL BE CHANGED TO:

   ".SUBTITLE PREFACE - REV 0"

   THE MINUS SIGN AFTER THE REVISION NUMBER IS MEANT TO INDICATE THAT THE REVISION IS NOT YET COMPLETE.

2. DATE: WILL HAVE ALREADY BEEN CHANGED
   BEFORE APPROVAL

3. IF THIS IS THE FIRST DISTRIBUTION (REV 0),

   APPROVED: WILL BE CHANGED TO:

   APPROVED: JOHN DOE

4. IF THIS IS A REVISION, THE APPROVAL ENGINEER'S INITIALS WILL BE ADDED TO THE NEWEST LINE ON THE REVISION HISTORY. FOR REV 1, BOTH REV 1 AND REV 0 WILL BE ADDED TO THE REVISION HISTORY SECTION.

8. RUNOFF COMMANDS

THE TYPIST WILL USE AS FEW RUNOFF COMMANDS IN THE SOURCE FILE (.SRC) AS POSSIBLE USING VTED. INSTEAD THE TYPIST WILL USE THE NATURAL FORMATTING CHARACTERS SPACE, TAB, RETURN, BLANK LINE, AND FORM FEED AS DESIRED. IN THIS WAY THE JUSTIFIED OUTPUT LISTING (.LST) WILL RESEMBLE THE INPUT (.SRC) FILE SO CLOSELY THAT THE TYPIST CAN EDIT THE SOURCE DIRECTLY FROM A MARKED UP JUSTIFIED LISTING (.LST). FOR EASE OF TYPING ALL RUNOFF COMMANDS WILL BE IN LOWER CASE.

".TITLE" WILL BE USED ON ALL CHAPTERS. IT WILL FOLLOW THE FIRST LINE OF THE SOURCE FILE, TWO TRAILING SPACES ENSURE THAT RUNOFF WILL LEAVE 2 SPACES BEFORE IT PUTS IN PAGE ON THE TOP OF EVERY PAGE. EXAMPLE:

".TITLE 1080,2040,2060 ENGINEERING FUNCTIONAL SPEC - CHAP M,N "

".SUBTITLE" WILL INDICATE THE TITLE OF THE CHAPTER AND THE REVISION NUMBER. IT WILL BE THE LINE FOLLOWING THE TITLE: LINE. EXAMPLE:

".SUBTITLE EXEC AND USER PAGE TABLES (EPT, UPT) - REV 0"

".NOFILL" WILL BE USED AS LITTLE AS POSSIBLE. IT WILL ONLY BE NEEDED FOR THE FLOW ALGORITHMS. IT WILL NOT BE USED FOR LISTS, TABLES, OR ASSEMBLY CODE EXAMPLES. LISTS AND TABLES WILL ALWAYS HAVE A BLANK LINE BETWEEN EACH ITEM SO RTRANS WILL NOT JUSTIFY. ASSEMBLY CODE HAS IMBEDDED TABS, SO RTRANS WILL NOT FILL AND JUSTIFY EVEN THOUGH IT DOESN'T HAVE BLANK LINES BETWEEN EACH LINE.

".FILL" WILL BE USED TO RESTORE FILLING AND JUSTIFYING.

NO OTHER RUNOFF COMMANDS WILL BE USED. INDEXING WILL BE GENERATED AUTOMATICALLY BY SOME SORTING SOFTWARE. THEN THE TYPIST WILL NOT HAVE TO USE ".INDEX" COMMANDS AND THE INDEX WILL BE MORE COMPLETE.

9. TYPING CONVENTIONS FOR RTRANS

THE FORMAT FOR EACH CHAPTER FOLLOWS THE "STANDARD FOR RUNOFF MANUALS, PDM# 005-003-026-00 WITH A FEW MINOR CHANGES. THE CHANGES ARE: (1) RIGHT MARGIN 63 INSTEAD OF 70 BECAUSE THE VT05 BLEEPS ON THE 64TH CHARACTER, (2) TEXT PART OF A LIST STARTS 8 SPACES IN RATHER THAN 9 SO THAT TAB CAN BE USED TO LINE UP TEXT, AND (3) LEVEL A PARAGRAPH NUMBER DOES NOT HAVE A 0 FRACTION. ALL INFORMATION TYPED FALLS INTO ONE OF THE FOLLOWING THREE CATEGORIES:

1. PARAGRAPHS

2. LISTS

3. TABLES

9.1 PARAGRAPHS

THERE ARE 4 LEVELS OF PARAGRAPHS: LEVEL A, B, C, D, DEPENDING ON THE NUMBER OF FRACTIONS FOLLOWING THE PARAGRAPH NUMBER. LEVEL A ALWAYS HAS NO FRACTION, LEVEL B HAS FRACTION .1 TO .NN, LEVEL C HAS FRACTION .1.1 TO .NN.NN, AND LEVEL D HAS FRACTION .1.1.1 TO .NN.NN.NN. LEVEL A ALWAYS HAS A HEADER LINE IN ALL CAPITAL LETTERS, PRECEDED BY 3 BLANK LINES AND FOLLOWED BY 1 BLANK LINE. LEVEL B MAY OR MAY NOT HAVE A HEADER LINE WITH THE FIRST LETTER OF EACH WORD CAPITALIZED. IF IT DOES HAVE A HEADER, THE HEADER IS PRECEDED AND FOLLOWED BY 1 BLANK LINE. LEVEL C MAY OR MAY NOT HAVE A TITLE. IF IT DOES HAVE A TITLE, THE TITLE APPEARS ON THE SAME LINE AS THE TEXT, SEPARATED BY A HYPHEN. LEVEL D IS SUGGESTED TO BE AVOIDED. ALL PARAGRAPHS AT ALL LEVELS ARE BLOCK STYLE AND FLUSH LEFT.

EXAMPLE SHOWING ALL 4 LEVELS:

1. THIS IS A LEVEL A HEADER

THIS IS THE TEXT IN A LEVEL A PARAGRAPH OR PARAGRAPHS. NOTE THAT IT IS FLUSH LEFT.

1.1 THIS IS A LEVEL B HEADER

THIS IS THE TEXT IN A LEVEL B PARAGRAPH OR PARAGRAPHS. NOTE THAT IT IS FLUSH LEFT ALSO.

1.1.1 THIS IS A LEVEL C HEADER - THIS IS THE TEXT IN A LEVEL C PARAGRAPH OR PARAGRAPHS. NOTE THAT IT IS FLUSH LEFT.

1.1.1.1 THIS IS THE TEXT IN A LEVEL D PARAGRAPH. NOTE THAT LEVEL D NEVER HAS A TITLE AND IS USUALLY TO BE AVOIDED.

9.2 LISTS

LISTS WILL ALWAYS HAVE EACH ITEM NUMBERED. A NUMBERED LIST IS DISTINGUISHED FROM PARAGRAPHS BY BEING INDENTED. USUALLY THE TEXT PRECEDING A LIST WILL END IN A COLON TO INDICATE THAT A LIST FOLLOWS. LIST ITEMS ARE NUMBERED WITH INTEGERS FOLLOWED BY A PERIOD. SINGLE DIGITS ARE PRECEDED BY 4 SPACES AND FOLLOWED BY A TAB SO THAT THE TEXT IS SEPARATED FROM THE NUMBER OF 2 SPACES. TWO DIGIT NUMBERS MUST BE PRECEDED BY 3 SPACES SO THAT THE TAB WILL STIL MOVE TWO CHARACTER POSITIONS. THE FOLLOWING IN AN EXAMPLE OF A LIST:

1. FIRST ITEM IN LIST

2. SECOND ITEM IN LIST

3. THIRD ITEM IN LIST

IF THE TEXT IN AN ITEM OF A LIST EXCEEDS ONE LINE, THEN THE TYPIST MUST TYPE A TAB ON SUCCEEDING LINES. THE TYPIST WILL PUT 1 BLANK LINE BETWEEN EACH ITEM IN A LIST. EXMPLE OF A LIST WITH ITEMS WHICH OVERFLOW A LINE:

1. THIS IS AN EXAMPLE OF A LONG ITEM WHICH OVERFLOWS A SINGLE LINE.

2. THIS IS A SECOND EXAMPLE.

3. THE TYPIST MUST TYPE A TAB AFTER TYPING A RETURN.

9.3 TABLES

TABLES ARE JUST LIKE LISTS EXCEPT THAT THE ITEMS ARE NOT USUALLY NUMBERED. TABLES USUALLY HAVE MORE THAN ONE COLUMN SEPARATED BY TABS. TABLES USUALLY HAVE COLUMN HEADINGS. EACH ENTRY WILL BE SEPARATED BY A BLANK LINE. TABLES WILL USUALLY BE INDENTED ONE FULL TAB IF THERE IS ROOM.

EXAMPLE OF A TABLE:

NO.	ITEM
3	WAMBATS
5	ZAMBONIS
2	KL10S

9.4 FORMAT

GENEROUS USE OF FORM FEEDS (CONTROL L) WILL BE USED TO CREATE PAGE BREAKS SO THAT INSERTS CAN BE MADE WITHOUT CHANGING THE PAGE NUMBERING. IN THIS WAY UPDATE PAGES CAN BE DISTRIBUTED

RATHER THAN THE ENTIRE CHAPTER, AN ATTEMPT WILL BE MADE TO PUT PAGE BREAKS IMMEDIATELY BEFORE LEVEL A PARAGRAPHS.

VTED (ACTUALLY RTRANS) DOES NOT CAUSE JUSTIFICATION OF LINES WHICH HAVE TABS EMBEDDED IN THEM. THUS ASSEMBLY CODE EXAMPLES CAN BE PREPARED WITHOUT BLANK LINES. LEADING TABS DO NOT PREVENT JUSTIFICATION BECAUSE THEY ARE NOT EMBEDDED. LEADING TABS INDICATE AN INDENTED LEFT MARGIN.

10. STORAGE OF FILES

ALL .SRC, .RNO, .LST, AND .SCM FILES WILL BE KEPT IN AN UNPUBLISHED DIRECTORY ON VIROS. STRICT SECURITY MEASURES ARE BEING TAKEN. A ONE DOLLAR REWARD IS OFFERED TO EACH PERSON OUTSIDE OF THE VIROS GROUP WHO CAN OBTAIN ANY OF THE FUNCTIONAL SPECIFICATIONS FROM THE MACHINE. THE PERSON MUST FIND A NEW WAY AND CANNOT GO INTO THE COMPUTER ROOM. DECISIONS OF THE VIROS GROUP JUDGES ARE FINAL. CONTEST IS VOID WHERE PROHIBITED. THE FILE NAME WILL BE THE SAME AS THE CHAPTER NUMBER, I.E. CHAPTER 2,3 WILL HAVE FILE NAME 2P3RN.SRC, WHERE N IS THE REVISION NUMBER.

11. FLOW LANGUAGE

ALGORITHMS AND PROCEDURES WILL BE DONE USING A SIMPLE ENGLISH-LIKE PROGRAMMING LANGUAGE WITH A SYNTAX LIKE ALGOL. THE LANGUAGE IS AN EXPERIMENT IN STRUCTURED PROGRAMMING IN THAT THE LANGUAGE IS BLOCK STRUCTURED WITH NO GOTOS.

11.1 NOTATION

ALL REGISTER NAMES AND BITS WILL BE CAPITALIZED. ALL KEYWORDS WILL BE CAPITALIZED. COMMENTS WILL BE INCLUDED IN PARENTHESESIS. ALL STATEMENTS WILL INCLUDE THE NUMBER OF BITS INVOLVED IN EACH MENTIONED. THE BEGINNING OF A PROCEDURE WILL DECLARE ALL REGISTERS USED AND THEIR BIT WIDTHS. THE BITS ARE SPECIFIED IN DECIMAL INSIDE SQUARE BRACKETS WITH NO SPACES SURROUNDING THE HYPHEN.

EXAMPLE:

```
COPY AR [18-35] TO - VMA [18-35]
```

ANY BITS OF A SOURCE OR DESTINATION REGISTER WHICH EXIST BUT ARE NOT MENTIONED ARE ASSUMED TO REMAIN UNCHANGED.

11.2 STATEMENTS

STATEMENTS ARE TERMINATED BY A SEMI-COLON. STATEMENTS MAY OCCUPY MULTIPLE LINES. THE TYPIST MUST INDENT CONTINUATION LINES 3 SPACES BEYOND THE INDENTATION OF THE FIRST LINE OF THE STATEMENT.

EXAMPLES:

```
THIS IS A STATEMENT;
THIS IS A MULTI-LINE STATEMENT WHICH HAS BEEN
   CONTINUED ON THE NEXT LINE;
```

11.3 COMPOUND STATEMENTS

A COMPOUND STATEMENT IS A SEQUENCE OF STATEMENTS BEGINNING WITH BEGIN AND ENDING WITH END. A COMPOUND STATEMENT MAY BE USED ANYWHERE A STATEMENT MAY BE USED. FOR EASE OF READING THE BEGIN AND END ARE PUT ON SEPARATE LINES. THEY ARE INDENTED THE SAME NUMBER OF TAB STOPS AS THE STATEMENTS IN THE BODY OF THE COMPOUND STATEMENT.

EXAMPLE:

```
BEGIN
CLEAR FIRST-PART DONE FLOP;
COPY PREVIOUS AC P [10-17] TO HR [18-35];
END
```

11.4 CONDITIONAL STATEMENT

CONDITIONAL STATEMENTS ARE USED TO MAKE DECISIONS AND ACT ACCORDINGLY. A CONDITIONAL STATEMENT MAY OCCUR ANYWHERE THAT A STATEMENT CAN OCCUR. THE SIMPLEST FORM OF CONDITIONAL STATEMENT EITHER PERFORMS A SPECIFIED STATEMENT OR IT DOESN'T DEPENDING ON THE CONDITION. FORM:

```
IF CONDITIONAL EXPRESSION, THEN STATEMENT;
```

EXAMPLE:

```
IF AC BIT 9 IS 1, THEN COPY PCS TO VMA [13 - 17];
```

THE FOLLOWING EXAMPLE SHOWS A COMPOUND STATEMENT:

```
IF AC BIT 9 IS 1, THEN

        BEGIN
        COPY PCS TO VMA [13-17];
        ADD XR [0-35] TO AC [0-35];
        END
```

NOTE THAT THE COMPOUND STATEMENT IS INDENTED ONE ADDITIONAL TAB FROM THE PREVIOUS LINE AND IS SET OFF BY BLANK LINES. A CHOICE BETWEEN TWO STATEMENTS CAN BE MADE BY USING THE ELSE CONSTRUCTION. FORM:

```
IF CONDITIONAL EXPRESSION, THEN STATEMENT 1

ELSE STATEMENT 2;
```

STATEMENT 1 IS EXECUTED IF THE CONDITIONAL EXPRESSION IS TRUE, OTHERWISE STATEMENT 2 IS EXECUTED. A CHOICE BETWEEN MORE THAN TWO STATEMENTS CAN BE MADE TO GIVE A GENERAL CASE ANALYSIS CAPABILITY. FORM:

```
IF CONDITIONAL EXPRESSION 1, THEN STATEMENT 1

ELSE

IF CONDITIONAL EXPRESSION 2, THEN STATEMENT 2

ELSE

IF CONDITIONAL EXPRESSION 3, THEN STATEMENT 3

.   .

.   .

.   .

ELSE

        STATEMENT N + 1;
```

EXAMPLE: SHOWING TYPING FORMAT AS WELL AS SYNTAX:

```
IF TEMPERATURE IS GREATER THEN 60, THEN

        DO NOT WEAR OVERCOAT

ELSE

IF TEMPERATURE IS IN RANGE 30 TO 60, THEN

        WEAR OVERCOAT

ELSE

        WEAR OVERCOAT AND SCARF;
```

11.5 LOOP STATEMENTS

REPETITION OF STATEMENTS IS DONE BY PRECEDING A STATEMENT WITH THE WORD DO. FORM:

```
DO STATEMENT;
```

STATEMENT IS REPEATED FOREVER. SEE BELOW HOW TO TERMINATE LOOP.

11.6 STATEMENT LABELS

ANY SIMPLE OR COMPOUND STATEMENT MAY BE LABELED BY PRECEDING IT BY AN IDENTIFIER TERMINATED BY A COLON. FORM:

```
LABEL: STATEMENT;
```

EXAMPLE:

```
SUM:    BEGIN
        ADD A(I) + TOTAL TO TOTAL;
        ADD I + 1 TO I;
        END
```

11.7 EXIT LOOP STATEMENT

THE EXIT LOOP STATEMENT IS USED TO TERMINATE LOOPS. IN ORDER TO SPECIFY WHICH SYNTACTICALLY NESTED BLOCK TO EXIT, THE EXIT LOOP STATEMENT SPECIFIES A (COMPOUND) STATEMENT LABEL. FORM:

```
EXIT LOOP LABEL;
```

EXAMPLE:

```
SET TOTAL TO 0;
SET I TO 1;
```

```
        DO
SUM:            BEGIN
                ADD A(I) + TOTAL TO TOTAL;
                ADD I + 1 TO I;
                IF I IS 10, THEN EXIT LOOP SUM;
                END
```

12. REGISTER LAYOUTS

REGISTER LAYOUTS WILL BE DONE DOWN THE PAGE IN ASCENDING BIT POSITION. ONE BLANK LINE WILL SEPARATE EACH BIT. THREE BLANK LINES WILL SEPARATE OCTAL DIGITS FOR EASE OF READING. THE MNEMONIC FOR EACH BIT OR FIELD (REGISTER) WILL BE GIVEN IN UPPER CASE FOLLOWED BY A COLON. THE SAME MNEMONIC WILL BE USED IN THE PRINTS, DIAGNOSTICS, SYSTEM SOFTWARE, THE HARDWARE MANUALS, THE MAINTENANCE MANUALS, AND ERROR REPORTS. ALL WORDS OF ALL BITS AND REGISTER NAMES WILL BE CAPITALIZED. IN TEXT THE FIRST FEW REFERENCES TO A BIT OR REGISTER WILL HAVE THE REGISTER NAME AND MNEMONIC IN PARENTHESES. TABLES WILL ALWAYS CONTAIN BOTH SINCE THEY ARE USED FOR REFERENCE.

REGISTER NAME (RN)

0	LPCS: IF 1, LOAD PREVIOUS CONTEXT SECTION REGISTER (PCS)
1	LCAC: IF 1, LOAD CURRENT AC REGISTER (CAC) FROM BITS 3-5
2	LPAC: IF 1, LOAD PREVIOUS AC REGISTER (PAC) FROM BITS 6-8
3-5	CAC: CURRENT AC REGISTER
6-8	PAC: PREVIOUS AC REGISTER

13. SPECIAL WORDS

ALL MNEMONICS, AND OPTION NUMBERS WILL BE IN ALL CAPS. EG. MBOX, MBUS, RH20. AN EXCEPTION IS MASSBUS WHICH IS COPYRIGHTED. BIT NAMES WILL HAVE THE FIRST LETTER CAPITALIZED, INCLUDING ANY GENERIC WORDS, IE. CACHE BIT. THE SYMBOL FOR MICRO-SECOND WILL BE "USEC".

[END OF CH0S02.SPC]

1080,2040,2060 ENGINEERING FUNCTIONAL SPEC - CHAP 1,1

TO: KL10 LIST, J. PARSLOW (200 FILE)

TITLE: 1080,2040,2060 PRODUCT DESCRIPTION - REV 0

STATUS: THIS CHAPTER IS THE PRODUCT DESCRIPTION FOR THE KL10, THE 1080 SECTION IS MISSING.

FILE: [EFS]CH1S01.SPC

PDM #: 200-200-036-00

DATE: 1 APRIL 74

SUPERSEDED MEMOS: DECSYSTEM 2060 PRODUCT DESCRIPTION, R. BINGHAM, 19 MARCH 74

SUPERSEDED SPECS:

ENGINEER: R. BINGHAM

APPROVED:

EDITOR: T. HASTINGS

TYPIST: M. PROUTY

REVIEWED:

ABSTRACT

THIS CHATER GIVES AN OVERVIEW OF THE 1080, 2040, AND 2060 SYSTEMS FROM THE PRODUCT POINT OF VIEW.

REVISION HISTORY

REV	DESCRIPTION	CHG NO	ORIG	DATE	APPD BY	DATE

THE DECSYSTEM 2060 REPRESENTS THE CULMINATION OF OVER TEN YEARS OF DEVELOPMENT OF THE STATE OF THE ART INTERACTIVE SYSTEMS BY DIGITAL EQUIPMENT CORPORATION. THE DECSYSTEM 2060 WITH THE VIRTUAL OPERATING SYSTEM PROVIDES SUCH ADVANCE FEATURES AS:

- 36 MILLION BYTES (8 MILLION 36 BIT WORDS) USER VIRTUAL ADDRESS SPACE
- SECTION ARCHITECTURE
- 398 INSTRUCTIONS INCLUDING VARIABLE BYTE LENGTH OPERATIONS, 72 BIT ARITHMETIC INSTRUCTIONS AND STACK OPERATION INSTRUCTIONS.
- AN ADVANCED BUSINESS INSTRUCTION SET WHICH HANDLES IBCDIC, ASCII, OR ANY OTHER CHARACTER REPRESENTATIONS UP TO 15 BITS IN LENGTH AND PROVIDES EXTENSIVE EDITING, TRANSLATION CHARACTER MANIPULATIONS AND RADIX CONVERSION CAPABILITIES
- INSTRUCTION RATE OF 1.8 MILLION INSTRUCTIONS PER SECOND OR 20% FASTER THAN 370/158
- UP TO 8 UNIVERSAL MASS STORAGE PROCESSORS FOR HIGH SPEED MAG TAPE, HIGH SPEED DRUM AND DISK
- AN IO BAND WIDTH OF ABOUT 9 MILLION BYTES PER SECOND (2 MILLION, 36 BIT WORDS PER SECOND)
- UP TO 4 FRONT END PROCESSORS FOR CARD READERS, LINE PRINTERS AND COMMUNICATIONS
- UP TO 2.3 MILLION BYTES (512K, 36 BIT WORDS) OF INTERNAL MEMORY
- 9,216 BYTES (2048 36 BIT WORDS) OF 125 NS CACHE MEMORY
- 8 SETS OF 16 REGISTERS
- ADVANCE MAINTAINABILITY FEATURE INCLUDING CACHE PARITY CHECKING, IO PROCESSORS LOOP-BACK FEATURES, MEMORY READ AND WRITE PARITY, AND REMOVABLE COMPONENT DIAGNOSTIC ISOLATION
- COMPACT REQUIRES A MINIMUM OF FLOOR SPACE
- ADVANCED VIRTUAL MEMORY OPERATING SYSTEM FEATURING:

  PROCESS STRUCTURED ARCHITECTURE

  MULTIPLE FORKING CAPABILITY

  VIRTUAL MEMORY CAPABILITY

MULTIPLE COMMAND LANGUAGE CAPABILITY

SECTION ARCHITECTURE

THE DECSYSTEM-2060 IS ORGANIZED INTO TWO ADDRESS SPACES - ONE FOR THE MONITOR AND ONE FOR THE USER PROGRAM. EACH CONSISTS OF 32, 512 PAGE SECTIONS WHERE EACH PAGE CONTAINS 512, 36 BIT WORDS. THUS, EACH USER HAS 32, 256K SECTIONS OF ADDRESS SPACE WHICH ALLOWS HIM A GREAT DEAL OF EXPANSION, PROTECTION, AND MODULARITY IN ORGANIZING HIS SYSTEMS STRUCTURE. FOR EXAMPLE, THE FORTRAN LIBRARY ROUTINES COULD RESIDE IN ONE SECTION THE COMPILER IN ANOTHER SECTION, AND THE USER COMPILED CODE IN YET ANOTHER SECTION. THUS, AND ERROR IN THE USERS CODE COULD NOT DESTROY THE FORTRAN LIBRARY ROUTINES WHICH MAY BE SHARED BY A LARGE NUMBER OF OTHER USERS. THE CONCEPT OF PAGE SHARING IS IMPLEMENTED IN VIROS SO A SINGLE PAGE MAY BE SHARED BY A NUMBER OF USERS THUS GIVING AN IMPROVED EFFICIENCY IN MEMORY UTILIZATION. THUS, A USER CAN HAVE A TOTAL PROGRAM SPACE WHICH EXCEEDS PHYSICAL MEMORY AND CAN BE AS LARGE AS 8 MILLION 36 BIT WORDS.

INSTRUCTION SET

THE DECSYSTEM-202 INSTRUCTION SET IS AN EXTENSION OF THE LOGICALLY DEVELOPED AND HIGHLY SUCCESSFUL DECSYSTEM-10 INSTRUCTION SET. THE DECSYSTEM-10 INSTRUCTION SET HAS BEEN AMPLIFIED TO INCLUDE SUCH THINGS AS VARIABLE LENGTH BYTE ADDRESSING TO ALLOW BYTES TO CROSS WORD BONDARIES THUS YIELDING, FOR INSTANCE, 12,5% INCREASE IN THE STORAGE EFFICIENCY OF 8 BIT BYTES. FURTHER, THE BYTE POINTER MAY BE ADJUSTED FORWARD OR BACKWARD TO FACILITATE INDEXING AND MANIPULATION OF BYTE STRING.

IN ORDER TO IMPLEMENT THE SECTION ARCHITECTURE, AN ADDITIONAL SPECIAL EXECUTE INSTRUCTION HAS BEEN IMPLEMENTED TO ALLOW COMMUNICATION ACROSS SECTION BOUNDARIES. AS IN THE DECSYSTEM-10, THE 2060 INCLUDES 72 BIT DOUBLE PRECISION ARITHMETIC FOR INCREASE COMPUTATIONAL ACCURACY. PUSH AND POP STACK INSTRUCTIONS ALLOW RECURSIVE SUBROUTINING. IN ORDER TO PROVIDE IMPROVED CAPABILITY IN THE BUSINESS DATA PROCESSING AREA, THE DECSYSTEM-2060 PROVIDES ADVANCED EDITING AND CHARACTER HANDLING CAPABILITY. FOR EXAMPLE, THE EDIT INSTRUCTION PROVIDES SUCH FEATURES AS CHECK PROTECTION FLOATING OF CURRENCY SIGNS, THE ABILITY TO SUPPRESS LEADING ZEROS AND TO BLANK A FIELD WHEN ZERO, THE ABILITY TO TRANSLATE FROM ONE CODE SET TO ANOTHER WHILE AT THE SAME TIME EDITING THE DATA FOR OUTPUT. THE ABILITY TO SUPPRESS LEADING BLANKS FOR INSERTION OF NUMERIC VALUES INTO TEXTIS ALSO PROVIDED. COMPLETE FLEXIBILITYIS PROVIDED TO CHOOSE WHATEVER CURRENCY AND PLACE VALUE SYMBOLS YOU MAY DESIRE IN EDITING NUMERICAL VALUES FOR OUTPUT. ALSO PROVIDED ARE STRING COMPARE, BINARY TO DECIMAL CONVERSION AND DECIMAL TO BINARY CONVERSION. THESE

INSTRUCTIONS SHOULD IN FACT BE CALLED CHARACTER TO BINARY AND BINARY TO CHARACTER SINCE ALL OF THE STRING TO BINAR CONVERSIONS ARE CODE INDEPENDENT, THE ONLY REQUIREMENT BEING THAT THE DIGITS IN THE CODE BE CONTIGUOUS, ALSO PROVIDED IS A SCAN CAPABILITY WHICH ALLOWS CHARACTER STRINGS TO BE SCANNED FOR ANY ONE OF A TABLE OF SIGNIFICANT CHARACTERS, I,E., COMMAS, PERIODS, EQUAL SIGNS OR OTHER SYMBOLS THAT WOULD HAVE SIGNIFICANCE IN A SYNTAX SCANNING APPLICATION, ALSO PROVIDED IN THE BUSINESS INSTRUCTION SET IS A CHARACTER STRING MOVE INSTRUCTION SO THAT BLOCKS OF CHARACTERS MAY BE MOVED,

PROCESSOR PERFORMANCE

IN ORDER TO OBTAIN THE OUTSTANDING PRICE PERFORMANCE CHARACTERISTICS THAT THE DECSYSTEM-2060 EXHIBITS, THE PROCESSOR SECTION OF THE MACHINE IS IMPLEMENTED IN HIGH SPEED E.C.L. WITH GATE DELAYS IN THE ORDER OF 3 NANO SECONDS,

THE KL10 ALSO FEATURES A 2,048 WORD SOLID STATE MEMORY BUFFER STORE WHICH TYPICALLY ALLOWS 90% OF THE CPU MEMORY REFERENCES TO OCCUR IN A 125 NANO SECONDS, ANOTHER ADVANCED FEATURE OF THE DECSYSTEM-2060 CACHE IS THAT IT DOES NOT REQUIRE CACHE WRITE THROUGHS AS OTHER CURRENT MEMORY CACHE DESIGNS DO, THIS ELIMINATES, FOR INSTANCE, THE NECESSITY OF WRITING BACK INTO MAIN MEMORY EACH VALUE OF A LOOP INDEX IN A SMALL INSTRUCTION LOOP, IO IS CHECKED AGAINST THE CACHE TO MAKE SURE THE MOST CURRENT DATA IS SENT TO THE IO DEVICES, LIKEWISE, PAGES IN THE CACHE ARE MARKED INVALID AS DATA FOR THAT PAGE COMES FROM AN IO DEVICE, PAGES ARE ONLY WRITTEN BACK INTO THE MAIN MEMORY FROM THE CACHE IN ORDER TO MAKE ROOM FOR NEW PAGES, ALL THIS ADDS UP TO 1,85 MIPS AVERAGE INSTRUCTION RATE PROCESSOR, ADD TIMES, FOR INSTANCE, OCCUR IN 550 NANO SECONDS, A FLOATING 36 BIT ADD AND ROUND REQUIRES ONLY 1,72 MICROSECONDS,

INPUT/OUTPUT

PACKAGED WITHIN THE KL10 ITSELF ARE UP TO 8 MASS STORAGE PROCESSORS, UP TO 8 DEVICES MAY BE ATTACHED TO ANY SINGLE PROCESSOR, THESE DEVICES CAN BE A MIX OF A MAG TAPE, RP04 MOVING HEAD DISK, AND RS04 DISK, FOUR OF THE CHANNELS CAN OPERATE UP TO 1 MICROSECOND PER WORD TRANSFER RATE AND FOUR OF THE CHANNELS AT TWO MICROSECONDS PER WORD TRANSFER RATE, HOWEVER, THE COMPLETE SIMULTANEOUS TRANSFER RATE IS SOMEWHERE IN THE ORDER OF 2 MILLION TO 2-1/2 MILLION WORDS PER SECOND, THE I/O PROCESSORS EXECUTE INSTRUCTION LISTS IN MAIN MEMORY AND HAVE THE ABILITY TO LOOK AHEAD ONE INSTRUCTION SO THAT, FOR INSTANCE, IN THE CASE OF A FIXED HEAD DISK, THE I/O PROCESSOR CAN CHANGE FROM READ TO WRITE IN THE SECTOR GAP TIME, SOME OF THE OTHER FEATURES OF THE MASS STORAGE PROCESSORS RELATE TO RELIABILITY, THE MASS STORAGE PROCESSORS PROVIDE COMPLETE PARITY CHECKING FROM THE DEVICE TO THE MEMORY, THEY ALSO HAVE LOOP-BACK FEATURES WHICH ENABLE THE CENTRAL PROCESSOR TO

ISOLATE FAULTS IN THE MASS STORAGE PROCESSOR WITHOUT HAVING ANY DEVICE CONNECTED TO IT.

EACH DECSYSTEM-2060 HAS UP TO FOUR FRONT ENDS I/O PROCESSORS FOR LOW SPEED PERIPHERALS SUCH AS CARD READERS, LINE PRINTERS SYNCHRONOUS, AND ASYNCHRONOUS COMMUNICATIONS. THESE FOUR FRONT END PROCESSORS HAVE THE COMPLETE INSTRUCTION SET OF THE PDP-11 AND HELP TO RELIEVE THE PROCESSING BURDEN OF THE DECSYSTEM-2060 PROCESSOR ASSOCIATED WITH LOW SPEED PERIPHERAL CONTROL. THE PDP-11 FRONT END PROCESSOR ACTS AS A CONSOLE PROCESSOR FOR THE DECSYSTEM-2060 AND FUNCTIONS EXTENSIVELY AS A DIAGNOSTIC PROCESSOR TO ALLOW INTERNAL REGISTERS IN THE DECSYSTEM-2060 TO BE READ IN ORDER TO DIAGNOSE FAILURES IN THE PROCESSOR. THE PDP-11 ALSO PROVIDES SERVICES SUCH AS THE LOADING OF THE CONTROL STORE OF THE DECSYSTEM-2060 MICROPROCESSOR.

MEMORY

WITHIN THE PROCESSOR ITSELF, THE DECSYSTEM-2060 MAY CONTAIN UP TO ONE HALF MILLION (512K) WORDS OF MEMORY. THE MEMORY IS 4 WORDS WIDE SO THAT DURING A SINGLE MEMORY ACCESS, THE MAIN MEMORY FOUR WORDS SIMULTANEOUSLY OR WRITES UP TO FOUR WORDS SIMULTANEOUSLY THUS GIVING A MAXIMUM BAND WIDTH OF 4 MILLION WORDS PER SECOND. THE MEMORY IS ORGANIZED INTO 32K BLOCKS. THE MEMORY BLOCK STARTING ADDRESS MAY SWITCH UNDER PROGRAM CONTROL FOR DYNAMIC RECONFIGURATION OF THE MEMORY SYSTEM IF ANY 32K BLOCK IS FAILING. CONSISTENT WITH DYNAMIC RECONFIGURATION, THE WIDTH OF THE MEMORY MAY ALSO BE PROGRAMMED. THAT IS, IT MAY BE SET UP AT 1, 2 OR 4 WORDS WIDE SO THAT IN A 128K SYSTEM; FOR EXAMPLE, IF ANY 32K BZLOCK SHOULD FAIL, IT MAY CONTINUE TO OPERATE 2 BLOCKS SIMULTANEOUSLY WITH REDUCED I/O PERFORMANCE.

VIROS

THE DECSYSTEM 2060 OFFERS THE USER THE LATEST IMPROVEMENTS IN OPERATING SYSTEM TECHNOLOGY INCLUDING PROCESS STRUCTURING OF THE MONITOR SO THAT THE MONITOR ITSELF MAY BE PAGED AND HENCE INFREQUENTLY USED PORTION OF THE MONITOR NEED NOT TAKE UP VALUABLE MAIN MEMORY SPACE. PROCESS STRUCTURING IN THIS WAYLEADS TO A SMALL CORE RESIDENT REQUIREMENT, ROUGHLY 22K. VIROS CREATES A DEMAND PAGING VIRTUAL MEMORY ENVIRONMENT FOR THE USER SO THAT HIS PROGRAM MAY ACTUALLY EXCEED THE PHYSICAL SPACE LIMITS OF THE SYSTEM AND STILL RUN EFFICIENTLY. ADDITIONALLY, VIROS PROVIDES PROCESS FORKING ALLOWING ONE JOB (FORK) TO SPAWN SEVERAL PARALLEL TASKS (FORKS) WHICH CAN THEN BE SCHEDULED FOR COMPLETION BY THIS MONITOR INDEPENDENTLY, THUS IMPROVING A SYSTEM IMPLEMENTORS, FLEXIBILITY AND SYSTEM EFFICIENCY.

VIROS ALSO OFFERS MULTIPLE COMMAND LANGUAGES. PRESENTLY, TWO COMMAND LANGUAGES ARE BEING OFFERED - TOPS-10 FOR COMPATIBILITY WITH DECSYSTEM-10 AND VIROS WHICH IS A DERIVATIVE OF BBN'S

TENEX. IN THE FUTURE, COMMAND LAGUAGES CAN BE EASILY ADDED TO PROVIDE COMPATIBILITY WITH OTHER WIDELY USED OPERATING SYSTEMS FOR THE DECSYSTEM-10 TOPS-10 USERS. VIROS PROVIDES EXTENSIVE COMPATIBILITY FEATURES ALLOWING MOST TOPS-10 PROGRAMS TO RUN UNMODIFIED UNDER VIROS AT SOME LOSS IN PROGRAM EFFICIENCY.

KL10 PERIPHERALS

- TU16 MAGNETIC TAPE
  - 1600 BPI
  - 45 IPS
- RS04
  - 250,000 WORD CAPACITY
  - TRANSFER RATE IS 4 MICROSECONDS FOR 36 BIT WORD
  - IT HAS A LATENCY OF 8.3 MS
- RP04
  - 100 MEGABYTE CAPACITY
  - AVERAGE ACCESS TIME 30 MS
  - TRANSFER RATE 5.6 MICROSECONDS PER 36 BIT WORD
  - LATENCY IS 8.3 MS
- TU70
  - YET TO BE DEFINED

(CONFIGURATIONS TO BE FILLED IN LATER)

ALL DATA IS TO BE CONSIDERED PRELIMINARY AND WILL BE REFINED AND EXPANDED AS TIME PERMITS. THUS, THE DECSYSTEM-2060 SUPPLIES UNEQUALED PRICE PERFORMANCE IN THE MEDIAN TO HIGH PERFORMANCE GENERAL COMPUTER MARKET.

[END OF CH1S01.SPC]

1080,2040,2060 ENGINEERING FUNCTIONAL SPEC - CHAP 1,3

TO: KL10 LIST, J. PARSLOW (200 FILE)

TITLE: DIFFERENCES FROM THE KI10 - REV 1

STATUS: THIS CHAPTER REFLECTS THE CHANGES MADE DURING DEVELOPMENT OF THE 1080. BECAUSE THIS CHAPTER MENTIONS UNANNOUNCED CAPABILITIES, IT IS COMPANY CONFIDENTIAL. THESE CAPABILITIES ARE FLAGGED WITH *.

FILE: [EFS]CH1S03.SPC

PDM #: 200-200-017-01

DATE: 4 JUN 75

SUPERSEDED MEMOS: DIFFERENCES BETWEEN KI AND KL HARDWARE - VERSION 2, T. HASTINGS, 24 MAY 73

SUPERSEDED SPECS:

ENGINEER: T. HASTINGS

APPROVED:

EDITOR: T. HASTINGS

TYPIST: M.PROUTY

REVIEWED: 21 MAY 74

ABSTRACT

THIS CHAPTER DESCRIBES THE DIFFERENCES BETWEEN THE KI AND THE KL HARDWARE. A SHORT REASON FOR EACH DIFFERENCE IS USUALLY GIVEN. IT IS NOT A SYSTEM DESCRIPTION, SO SOME OF THE DIFFERENCES MAY NOT BE SUPPORTED BY SOFTWARE. THIS LIST DOES NOT GIVE DETAILS OF EACH DIFFERENCE. IT IS NOT A SUBSTITUTE FOR FUNCTIONAL SPECIFICATIONS. INSTEAD, THE LIST IS AN INDEX TO THE MORE DETAILED DOCUMENTATION. IT ALSO SHOWS WHERE MORE DETAILED DOCUMENTATION IS NEEDED. NEW ITEMS WILL BE ADDED TO THIS LIST AFTER THEY HAVE HAD A DETAILED SPEC WRITTEN AND APPROVAL HAS BEEN GIVEN. THUS, THIS LIST WILL SERVE AS A RECORD OF THE APPROVAL FOR CHANGES. PROPOSED DIFFERENCES APPEAR IN THE APPENDIX. ITEMS FLAGGED WITH * ARE NOT PRESENT IN THE 1080.

REVISION HISTORY

REV DESCRIPTION	CHG NO	ORIG	DATE	APPD BY	DATE

0, CONTENTS

1. SUMMARY OF MAJOR DIFFERENCES

1.1 FASTER THAN KI.

1.2 CHEAPER THAN KI.

1.3 SMALLER IN SIZE. TWO KL CABINETS HOLD WHAT REQUIRED 22 CABINETS ON KI.

1.4 BUSINESS INSTRUCTION SET ADDED.

*1.5 EXTENDED ADDRESSING ADDED SO THE MONITOR CAN ADDRESS 8M WORDS AND THE USER CAN ADDRESS 8 M WORDS. FUTURE MACHINES CAN HAVE UP TO 30-BIT ADDRESSES.

*1.6 GENERALIZED PAGING. HARDWARE (MICRO-CODE) HELPS WITH PAGE REPLACEMENT ALGORITHMS AND WORKING SET DETERMINATION.

*1.7 INTERNAL MEMORY IS BUILT INTO PROCESSOR. 4 WORD REFERENCES ARE MADE IN PARALLEL. EXTERNAL MEMORY ALSO AVAILABLE INTERFACING OVER THE KI MEMORY BUS.

1.8 MEMORY REFERENCES ARE CACHED. THE 2K CACHE HAS WRITEBACK AND 4 WAY CONTENTION LOGIC.

*1.9 UP TO 8 MASSBUS CONTROLLERS ARE BUILT INTO THE CPU.

1.10 UNIT RECORD EQUIPMENT AND COMMUNICATIONS GEAR WILL BE ATTACHED TO A FRONT END PDP-11/40.

1.11 THE HARDWARE IS BEING DESIGNED FOR UP TO 4 CPUS MULTI-PROCESSOR TIGHTLY COUPLED CONFIGURATION. (NEEDS SOFTWARE ARCHITECTURE WORK CURRENTLY NOT BUDGETED.)

1.12 MOST BUSSES AND DATA PATHS HAVE PARITY FOR BETTER ERROR DETECTION.

1.13 THE FRONT END CAN DIAGNOSE THE KL10.

1.14 THE MACHINE HAS A MICRO-CODE DESIGN. THE MICRO-CODE IS LOADED AT SYSTEM STARTUP INTO A RAM.

1.15 MANY DATA PATHS HAVE PARITY. ALSO RAMS AND CACHE.

1.16 THE SYSTEM IS BEING BUILT WITH LARGE MODULES. THUS MANY SIGNALS TRADITIONALLY USED IN TROUBLE SHOOTING WITH A SCOPE WILL NOT BE PRESENT ON THE BACK PANEL. THUS FIELD SERVICE WILL RELY MORE ON THE DIAGNOSTICS TO DETECT WHICH BIG BOARD IS FAILING THAN THEY DID ON THE KI10. (ON THE KI THE DIAGNOSTICS PERFORMED MORE OF AN EXCERCISER ROLE WHILE FIELD SERVICE RELIED ON THEIR SCOPES TO FIND BAD MODULES).

2, NEW EXEC/USER INSTRUCTIONS

2,1 EXTEND

PART OF BUSINESS INSTRUCTION SET (BIS), DECIMAL TO BINARY, BINARY TO DECIMAL, STRING EDIT, MOVE, COMPARE, TRANSLATE, SEE CHAPTER 2,10, EXTEND INSTRUCTION,

2,2 DADD, DSUB, DMUL, DDIV

PART OF BIS, DOUBLE PRECISION (70 BIT) INTEGER ADD, SUBTRACT, MULTIPLY AND DIVIDE, SEE CHAPTER 2,4, BUSINESS INSTRUCTION SET,

2,3 ADJSP

ADJUST PUSHDOWN STACK POINTER FORWARD AND BACKWARD,

2,4 JSYS

RESERVED FOR CALLING THE MONITOR, LIKE A UUO,

3. CHANGED EXEC/USER INSTRUCTIONS

3,1 PUSH

IF RH(AC) ADDRESSES AC AFTER INCREMENTING, KL STORES UPDATED POINTER, DESTROYING COPY OF (E). KA, KI STORED POINTER THEN COPY OF (E), WHICH CLOBBERED POINTER.

3,2 POP AC,AC

LEAVES DECREMENTED POINTER IN AC (LIKE PDP-6). KA, KI LEAVE WORD FROM STACK.

3,3 IBP (ADJBP)

IF AC FIELD IS NON-ZERO, THE BYTE POINTER IS INCREMENTED OR DECREMENTED BY THE NUMBER OF TIMES SPECIFIED BY THE CONTENTS OF THE AC, AND THE RESULT IS PLACED IN THE AC BUT NOT MEMORY. THE MNEUMONIC IS ADJBP WHEN AN AC FIELD IS SPECIFIED BECAUSE THE

OPERATION OF THE INSTRUCTION IS SO DIFFERENT.

3,4 MAP

NEED SPEC. NO LONGER LEGAL IN USER UNLESS IN USER I/O MODE.

3,5 DFDV

ROUNDS ON THE KL10 FOR EXTRA NUMERIC ACCURACY. TRUNCATES ON THE KI10.

3,6 FSC

ON THE KA ANDP KI, IF FSC OVERFLOWS TOO FAR, THE UNDERFLOW (FXU) FLAG ALSO GETS SET. ALSO, IF FSC UNDERFLOWS TOO FAR, THE UNDERFLOW (FXU) FLAG IS NOT SET. ON THE KL, FXU GETS SET IF AND ONLY IF AN UNDERFLOW OCCURS. THIS SHOULD NOT CAUSE ANY SERIOUS PROGRAM INCOMPATIBILITIES (EXCEPT DIAGNOSTICS).

3,7 FAD, FSB, FMP, FDV (-, M, B)

THESE INSTRUCTIONS SHOULD BE AVOIDED, BECAUSE ADDING A VERY SMALL NEGATIVE NUMBER WILL ACTUALLY DECREASE THE RESULT BY 1 IN THE LEAST SIGNIFICANT BIT, EXCEPT IF THE EXPONENTS DIFFER BY MORE THAN 64 (KA,KI) OR 72 (KL). THE "ROUNDING" VERSIONS (FADR, FSBR, FMPR, FDRV) ARE BETTER NUMERICALLY. WE CAN FIND NO PLACES WHERE THESE CANNOT BE REASONABLY REPLACED BY THE CORRESPONDING ROUNDING INSTRUCTIONS. OUR FORTRAN, COBOL, AND ALGOL COMPILERS ALL USE THE ROUNDING INSTRUCTIONS. THESE INSTRUCTIONS ARE BEING IMPLEMENTED SOLELY SO THAT WE CAN SAY THAT THE KL IS "COMPATIBLE" WITH KA AND KI.

3,8 DMOVEM, DMOVNM

ON THE KI, DMOVEM AC,AC+1 COPIES C(AC) TO AC+1 AND AC+2. ON THE KL AC+1 IS COPIED TO AC+2 AND AC IS COPIED TO AC+1. THIS IS ACCOMPLISHED BY FETCHING BOTH OPERANDS BEFORE STORING EITHER ONE.

3,9 DMOVE, DMOVN, DMOVEM, DMOVNM

ON THE KL, THESE INSTRUCTIONS ALWAYS FETCH OR STORE NEITHER OR BOTH WORDS BEFORE AN INTERRUPT OR A TRAP CAN OCCUR. THEREFORE THESE INSTRUCTIONS DO NOT SET OR TEST FIRST PART DONE FLAG.

3,10 DMOVN, DMOVNM

ON KI, NEGATIVE DOUBLE PRECISION INSTRUCTIONS NEVER SET ANY PC FLAGS BECAUSE THEY ARE USED ONLY FOR DOUBLE PRECISION FLOATING POINT NUMBERS. ON THE KL, THEY ARE ALSO USED FOR DOUBLE PRECISION INTEGERS. AN OVERFLOW ERROR CAN OCCUR IF AN ATTEMPT IS MADE TO NEGATE THE MOST NEGATIVE NUMBER (-2 TO THE 70TH POWER). THEREFORE ON THE KL IF THE SOURCE OPERAND IS NEGATIVE, CRY0 AND CRY1 ARE SET, EXCEPT IF THE SOURCE = 1B0/0 IN WHICH CASE AROV AND CRY0 ARE SET.

3,11 JRST (PORTAL, JRSTF, HALT, JEN)

THE AC FIELD OF THE JRST INSTRUCTION WILL BE DISPATCHED ON BY THE MICRO CODE. THE ONLY VALID AC FIELD COMBINATIONS WILL BE:

	AC FIELD	MNEUMONIC	
	0	JRST	
	1	PORTAL	
	2	JRSTF	
	4	HALT	
*	5	RPCF	RESTORE PC AND FLAGS
*	6	RPCFD	RESTORE PC, FLAGS AND DISMISS
*	7	EPCF	EXCHANGE PC AND FLAGS
	12	JEN	
*	14	SFM	SAVE FLAGS IN MEMORY

ANY COMBINATIONS OF THE ABOVE WILL TRAP ON KL. THE KA AND KI PERFORMED THE OR OF THE INDICATED FUNCTIONS.

3,12 MOVXA MOVE EXTENDED ADDRESS

MOVXA WORKS LIKE MOVEI, EXCEPT THE LH OF THE AC IS SET TO THE EFFECTIVE SECTION NUMBER. THE OPCODE IS SETMI WHICH WORKS AS ON THE KI IN SECTION 0.

3,13 BLT

THE AC OF BLT IS ALWAYS UPDATED WITH A COPY OF THE POINTER USED FOR THE LAST TRANSFER WITH BOTH HALVES INCREMENTED. THIS IS

DONE JUST BEFORE THE FIRST WORD IS TRANSFERRED SO THAT BLT 17,17 STILL WORKS TO RESTORE ALL OF THE ACS. THIS CHANGE IS INTRODUCED SO THAT BLT ALWAYS LEAVES THE ACS THE SAME NO MATTER WHETHER AN INTERRUPT OCCURRED OR NOT. THIS DOES NOT CATCH BUGS OF THE FORM BLT A, N (A).

4. DISCONTINUED EXEC/USER INSTRUCTIONS

4,1 UJEN (100)

NOT AVAILABLE ON KL TO DISMISS REALTIME INTERRUPTS AS IT WAS NOT ON KI. UJEN IS AVAILABLE ONLY ON KAS.

4,2 UFA, DFN, FADL, FSBL, FMPL, FDVL ARE PROVIDED ON THE KL10 ONLY FOR COMPATIBILITY WITH THE KA10. (KA10 SOFTWARE USES THEM TO SIMULATE DOUBLE PRECISION FLOATING POINT). THESE INSTRUCTIONS MAY BE DELETED ON THE KL10 AT SOME FUTURE TIME AND MAY NOT BE IMPLEMENTED ON A SUCCESSOR MACHINE AT ALL. THESE INSTRUCTIONS SHOULD BE USED ONLY FOR KA10 DOUBLE PRECISION COMPATIBILITY. MOST OF DEC SUPPORTED SOFTWARE CONDITIONALLY ASSEMBLES KA OR KI/KL VERSIONS OF DOUBLE PRECISION FLOATING POINT. SOFTWARE SIMULATION OF DOUBLE PRECISION FLOATING POINT ON A KI OR KL USING THESE INSTRUCTIONS RUNS A FACTORS OF 8 SLOWER THAN USING THE KI OR KL DOUBLE PRECISION FLOATING POINT INSTRUCTIONS. A SECOND REASON FOR PHASING THESE OUT IS TO SAVE MICRO CODE SPACE.

5. NEW EXEC MODE ONLY INSTRUCTIONS

NONE. SEE NEW DEVICES CODES. SOME I/O INSTRUCTIONS TO THE INTERNAL DEVICES REALLY PERFORM ARBITRARY FUNCTIONS NOT NECESSARILY RELATED TO THE INTERNAL DEVICE. SEE CHAPTER 2,5, INTERNAL DEVICES.

6. CHANGED EXEC MODE ONLY INSTRUCTIONS

6,1 EXECUTIVE XCT HAS BEEN RENAMED PXCT (PREVIOUS - CONTEXT XCT). PXCT PERMITS INDEXING IN PREVIOUS CONTEXT. THE DEFINITIONS OF BITS 9-12 OF PXCT INSTRUCTION HAVE CHANGED FROM KI.

7. DISCONTINUED EXEC MODE ONLY INSTRUCTIONS

NONE.

8. NEW IO DEVICE CODES

8.1 MTR - ACCOUNTING METERS

SEE CHAPTER 2,7, KL10 PROGRAM CLOCKS (TIM,MTR).

8.2 TIM

INTERVAL TIMER, TIME BASE, AND PERFORMANCE ANALYSIS COUNTER.
SEE CHAPTER 2,7, KL10 PROGRAM CLOCKS (TIM, MTR).

8.3 DTE0, DTE1, DTE2, DTE3

10/11 INTERFACE. SEE CHAPTER 4.3, 10/11 INTERFACE - DTE20.

*8.4 MBC0,...,MBC7

MASSBUS CONTROLLER. SEE CHAPTER 4.1, MASSBUS CONTROLLER - RH20

8.5 CCA - CACHE SWEEP

9. CHANGED IO DEVICES CODES

9.1 APR

BITS REARRANGED, MADE MORE REGULAR, NO 60 CYCLE LINE FREQUENCY INTERRUPT. SETS ADDRESS COMPARE REGISTER.

9.2 PI

SOME FUNCTIONS DROPPED AND SOME HAVE MOVED TO APR

9.3 PAG

SOME CHANGES, INCLUDING NEW, KL PAGING.

10. DISCONTINUED IO DEVICE CODES

10.1 CTY

CONSOLE TTY - MUST USE -11 INSTEAD.

10.2 PTR

PAPER TAPE READER - NONE PROVIDED, EVEN IN FRONT END. DATAO PTR, NO LONGER SETS ADDRESS BREAK. SEE APR.

10.3 PTP

PAPER TAPE PUNCH - NONE PROVIDED, EVEN IN FRONT END.

*11. DISK DIFFERENCES FROM RP10 TO RH20

11.1 DISK, DRUM, AND TAPE WILL BE ATTACHED TO ONE OR MORE MASSBUS CONTROLLERS. SEE CHAPTER 4.1, MASSBUS CONTROLLERS (RH20) AND CHANNELS (MBOX).

11.2 UP TO 8 MEMORY CHANNEL/CONTROLLERS CAN BE ATTACHED TO THE SYSTEM. EACH CONTROLLER MUST HAVE ITS OWN CHANNEL (UNLIKE THE DF10). THESE CHANNELS ARE BUILT INTO THE CPU.

11.3. EACH CONTROLLER CAN HAVE UP TO 8 DEVICES. FOR MAGTAPE A DEVICE IS THE FORMATTER WHICH IN TURN CONTROLS UP TO 8 SLAVE DRIVES. THE MAXIMUM DISK, DRUM, AND FORMATTER DEVICES IS 8*8 = 64 PER SYSTEM.

11.4 BECAUSE THE SAME CONTROLLER CAN CONTROL THE THREE DIFFERENT TYPES OF DEVICES, A GREATER NUMBER OF SUPPORTED CONFIGURATIONS IS POSSIBLE. FOR EXAMPLE, THE CHEAPEST SYSTEM COULD HAVE ONLY ONE CONTROLLER FOR BOTH DISKS AND MAGTAPE ALTHOUGH THIS MAY NOT BE SUPPORTED BY SOFTWARE.

11.5 BECAUSE THE CONTROLLERS ARE SO LOW IN COST (THE DEVICE DEPENDENT COMPLEXITY IS IN EACH DEVICE), IT IS MORE LIKELY THERE WILL BE MORE THAN ONE DISK CHANNEL. IF THERE ARE CHANNELS AND N DRIVES, N/M DRIVES WILL BE ATTACHED TO EACH CHANNEL. THE SAME FOR DRUMS. THIS WILL IMPROVE THE THRUPUT.

11.6 IT WILL BE EASIER TO CONFIGURE A SYSTEM WHICH EXCEEDS THE BANDWIDTH OF THE MEMORY. THE SOFTWARE MUST DO BANDWIDTH SCHEDULING OF CHANNELS AND MEMORY.

11.7 THE CONTROLLER HAS A BACKUP COMMAND REGISTER, SO THE SOFTWARE CAN SWITCH TRACKS BETWEEN SECTORS.

11.8 SEEKS CAN BE INITIATED ON DRIVES DURING A DATA TRANSFER ON ANOTHER DRIVE.

11.9 SEEK COMPLETION INTERRUPTS CAN BE ENABLED OR DISABLED DURING A DATA TRANSFER.

11.10 RP04 HAS ECC CORRECTION HARDWARE WHICH COMBINED WITH SOFTWARE CAN CORRECT 11 BIT BURST ERRORS.

11.11 RP04 HAS HEAD OFFSET ERROR RECOVERY HARDWARE.

11.12 PDP-10 OR PDP-11 FORMAT RP04 PACKS CAN BE READ OR WRITTEN ON THE PDP-10 (RH20) OR THE PDP-11 (RH11). PDP-11 16 BIT WORDS (BITS 15-0) COME INTO PDP-10 BITS 2-17 AND 20-35. SINCE PDP-11 INCREMENTS BYTES FROM RIGHT TO LEFT, BYTE DATA APPEARS REVERSED WITHIN EACH HALF -10 WORD.

11.13 PARITY ON MASSBUS CONTROL BUS, SO SOFTWARE HAS TO CHECK FOR ERRORS IN COMMANDS AS WELL.

11,14 RP04 HAS UNLOAD CAPABILITY.

11,15 SOFTWARE MUST ACKNOWLEDGE THAT PACK HAS BEEN MOUNTED BY SETTING VOLUME VALID BIT AFTER A POWER ON INTERRUPT FROM RP04 DRIVE.

11,16 RP04 HAS AUTOMATIC RECALIBRATE TO 0 ON SEEK INCOMPLETE.

11,17 RP04 HAS MID TRANSFER SEEK CAPABILITY (SPIRAL READ/WRITE). USED BY SIMPLE MINDED SOFTWARE ONLY.

11,18 RP04 HAS IMPLIED SEEK (BUT THIS TIES UP THE CONTROLLER SO PDP-10 WILL NOT USE). USED BY SIMPLE MINDED SOFTWARE ONLY.

11,19 RP04 HAS DUAL PORT OPTION.

11,20 MASSBUS CONTROLLER IS DESIGNED SO THAT TWO OF THEM CAN BE ELECTRICALLY CONNECTED TO ONE MASSBUS. THE SOFTWARE MUST ENABLE ONE AND DISABLE THE OTHER. IT IS INTENDED FOR FAILSAFE RECOVERY, RATHER THAN DYNAMIC SWITCHING.

11,21 COMMAND LIST MUST RUN OUT ON A SECTOR BOUNDARY, ELSE AN ERROR BIT. THIS HELPS INSURE THAT WORDS DO NOT GET ADDED OR DROPPED. PARTIAL SECTORS CAN BE READ AND WRITTEN BY FILLING OUT WITH ANOTHER IOWD WITH BITS 14-35 ZERO.

11,22 THE CONTROLLER HAS A BACKUP COMMAND REGISTER SO THAT A NEW COMMAND ON A DIFFERENT TRACK CAN BE INITIATED DURING A SECTOR GAP TIME. THIS SHOULD DOUBLE THRUPUT ON HEAVILY LOADED SYSTEMS BECAUSE RANDOM CONSECUTIVE SECTORS CAN BE READ AND WRITTEN.

11,23 RH20 DOES NOT PROVIDE READ-IN. MUST BE DONE BY THE FRONT END -11.

*12. CHANNEL DIFFERENCES

12.1 KI DF10 DEVICES REQUIRE FLUSHING THE CACHE BEFORE AND AFTER THE TRANSFER, SINCE THE DF10 CAN NOT ACCESS THE CACHE. [A. KOTOK, KL10 CACHE, OCT 19] SINCE MOST USER PROGRAMS DO NOT MOVE BUFFERS AROUND, TOPS10 CAN SWEEP THE CACHE ON THE FIRST USE OF THE BUFFER AND TURN OFF CACHE BIT FOR THAT PAGE. THIS WILL IMPROVE IO THRUPUT AT THE EXPENSE OF CPU TIME, BUT SHOULD KEEP SYSTEM BALANCED.

12.2 THE CHANNEL COMMAND LIST HAS 22 BIT ADDRESSES ONLY.

12.3 ADDRESS BITS 14-35 SPECIFY THE FIRST ADDRESS, NOT FIRST ADDRESS -1.

12.4 THE WORD COUNT IS SPECIFIED IN BITS 3-13 INSTEAD OF 0-17.

12.5 WORD COUNT BITS 3-13 SPECIFY THE POSITIVE NUMBER OF WORDS, NOT THE NEGATIVE.

12.6 THE CHANNEL CAN READ REVERSED (BIT 2 = 1).

12.7 THE LAST IOWD CAN OPTIONALLY SPECIFY HALT AT END (BIT 1 = 1) RATHER THAN REQUIRING ANOTHER COMMAND WORD OF ALL ZEROES TO BE FETCHED. THIS HELPS REDUCE CHANCES OF MISSING LATENCY WHERE THE NEXT COMMAND IS WAITING.

12.8 THE JUMP WORD FORMAT REQUIRES BITS 0-2 TO BE 2.

12.9 ERROR STATUS INFORMATION IS STORED IN THE EPT.

12.10 IF BITS 14-35 ARE 0, THE READ OPERATION SKIPS AND THE WRITE OPERATION WRITES ZEROES.

*13. TAPE DIFFERENCES

13.1 DECTAPES (TU56) ON THE -11 WILL BE USED FOR BOOTSTRAP, FOLLOWING DEC STANDARD FOR PDP-10 [SIC] TAPES. PDP-11 FORMAT TAPES WILL NOT BE USED. THUS THE TAPES CAN BE WRITTEN DURING TIME SHARING ON SYSTEMS WHICH HAVE TU56 ON DIA20 I/O BUS ADAPTER. NOTE: NO TIME SHARING SUPPORT IS PLANNED FOR -11 TU56S.

13.2 MAGTAPE CAN READ AND WRITE BACKWARDS. THE DATA WILL END UP IN MEMORY INDEPENDENT OF THE DIRECTION PROVIDED THE MONITOR SETS UP THE COMMAND LIST TO POINT TO THE BEGINNING OF BUFFER FOR FORWARD AND THE END FOR BACKWARD. THE RECORD MUST EVENLY FILL THE LAST WORD OF THE CORE BUFFER.

13.3 MAGTAPE HAS A NEW MODE, CALLED ASCII. ALL 36 BITS ARE WRITTEN ON THE TAPE IN FIVE FRAMES. BIT 35 IS WRITTEN AS THE HIGH ORDER BIT OF LAST FRAME. THUS, THIS MODE CAN REPLACE CORE DUMP MODE AND INDUSTRY COMPATIBLE MODE FOR MOST 9-TRACK PE TAPES. THE FOUR MODES IN FORWARD AND 3 MODES REVERSE WILL BE DONE BY A SPECIAL PDP-10 MICRO PROGRAMMED BIT FIDDLER BOARD. THE PDP-11 WILL HAVE SIMILAR BUT SIMPLER AND CHEAPER BOARD. THE TU16 MASTER WILL DIFFER BETWEEN -10 AND -11 ONLY IN THIS BOARD.

13.4 BACKUP REGISTER IN RH20 CANNOT BE USED FOR TAPE OPERATIONS. SOFTWARE MUST RESPOND TO INTERRUPT WITHIN 1 MILLISECOND AS ON KI IN ORDER TO KEEP THE TAPE FROM SHUTTING DOWN.

14. OTHER IO DIFFERENCES

14.1 NO DK10, INSTEAD KL WILL HAVE BUILT-IN REAL TIME CLOCK WITH INTERNAL DEVICE CODES TIM AND MTR.

14.2 THE KL10 WILL HAVE A BUILT-IN 52 BIT TIME BASE CLOCK (TIM).

14.3 THE KL10 WILL HAVE A BUILT-IN ACCOUNTING METER TO COUNT EBOX USEFUL TIME AND MBOX REFERENCES.

14.4 THE KL10 WILL HAVE BUILT-IN PERFORMANCE ANALYSIS COUNTER (MTR) FOR FINDING HARDWARE AND SOFTWARE BOTTLENECKS.

14.5 NO LINE FREQUENCY CLOCK. PDP-11 WILL DO TIME OF DAY.

14.6 UNIT RECORD EQUIPMENT WILL BE ON THE 11.

14.7 THERE WILL BE NO CTY, PTR, OR PTP BUILT INTO THE PROCESSOR. ALL STAND-ALONE PROGRAMS WILL USE A COMMON ROUTINE TO TALK TO A TERMINAL ON THE 11.

14.8 THERE WILL NOT BE AN IO BUS, UNLESS A CUSTOMER BUYS AN IO BUS ADAPTER (DIA20).

14.9 THE 10/11 INTERFACE IS QUITE DIFFERENT FROM THE DL10 FROM A PROGRAMMING POINT OF VIEW. IT IS SIMPLER, HAS HIGHER THRUPUT AND REFLECTS WHAT WE LEARNED FROM THE DL10/DC75 PROJECT. TRAFFIC ON THE INTERFACES STEALS TIME FROM THE CPU INSTEAD OF ONLY COMPETING FOR MEMORY CYCLES. SEE CHAPTER 4.3, FRONT END INTERFACE (DTE20).

14.10 UP TO 4 PDP-11S CAN BE ATTACHED TO THE KL10, EACH WITH ITS OWN 10/11 INTERFACE.

14.11 UP TO 4 DTE20S CAN BE CONNECTED TO A SINGLE UNIBUS.

14.12 CPU DOES NOT PROVIDE READ-IN FUNCTION. THE FRONT END MUST PROVIDE

14.13 A SPECIAL PRIORITY LEVEL OF 0 HAS BEEN INTRODUCED FOR THE DTE20. IT HAS HIGHER PRIORITY THAN 1-7, IS NOT DISABLED WHEN THE PI SYSTEM IS TURNED OFF, AND WORKS EVEN IF THE KL10 IS IN A HALT LOOP. IT IS USED FOR EXAMINES, DEPOSITS AND BYTE TRANSFERS.

14.14 PROGRAM INTERRUPT DEVICE PRIORITY IS DETERMINED BY DEVICE NUMBER AT A GIVEN PI LEVEL. THE ORDER IN DECREASING PRIORITY IS: MBC0, ... , MBC7, DTE0, ..., DTE3, DIA20, (KI I/O BUS), KA, PDP-6 DEVICES.

14.15 IF KL HALTS AND THEN CONTINUES, AN AOSA INSTRUCTION IN AN INTERRUPT LOCATION, WILL LOSE ONE COUNT IF INTERRUPT REQUEST HAPPENS AFTER MACHINE HALTS.

15. ADDRESSING DIFFERENCES

*15.1 EXTENDED ADDRESSING HAS BEEN ADDED SO THAT A PROCESS CAN ADDRESS 8 MILLION WORDS IN USER MODE AND 8 MILLION WORDS IN EXEC MODE. THIS IS DONE BY CREATING 32 ADDRESS SPACE SECTIONS, EACH 256K LONG FOR EACH MODE. SEE CHAPTER 2,2 USER INTERFACE TO EXTENDED ADDRESSING. FUTURE MACHINES MAY HAVE UP TO 30-BIT ADDRESSES TO GIVE 1 BILLION WORDS.

15,2 SHADOW MODE AC REFERENCES ARE NOT PERMITTED. INSTEAD, THE MONITOR WILL USE PXCT WITH 2 BLOCKS OF ACS. FOR MONITORS WHICH RESCHEDULE IN EXEC MODE, THIS WILL REQUIRE 2 AC BLOCK SAVES AND RESTORES ON EACH CONTEXT SWITCH INSTEAD OF 1. THE KI AC STACK POINTER IS NOT NEEDED ON THE KL AND IS NOT PROVIDED. RECURSION IS HANDLED BY BLTING AC BLOCKS. THIS WAS DONE SO THAT EFFECTIVE ADDRESS CALCULATION IN CALLING CONTEXT WOULD NEVER NEED TO REFERENCE MEMORY FOR INDEXING.

15,3 THERE WILL BE 8 AC BLOCKS (0 - 7). AC BLOCK 7 IS RESERVED FOR THE MICRO-CODE. TRAPS TO EXEC MODE DO NOT CHANGE THE AC BLOCK #. THUS BLOCKS 0-6 CAN BE USED BY THE SOFTWARE.

15,4 NEW REGISTERS

* PCS (PREVIOUS CONTEXT SECTION) - 5 BITS
PAC (PREVIOUS AC #) - 3 BITS
CAC (CURRENT AC #) - 3 BITS

16. PAGING DIFFERENCES

16.1 THERE WILL BE TWO TYPES OF PAGING PROVIDED ON THE MACHINE, KL-PAGING AND KI-PAGING. KI-PAGING IS VERY SIMILAR TO THE PAGING ON THE KI CPU. THE TWO TYPES OF PAGING WILL BE SELECTED BY LOADING THE APPROPRIATE VERSION OF THE MICRO-CODE. A STATUS BIT, "KI PAGING", WILL BE ON A 1 IF KI PAGING AND 0 IF KL PAGING. SEE CHAPTER 2.8, PAGING. KL-PAGING INCLUDES:

A. SHARED POINTERS

B. INDIRECT POINTERS

C. REFERENCE STRING HARDWARE SUPPORT

D. CORE STATUS TABLE

E. ALL OF EXEC VIRTUAL CORE IS MAPPED.

16.2 PAGE FAIL TRAPS WILL STORE THE PAGE FAIL WORD AND PC IN THE UPT AND WILL LOAD PC FROM UPT INSTEAD OF EXECUTING AN INSTRUCTION IN THE UPT. THIS MAKES IT EASIER FOR A MONITOR CALL TO TAKE A PAGE TRAP. LOCATIONS 420, 426, AND 427 ARE CHANGED ACCORDINGLY.

16.3 INSTEAD OF A SET OF SEPARATE PAGE FAIL LOCATIONS DEPENDING ON EXEC/USER MODE, THERE WILL BE ONLY ONE. THERE IS NO NEED FOR TWO.

16.4 MULTI-PROCESSING SOFTWARE MUST TURN OFF THE CACHE (C) BIT FOR SHARED DATA PAGES. THE CACHE BIT IS THE LAST SPARE BIT IN THE 18-BIT PAGE MAP ENTRY.

16.5 IN BOTH KI AND KL PAGING, ALL OF THE EXEC ADDRESSING SPACE IS MAPPED.

16.6 THERE IS NO SMALL USER BIT. MOST PROGRAMS WILL USE PAGES AT TOP OF VIRTUAL CORE.

16.7 AS A PIECE OF PHILOSOPHY, KL PAGE MAPS ARE ASSOCIATED WITH SECTIONS (DOMAIN INCARNATIONS) RATHER THAN PROCESSES. EACH PROCESS (MAP) HAS A LIST OF 32 POINTERS TO THE USER SECTION MAPS AND 16 POINTERS TO THE EXEC PER PROCESS MAP. THE EXEC MAP HAS 16 EXEC PER CPU POINTERS. EACH SECTION MAP IS A FULL PAGE WITH FULL WORD ENTRIES.

16.8 THE PAGE FAIL WORD SPECIFIES THE ENTIRE 23 BIT VIRTUAL ADDRESS INCLUDING THE SECTION NUMBER, NOT JUST VIRTUAL PAGE. THE FAIL CODE HAS MOVED TO BITS 0-5.

16.9 A NUMBER OF NEW LOCATIONS HAVE BEEN ADDED TO THE EPT. SEE CHAPTER 2.9, EXEC AND USER PROCESS TABLES (EPT, UPT). THEY ARE:

1. 0-37 RH20 CHANNEL LOGOUT AREA

2. 60-63 CHANNEL FILL WORDS

3. 64-137 RESRVED FOR FUTURE DEC HARDWARE

4. 140-177 DTE20 AREA

5. 510-514 SYSTEM CLOCK AND TIMER LOCATIONS

6. 600-757 EXEC MAP FOR PAGES 0-337 IS KI MODE.

16,10 A NUMBER OF NEW LOCATIONS HAVE BEEN ADDED TO THE UPT, SEE CHAPTER 2,9, EXEC AND USER PROCESS TABLES (EPT, UPT). THEY ARE:

* 1. 440-477 USER SECTION TABLE

2. 426 PROCESS CONTEXT WORD

3. 504-507 PER PROCESS ACCOUNTING METER LOCATION

16,11 A NEW INSTRUCTION (INTERNAL DEVICE CODE) WILL ALLOW THE OPERATING SYSTEM TO INVALIDATE A SPECIFIC ENTRY IN THE PAGING MEMORY. THUS THE MONITOR CAN CHANGE A SINGLE ENTRY IN A PAGE MAP WITHOUT CLEARING ALL OF THE PAGING MEMORY AND SUFFERING THE MICRO-CODE OVERHEAD OF REFILLING IT AGAIN.

17. MISCELLANEOUS DIFFERENCES

17,1 THERE ARE NO LIGHTS AND SWITCHES. AN ADDRESS COMPARATOR REGISTER CAN BE LOADED FROM THE 10 (DATAO APR) FOR ADDRESS BREAK. NO ADDRESS STOP IS POSSIBLE.

17,2 THE -11 CAN EXAMINE AND DEPOSIT DIRECTLY INTO THE -10 MEMORY BY INTERRUPTING THE -10 MICRO CODE. THE -11 CANNOT GET THE -10 PC, EXCEPT IN MAINTENANCE MODE. THE -11 CAN READ THE STATE OF THE PI SYSTEM (LEVELS IN PROGRESS, PI ON) AS WELL AS CPU RUN FLOP (NOT HALTED) AND EBOX CLOCK RUNNING WHILE THE SYSTEM RUNS OVER THE DS LINES.

17,3 POWER FAIL CONTINUE REQUIRES MORE SOFTWARE THAN ON THE KI. CACHE MUST BE DUMPED (700 MICROSECS MAX 4-WAY INTERLEAVE, MORE IF LESS INTERLEAVE) AND MICROCODE RELOADED.

17,4 POWER FAIL RELOAD AND RESTART IS NOT IN KL. PDP-11 POWER FAIL RESTART OPTION MUST BE USED. DOES IT HAVE RESTART OPTION? NOTE: POWER FAIL RELOAD IS MUCH EASIER IN HARDWARE THAN COMPLETE CONTINUE FROM FAILURE HALT.

17,5 MICRO-CODE MUST BE LOADED FROM -11 BEFORE SYSTEM CAN BE STARTED. THE TU56 DECTAPE WILL BE USED FOR THIS.

17,6 NO MI REGISTER, SENSE SWITCHES. PDP-11 CAN SIMULATE.

17,7 NO VOLTAGE MARGINS.

17,8 COMPREHENSIVE DIAGNOSTIC MODE SO MACHINE CAN BE DIAGNOSED FROM THE -11.

17,9 UP TO 4 KL PROCESSORS CAN BE ATTACHED TOGETHER IN A TIGHTLY COUPLED SYSTEM. INSTEAD OF 2 (KI MEMORY LIMIT WAS 2).

17,10 THE SYSTEM HAS TWO KINDS OF MEMORY: INTERNAL AND EXTERNAL. INTERNAL IS FASTER AND CHEAPER THAN EXTERNAL. INTERNAL MEMORY HAS ONLY ONE PORT. THE CACHE MULTIPLEXES ITS CPU AND MASSBUS CONTROLLERS INTO THIS SINGLE PORT. DF10 CHANNELS CAN ONLY TALK TO EXTERNAL MEMORY. INTERNAL MA20 MEMORIES ARE 16K SENSE. IN FUTURE INTERNAL MB20 WILL BE 32K SENSE. A MAXIMUM OF 256K OF MA20 OR 512K OF MB20 IS POSSIBLE. THE REST OF MEMORY UP TO 4M WORDS MUST BE EXTERNAL. A TIGHTLY COUPLED, SHARED MONITOR, MULTI-PROCESSING SYSTEM MUST HAVE EXTERNAL MEMORY FOR SHARED PAGES. FOR BEST PERFORMANCE SYSTEMS SHOULD BE 4-WAY INTERLEAVED. INSTEAD OF REQUIRING OPERATOR INTERVENTION, THE SOFTWARE CAN CHANGE THE INTERNAL MEMORY INTERLEAVING AND ADDRESSES. NO HAND SWITCHES ARE ENVISIONED. [NEED SPECIFICATIONS]. ALSO, THE SOFTWARE MUST BE PREPARED TO RUN WITH NON-CONTIGUOUS INTERNAL MEMORY, HENCE THE FULL MAPPING OF EXEC CORE.

17,11 PRESENT EXTERNAL MEMORY ARE ME10 AND MF10 MEMORY WITH RECONFIGURATION SWITCHES. EXTERNAL MEMORY MUST BE RECONFIGURED BY THE OPERATOR USING MEMORY ADDRESS SWITCHES.

17,12 THE INTERNAL MEMORY WILL HAVE ADDRESS PARITY AS WELL AS DATA PARITY.

17,13 THE FOLLOWING OTHER PARITY CHECKS ARE ALSO PROVIDED:

1. FAST MEMORY (ACS)
2. DTE20 TO EBOX (BYTE TRANSFER DATA, DEPOSIT DATA)
3. EBOX TO DTE20 (BYTE TRANSFER DATA, EXAMINE DATA)
4. CACHE DATA RAMS
5. CACHE ADDRESS RAMS
6. PAGING RAMS
7. CHANNEL TO RH20
8. RH20 TO MASSBUS DEVICES
9. UNIBUS (DTE20 ONLY SO FAR)

17,14 UNLIKE KA, KI, NXM CAN CAUSE PARITY TRAP (PF CODE 36 OR 37).

18. ACCEPTED SIMILARITIES

18.1 THE KL WILL DO THE SAME AS KI WITH INSTRUCTIONS IN AN INTERRUPT OR A TRAP LOCATION, EVEN THOUGH THIS MAY COST SOMETHING.

18.2 JSYS WILL TRAP LIKE ANY OTHER ILLEGAL INSTRUCTION. JSYS WILL BE USED TO CALL VIROS.

18.3 THERE WILL BE NO PC OVERFLOW TRAP WHEN A PROGRAM RUNS OFF THE END OF A SECTION. THE PC STAYS IN THE SAME SECTION AND WRAPS AROUND TO THE ACS.

18.4 DECTAPES WILL BE AVAILABLE ON THE -10 WITH THE 10 I/O BUS ADAPTER (DIA20) OR ON THE -11. MOST SYSTEMS WILL NOT HAVE DECTAPES. THE SOFTWARE IS NOT CURRENTLY PLANNING TO SUPPORT THE DECTAPE DRIVES ON THE -11 DURING TIME SHARING.

18.7 MAGTAPE FORMATTER (TU16) PROVIDES INDUSTRY COMPATIBLE -10 CORE DUMP, AND TRACK FORMAT FOR NRZI AND PE TAPES, AND 9 CHANNEL.

18.8 SERIAL NUMBER WILL START AT 1025 (DECIMAL).

18.9 PROPRIETARY VIOLATION WILL CONTINUE TO TRAP ON NEXT INSTRUCTION FETCH AND IS THEREFORE A NON-RECOVERABLE ERROR.

19. REJECTED DIFFERENCES

19,1 INTERRUPTS WILL NOT STORE PC 40+2*N AND LOAD PC FROM C(41+2*N).

19,2 TRAP ON UNUSED FIELDS OF INSTRUCTIONS WILL NOT BE DONE BECAUSE OF HIGH COST OF INVALIDATING EXISTING PROGRAMS. (ALL UNUSED FIELDS OF NEW INSTRUCTIONS WILL TRAP, SO NEW FUNCTIONS CAN BE ADDED.)

POPJ BITS 18-35
AC FIELD OF XCT
AC FIELD OF JSR
BITS 0-12 INDIRECT ADDRESSES (BECAUSE OF SOFTWARE
MANUAL FLAGS

19,3 TU16 WILL HAVE ADDRESS PLUG FOLLOWING DRIVE SELECTION STANDARD. SINCE MASTER DRIVE (FORMATTER) IS THE ONLY MASSBUS DEVICE AND HAS A FIXED DEVICE NUMBER WITH RESPECT TO THE CONTROL, PULLING ADDRESS PLUGS CAN NEVER CONFLICT WITH OTHER MASSBUS DEVICES LIKE DISK.

19,4 EXEC AND USER SECTIONS ARE NOT INDICATED IN AN ADDRESS WITH A BIT SO THAT EXTENDED ADDRESSES WOULD BE INDEPENDENT OF THE EXEC/USER MODE OF THE CPU.

19,5 DO NOT DO EXTENDED ADDRESSING BY HAVING 4 USER LOADABLE SECTION NUMBER REGISTERS SPECIFIED BY BITS 18 AND 19 OF ADDRESS. THIS WOULD BE MORE EFFICIENT THAN SXCT FOR LOOPS [T. HASTINGS MEMO].

19,6 DO NOT IMPLEMENT FULL SEGMENTATION USING SYSTEM WIDE SEGMENT NUMBERS [M. SPIER MEMO].

19,7 JSYS WILL CONTINUE AS AN ILLEGAL INSTRUCTION.

19,8 JUMP TRACE REGISTER WILL NOT BE INCLUDED.

19,9 BYTE POINTERS WILL NOT BE ABLE TO CROSS WORD BOUNDARIES (WHEN BIT 12 = 1).

19,10 BLT BACKWARD WILL NOT BE DONE (IF E IS LESS THAN DESTINATION ADDRESS), SINCE IT BREAKS TOO MANY PROGRAMS.

APPENDIX A - PROPOSED DIFFERENCES NEEDING A SPEC BEFORE APPROVAL

A,1 PORTAL INSTRUCTION MUST BE IN A READ ONLY PAGE.

A,2 DRUM SPLIT-LIKE SOLUTIONS TO DRUM OVERRUN PROBLEM.

APPENDIX B - REVIEWED DIFFERENCES WHICH WILL BE INCLUDED ONLY IF ROOM IN MICRO-CODE AND A SPEC APPROVAL.

B,1 CAL, RET, VAL, PUSHM FOR SUBROUTINE CALLING.

B,2 SPEEDING UP STRING TO INCLUDE A WAY TO DO A BYTE BLOCK TRANSFER WHICH ONLY REFERENCES EACH WORD ONCE.

[END OF CH1S03,SPC]

1080,2040,2060 ENGINEERING FUNCTIONAL SPEC CHAP 1,5

TO: KL10 LIST

TITLE: SYSTEM ERROR HANDLING AND MAINTAINABILITY - REV 0

STATUS: THIS CHAPTER WILL SERVE AS A BASIS FOR DESIGNING AND REVIEWING EACH OF THE ERROR DETECTION CAPABILITIES BEING ADDED TO THE MACHINE. THIS WILL BE REVIEWED ON WED., 13 MAR, AT 1:00 P.M. AT THE KL10 STEERING COMMITTEE.

FILE: [EFS]CH1S05.SPC

PDM #: 200-200-009-00

DATE: 5 MAR 74

SUPERSEDED MEMOS: NONE

ENGINEER: T. HASTINGS

APPROVED: T. HASTINGS

EDITOR: T. HASTINGS

TYPIST: L. NOWELL

REVIEWED:

ABSTRACT

THIS CHAPTER DESCRIBES THE VARIOUS STRATEGIES FOR ERROR DETECTION, CORRECTION, REPORTING, CORRECTIVE AND PREVENTATIVE MAINTENANCE. IT SERVES AS A FRAMEWORK FOR THE MAINTAINABILITY SECTION OF EACH CHAPTER DESCRIBING A FUNCTION COMPONENT.

REVISION HISTORY

REV	DESCRIPTION	CHG NO	ORIG	DATE	APPD BY	DATE

1. INTRODUCTION

THIS CHAPTER ADDRESSES THE GOALS AND STRATEGIES FOR MAINTENANCE OF THE SYSTEM. BECAUSE THIS IS A SYSTEM QUESTION, IT WILL INCORPORATE HARDWARE, MONITOR SOFTWARE AND DIAGNOSTIC CONSIDERATIONS. WE MUST SET THESE GOALS AND STRATEGIES JOINTLY IN ORDER TO SUCCEED. THIS CHAPTER SETS OUT A FRAMEWORK FOR THE DISCUSSION OF ERRORS. THIS IS INCORPORATED INTO THE MAINTAINABILITY SECTION OF EACH OF THE OTHER CHAPTERS WHICH DESCRIBE FUCTIONAL COMPONENTS. THUS ALL INFORMATION ABOUT A FUNCTIONAL COMPONENT IS KEPT IN A SINGLE PLACE. THE FRAMEWORK INCLUDES CLASSIFICATION OF ERRORS ALONG THE FOLLOWING DIMENSIONS:

1. DETECTION/CORRECTION
2. SEVERITY
3. RECOVERY ACTIONS
4. ERROR INFORMATION REQUIRED
5. CORRECTIVE MAINTENANCE STRATEGY
6. PREVENTATIVE MAINTENANCE STRATEGY

2. DETECTION/CORRECTION CLASSIFICATION OF ERRORS

EACH ERROR FALLS INTO ONE OF THE FOLLOWING CATEGORIES ACCORDING TO HOW IT IS DETECTED AND CORRECTED.

2.1 NOT DETECTED - BY HARDWARE OR SOFTWARE

EX. DROPPED BIT IN ADDER

2.2 DETECTABLE - BUT NO CORRECTION OR RECOVERY POSSIBLE

2.3 DETECTABLE AND USUALLY CORRECTABLE:

1. HARDWARE CORRECTABLE

   EX. NONE CURRENTLY

2. HARDWARE AND SOFTWARE CORRECT

   EX. - ECC ON RP04

3. HARDWARE DETECTS, SOFTWARE RETRIES

   EX. - DATA PARITY ON DISK

4. SOFTWARE DETECTS AND RETRIES

EX. - COMMUNICATIONS PARITY ERROR

2,4 DETECTABLE AND ALWAYS CORRECTABLE

EX. MEMORY ADDRESS PARITY ERROR ON A CONTROLLER READ

3. SEVERITY CLASSIFICATION OF ERRORS

EACH ERROR FALLS INTO ONE OF THE FOLLOWING CATEGORIES DEPENDING ON THE SEVERITY OF THE ERROR.

3,1 SYSTEM IS DOWN IF COMPONENT IS DOWN

EX. RH20 IF ONLY ONE

3,2 ALL JOBS ARE ABORTED:

1. SYSTEM MUST BE RELOADED

EX. DATA PARITY IN EXEC DATA PAGE

3,3 SOME JOBS ABORTED:

1. CPU MUST BE RESTARTED BY -11

EX. CRAM PARITY ERROR IN USER MODE

2. CPU CONTINUES TO RUN, MONITOR ABORTS THE JOB

EX. MEMORY PARITY ERROR IN USER JOB

3,4 I/O TRANSFER ABORTED

EX. RECOVERY FAILED ON DEVICE PARITY ERROR

3,5 RECOVERED ERROR

EX. RETRY SUCCEEDED ON DEVICE PARITY ERROR

4. RECOVERY ACTIONS

THE RECOVERY ACTION MUST BE SPECIFIED FOR EACH TYPE OF ERROR. THE RECOVERY ACTION CAN BE:

4.1 RETRY THE OPERATION WITH NO CHANGE

EX. DEVICE DATA PARITY ERROR

4.2 RETRY THE OPERATION IN A DIFFERENT PLACE

EX. HEAD OFFSET

4.3 RETRY THE OPERATION DIFFERENTLY TO GET DIAGNOSTIC INFORMATION

EX. STOP ON ERROR

4.4 REMOVE FAILED COMPONENT OR BAD REGION OF COMPONENT FROM FUTURE USE:

EX. BAD DISK BLOCK

1. AUTOMATICALLY BY SOFTWARE
2. MANUALLY BY OPERATOR

EX. MEMORY BANK

4.5 REPORT ERROR TO:

1. USER
2. OPERATOR
3. FIELD SERVICE

5. ERROR INFORMATION REQUIRED

FOR EACH ERROR, THE FOLLOWING QUESTIONS MUST BE ANSWERED ABOUT THE INFORMATION REPORTED TO THE FOUR AUDIENCES:

5.1 FIELD SERVICE:

1. IDENTIFY FUNCTIONAL COMPONENT

   EX. WHICH RP04

2. PERHAPS IDENTIFY PART OF COMPONENT SO CAN REPLACE OR REPAIR BOARD

   EX. WHICH BIT ON A PARITY ERROR

5.2 OPERATOR

1. WHICH FUNCTIONAL COMPONENT

5.3 USER

1. WHICH FILE, JOB OR PROGRAM

5.4 SOFTWARE:

1. FUNCTIONAL COMPONENT
2. REGION OF FUNCTIONAL COMPONENT

6. CORRECTIVE MAINTENANCE STRATEGY

FOR EACH OF THE ERRORS IN EACH OF THE FUNCTIONAL COMPONENTS, WE MUST SPECIFY ONE OR MORE OF THE FOLLOWING CORRECTIVE MAINTENANCE STRATEGIES:

6.1 WHILE SYSTEM RUNS WITHOUT SYSTEM RELOAD

1. OFF-LINE TEST
2. OFF-LINE TIME-SHARING DIAGNOSTICS
3. MONITOR ERROR REPORTING

6.2 WHILE SYSTEM RUNS BUT SYSTEM MUST BE RELOADED

1. OFF-LINE TEST
2. ON-LINE TIME-SHARING DIAGNOSTICS

6.3 SYSTEM NEEDED STAND-ALONE

1. LOCAL TESTING USING - 10
2. LOCAL TESTING USING - 11
3. REMOTE TESTING USING EXPERT IN MARLBORO

6.4 STOCKING OF SPARES

1. ON-SITE
2. IN REGION HEADQUARTERS

7. PREVENTATIVE MAINTENANCE

FOR EACH ERROR IN EACH FUNCTIONAL COMPONENT, WE MUST SPECIFY ONE OR MORE OF THE FOLLOWING PREVENTATIVE MAINTENANCE STRATEGIES.

7.1 WHILE SYSTEM RUNS WITHOUT SYSTEM RELOAD

1. OFF-LINE TEST

2. ON-LINE TIME SHARING DIAGNOSTICS

7.2 WHILE SYSTEM RUNS, BUT SYSTEM MUST BE RELOADED

1. OFF-LINE TEST

2. ON-LINE TIME-SHARING DIAGNOSTIC

7.3 SYSTEM NEEDED STAND-ALONE

[END OF CH1S05.SPC]

1080,2040,2060 ENGINEERING FUNCTIONAL SPEC - CHAP 1,6

TO: KL10 LIST, J. PARSLOW (200 FILE)

TITLE: KL10 ERROR DETECTION, RECOVERY AND REPORTING

STATUS: REVIEWED AND APPROVED 3 FEB 75

FILE: [EFS]CH1S06,SPC

PDM #: 200-200-025-01

DATE: 18 APR 75

SUPERSEDED MEMOS: NONE

SUPERSEDED SPECS:

ENGINEER: T. HASTINGS, P. GUGLIELMI, R. REID, R. DRUEKE

APPROVED:

EDITOR: T. HASTINGS, R. DRUEKE

TYPIST: J. MCCARTHY

REVIEWED: 3 FEB 75

ABSTRACT

THIS CHAPTER DESCRIBES THE SYSTEM ERRORS DETECTED BY THE KL10 CPU HARDWARE AND DESCRIBES HOW THE OPERATING SYSTEM SOFTWARE SHOULD ATTEMPT TO RECOVER AND WHAT INFORMATION SHOULD BE PRESERVED FOR THE ERROR RECORDING SYSTEM. THE APPENDIX DESCRIBES HOW HARDWARE FAULT INSERTION CAN BE PERFORMED BY PRIVELEGED USER PROGRAMS RUNNING UNDER TIMESHARING FOR MOST ERRORS. SEE ATTACHED BLOCK DIAGRAM FOR WHERE PARITY IS GENERATED AND CHECKED.

REVISION HISTORY

REV	DESCRIPTION	CHG NO	ORIG	DATE	APPD BY	DATE
0	ORIGINAL			20 JAN 74	1	ADDED APPENDIX

BECAUSE OF MACHINE-READAABLE LIMITATIONS, SOME OF THE TABLES AND FIGURES REFERENCED IN THIS CHAPTER WILL BE FOUND IN A SEPARATELY PUBLISHED ADDENDUM, THESE INCLUDE:

1. DIFFERENT TYPES OF ERRORS DETECTED BY THE KL10

TABLE 1 SUMMARIZES THE DIFFERENT TYPES OF ERRORS THAT ARE DETECTED BY THE KL10 HARDWARE. INCLUDED ARE THE ERRORS DETECTED BY THE INTERNAL CONTROLLER, THE RH20.

FIGURE 1 IS A DIAGRAM OF THE DATA PATH PARITY NETWORKS IN THE KL10 AND IS DESCRIBED IN 1.1.

FIGURE 2 IS A DIAGRAM OF THE APR ERROR STATUS REGISTER. IT IS DESCRIBED IN 1.2.

FIGURE 3 IS A DIAGRAM OF THE ERA REGISTER AND IS DESCRIBED IN 1.3.

FIGURE 4 IS A DIAGRAM OF THE SBUS DIAGNOSTIC CYCLES AND IS DESCRIBED IN 1.4.

TABLE 1

ERROR SUMMARY TABLE

SBUS		
	MA20:	INCOMPLETE CYCLE
		ADDRESS PARITY ERROR
	DMA20:	DATA PARITY ERROR
		ADDRESS PARITY ERROR
		NON-EXISTENT MEMORY (NXM) EROR
MBOX		
		SBUS ERROR
		SBUS ADDRESS PARITY ERROR
		DATA PARITY ERROR
		PAGE TABLE PARITY ERROR
		CACHE DIRECTORY PARITY ERROR
		NON-EXISTENT MEMORY (NXM) ERROR
EBOX		
		AR OR ARX PARITY ERROR
		EBUS PARITY ERROR
		FAST MEMORY PARITY ERROR
		D RAM PARITY ERROR
		C RAM PARITY ERROR
RH20		
		DATA PARITY ERROR

1.1 KL10 DATA PATH PARITY NETWORKS

PARITY IS GENERATED IN VERY FEW PLACES IN THE SYSTEM. THE GENERAL PHILOSOPHY IS TO HAVE PARITY GENERATED ONCE AND THEN TO HAVE IT PASSED ALONG FROM COMPONENT TO COMPONENT AND CHECKED AS IT GOES. IN THIS WAY, THE CHANCE THAT AN ERROR GOES UNDETECTED IS MINIMIZED. EFFECTIVELY THERE ARE ONLY TWO PARITY GENERATION NETWORKS (PG) IN THE SYSTEM. ONE IS ON THE OUTPUT SIDE OF THE ARITHMETIC REGISTER, AR (FIGURE 1), AND THE OTHER IS ON THE OUTPUT OF THE CHANNEL STATUS RAMS. WHENEVER DATA LEAVES THE ARITHMETIC REGISTER TO GO TO THE CACHE, TO FAST MEMORY, TO THE EBUS, OR TO THE MEMORY BUFFER, A PARITY BIT IS GENERATED AND THAT PARITY BIT REMAINS WITH THE DATA IN THE CACHE, IN FAST MEMORY, TO THE EBUS, OR THE MEMORY BUFFER.

THERE ARE INSTANCES WHERE THE DATA PATHS DECREASE FROM 36 BITS TO 18 BITS, OR EXPAND FROM 18 BITS TO 36 BITS. IN THESE CASES, A PARITY FOLDING GENERATION OCCURS. THIS HAPPENS ON THE INPUT SIDE OF THE PAGE TABLE AS DATA PASSES FROM THE MEMORY BUFFER INTO THE PAGE TABLE AND A CONVERSION IS MADE FROM 36 BITS TO 18 BITS (AND A PARITY BIT GENERATED TO CHECK EACH 18 BITS). THE SAME THING HAPPENS AS DATA PASSES FROM THE RH20 TO THE CHANNEL. THE RH20 DEALS WITH 18 BIT ENTITIES AND THE CHANNEL DEALS WITH 36.

THERE ARE SIX PARITY CHECKING NETWORKS IN THE MAJOR DATA PATH OF THE SYSTEM. THE FIRST PARITY CHECKING NETWORK IS ON THE OUTBOARD SIDE OF THE DMA20. THUS PARITY IS CHECKED ON DATA BEING RECEIVED FROM THE MF10 AND ENTERING THE DMA20, AND PARITY IS CHECKED AS DATA LEAVES THE DMA20 AND STARTS DOWN THE KBUS TO THE MF10. THE NEXT PARITY CHECKING NETWORK IS LOCATED AT THE OUTPUT OF THE MEMORY BUFFER (MB) REGISTERS. WHENEVER DATA LEAVES A MEMORY BUFFER, FOR THE DMA20, HARDWARE PAGE TABLE, CHANNEL, CACHE, ARITHMETIC REGISTER (AR), OR THE ARITHMETIC REGISTER EXTENDER (ARX), ITS PARITY IS CHECKED.

THE THIRD PARITY CHECKING NETWORK IN THE SYSTEM IS LOCATED ON THE OUTPUT SIDE OF THE PAGE TABLE. THUS PARITY IS CHECKED WHENEVER DATA COMES OUT OF THE HARDWARE PAGE TABLE.

THE FOURTH PARITY CHECKING NETWORK IS ON THE OUTBOARD SIDE OF THE RH20. THIS PARITY NETWORK CHECKS DATA PARITY COMING FROM THE ROTATING DEVICE INTO THE RH20 FROM THE MASSBUS AND CHECKS PARITY ON DATA LEAVING THE RH20 AND BEING PUT ON THE MASSBUS TO GO OUT TO THE ROTATING DEVICE.

THE FIFTH PARITY CHECKING NETWORK IS ON THE ARITHMETIC REGISTER. IT CHECKS DATA ARRIVING AT THE ARITHMETIC REGISTER FROM MEMORY.

THE LAST PARITY CHECKING NETWORK OF THE MAJOR DATA PATH IS ON THE ARITHMETIC EXTENSION REGISTER. IT CHECKS DATA ARRIVING AT THE ARX FROM MEMORY.

ADDITIONALLY, THERE ARE THREE OTHER PARITY NETWORKS IN THE SYSTEM. THE FIRST IS ON THE OUTPUT OF FAST MEMORY. IT CHECKS STORED PARITY AND DATA COMING FROM FAST MEMORY.

THE OTHER TWO ARE ON THE OUTPUT OF THE CRAM AND THE OUTPUT OF THE DRAM. HERE, THE PARITY IS CHECKED ON EACH WORD BEFORE IT IS USED BY THE MICRO WORD CONTROLS.

1,2 APR ERROR STATUS REGISTER - AS DESCRIBED IN CHAPTER 2,6, INTERNAL DEVICES, THE APR CONO AND CONI OPERATIONS ALLOW THE ENABLING, DISABLING, SETTING, AND CLEARING OF EIGHT ERROR CONDITIONS. THESE ERRROR CONDITIONS ARE SBUS ERROR, NON-EXISTENT MEMORY ERROR, PARITY ERROR, I/O PAGE FAIL ERROR, CACHE DIRECTORY PARITY ERROR, SBUS ADDRESS PARITY ERROR, POWER FAILURE, AND CACHE SWEEP DONE. FOUR BITS IN THE CONO WORD ARE USED TO INDICATE WHETHER OR NOT TO ENABLE THE SELECTED FLAGS, DISABLE THE SELECTED FLAGS, CLEAR THE SELECTED FLAGS, OR SET THE SELECTED FLAGS. THESE ARE BITS 20 - 23, RESPECTIVELY. BITS 24 - 31 ARE USED FOR THE SELECTED FLAGS. BIT 32 IS A 1 TO INDICATE THAT ONE OF THE SELECTED FLAGS IS TRUE. BITS 33 - 35 ARE THE PI LEVEL FOR THE APR DEVICE. BITS 6 - 13 OF THE CONI WORD INDICATE WHICH OF THE SELECTED FLAGS ARE ENABLED. BIT 19 IS A 1 ON A CONI IF THE CACHE SWEEPER IS BUSY. BIT 19, ON A 1 ON THE CONO, IS THE I/O RESET AND RESETS ALL THE I/O IN THE SYSTEM. THE SOFTWARE IS ABLE TO TEST FOR MANY OF THESE MEMORY ERRORS BY ARTIFICALLY INSERTING A REAL ERROR IN THE SYSTEM AND THEN OBSERVING WHAT HAPPENS WHEN THE SYSTEM REFERENCES THE MEMORY. CHAPTER 2,6 SHOWS HOW THE SOFTWARE CAN WRITE INCORRECT PARITY USING CONO PI ON BITS 18, 19, AND 20.

1.3 ERROR ADDRESS REGISTER - ERA

THE MBOX HAS AN INTERNAL REGISTER CALLED THE ERROR ADDRESS REGISTER, OR ERA (FIGURE 3). THIS REGISTER CONTAINS THE LAST ADDRESS REQUESTED FROM THE MEMORY SYSTEM ALONG WITH ERROR FLAGS INDICATING WHAT KIND OF OPERATION (READ, WRITE, ETC.) HAD BEEN REQUESTED OF THE MEMORY. THE ERA REGISTER LATCHES ON AN ERROR CONDITION SIGNALED BY THE MEMORIES AND IS NEVER CHANGED UNTIL THE SOFTWARE IS ABLE TO COPY OUT THE ERROR INFORMATION. THE SOFTWARE THEN INDICATES THAT IT HAS DONE THIS AND THE ERA REGISTER IS FREE TO RECORD THE NEXT ERROR THAT MAY OCCUR. IN OTHER WORDS UNTIL AN ERROR OCCURS, THE ERA REGISTER IS CHANGING WITH EVERY ADDRESS THAT IS BEING SENT BY THE MBOX TO THE MEMORY SYSTEM FOR READING OR WRITING. AS SOON AS AN ERROR OCCURS THE ERA REGISTER IS LATCHED AT THE LAST ADDRESS SENT BY THE MBOX TO THE MEMORY SYSTEM AND REMAINS THAT WAY UNTIL THE SOFTWARE IS ABLE TO COPY OUT THE ERROR ADRESS AND THE OPERATION BEING DONE AND HAS RECORDED IT FOR FIELD SERVICE IN THE SYSTEM ERROR LOG.

THERE ARE THREE ERROR BITS IN THE APR STATUS REGISTER WHICH INDICATE TO THE SOFTWARE (ON AN INTERRUPT ON THE APR) THAT THE ERA REGISTER CONTAINS ERROR INFORMATION. THESE THREE BITS IN THE APR REGISTER ARE: NXM ERR (BIT 25), MB PAR ERR (BIT 27), AND SBUS ADR PAR ERR (BIT 29). THE SOFTWARE THEN READS THE CONTENTS OF THE ERA REGISTER, USING THE APPROPRIATE INTERNAL DEVICE CODE, (BLKI PI). AFTER THE SOFTWARE HAS COPIED OUT THE INFORMATION IN THE ERA REGISTER, THE SOFTWARE MUST CLEAR THE THREE BITS IN THE APR REGISTER IN ORDER TO ALLOW THE ERA REGISTER TO BE READY FOR A SUBSEQUENT ERROR.

1,3,1 LAYOUT OF THE ERA REGISTER - BITS 14 - 35 CONTAIN THE 22-BIT PHYSICAL ADDRESS REQUESTED BY THE EBOX OF THE MBOX. BITS 0 AND 1 OF THE ERA REGISTER INDICATE IN WHICH OF THE FOUR WORDS (THAT THE MBOX REQUESTS OF THE MEMORY SYSTEM) THE ERROR OCCURRED. WHENEVER THE MBOX MAKES A REFERENCE TO MEMORY, IT REQUESTS FOUR WORDS AT A TIME, WITH THE WORD THAT THE EBOX NEEDS BEING REQUESTED FIRST AND THE OTHER THREE FOLLOWING IN QUICK SUCCESSION. BITS 0 AND 1 INDICATE WHICH OF THE FOUR WORDS IN THE 4-WORD BLOCK HAD THE ERROR, WHILE BITS 14 - 35 INDICATE THE 22-BIT PHYSICAL ADDRESS THAT IS NEEDED BY THE EBOX. BIT 2 IS NAMED THE CCA REF BIT. IT IS A 1 IF THE MEMORY ERROR OCCURRED WHILE THE CACHE WAS SWEEPING. BIT 3 IS CALLED THE CHAN REF BIT. IT IS A 1 IF THE ERROR OCCURRED WHEN THE CHANNEL WAS MAKING A REFERENCE TO MEMORY. BITS 4 AND 5 COMPRISE A DATA SOURCE CODE THAT INDICATES WHERE THE DATA IN THE MB'S ORIGINATED. THE DATA MAY COME FROM THE MEMORY ON A READ OR A READ-PAUSE-WRITE; DATA MAY COME FROM THE CHANNEL ON A STORING STATUS, I.E., WRITING INTO MEMORY; DATA MAY COME FROM THE AR ON AN EBOX WRITE; DATA MAY COME FROM THE CACHE ON A PAGE REFILL OR CHANNEL READ OPERATION; OR DATA MAY COME FROM THE CACHE ON A WRITE OPERATION TO MEMORY. BIT 6 IS THE WRITE REFERENCE BIT. IT IS A 1 IF A MEMORY WRITE OPERATION WAS IN PROGRESS WHEN THE ERROR OCCURRED. IT IS A 0 ON THE READ PORTION OF A READ-PAUSE-WRITE. THE DATA SOURCE FIELD IS NOT VALID ON AN ADDRESS PARITY ERROR.

1,4 SBUS DIAGNOSTIC CYCLES

THE SBUS DIAGNOSTIC CYCLE IS USED FOR COMMUNICATION BETWEEN THE MBOX AND THE INTERNAL MEMORY CONTROLLERS, AS THERE ARE NO SWITCHES ASSOCIATED WITH THE INTERNAL MEMORY CONTROLLERS, FUNCTIONS SUCH AS ADDRESS BOUNDARIES, INTERLEAVE MODE, MARGINS ERROR REPORTING, ETC,, ARE CONTROLLED USING THE DIAGNOSTIC CYCLE, A DIAGNOSTIC CYCLE IS EXECUTED BY THE BY THE MBOX ASSERTING THE SBUS DIAG, SIGNAL AND USING THE 36 SBUS DATA LINES FOR TRANSFERRING INFORMATION, THERE WILL BE A PDP-10 MACRO INSTRUCTION WHICH TAKES A 36 BIT INPUT ARGUMENT AND RETURNS A 36 BIT OUTPUT ARGUMENT, THE MICROCODE FOR THIS INSTRUCTION PUTS THE 36 BIT INPUT ARGUMENT IN THE AR AND TELLS THE MBOX TO EXECUTE AN SBUS DIAGNOSTIC CYCLE, WHEN THE MBOX TURNS ON ITS MBOX RESPONSE SIGNAL THE MICROCODE CLOCKS THE CONTENTS OF THE CACHE DATA LINES INTO THE AR AND RETURNS IT AS THE OUTPUT ARGUMENT

TWO FUNCTION CODES, 0 AND 1, HAVE BEEN DEFINED AND ARE FULLY DESCRIBED IN CHAPTER 3,7 OF THIS SPECIFICATION, DESCRIBED HERE IS A SUMMARY OF BOTH FUNCTIONS DURING THE 2ND HALF OF THE CYCLE, CALLED "FROM MEMORY", THESE ARE INTEREST IN THIS CHAPTER BECAUSE BOTH FUNCTIONS PROVIDE SEVERAL OF THE INDICATIONS AND STATUS REQUIRED DURING ERROR RECOVERY,

THE CONTROLLER ADDRESS NEEDED TO SPECIFY WHICH CONTROLLER THE MBOX IS COMMUNICATING WITH IS PUT IN BITS 0 TO 4 OF THE "TO MEMORY" WORDS FOR BOTH FUNCTIONS,

ADDRESSES 0 TO 3 ARE FOR THE INTERNAL CONTROLLERS, MA20 OR MB20, AND ADDRESS 4 IS FOR THE DMA20,

1,4,1 FUNCTION 0, "FROM MEMORY"

BITS 0-5	0-1	EXPANSION
	2	INCOMPLETE CYCLE
	3	READ PAR ERR
	4	WRITE PAR ERR
	5	ADR PAR ERR

THESE ARE ERROR REPORTING FLAGS IN THE MEMORY CONTROLLERS, WHICH ARE CLEARED WHEN BIT 5 IS ON A "ONE" IN THE 1ST HALF OF FIUNCTION 0, THE DMA20 DOES NOT IMPLEMENT THE INCOMPLETE CYCLE BIT AND THE MA20 AND MB20 CONTROLLERS DO NOT IMPLEMENT THE READ OR WRITE PAR ERROR BITS,

BITS 6-7 THESE BITS REPORT THE INTERLEAVE MODE OF THE CONTROLLER WITH THE 00 COMBINATION USED TO INDICATE AN OFF LINE OR NON EXISTENT MEMORY CONTROLLER,

BITS 8-35 THESE BITS CONTAIN THE ADDRESS INFORMATION OF THE MOST RECENT CYCLE OR OF THE 1ST PARITY ERROR, BITS 8-11

CONTAIN THE SBUS RQ0 - RQ3, BITS 12 AND 13 THE RD AND WR RQ AND BITS 14-35 THE SBUS ADR BITS. THIS FEATURE IS NOT IMPLEMENTED IN THE MA20 OR MB20.

1.4.2 FUNCTION 1, "FROM MEMORY"

BITS 0-7 THESE BITS INDICATE THE NUMBER OF STORAGE MODULES PER CONTROLLER. EACH BIT ON A "ONE" INDICATES THE PRESENCE OF A STORAGE MODULE. ALL BITS EQUAL 0 MEANS SIZE UNKNOWN. DMA20 SENDS BACK 0.

MA20 AND MB20 INDICATES STORAGE MODULES AS FOLLOWS.

```
0 1 2 3 4 5 6 7
0 0 0 0 0 0 0 1 SM0
0 0 0 0 0 0 1 0 SM1
0 0 0 0 0 1 0 0 SM2
0 0 0 0 1 0 0 0 SM3
```

BITS 8-11 MEMORY TYPE

0 = CUSTOMER
1 = MA20
2 = DMA20
3 = MB20
4-17(8) EXPANSION

BIT 12 LOOP BACK SET

NOTE: THE FOLLOWING BITS (13-35) ARE NOT IMPLEMENTED IN DMA20.

BIT 13 POWER SUPPLY FAILING

BIT 14-21 ADDRESS BOUNDARY SET UP IN 1ST HALF OF FUNCTION 1

BITS 22-25 ADDRESS BOUNDARY SET UP IN 1ST HALF OF FUNCTION 1

BITS 26-29 EXPANSION

BIT 30 MARGINS SELECTED

BIT 31 EXPANSION

BITS 32-35 INDICATES WHICH WORD RQS ARE ENABLED

32 - SBUS RQ0 ENABLE
33 - SBUS RQ1 ENABLE
34 - SBUS RQ2 ENABLE
35 - SBUS RQ3 ENABLE

A SUMMARY OF BOTH FUNCTIONS IS SHOWN IN FIGURE 4.

2. DETAILED DESCRIPTION OF EROR DETECTION

TABLE 2 CONTAINS A DETAILED DESCRIPTION OF THE VARIOUS ERROR CONDITIONS LISTED IN TABLE 1.

TABLE 2

DESCRIPTION OF ERRORS

PLACE	TYPE OF ERROR	DESCRIPTION OF ERROR
SBUS:		
MA20	INCOMPLETE CYCLE	OCCURS ON A READ-PAUSE-WRITE (RPW) CYCLE WHEN PROCESSOR FAILS TO SEND VALID DATA TO THE MEMORY TO INITIATE THE WRITE PORTION OF THE RPW CYCLE (8.0 USEC TIME-OUT) AND ON ANY CYCLE WHICH THE MEMORY FAILS TO COMPLETE BEFORE THE TIME-OUT. MA20 SIGNALS OCCURANCE OF THIS ERROR BY SENDING SBUS ERROR TO THE PROCESSOR.
	ADDRESS PARITY ERROR	CAN OCCUR AT THE START OF ANY MEMORY CYCLE. PROCESSOR WILL BE NOTIFIED OF THE OCCURRENCE OF AN ADDRESS PARITY ERROR WITHIN 125 USEC AFTER IT SEES THE FIRST ACKNOWLEDGE. MA20 SIGNALS OCCURRENCE OF THIS ERROR BY SENDING SBUS ADDRESS PARITY ERROR TO THE PROCESSOR. THE ADDRESS OF THE LOCATION THE PROCESSOR WAS ATTEMPTING TO REFERENCE WHEN THIS ERROR OCCURED WILL SBE LATCHED IN THE ERA REGISTER IN THE PROCESSOR. AN ADDRESS PARITY ERROR FLAG WILL BE SET IN THE MA20 TO INDICATE THE OCCURRENCE OF THIS ERROR.
DMA20	DATA PARITY ERROR	READ DATA PARITY AND WRITE DATA PARITY ARE BOTH CHECKED IN THE DMA20 DURING THE THREE TYPES OF MEMORY CYCLES. EACH TYPE OF PARITY ERROR SETS APPROPRIATE FLAGS IN THE DMA20 STATUS REGISTER. THE DMA20 ERROR DETECTION LOGIC ALSO STORES THE ADDRESS OF THE FIRST ERROR IN THE DMA20 ERROR ADDRESS

	REGISTER UNTIL THE ERROR BIT ASSOCIATED WITH THAT ERROR IS CLEARED. A CUMULATIVE ERROR SIGNAL (SBUS ERROR) IS SENT TO THE MBOX. ON WRITE CYCLES, THE ERROR SIGNAL MAY NOT ARRIVE AT THE PROCESSOR UNTIL 500 NSEC AFTER THE LAST ACKNOWLEDGE. ON READ CYCLES, THE ERROR SIGNAL MAY NOT ARRIVE AT THE PROCESSOR UNTIL 250 NSEC AFTER THE LAST WORD ARRIVES. DMA20 SIGNAL OCCURRENCE OF THIS ERROR BY SENDING SBUS ERROR TO THE PROCESSOR.
ADDRESS PARITY ERROR	CAN OCCUR AT THE START OF ANY MEMORY CYCLE. PROCESSOR WILL BE NOTIFIED OF THE OCCURRENCE OF AN ADDRESS PARITY ERROR WITHIN 125 USEC AFTER IT SEES THE FIRST ACKNOWLEDGE. DMA20 SIGNALS OCCURRENCE OF THIS ERROR BY SENDING SBUS ADDRESS PARITY ERROR TO THE PROCESSOR. THE ADDRESS OF THE LOCATION THE PROCESSOR WAS ATTEMPTING TO REFERENCE WHEN THIS ERROR OCCURED WILL BE LATCHED IN THE ERA REGISTER IN THE PROCESSOR. AN ADDRESS PARITY ERROR FLAG WILL BE SET IN THE DMA20 TO INDICATE THE OCCURRENCE OF THIS ERROR. THE ADDRESS THE THE DMA20 RECEIVED AT THE TIME OF THE ERROR WILL BE LATCHED IN THE DMA20 ERROR ADDRESS REGISTERS. NOTE: THAT ADDRESS PARITY IS ONLY CHECKED FOR SBUS ADDRESSES AND DOES NOT INCLUDE KBUS ADDRESSES.
NXM ERROR	IF AN ACKNOWLEDGE (ACK) IS RECEIVED FROM THE FIRST STORAGE MODULE, AND ANY SUBSEQUENT REFERENCE IN THE CYCLE DOES NOT RECEIVE AN ACK (I.E., A STORAGE MODULE DID NOT RESPOND), THEN A NON-EXISTENT MEMORY (NXM) FLAG IS SET IN THE DMA STATUS REGISTER AND THE DMA20 EROR ADDRESS REGISTER IS FROZEN IN THE SAME WAY AS AN ADDRESS OR DATA PARITY ERROR. THE TIME

		OUT FOR THIS ERROR IS 28 USEC. THE PROCESSOR WILL BE NOTIFIED IMMEDIATELY AFTER THE TIME OUT BY THE DMA20 SENDING SBUS ERROR.
MBOX	SBUS ERROR	WHEN THE MBOX SEES SBUS ERROR FROM THE MA20 OR DMA20 IT SETS THE SBUS ERROR APR FLAG IN THE EBOX. NO VALID INFORMATION IS HELD IN THE ERA ON OCCURRENCE OF THIS EROR.
	SBUS ADDRESS PARITY ERROR	WHEN THE MBOX SES SBUS ADDRESS PARITY ERROR FROM THE MA20 OR DMA20 THE MBOX SETS THE SBUS ADR PAR ERR FLAG IN APR STATUS REGISTER. ERA REGISTER IN MBOX IS LATCHED WHEN ERROR IS DETECTED AND DOES NOT UNLATCH UNTIL SOFTWARE CLEARS THE APR FLAG AFTER HAVING COPIED THE DATA OUT OF THE ERA.
	DATA PARITY ERROR	THE PARITY OF ALL DATA LEAVING THE MB'S IS CHECKED. IF THE PARITY IS FOUND TO BE INCORRECT, THEN THE ADDRESS OF THE LOCATION REFERENCED, A 3-BIT CODE INDICATING THE SOURCE OF THE DATA, AND TWO BITS INDICATING THE MB THAT HELD THE BAD WORD WILL BE HELD IN THE ERA. THIS INFORMATION WILL BE HELD UNTIL THE MB PARITY ERROR FLAG IN THE EBOX (APR STATUS WORD) IS CLEARED.
	PAGE TABLE PARITY ERROR	THIS TYPE OF ERROR FORCES A PAGE FAIL TRAP WITH A CODE OF 25. THE ERROR COULD HAVE BEEN THE RESULT OF A MEMORY OR CACHE DATA PARITY ERROR ON A PAGE REFILL. IF NOT, THEN THE HARDWARE PAGE TABLE MAY HAVE A FATAL ERROR. IN THIS LAST CASE, IF THE FAILURE IS SOLID THEN THE HARDWARE PAGE TABLE IS UNUSABLE.
	CACHE DIRECTORY PARITY ERROR	FLAG IS SET IN APR REGISTER WHEN A PARITY ERROR IS DETECTED IN CACHE DIRECTORY. THE CACHE IS EFFECTIVELY TURNED OFF AT

		THE END OF THE CURRENT MEMORY REFERENCE. THE CACHE WILL REMAIN "OFF" UNTIL THE CACHE DIR PAR ERR FLAG IN THE APR STATUS REGISTER IS CLEARED. IN LINE WITH THE ORIGINAL PARITY SCHEME GOALS, THIS ERROR IS NON-RECOVERABLE BY THE KL10 SOFTWARE BUT THE 11 IS ABLE TO SCAN THE CACHE IN A DIAGNOSTIC MODE AND TURN OFF THE AFFECTED CACHE BEFORE MONITOR RELOAD IF THE ERROR IS REPRODUCEABLE.
	NXM ERROR	AN ERROR FLAG IS SET IN THE APR REGISTER WHEN A NON-EXISTENT MEMORY LOCATION IS ADDRESSED. ON MBOX READS FROM MEMORY, THE MBOX SUPPLIES FOUR WORDS OF ZEROES TO THE EBOX WITH BAD PARITY, SO EBOX WILL GET A DATA PARITY TRAP. ON MBOX WRITES TO MEMORY, THE DATA IN THE MB'S IS DISCARDED. THE ERA REGISTER IN THE MBOX IS LATCHED WHEN THIS TYPE OF ERROR OCCURS.
EBOX	AR/ARX PARITY ERROR	ALL DATA ENTERING THE AR/ARX TO BE USED BY THE EBOX IS PARITY CHECKED BEFORE IT IS USED. IF A FAILURE IS DETECTED, A PAGE FAIL TRAP OCCURS WITH A CODE OF 26.
	EBUS PARITY ERROR	ALL WORDS ENTERING THE AR/ARX FROM THE EBUS ARE CHECKED FOR CORRECT PARITY IF PARITY HAS BEEN GENERATED BY THE TRANSMITTING DEVICE. IF AN ERROR IS DETECTED, AN I/O PAGE FAIL INTERRUPT OCCURS AND THE MONITOR HAS THE ABILITY TO DETERMINE THAT THE BAD DATA WAS RECEIVED FROM THE EBUS. NOTE THAT AN -11 DEPOSIT TO -10 MEMORY CAN CAUSE A PARITY INTERRUPT WHICH HAS NOTHING TO DO WITH THE -10 INSTRUCTION BEING EXECUTED.
	FAST MEMORY PARITY ERROR	EACH WORD STORED IN FAST MEMORY HAS ODD PARITY GENERATED AS IT LEAVES THE AR. PARITY IS CHECKED IF THE FAST MEMORY WORD

		IS BEING READ OUT THROUGH THE ADB MISER. IF AN ERROR IS DETECTED, THE EBOX CLOCK IS HALTED IF FAST MEMORY PARITY CHECKING IS ENABLED BY THE CONSOLE PROCESSOR (USUAL).
	CRAM & DRAM PARITY ERRORS	ODD PARITY IS GENERATED AND STORED FOR EACH WORD BY THE MICRO-CODE ASSEMBLY PROGRAM AND LOADED INTO THESE RAMS EACH TIME THEY ARE LOADED. THIS ODD PARITY IS CHECKED EACH TIME A WORD IN EITHER RAM IS REFERENCED. IF AN ERROR OCCURS AND THE HALT ON ERROR DIAGNOSTIC FUNCTION IS ENABLED (USUAL), THE EBOX CLOCK IS HALTED.
RH20	DATA PARITY ERROR	OCCURS WHEN RH20 DETECTS BAD PARITY ON A DATA TRANSFER FROM THE DEVICE. IF DISABLE XFER ERR STOP IS SET, THE X-FER WILL CONTINUE AFTER A STATUS FLAG IS SET. IF (NORMAL CASE) THE DISABLE XFER ERR STOP IS NOT SET, THE EXC IS ALSO SET AND AN INTERRUPT IS SENT TO THE EBOX.

3. DETAILED DESCRIPTION OF ERROR RECOVERY AND REPORTING

3.1 DATA PARITY ERRORS

TABLE 4 IS A COMPREHENSIVE CHART OF DATA PARITY ERRORS DETECTED THROUGHOUT THE SYSTEM. THE CHART SHOWS HOW THE ERRORS ARE INDICATED TO THE SOFTWARE, WHAT CONDITIONS CAUSE THESE ERRORS, WHAT PART OF THE SYSTEM IS BROKEN, AND HOW THE SOFTWARE RECOVERS FROM AN ERROR. THE COLUMN INDICATING WHICH PART IS BROKEN IS HELPFUL IN DESIGNING THE ERROR REPORTING SYSTEM SO THAT THE AREA OF FAILURE CAN BE QUICKLY ISOLATED AND REPORTED TO FIELD SERVICE.

3.1.1 DESCRIPTION OF PARITY ERROR TABLE

THE FIRST DATA COLUMN OF TABLE 4 INDICATES WHETHER THE CACHE IS BEING USED BY THE SYSTEM OR NOT: I.E., IS IT ON OR OFF. NOTE: IF C BIT (CACHE) IS OFF IN THE PAGE TABLE ENTRY AND THE DATA IS NOT IN THE CACHE, THE POSSIBILITIES ARE THE SAME AS THE ERRORS WITH CACHE OFF. THE SECOND DATA COLUMN OF THE TABLE INDICATES WHETHER THE OPERATION BEING PERFORMED IS A READ, WRITE, OR READ-PAUSE-WRITE TO THE MEMORY SYSTEM. THE THIRD DATA DOLUMN INDICATES WHETHER THE MEMORY INVOLVED IN THE ERROR IS INTERNAL MEMORY OR EXTERNAL MEMORY. THE FOURTH DATA COLUMN, LABELED WHERE DETECTED, INDICATES WHICH SYSTEM COMPONENT DETECTED THE PARITY ERROR. IN MANY CASES, MORE THAN ONE FUNCTIONAL COMPONENT WITHIN THE SYSTEM WILL DETECT THE ERROR. FUNCTIONAL COMPONENTS LISTED ARE: THE DMA20, THE MA20, THE MEMORY BUFFER REGISTERS (CALLED MB'S), THE ARITHMETIC REGISTER IN THE EBOX (CALLED AR), THE PAGE TABLE DIRECTORY (CALLED PT), AND THE RH20.

THE LAST THREE ENTRIES IN THE WHERE DETECTED COLUMN INDICATES WHICH OF THREE SYSTEM COMPONENTS IS MAKING A MEMORY REFERENCE OR IF IN FACT A MEMORY REFERENCE IS INVOLVED AT ALL. THESE COMPONENTS ARE: THE MBOX, IN RESPONSE TO THE EBOX; THE MB AS PART OF THE CACHE SWEEP OPERATION (CCA REF); AND THE CHANNELS (CHAN REF).

THE NEXT DATA COLUMN IN TABLE 4 IS LABELED DATA CODE. IT IS THE 3-BIT DATA CODE RETURNED TO THE SOFTWARE WHEN IT READS THE ERA STATUS REGISTER. OF THE EIGHT POSSIBLE COMBINATIONS OF 3 BITS, ONLY SIX ACTUALLY OCCUR IN PRACTICE. THE FIRST TWO BITS INDICATE THE DATA SOURCE CODE AND THE THIRD BIT, IF A 1, INDICATES THAT A WRITE OR WRITE PORTION OF A READ-PAUSE-WRITE WAS BEING PERFORMED.

THE SEVENTH DATA COLUMN IN TABLE 4 INDICATES THE SOURCE OF THE DATA: I.E., WHERE THE DATA WAS COMING FROM WHEN THE ERROR WAS DETECTED. FOR EXAMPLE, ON A READ FROM MEMORY, THE DATA SOURCE WOULD BE LISTED AS MEMORY READ OR READ-PAUSE-WRITE. IN OTHER CASES, THE SOURCE OF DATA COULD BE THE MBOX IN THAT THE ERRORS WERE DETECTED DURING THE WRITE PART OF THE READ-PAUSE-WRITE.

THE NEXT DATA COLUMN CONTAINS FIVE BITS FROM THE APR REGISTER AND A SIXTH BIT TO INDICATE AN INTERRUPT FROM THE RH20. THE FIRST BIT IS A 1 IF THE ERROR OCCURRED ON DATA COMING FROM MEMORY; THATA BIT IS CALLED MEM DATA. THE NEXT TWO BITS ARE A 1 IF THE ERROR INVOLVED THE SBUS. THE FIRST IS LABELED SBUS ERROR. SBUS ERRORS ARE THEN ERRORS IN THE 36-BIT WORD BEING SENT ON THE SBUS OR FAILURES OF THE MEMORY TO COMPLETE ITS CYCLE. THE NEXT BIT IS LABELED SBUS ADDRESS PARITY ERROR AND IS A 1 IF THE ERROR WAS DETECTED IN THE ADDRESS BEING SENT ON THE SBUS, AS OPPOSED TO DATA.

THE FOURTH BIT IS A 1 IF THE PARITY ERROR OCCURRED IN THE CACHE DIRECTORY. IT IS CALLED THE CACHE DIRECTORY PARITY ERROR. THE FIFTH BIT LABELED NXM, IS A 1 IF A NON-EXISTENT MEMORY IS DETECTED. THE SIXTH BIT INT FROM RH20, IS DISCUSSED IN PARAGRAPH 3,3,2. NOTE, IN THIS 6-BIT DATA COLUMN, THAT SEVERAL ERROR BITS MAY SIMULTANEOUSLY BE 1.

3.2 SUMMARY OF ERROR RECOVERY

THERE ARE TWO WAYS IN WHICH PARITY ERRORS ARE INDICATED TO THE -10 SOFTWARE. THE FIRST WAY IS WITH A TRAP. A TRAP IS AN INTERRUPTION OF THE INSTRUCTION SEQUENCE OF THE CPU CAUSED BY THE ATTEMPT TO EXECUTE THAT INSTRUCTION. CONTROL THEN PASSES TO A TRAP ROUTINE WHICH ATTEMPTS TO SLVE THE PROBLEM. IN ALL CASES, WHEN A TRAP OCCURS, THE SOFTWARE IS ABLE TO RETRY AND, IF THE ERROR DOES NOT REPEAT, THE CONDITION HAS BEEN SUCCESSFULLY RECOVERED. THE TRAPS ARE:

1. PAGE FAIL TRAP - PAGE TABLE PARITY ERROR, CODE 25

2. PAGE FAIL TRAP - AR/ARX DATA PARITY ERROR, CODE 26

THE SECOND WAY IN WHICH ERRORS ARE INDICATED TO THE SOFTWARE IS BY AN INTERRUPT ON THE APR. INTERRUPTS DIFFER FROM TRAPS IN THAT THEY ARE USUALLY CAUSED BY AN EXTERNAL ACTIVITY OTHER THAN AN EBOX REQUEST FOR A MEMORY WORD, FOR EXAMPLE: AN I/O OPERATION, THE CACHE DOING SOME OPERATION SUCH AS WRITE BACK WHICH IS NOT IN RESPONSE TO A REQUEST BY THE EBOX, OR THE READING OF THE REMAINING THREE WORDS TO FILL THE CACHE. THESE ARE THEN ASYNCHRONOUS EVENTS AS DISTINGUISHED FROM SYNCHRONOUS EVENTS. IN MANY CASES, THE SOFTWARE IS UNABLE TO RECOVER BY RETRYING ON ERRORS REPORTED BY INTERRUPTS ON THE APR. IN OTHER CASES, SUCH AS CHANNEL ERRORS, THE SOFTWARE IS ABLE TO RECOVER BY MERELY RETRYING THE I/O OPERATION. HOWEVER, IN ALL CASES OF AN APR INTERRUPT FOR A DATA PARITY ERROR, PHYSICAL CORE MEMORY WILL BE SWEPT LOOKING FOR AND RECORDING MORE ERRORS.

THE NEXT DATA COLUMN OF TABLE 4 IDENTIFIES THE TWO KINDS OF TRAPS. IF A 1, THE FIRST ENTRY INDICATES THAT THE ERROR OCCURRED IN THE HARDWARE PAGE TABLE, IS LABELED PAGE TABLE PARITY ERROR, AND HAS TRAP NUMBER 25. THE SECOND ENTRY IS A TRAP ON A PARITY ERROR DETECTED BY THE EBOX WHEN DATA IS MOVED INTO OR OUT OF THE AR OR ARX REGISTERS (CODE 26). THIS ENTRY IS LABELED AR-ARX DATA PAR. IF NEITHER OF THESE ENTRIES IS MARKED WITH A 1, THEN NO TRAP OCCURS - ONLY AN INTERRUPT, INDICATED IN THE PREVIOUS DATA COLUMN.

THE NEXT DATA COLUMN IN TABLE 4 INDICATES WHICH PART OF THE MACHINE IS BROKEN WHEN A NON-RECOVERABLE ERROR OCCURS. THIS CAN BEST BE SEEN BY REFERRING TO THE BLOCK DIAGRAM OF THE SYSTEM (FIGURE 1). THE ENTRIES IN THIS COLUMN ARE PAIRS OR, IN SOME CASES, TRIPLETS. THEY INDICATE WHERE THE ERRORS MAY HAVE OCCURRED.

THE MAJOR DATA PATH CAN BE DIVIDED INTO 11 PARTS, OR REGIONS, FROM THE POINT OF VIEW OF IDENTIFYING AREAS THAT MAY HAVE BROKEN (SEE FIGURE 1). THE FIRST REGION IS FROM THE DMA20 OUT TO THE MF10 AND BACK TO THE DMA20. THE SECOND REGION IS FROM THE DMA20 TO THE MB'S; THE THIRD IS FROM THE MB'S TO THE AR/ARX; THE FOURTH IS FROM THE MB'S TO THE PAGE TABLE OR IN

THE PAGE TABLE; THE FIFTH IS FROM THE MB'S TO THE RH20; THE SIXTH IS FROM THE RH20'S OUT TO THE DRIVES; THE SEVENTH IS FROM THE RH20'S TO THE MB'S; THE EIGHTH IS FROM THE AR/ARX TO THE MB'S; THE NINTH IS FROM THE MB'S TO THE DMA20; THE TENTH IS FROM THE MB'S TO THE MA20'S TO THE MB'S; AND THE ELEVENTH IS FROM THE AR/ARX TO THE CACHE TO THE AR/ARX. THESE ZONES ARE SUMMARIZED IN TABLE 3.

TABLE 3.

ZONE #	DATA PATH
1	DMA20 TO MG10 OR MF10 TO DMA20
2	DMA20 TO MB'S
3	MB'S TO AR/ARX
4	MB'S TO PT
5	MB'S TO RH20
6	RH20 TO MASSBUS DEVICE TO RH20
7	RH20 TO MB'S
8	AR/ARX TO MB'S
9	MB'S TO DMA20
10	MB'S TO MA20 TO MB'S
11	AR/ARX TO CACHE TO AR/ARX

THE NEXT COLUMN IN TABLE 4 INDICATES WHETHER THE SOFTWARE CAN RECOVER BY RETRYING OR BY ABORTING THE PROCESS THAT WAS IN PROGRESS. NOTICE THAT, IN A LARGE NUMBER OF CASES, THE SOFTWARE CAN RETRY (Y = YES, N = NO). IF THE ERROR DOES NOT RE-OCCUR UPON RETRYING, THEN THE SOFTWARE HAS SUCCESSFULLY RECOVERED FROM THE HARDWARE FAILURE. THIS KIND OF ERROR IS CALLED A "SOFT" ERROR. SOMETIMES THE SOFTWARE WILL NOT BE ABLE TO SUCCESSFULLY RECOVER AFTER A NUMBER OF RETRYS. THIS KIND OF ERROR IS TERMED A "HARD" ERROR. NOTICE THEN THAT THE DISTINCTION BETWEEN SOFT ERROR AND HARD ERROR IN MANY OF THE ERRORS LISTED IS DETERMINED DYNAMICALLY BY RETRYING. THOSE ERORS THAT ARE LABELED WITH A NO IN THE RETRY COLUMN EFFECTIVELY ARE HARD ERRORS UPON THEIR FIRST OCCURRENCE.

THE NEXT COLUMN, LABELED CACHE SWEEP, IS COVERED IN PARAGRAPH 3.3.1.

THE LAST TWO COLUMNS IN TABLE 4 LABELED CONFIGURATION, ARE PROVIDED AS A QUICK REFERENCE TO THOSE ERRORS APPLICABLE TO EITHER A 1080 OR 2040 CONFIGURATION.

TABLE 3 ITSELF IS DIVIDED INTO TWO HALVES. THE FIRST HALF IS REFERRED TO AS "EBOX DATA PARITY ERRORS". THESE ARE PARITY ERRORS DETECTED WHEN THE EBOX IS MAKING A REFERENCE TO MEMORY. THE SECOND HALF OF THE TABLE IS CONCERNED WITH ERRORS WHICH OCCUR WHEN THE CHANNEL IS MAKING REFERENCES TO MEMORY. THESE ARE REFERRED TO AS "CHANNEL ERRORS".

3.3 DETAILED DESCRIPTION OF ERROR RECOVERY

3.3.1 EBOX DATA PARITY ERRORS

ON THE FIRST TWELVE ERRORS (LABELED 01 TO 12 IN LEFT HAND MARGIN OF TABLE 3), THE SOFTWARE GETS A TRAP. THE SOFTWARE WILL THEN RETRY TO SEE IF IT CAN RECOVER FROM THE ERROR. THE SOFTWARE WILL ALSO REPORT THE ERROR AND WAIT TO SEE WHETHER OR NOT AN APR INTERRUPT WILL OCCUR. THIS CAN HAPPEN IF AN SBUS ERROR OR A MEMORY DATA (MEM DATA) ERROR OCCURS. THE SOFTWARE WILL REPORT WHAT BROKE, IT WILL SWEEP CORE DURING THE INTERRUPT AND IT WILL RETRY THE INSTRUCTION THREE TIMES. IF A TRAP DOES NOT OCCUR, THEN THE ERROR WAS A SOFT ERROR. IF THE ERROR STILL OCCURS AFTER THREE TIMES, IT IS A HARD ERROR AND THE MONITOR MUST DECIDE WHETHER THIS IS A SERIOUS SYSTEM ERROR OR WHETHER IT IS A NON-SERIOUS ERROR AND THE SYSTEM CAN CONTINUE TO RUN. IF THE HARD ERROR WAS IN A MONITOR WRITEABLE PAGE, THEN THE OPERATING SYSTEM WILL DECCIDE TO CRASH THE SYSTEM. IT IS TOO RISKY TO CONTINUE RUNNING WITH AN ERROR IN A WRITEABLE PAGE WHICH COULD CONTAIN DATA FOR ALL USERS. IF THE HARD EROR IS IN ANY READ-ONLY PAGE, USER OR MONITOR, THE MONITOR WILL THEN READ THAT PAGE IN AGAIN FROM THE DISK SINCE A GOOD COPY IS ALWAYS THERE. IF THE TRAP WHICH OCCURS IS A PAGE TABLE TRAP, THEN THE SOFTWARE WILL JUST CLEAR THE HARDWARE PAGE TABLE AND RETRY. ON ERROR 6 ON THE APR INTERRUPT, THE SOFTWARE WILL INVALIDATE THE HARDWARE PAGE TABLE AND THEN RETRY 3 TIMES. IF THE HARD ERROR OCCURRED IN A WRITEABLE PAGE WHICH WAS PRIVATE OR BELONGED TO A SMALL NUMBER OF USERS,S THEN THE MONITOR WILL CRASH ONLY THOSE USERS WHO WERE USING THAT PAGE. THUS THE SYSTEM CONTINUES TO RUN WHENEVER THERE IS NO DANGER OF USING BAD DATA.

ON ERRORS 13-15, THE SOFTWARE WILL WAIT FOR AN APR SBUS ERROR OR MEM DATA. IT WILL REPORT WHAT IS BROKEN, SWEEP CORE, AND CRASH THE USER IF THE ERROR IS IN USER MODE. IT WILL NOT RETRY ON ERRORS 13-15. THE SOFTWARE ERROR RECOVERY FOR ERRORS 16-18 IS THE SAME AS THAT FOR ERROS 1-12.

ERRORS 19 AND 20 ARE READ-PAUSE-WRITE OPERATIONS IN WHICH THE ERROR OCCURS DURING THE WRITE PART OF THE OPERATION. IN THESE TWO CASES, THE SOFTWARE IS UNABLE TO RECOVER AND A HARD ERROR OCCURS ON THE FIRST SUCH FAILURE. ERRORS 21 AND 22 ARE READ-PAUSE-WRITE ERRORS ON THE READ PART WITH AN INTERNAL MEMORY. IN THESE TWO CASES, THE SOFTWARE IS ABLE TO RECOVER IN THE SAME WAY AS ERRORS 1-12. ERROR 23 IS A READ-PAUSE-WRITE INSTRUCTION WITH INTERNAL MEMORY AND THE ERROR OCCUR DURING THE WRITE PORTION. HERE AGAIN THE SOFTWARE IS UNABLE TO RECOVER AND MUST CRASH THE USER OR THE SYSTEM IMMEDIATELY AND NOT RETRY.

ERRORS 24-52 ARE DATA PARITY ERRORS OCCURRING WHEN THE CACHE IS TURNED ON AS OPPOSED TO BEING OFF. THE ONLY DIFFERENCE BETWEEN THE SOFTWARE RECOVERY WHEN THE CACHE IS ON AND WHEN THE CACHE

IS OFF IS THAT, IN ADDITION TO THE OTHER ERROR RECOVERY PROCEDURES, THE SOFTWARE MUST SWEEP THE PAGE OUT OF THE CACHE IN WHICH THE ERROR OCCURRED. THIS IS BECAUSE THE MBOX LOOKS IN CACHE BEFORE GOING TO CORE AND WOULD REFERENCE THE BAD DATA FROM THE CACHE FIRST. IN ADDITION IF THE WORD HAS ITS WRITTEN BIT ON, AN ERROR FOR THAT PAGE WOULD OCCUR DURING THE SWEEP AND THAT USER WILL HAVE TO BE CRASHED.

AS WHEN THECACHE IS OFF, IF A TRAP OCCURS, THEN THE SOFTWARE IS ABLE TO RETRY TO SEE WHETHER OR NOT IT CAN RECOVER. IF ONLY AN APR INTERRUPT OCCURS, THEN IT IS A FATAL ERROR. ON SOME ERRORS, BOTH AN APR INTERRUPT AND A TRAP OCCUR IN WHICH CASE THE TRAP ROUTINE RETRIES AND THE APR INTERRUPT REPORTS THE ERROR, AS USUAL. ERRORS 24-39 ARE ERRORS WHEN THE CACHE IS ON AND A READ OPERATION IS OCCURRING IN EXTERNAL OR INTERNAL MEMORY. THERE IS ONE DIFFERENCE IN THE ERROR RECOVERY PROCEDURE WHEN THE CACHE IS ON AND WHEN THE CACHE IS OFF, CAUSED BY THE FACT THAT PARITY ERRORS CAN OCCUR ON AHY OF THE OTHER THREE WORDS THAT THE CACHE READS FROM MEMORY WHEN THE EBOX REFERENCES A SINGLE WORD. IN THESE CASES, THE HARDWARE DOES NOT CAUSE A TRAP SINCE THE EBOX HAS NOT TRIED TO USE THAT DATA. INSTEAD, AN APR INTERRUPT IS CAUSED WITH ENOUGH INFORMATION IN THE ERA REGISTER TO ACCURATELY REPORT THE PROBLEM EVEN THOUGH NO USER PROCESS OR THE SYSTEM NEEDS TO BE CRASHED AT THIS TIME. IF, AT A LATER TIME, THE EBOX USES ONE OF THESE BAD WORDS, THEN A TRAP WILL OCCUR AND THE NORMAL RETRY PROCEDURE CAN BE USED.

THIS NEW KIND OF ERROR IN THE OTHER THREE WORDS OCCURS FOR ERRORS 26, 28, 31, 33, 36, AND 39. THESE ERRORS ARE INDICATED WITH AN R IN THE RETRY COLUMN RATHER THAN A YES OR NO, SINCE NO USER PROCESS NOR THE SYSTEM IS EVER CRASHED AT THE INSTANT THAT AN EROR OCCURS IN ONE OF THE OTHER THREE WORDS READ FROM MEMORY BY THE CACHE. AGAIN, AS WHEN THE CACHE IS OFF, AN APR INTERRUPT ALWAYS CAUSES THE SOFTWARE TO SWEEP ALL OF CORE AND REPORT ALL PARITY ERRORS THAT MIGHT OCCUR DURING THE SWEEP. ANOTHER COMPLEXITY WITH THE CACHE BEING ON IS A NEW KIND OF OPERATION - THAT OF THE SWEEPER, OR CCA WRITE. IN THE CASE OF AN ERROR ON A CCA WRITE SWEEP, NO RECOVERY IS POSSIBLE AND THE SOFTWARE MUST ATTEMPT TO CRASH THE USER (OR THE SYSTEM) WHO LAST MADE USE OF THAT PAGE. THIS CAN BE DONE BY CONSULTING THE HARDWARE CORE STATUS TABLE (CST) TO SEE WHICH PROCESS MODIFIED THE PHYSICAL PAGE IN WHICH THE SWEEP WRITE ERROR OCCURRED.

3,3,2 CHANNEL ERRORS

ERRORS 53 - 80 IN TABLE 3 ARE ERORS IN WHICH THE OPERATION IS THE CAHNNEL ACCESSING THE MEMORY INSTEAD OF THE EBOX. ERRORS 53 - 68 ARE WITH THE CACHE OFF, AND ERORS 69 - 80 ARE WITH THE CACHE ON. ON CHANNEL ERRORS, THE SOFTWARE IS NOTIFIED IN ONE OF TWO WAYS AND SOMETIMES BOTH WAYS. THE FIRST WAY IS WITH A INTERRUPT FROM THE RH20 CONTROLLER DURING OR AFTER THE TRANSFER HAS OCCURRED, WITH AN ERROR FLAG SAYING THAT A PARITY ERROR HAS OCCURERED. THE SECOND WAY IS BY AN APR INTERRUPT IN WHICH THE CHANNEL SIGNALS THE EBOX THAT A MEM DATA OR SBUS ERROR HAS OCCURRED. THE SOFTWARE INTERRUPT ROUTINE WILL RETRY EVERY OPERATION WHEN THE RH20 INTERRUPT INDICATES A PARITY ERROR OCCURS. THE APR INTERRUPT CODE WILL SWEEP CORE AND REPORT ALL PARITY ERRORS THAT HAVE OCCURRED DURING THE SWEEP. IT WILL ALSO REPORT THE CONTENTS OF THE ERA REGISTER AND OTHER ERROR INFORMATION GIVEN BY THE APPROPRIATE CONI.

OCCASIONALLY, IT IS POSSIBLE FOR THE RH20 TO THINK THAT A TRANSFER TO MEMORY HAS COMPLETED SUCCESSFULLY BECAUSE IT TRANSFERRED EVERYTHING CORRECTLY TO THE CHANNEL. THE CHANNEL, HOWEVER; IN STORING THE LAST FEW WORDS OF A TRANSFER, ENCOUNTERED AN ERROR ON THE WRITE TO MEMORY SUCH AS AN SBUS ERROR. IN THESE RARE CASES, THE CONTROLLER INTERRUPT ROUTINE WILL THINK THAT TRANSFER OPERATION WAS SUCCESSFUL WHEN IN FACT THE TRANSFER HAD AN ERROR. THEREFORE, THERE MUST BE SOME COMMUNICATION BETWEEN THE APR INTERRUPT ROUTINE AND THE RH20 CONTROLLER ROUTINE TO MAKE SURE THAT THIS KIND OF ERROR DOES NOT GO ON UNDETECTED WITHOUT AN APPROPRIATE RETRY BY THE RH20 SERVICE ROUTINE. ERROR 58 IS AN EXAMPLE OF A CASE WHERE THE RH20 SERVICE ROUTINE DOES NOT GET AN INTERRUPT SAYING THERE WAS AN ERROR, AND ONLY THE APR INTERRUPT OCCURS. IN THIS CASE, THE APR INTERRUPT ROUTINE IS NOT SURE WHETHER THE ERROR OCCURRED IN THE CHANNEL OR IN THE PROCESSOR. ERROR 58, BY THE WAY, IS A CHANNEL DATA WRITE INTO ERROR EXTERNAL MEMORY. THEREFORE, THE SOFTWARE IN THE APR INTERRUPT ERROR ROUTINE MUST LOOK AT THE CST IF THE REFERENCED OR MODIFIED BIT IS NOT ON FOR THAT PAGE. THEN THE SOFTWARE CONCLUDES THAT IT MUST BE THE CHANNEL AND IT SHOULD GO AND CHECK THE CHANNEL AND HAVE THE RH20 SERVICE ROUTINE RETRY THE OPERATION. ON ERRORS 59 AND 61, THE RH20 IS NOT AWARE OF THE ERROR AND ONLY AN APR INTERRUPT OCCURS. IN THIS CASE, THE APR INTERRUPT ROUTINE IS TOLD THAT IT IS THE CHANNEL, IT MUST DETERMINE WHICH CHANNEL HAD THE ERROR, AND RETRY THAT PARTICULAR OPERATION. ERRORS 60 AND 62 LOOK LIKE ERRORS 59 AND 61; HOWEVER, THE RH20 GETS AN INTERRUPT TOO AND SO THE RH20 SERVICE ROUTINE MAY RETRY THE OPERATION AS PART OF ITS NORMAL ERROR RECOVERY. IN ERRORS 63 - 65 THE PARITY ERROR OCCURS WHEN THE CHANNEL READS THE COMMAND WORD INSTEAD OF THE DATA. HERE THE SOFTWARE MUST RETRY THE OPERATION AFTER BUILDING A NEW COMMAND LIST. ERRORS 66 - 68 HAVE APR INTERRUPTS ONLY AND NOT RH20 INTERRUPTS, WHERE THE ADDRESS IS IN THE EPT CHANNEL LOGOUT AREA AND THE ERROR OCCURRED WHEN THE CHANNEL ATTEMPTED TO WRITE THE LOGOUT INFORMATION. HERE THE

SOFTWARE MUST RETRY THE OPERATION AGAIN. ON ERRORS 71, 74, 77, AND 80, THE RH20 SERVICE ROUTINE HAS AN EROR INDICATION AND, IN THESE CASES, SINCE THE CACHE IS ON, THE CACHE MUST BE INVALIDATED BEFORE THE SOFTWARE RETRYS THE OPERATION.

3.3.3 ADDRESS PARITY ERRORS

ADDRESS PARITY ERRORS ARE A NEW KIND OF ERROR DETECTED ON THE KL10; THEY WERE NOT DETECTED ON THE KI10. THEY CAN OCCUR BOTH ON READS AND WRITES TO MEMORY. ADDRESS PARITY ERROR IS DETECTED BY THE MA20 OR THE DMA20, ON A READ, WRITE, OR READ-PAUSE-WRITE, WHEN THEY LOOK AT THE ADDRESS THAT THE MBOX IS REQUESTING TO BE READ OR WRITTEN. ADDRESS PARITY IS REALLY A SLIGHT MISNOMER IN THAT ADDRESS PARITY IS COMPUTED OVER THE 22-ADDRESS BITS, 4-WORD REQUESTS, READ REQUEST, AND WRITE REQUEST. ADRESS PARITY IS A WAY OF CHECKING THE BUS AND THE DRIVERS WHICH EXIST BETWEEN THE MBOX AND THE MA20 OR THE DMA20.

3.3.3.1 MA20 RESPONSE - ON A READ FROM MEMORY IN WHICH THE MA20 DETECTS AN ADDRESS PARITY ERROR ON THE ADDRESS LINES COMING FROM THE MBOX, THE MA20 WILL RETURN FOUR WORDS OF ZEROES WITH BAD PARITY TO THE MBOX. THIS BAD PARITY SHOULD CONTINUE TO PERMEATE THROUGH THE SYSTEM, BEING DETECTED AT EACH LEVEL BY THE PARITY NETWORK. THIS APPROACH DEMONSTRATES THE PRINCIPLE OF NEVER REGENERATING PARITY BUT ALWAYS CHECKING PARITY AND PASSING IT ALONG IF IT IS BAD TO THE NEXT LEVEL WHICH WILL ALSO CHECK IT AND USUALLY FIND IT BAD ALSO. ON A WRITE FROM THE MBOX TO THE MA20, THE MA20 WILL THROW THE DATA AWAY. THE MA20 WILL NOT WRITE THE DATA INTO MEMORY BECAUSE IT IS NOT CLEAR WHICH ADDRESS SHOULD BE WRITTEN INTO, SINCE THERE IS A PARITY ERROR IN THE ADDRESS.

3.3.3.2 DMA20 RESPONSE - ON BOTH READS AND WRITES, IF AN ADDRESS PARITY ERROR OCCURS, THE DMA20 WILL NOT RESPOND TO THE MBOX. INSTEAD, THE MBOX WILL TIME OUT AND GIVE A NORMAL NON-EXISTENT MEMORY INTERRUPT TO THE CPU ON THE APR. THE DMA20 WILL NOT ATTEMPT TO READ OR WRITE MEMORY, AS WITH THE MA20. IN ADDITION, THE DMA20 ALWAYS CHECKS ALL ADDRESSES ON THE MEMORY BUS, WHETHER THE ADDRESSES ARE INTENDED FOR THE MA20 OR THE DMA20. THEREFORE, THE DMA20 MAY SEND BACK AN ADDRESS PARITY ERROR CONDITION TO THE CPU VIA SBUS ADDRESS PARITY ERROR EVEN IF THE ADDRESS IS FOR THE MA20.

3.3.3.3 ADDRESS PARITY ERROR RECOVERY AND REPORTING - RECOVERY IS DEPENDENT ON THE OPERATION IN PROGRESS AT THE TIME OF FAILURE. THE OPERATION CAN BE DETERMINED BY EXAMINING BITS 2, 3, AND 6 OF THE ERA. THESE ARE CCA, CHAN, AND WRITE.

ALL READ (FROM MEMORY) OPERATIONS MAY BE RETRIED 3 TIMES SINCE THE DMA NEVER STARTED THE EXTERNAL MEMORY CYCLE AND THE MA20 SENT 4 WORDS OF 0 WITH BAD PARITY BUT DID NOT DESTROY THE CORE CONTENTS. THOSE READ OPERATIONS INVOLVING THE CHANNEL (ERA BIT 3=1) MAY BE RETRIED BY RE-ISSUING THE COMMAND TO THE RH20.

IF, AFTER ATTEMPTING 3 RETRIES, THE FAILURE PERSISTS, THE

MONITOR SHOULD DETERMINE THE OWNER OF THE AFFECTED PAGE OR CORE AREA AND CRASH THAT OWNER, EITHER A USER OR THE MONITOR ITSELF. IF THE OWNER IS ONLY A USER, THE MONITOR SHOULD ALSO CONSIDER RECONFIGURATION OF THE DEFECTIVE HARDWARE.

IF THE OPERATION IN PROGRESS WAS A MEMORY WRITE CYCLE, RECOVERY IS LIMITED. ONLY THOSE OPERATIONS INVOLVING THE CHANNEL MAY BE RETRIED BY RE-ISSUING THE COMMANDS TO THE RH20.

IN ALL OTHER CASES, THE ONLY RECOVERY POSSIBLE IS TO DETERMINE, BY EXAMINING THE ERA, THE OWNER OF THE PAGE OR CORE AREA IN WHICH THE BAD ADDRESS IS LOCATED. IF THE OWNER IS A USER ONLY THAT USER MUST BE CRASHED. IF THE OWNER WAS THE MONITOR THEN THE SYSTEM MUST BE CRASHED. THERE IS NO POSSIBILITY OF RESTORING READ-ONLY PAGES FOR THE USER OR MONITOR IN THIS CASE SINCE IF THE AFFECTED PAGES WERE READ-ONLY, THE FAILURE ON A MEMORY WRITE CYCLE IS ALSO INDICATING A "WRITE PROTECTION" FAILURE.

FOR THE ERROR REPORTING SYSTEM THE MONITOR SHOULD COLLECT AND PRESERVE THE FOLLOWING INFORMATION:

1. SBUS DIAGNOSTIC CYCLE INFORMATION FOR ALL MEMORY CONTROLLERS. (MA20'S AND DMA20).

2. ERA

3. CONI APR INFORMATION

4. RECOVERY PROCEDURE ATTEMPTED, RESULTS, AND RETRY COUNT.

5. WHERE THE FAILURE WAS DETECTED AND WHAT BROKE.

3.3.4 NON-EXISTENT MEMORY (NXM)

ALL OF THE ERA REGISTER IS VALID WHEN A NON-EXISTENT MEMORY ERROR OCCURS AND AN APR INTERRUPT IS CAUSED. THIS ERROR IS DETECTED BY THE MBOX BECAUSE THERE IS NO RESPONSE FROM THE MA20 OR THE DMA20 AFTER A PERIOD OF 64 MICRO-SECONDS. THE SOFTWARE CAN THEN DETERMINE EXACTLY WHICH WORD RECEIVED THE NON-EXISTENT MEMORY AND WHAT WAS THE SOURCE OF THE REQUEST. IF ADDRESS PARITY ERROR IS ALSO SET, THE ERROR IS AN ADDRESS PARITY ERROR - NOT A NON-EXISTENT MEMORY ERROR; THAT IS, THE DMA20 WILL NOT RESPOND TO AN ADDRESS IF THERE IS A PARITY ERROR IN THE ADDRESS. THIS WILL CAUSE THE MBOX TO CONCLUDE THAT THERE IS A NON-EXISTENT MEMORY BECAUSE OF THE LACK OF A RESPONSE. THUS THE SOFTWARE SHOULD BE AWARE OF WHAT THE REAL FALURE IS WHEN BOTH NON-EXISTENT MEMORY AND ADDRESS PARITY ERRORS ARE INDICATED IN THE APR ERROR STATUS WORD. IF THE MBOX IS READING FROM MEMORY WHEN THE NON-EXISTENT MEMORY OCCURS, THE MBOX WILL SUPPLY FOUR WORDS OF ZEROES TO THE EBOX WITH BAD PARITY, SO THAT THE EBOX WILL GET A DATA PARITY TRAP WHEN IT USES THE DATA. THE SOFTWARE SHOULD CHECK TO SEE WHETHER OR NOT THE ADDRESS BEING REFERENCED IS IN BOUNDS, I.E., IS IN A MEMORY IN WHICH THE SOFTWARE EXPECTS IT TO EXIST AND, IF IT DOES, THEN THE SOFTWARE SHOULD RETRY USING THE NORMAL RETRY MECHANISM AS IF A DATA PARITY ERROR OCCURRED. THIS RETRY PROCEDURE IS PROGRAMMED IN THE PARITY TRAP ROUTINE AND WILL HAPPEN NATURALLY SINCE THE EBOX WILL HAVE DETECTED A PARITY ERROR ON THIS NON-EXISTENT MEMORY CONDITION. NOTE THAT AN APR INTERRUPT WILL OCCUR AS WELL, WITH ERA LATCHED TO THE ERROR WORD, SO THAT THE SOFTWARE CAN LOG THE ERROR IN THE APR INTERRUPT ROUTINE.

ON WRITES TO MEMORY, AND A NON-EXISTENT MEMORY, THE DATA IN THE MB'S IS LOST. HOWEVER, EVERYTHING IN THE ERA REGISTER IS VALID.

THERE IS ANOTHER FORM OF NON-EXISTENT MEMORY ERROR WHICH CAN OCCUR IN THE DMA20. THIS ERROR OCCURS IF THE FIRST WORD REQUESTED BY THE MBOX IS ACKNOWLEDGED; HOWEVER, IF ONE OF THE SUBSEQUENT THREE WORDS DOES NOT SEND BACK AN ACKNOWLEDGE WITHIN 28 USEC A NON-EXISTENT MEMORY CONDITION IS INDICATED TO THE SOFTWARE VIA AN SBUS ERROR INTERRUPT ON THE APR INSTEAD OF VIA AN NXM ERROR INTERRUPT ON THE APR. THE DMA20 REMEMBERS THE ADDRESS OF THE SPECIFIC WORD WHICH RECEIVED A NON-EXISTENT MEMORY CONDITION. THE SOFTWARE MAY THEN READ THIS ADDRESS FROM THE DMA20 USING THE SBUS DIAG FUNCTION. IN THIS CASE THE ERA IS NOT VALID.

IN ALL CASES OF AN NXM OCCURRING ON A MEMORY WRITE CYCLE FOR THE EBOX, THE ONLY RECOVERY PROCEDURE IS TO CRASH THE AFFECTED USER. IF THE OPERATION INVOLVED A CHANNEL, THEN THE SOFTWARE MAY RETRY THE I/O OPERATION.

THE INDICATIONS AVAILABLE TO THE SOFTWARE FOR BOTH ADDRESS PARITY AND NXM ERORS ARE SUMMARIZED IN TABLE 5. THE ERROR

NUMBERS ARE SEQUENTIAL TO PROVIDE EASY REFERENCE FOR ALL ERRORS IN THIS CHAPTER.

3.3.5 CRAM AND DRAM PARITY ERRORS

ODD PARITY IS GENERATED AND STORED FOR EACH WORD BY THE MICRO-CODE ASSEMBLY PROGRAM AND LOADED INTO THESE RAMS EACH TIME THEY ARE LOADED. THIS ODD PARITY IS CHECKED EACH TIME A WORD IN EITHER RAM IS REFERENCED. IF AN ERROR OCCURS AND THE HALT ON ERROR DIAGNOSTIC FUNCTION IS ENABLED (USUAL), THE EBOX CLOCK IS HALTED. THE PDP-11 MUST DETECT THIS AND SAVE THE ENTIRE INTERNAL HARDWARE STATE OF THE MACHINE USING DIAGNOSTIC FUNCTIONS FOR LATER INCLUSION IN THE SYSTEM ERROR FILE AND PRESENT A CONDENSED FORM OF THIS REPORT ON THE OPERATOR'S CONSOLE.

IF EITHER OF THESE EVENTS OCCUR, THE ERROR RECORDING OVERLAY IN THE FRONT END PROCESSOR VALIDATES THE ASSOCIATED RAM. IT SHOULD GET A COPY OF THE "GOOD" CONTENTS (FROM A DISK FILE) AND DO A WORD FOR WORD COMPARE WITH THE CONTENTS OF THE RAM.

3.3.5.1 SOFT RAM PARITY ERROR

IF NO FAILURES ARE DETECTED, THE KL10 AND ITS MONITOR ARE IN A CONTINUABLE STATE AND THE ERROR RECORDING OVERLAY SHOULD INFORM THE OPERATOR OF THE ORIGINAL FAILURE INCLUDING THE ERROR, BAD DATA AND ADDRESS, AND THE FACT THAT A CONTINUE IS BEING ATTEMPTED. IT SHOULD THEN COPY OUT TO THE RP04 THE FAILURE INFORMATION, TURN OFF THE -10 PI SYSTEM, AND ASK THE KERNAL TO RESTART THE -10 AT THE "WARM RESTART" LOCATION.

THIS -10 RESTART ROUTINE WILL RESTART I/O AND DETERMINE WHICH USER (OR THE MONITOR) WAS RUNNING AT THE TIME OF THE ERROR AND CRASH THAT USER.

3.3.5.2 HARD RAM PARITY ERROR

IF THE VERIFICATION OF THE RAM SHOWS ANY ERRORS, THEN EACH BAD DATA WORD AND ADDRESS SHOULD BE PRESENTED TO THE OPERATOR ALONG WITH THE "GOOD DATA" FROM THE DISK FILE. THIS INFORMATION SHOULD ALSO BE INCLUDED IN THE RP04 FILE ALONG WITH THE ENTIRE INTERNAL HARDWARE STATE OF MACHINE. THEN THE FRONT END MUST RELOAD AND VERIFY ALL RAMS. IF VERIFICATION ERRORS STILL OCCUR, INFORM OPERATOR TO CALL FIELD SERVICE. IF VERIFICATION SUCCEEDS, TURN OFF -10 PI AND RESTART -10 MONITOR AT THE "WARM RESTART" LOCATION. THE MONITOR WILL ATTEMPT THE SAME RECOVERY AS ABOVE.

THE AFFECTS OF THIS ERROR RECOVERY MAY BE SUMMARIZED WITH THE FOLLOWING TABLE. AGAIN, THE ERROR #'S ARE SEQUENTIAL FOR EASY REFERENCE.

ERROR #	FAILURE TYPE	ACTION IMED. VERIFY	RELOAD VERIFY	CRAM RESULT	DRAM RESULT

89	SOFT-NONRELOAD RAM	PASS	-	CRASH USER	CONT. USER
90	SOFT-RELOAD RAM	FAIL	PASS	CRASH USER	CONT. USER
91	HARD	FAIL	FAIL	CALL FIELD SERVICE	CALL FIELD SERVICE

3,3,6 CACHE DIRECTORY PARITY ERROR

IN THIS CASE THE ERROR IS FIRST DETECTED BY THE -10 MONITOR WHOSE RECOVERY PROCEDURE IS TO HALT LEAVING INDICATION OF WHY AVAILABLE TO THE -11 KERNAL. THE KERNAL, HAVING SEEN THE HALT BECAUSE OF THIS ERROR, WILL STOP THE EBOX CLOCK AND CALL THE ERROR RECORDING OVERLAY.

AFTER THE OVERLAY HAS DETERMINED THAT THIS IS THE FAILURE IT SHOULD USE DIAGNOSTIC FUNCTIONS TO SCAN ALL OF THE CACHE LOOKING FOR MORE ERRORS. IF ERRORS ARE FOUND, THE CACHE INFORMATION INCLUDING CACHE # AND ADDRESS SHOULD BE PRESERVED.

AT THE END OF THE SWEEP, IF ERRORS WERE DETECTED, THE OVERLAY SHOULD ASK THE OPERATOR ABOUT RECONFIGURING THE CACHE WITH THE SUGGESTION THAT THE BAD CACHE BE TURNED OFF. IF NO RESPONSE IS GIVEN (UNATTENDED OPERATION) THE OVERLAY SHOULD TURN OFF ALL OF CACHE AND INFORM THE OPERATOR.

IF MORE THAN 1 CACHE DETECTED A FAILURE DURING THE SWEEP, ALL OF THE CACHE SHOULD BE TURNED OFF DURING THE OPERATOR DIALOGUE SINCE THIS FAILURE IS MORE INDICATIVE OF A CONTROL PROBLEM VERSUS A CACHE RAM PROBLEM.

IF NO ERRORS WERE DETECTED BY THE SWEEP, THE OVERLAY SHOULD ASK THE OPERATOR WHETHER TO TURN OFF ALL OF CACHE OR NOT WITH A SUGGESTION TO LEAVE CACHE ON IF THIS IS THE FIRST FAILURE. AGAIN, IF NO RESPONSE IS RECEIVED, ALL OF THE CACHE SHOULD BE TURNED OFF, BUT ALWAYS THE OVERLAY WILL WRITE OUT THE INFORMATION ON THE RP04 AND TELL THE OPERATOR OF THE FAILURE AND ACTION TAKEN.

WITH ALL FAILURES THE OVERLAY WILL INITIATE A FULL RELOAD OF THE SYSTEM AFTER RECORDING THE INFORMATION AND RECONFIGURING THE CACHE.

THE DEFAULT ACTION DURING UNATTENDED OPERATION (LEAVE THE CACHE ON OR OFF) SHOULD BE A SITE PARAMETER BUT THE SAFEST METHOD IS TO TURN THE CACHE OFF.

3.3.7 FAST MEMORY PARITY ERROR

EACH WORD STORED IN FAST MEMORY HAS ODD PARITY GENERATED AS IT LEAVES THE AR. PARITY IS CHECKED IF THE FAST MEMORY WORD IS BEING READ OUT THROUGH THE ADB MIXER. IF AN ERROR IS DETECTED, THE EBOX CLOCK IS HALTED IF THE HALT FUNCTION IS ENABLED BY THE CONSOLE PROCESSOR (USUAL).

AS WITH CRAM AND DRAM PARITY ERRORS, THE PDP-11 MUST DETECT THE ERROR AND THE ERROR RECORDING OVERLAY WILL RECORD THE HARDWARE MACHINE STATE. AFTER THE MACHINE STATE HAS BEEN SAVED, THE ERROR RECORDING OVERLAY SHOULD ALSO SCAN ALL OF FAST MEMORY LOOKING FOR MORE ERRORS AND PROVIDE LOGICAL "ANDS" AND "ORS" OF THE BAD DATA AND ADDRESS WORDS IF ANY ERRORS OCCUR.

3.3.7.1 SOFT FAST MEMORY PARITY ERROR - IF NO ERRORS ARE DETECTED OR ALL THE ERRORS ARE IN THE SAME BLOCK OF FAST MEMORY, THE ERROR RECORDING OVERLAY SHOULD INFORM THE OPERATOR OF ALL FAILURE INFORMATION, WRITE OUT THE RP04 FILE, TURN OFF THE -10 PI SYSTEM, AND ASK THE KERNAL TO RESTART THE -10 AT THE "WARM RESTART" LOCATION. THE -10 MONITOR WILL THEN CRASH ONLY THE AFFECTED USER.

3.3.7.2 HARD FAST MEMORY PARITY ERROR - IF ERRORS ARE FOUND IN MORE THAN 1 BLOCK OF FAST MEMORY ADDRESSES, THE ERROR RECORDING OVERLAY WILL WRITE OUT THE RP04 FILE, AND INFORM THE OPERATOR ALL FAILURE DATA WITH THE ADDITIONAL REQUEST TO CALL FIELD SERVICE.

3,3,8 EBUS PARITY ERROR

THE AR PARITY TREE GENERATES ODD PARITY FOR ALL DATAO AND CONO WORDS DESTINED TO BE OUTPUT ON THE EBUS. ALL DEVICES MAY CHECK THIS PARITY. CURRENTLY THE DTE20 CHECKS PARITY ON DATAO OPERATIONS, BUT NOT ON CONO. ALL DEVICES MAY GENERATE ODD PARITY FOR ALL WORDS THAT THEY TRANSMIT TO THE EBOX OVER THE EBUS IF THEY DESIRE AND GENERATE THE APPROPRIATE EBUS SIGNAL INDICATING PARITY HASBEEN GENERATED.

ALL WORDS ENTERING THE AR/ARX FROM THE EBUS ARE CHECKED FOR CORRECT PARITY IF PARITY HAS BEEN GENERATED BY THE TRANSMITTING DEVICE. IF AN ERROR IS DETECTED, AN I/O PAGE FAIL INTERRUPT OCCURS AND THE MONITOR HAS THE ABILITY TO DETERMINE THAT THE BAD DATA WAS RECEIVED FROM THE EBUS. THE INFORMATION COLLECTED FOR ERROR REPORTING SHOULD BE THE SAME AS FOR OTHER OCCURANCES OF THE SAME APR INTERRUPT. NOTE THAT AN -11 DEPOSIT TO -10 MEMORY CAN CAUSE A PARITY INTERRUPT WHICH HAS NOTHING TO DO WITH THE -10 INSTRUCTION BEING EXECUTED. THIS INTERRUPT SHOULD BE TREATED LIKE AN INTERRUPT BY THE -10 MONITOR. THE CURRENT USER SHOULD BE CONTINUED, AS WITH ALL PARITY TRAPS. THE -10 MONITOR WILL TELL THE -11 THAT THE DEPOSIT FAILED AND THE -10 WILL LOG THE FAILURE IN THE LOG FILE.

THE RECOVERY PROCEDURE SHOULD RETRY THE I/O OPERATION 3 TIMES AND IF THE FAILURE PERSISTS CRASH THE SYSTEM SINCE A VERY BASIC PART OF THE SYSTEM HAS FAILED.

4. GENERALIZED FLOW CHARTS FOR ERROR RECOVERY

INFORMATION IS PRESENTED HERE TO DESCRIBE THE ACTIONS OF THE VARIOUS ERROR RECOVERY ROUTINES. THE FLOW CHARTS START AT THE POINT AFTER THE ERROR HAS BEEN DETERMINED. FOR EXAMPLE, AN APR INTERRUPT MAY OCCUR FOR MANY REASONS BUT THE FLOW FOR A MEMORY DATA PARITY ERROR STARTS WHEN IT IS KNOWN THAT THIS IS THE REASON FOR THIS INTERRUPT.

IN SOME CASES RECOVERY IS NOT POSSIBLE BY THE -10 MONITOR, SUCH AS CACHE DIRECTORY PARITY ERRORS. THESE ARE NOT COVERED HERE.

4.1 PAGE FAIL TRAP FOR AR/ARX PARITY ERROR. (CODE 26)

AFTER DETERMINING THAT THIS PAGE FAIL TRAP HAS OCCURED BECAUSE OF AN AR/ARX PARITY ERROR, THE TRAP ROUTINE SHOULD CHECK TOSEE IF AN APR INTERRUPT IS IN PROGRESS (SOFTWARE FLAG "A") AND IF SO, SAVE THE BAD DATA WORD FROM AC BLOCK 7 WORD, SET THE ERROR DETECTED FLAG (SOFTWARE FLAG "D"), AND RETURN TO THE INTERRUPT.

IF FLAG "A" IS NOT SET, THE ROUTINE CHECKS TO SEE IF THE PHYSICAL ADDRESS IS THE SAME AS THE LAST BAD PHYSICAL ADDRESS SEEN FOR THIS PROCESS. IF THE ADDRESS IS NOT THE SAME, THE ROUTINE WILL SAVE INFORMATION FOR ERROR RECORDING, INITIALIZE THE RETRY COUNT, AND PERFORM THE FIRST RETRY BY ADDRESSING THE BAD PHYSICAL WORD. IF THE REFERENCED WORD IS STILL BAD ANOTHER TRAP WILL OCCUR AND ACTION FOR THIS IS DESCRIBED IN PARAGRAPH 4.1.1.

IF THIS REFERENCE IS SUCCESSFUL (NO RECURSIVE TRAP) WE MAY ASSUME THE USER WILL RECOVER SO THE ERROR LOGGER IS CALLED FOR A SOFT ERROR. THIS FIRST RETRY IS DONE BY THE TRAP ROUTINE SO THAT IF THE REFERENCE IS SUCCESSFUL THE TRAP ROUTINE IS STILL ACTIVE AND CAN CALL THE LOGGER.

4.1.1 IF THE PHYSICAL ADDRESS IS THE SAME AS THE LAST "BAD" PHYSICAL ADDRESS SEEN FOR THIS PROCESS, THE ROUTINE WILL DECREMENT THE RETRY COUNT AND IF THE COUNT IS GREATER THAN 0, DISMISS BACK TO THE USER VIA THE SAVED PC FOR ANOTHER RETRY, IF THE RETRY COUNT IS NOT GREATER THAN 0 (I.E., = 0, 3 RETRIES PERFORMED) THE TRAP ROUTINE WILL CALL THE ERROR LOGGER TO RECORD A HARD ERROR AND THEN CRASH THE AFFECTED PROCESS (EITHER THE USER OR SYSTEM).

UNDER CERTAIN CONDITIONS, 2 ENTRIES FOR THE ERROR RECORDING SYSTEM MAY OCCUR FOR THE SAME ERROR. THIS CAN HAPPEN IF THE FIRST RETRY (MADE BY THE TRAP ROUTINE) IS SUCCESSFUL AND THE ROUTINE CALLS THE LOGGER FOR A SOFT ERROR; BUT ALL 3 OF THE RETRIES PERFORMED BY THE USER ARE UNSUCCESSFUL AND THE TRAP ROUTINE AGAIN CALLS THE ERROR LOGGER TO RECORD THE HARD ERROR. THIS IS EXPECTED TO BE A RARE CASE BUT SHOULD BE WELL DOCUMENTED IN THE ERROR REPORTING DOCUMENTATION.

4.2 APR INTERRUPT FOR MEMORY DATA PARITY ERROR

AFTER DETERMINING THAT THIS INTERRUPT WAS CAUSED BY A MEMORY DATA PARITY ERROR (FOR EITHER A READ OR WRITE CYCLE) THE INTERRUPT ROUTINE WILL CHECK TO SEE IF A CORE SWEEP IS ALREADY IN PROGRESS. IF NOT IT WILL SET THE "SWEEPING" FLAG, SAVE THE ERROR INFORMATION REQUIRED, SET THE SWEEP FLAG "A" FOR THE AR/ARX TRAP ROUTINE, INITIALIZE THE SWEEP ADDRESS AND CLEAR THE APR FLAGS. THE ROUTINE WILL THEN ADJUST THE INTERRUPTED PC VALUE TO POINT AT THE SWEEP REFERENCE ROUTINE AND DISMISS THE INTERRUPT.

THE SWEEP REFERENCE ROUTINE WILL REFERENCE THE SWEEP ADDRESS (I.E., MOVE AC, SWEEP ADDR) AND AN AR/ARX TRAP MAY OCUR SAVING THE BAD DATA WORD. WHEN THIS TRAP ROUTINE IS FINISHED, IT WILL RETURN TO THE SWEEPER AFTER SETTING THE "D" FLAG.

THE SWEEPER WILL CHECK THE "D" FLAG AND, IF SET, SAVE THE BAD DATA WORD IN THE SWEEP ERROR TABLE. IT WILL THEN CHECK TO SEE IF THE CURRENT LOCATION IN THE SWEEPER ERROR TABLE HAS BEEN USED. THIS LOCATION CAN BE USED BY EITHER THE AR/ARX TRAP ROUTINE OR THE APR INTERRUPT ROUTINE. IF THE CURRENT POSITION IN THE ERROR TABLE HAS BEEN USED THE POINTER WILL BE INCREMENTED; THE SWEEP REFERENCE ADDRESS WILL ALWAYS BE INCREMENTED. IF THE SWEEP IS NOT FINISHED, THE ROUTINE WILL REFERENCE THE NEXT ADDRESS AND CONTINUE THE SWEEP.

WHEN THE SWEEP IS FINISHED THE ROUTINE WILL CLEAR THE "SWEEPING" FLAG AND THE "A" FLAG, CALL THE ERROR-LOGGER AND DISMISS VIA THE ORIGINAL SAVED P.C.

4.2.1 ERROR DETECTED DURING THE SWEEP WILL CAUSE ANOTHER APR INTERRUPT OR AN AR/ARX PARITY ERROR TRAP OR BOTH. THE AR/ARX PARITY ERROR TRAP ROUTINE IS DESCRIBED IN PARAGRAPH. THE INTERRUPT ROUTINE WILL SEE THE "SWEEPING" FLAG SET, SAVE THE ERROR INFORMATION IN THE CURRENT LOCATION IN THE SWEEP ERROR TABLE, INCREMENT THE SWEEP ERROR COUNTER, CLEAR THE APR FLAGS AND DISMISS THE INTERRUPT (BACK TO THE AR/ARX TRAP ROUTINE OR THE CORE SWEEPER).

THE BAD DATA DETECTED BY A CORE SWEEP IS ONLY HELD FOR ERROR REPORTING AND IS NOT CORRECTED (RE-WRITTEN WITH GOOD PARITY) NOR ARE ANY RETRIES PERFORMED. NO USERS ARE CRASHED BECAUSE IT IS NOT ALWAYS CLEAR HOW MANY USERS AND WHO SHOULD GET THE AXE. INSTEAD IT IS A FUNCTION OF THE AR/ARX PARITY ERROR TRAP ROUTINE AND/OR THE RH20 ROUTINES TO PERFORM THE RETRYS AND CRASH USERS IF UNSECCESSFUL.

4,3 PAGE FAIL TRAP FOR PAGE TABLE PARITY ERROR (CODE 25)

TO BE SUPPLIED

4,4 APR INTERRUPT FOR ADDRESS PARITY ERROR

TO BE SUPPLIED

4,5 APR INTERRUPT FOR NXM

TO BE SUPPLIED

5. SYSERR SAMPLE OUTPUT

THE FOLLOWING ARE SAMPLE OUTPUTS FROM SYSERR, THE REPORT GENERATING PORTION OF THE MONITOR ERROR REPORTING PACKAGE. THESE SHOULD NOT BE CONSIDERED ABSOLUTE WITH REGARD TO FORM AND CONTENT BUT ARE INCLUDED AS GUIDELINES TO DETERMINE WHAT INFORMATION SHOULD BE PRESERVED BY THE OPERATING SYSTEM.

5.1 FAST AC PARITY ERROR

THE LOGICAL "AND" AND "OR" OF THE BAD ADDRESSES SHOULD INCLUDE THE AC BLOCK NUMBER.

```
*******************************
FAST AC PARITY ERROR SYS # 1037
AT:  10:14       ON: 15-JULY-74
*******************************
        ***(NOTE:  INFORMATION MUST BE COLLECTED BY THE 11.)***

STATUS AT FAILURE:

        BLOCK:          2
          AC#:          12
        BAD DATA:       123456,,654321

RECOVERY:       CONTINUE-CRASH USER
                SYSTEM HALT
                ETC.
SWEEP INFORMATION:
        # ERRORS DETECTED:                3
        LOGICAL "AND" OF BAD ADDRESSES:          ABC
        LOGICAL "OR" OF BAD ADDRESSES:           CBA
        LOGICAL "AND" OF BAD DATA:               XWZ
        LOGICAL "OR" OF BAD DATA:                ZWX
```

5,2 EBUS PARITY ERROR

THIS REPORT COVERS THOSE ERRORS DETECTED BY THE CPU, EBUS PARITY ERRORS DETECTED BY I/O DEVICES SHOULD BE REPORTED UNDER HEADINGS OF THAT DEVICE WHICH DETECT THE ERROR,

```
*******************************
E-BUS PARITY ERROR SYS # 1037
AT:  10:14      ON:  15-JULY-74
*******************************

STATUS AT ERROR:
        OPERATION:      DATAI XYZ
        BAD DATA WORD:  123456,,654321
        RECOVERY:       RECOVERABLE
        RETRY COUNT:    2
```

5,3 CRAM PARITY ERROR

INFORMATION IN PARENTHESES IS NOT INCLUDED IN FINAL OUTPUT BUT ARE ONLY EXPLANATORY NOTES,

```
*******************************
C-RAM PARITY ERROR  SYS # 1037
AT:  13:03      ON:   28-OCT-75
*******************************
        ***(NOTE:  INFORMATION MUST BE COLLECTED BY THE 11,)***

STATUS AT FAILURE:
        IF:                     254000
        LAST ADDR ACCESSED:     432
        CR WAS:                 123456123456,,654321654321
        SHOULD BE:              123456123456,,454321654321
        DIFF:                   000000000000,,200000000000

RECOVERY:       SOFT-NONRELOAD,CRASH USER
                HARD,SYSTEM CRASH
                ETC,

SWEEP INFORMATION:
        # ERRORS DETECTED:      3
        LOGICAL "AND" CF ADDRESSES:     XYZ
        LOGICAL "OR" CF ADDRESSES:      XYZZ
        LOGICAL "AND" CF DATE:  ABC
        LOGICAL "OR" CF DATA:   ABCD

ADDR,   GOOD DATA               OBSERVED DATA   DIFFERENC#E

(INFORMATION PRESENTED HERE FOR FIRST & FAILURES DETECTED
DURING SWEEP)
```

5,4 D-RAM PARITY ERROR

INFORMATION IN PARENTHESIS IS NOT INCLUDED IN FINAL OUTPUT BUT ARE ONLY EXPLANATORY NOTES.

```
**********************************
D-RAM PARITY ERROR SYS # 1037
AT:  13:05      ON:13-JUN-74
**********************************

        ***(NOTE:  INFORMATION MUST BE COLLECTED BY THE 11.)***

STATUS AT FAILURE:
        IF:             254000
        DR ADDRESS:     254
        D-RAM J:        123
        DR WAS:         12345670
        SHOULD BE:      12345674
        DIFF:           00000004

RECOVERY:       SOFT-RELOAD, CONTINUE USER
                HARD, CRASH SYSTEM
                ETC.
SWEEP INFORMATION:
        # ERRORS DETECTED:      3
        LOGICAL "AND" OF ADDRESSES:     XYZ
        LOGICAL "OR" OF ADDRESSES:      XYZZ
        LOGICAL "AND" OF DATA:  ABC
        LOGICAL "OR" OF DATA:   ABCD

ADDR>   GOOD DATA               OBSERVED DATA   DIFFERENCE
```

(INFORMATION PRESENTED HERE FOR FIRST 6 FAILURES DETECTED DURING SWEEP)

5.5 KL10 DATA PARITY ERROR

THIS REPORT WILL BE GENERATED FOR ANY ERROR DETECTED IN THE MAJOR DATA PATH BY THE DMA, MB, PT, OR AR/ARX.

THE "BAD DATA WORD" IN THE SECOND SECTION IS LISTED AS "(SUSPECTED)" IF THE ORIGINAL FAILURE DID NOT INVOLVE THE EBOX AND THE MOVE INSTRUCTION (TO GET THE DATA TO THE EBOX) FAILED TO CAUSE A TRAP.

```
**********************************
KL10 DATA PARITY ERROR  SYS # 1037
AT:  10:14      ON:  15-JULY-74
**********************************

DATA PARITY ERROR TOTALS FOR CPL0
        REPRODUCABLE:            2
        NON-REPRODUCABLE:        0
        USER ENABLED:            0
        CORE SWEEPS:             1
        DETECTED BY DATA
         CHANNEL BUT NOT
         BY CPU:                 0
        CONTINUES AFTER PE:      0

STATUS AT ERROR:
        CONI APR:        12345 = TEXT
        ERA:             123456,,654321 = TEXT
        PHYSICAL MEM ADDR.
         AT FAILURE;     12345670
        (SUSPECTED) BAD DATA WORD:      123456,,654321

        ERROR FIRST DETECTED BY:        MB
        PROBABLE BAD DATA PATH: SBUS TC MB, MEMORY

        RECOVERY:        CONTINUE USER
        RETRY COUNT:     1

SWEEP INFORMATION:
        ERRORS DETECTED:         2
        ERROR FIRST DETECTED BY:        MB
        PROBABLE BAD DATA PATH: SBUS TO MB, MEMORY
        LOGICAL "AND" OF BAD
        PHYSICAL ADDRESSES:      654321
        LOGICAL "OR" OF BAD
        PHYSICAL ADDRESSES:      654321
        LOGICAL "AND" OF BAD DATA:      000001,000000
        LOGICAL "OR" OF BAD DATA:       777776,777777

SYSTEM MEMORY MAP:
        INTERLEAVE MODE:         1,2,OR 4

        CONTROLLER #0:  MA20, 32K
```

STARTING ADDR: 000000

(REPEATED FOR EACH CONTROLLER ON THE SYSTEM)

DMA-20: 128K
STARTING ADDR: 177777

5.6 KL10 ADDRESSING FAILURE

THIS REPORT WOULD BE GENERATED ANY TIME EITHER ON NXM OR ADDRESS PARITY ERROR IS DETECTED. THE "BAD ADDRESS" FROM THE DMA IS LISTED ONLY IF THE DMA DETECTED THE ERROR.

```
**********************************
KL10 ADDRESSING FAILURE  SYS # 1037
AT:  10:14      ON: 15-JULY-74
**********************************

ADDRESS PARITY ERROR TOTALS FOR CPL0
        REPRODUCABLE:              2
        NON-REPRODUCABLE:          0
        USER ENABLED:              0
        CORE SWEEPS:               1
        DETECTED BY DATA
         CHANNEL BUT NOT
         BY CPU:                   0
        CONTINUES AFTER PE:        0
NXM ERROR TOTALS FOR CPL0
        REPRODUCABLE:              2
        NON-REPRODUCABLE:          0
        USER ENABLED:              0
        CORE SWEEPS:               1
        DETECTED BY DATA
         CHANNEL BUT NOT
         BY CPU:                   0
        CONTINUES AFTER PE:        0

STATUS AT ERROR:
        ERROR:              S-BUS ADDR, PARITY ERROR
                            DETECTED BY DMA-20
        CONI APR:           12345 = TEXT
        ERA:                123456,,654321 = TEXT
        BAD ADDR, (FROM DMA):  12345670

        RECOVERY:           CONTINUE USER
        RETRY COUNT:        1

SWEEP INFORMATION:
        ERRORS DETECTED:           2
        LOGICAL "AND" OF BAD
        PHYSICAL ADDRESSES:        654321
        LOGICAL "OR" OF BAD
        PHYSICAL ADDRESSES:        654321

SYSTEM MEMORY MAP:
        INTERLEAVE MODE;           1,2,OR 4

        CONTROLLER #0:  MA20, 32K
        STARTING ADDR:  000000
```

(REPEATED FOR EACH CONTROLLER ON THE SYSTEM)

DMA-20: 128K
STARTING ADDR; 177777

5.7 CACHE DIRECTORY PARITY ERROR

THE CACHE RECONFIGURATION IS THE RESULT OF THE OPERATOR DIALOGUE AFTER THE CACHE SWEEP BY THE -11. IF ONLY 1 CACHE FAILED AND WAS TURNED OFF, ONLY THAT CACHE NUMBER WOULD BE LISTED.

```
**********************************
CACHE DIRECTORY PARITY ERROR  SYS # 1037
AT:  10:14       ON: 23-SEPT-74
**********************************

        **NOTE: INFORMATION MUST BE COLLECTED BY THE 11***

SWEEP INFORMATION:
        # ERRORS DETECTED:        3
        LOGICAL "AND" OF ADDRESSES:      ABCD
        LOGICAL "OR" OF ADDRESSES*       DCBA

        CACHE #'S WITH FAILURE:          2,3
        CACHE CONFIGURATION AT ERROR:    ALL CACHE ON
        CACHE RECONFIGURATION:           ALL CACHE OFF
```

APPENDIX

TESTING KL10 ERROR REPORTING LOGIC

THE FOLLOWING IS A DESCRIPTION OF HOW TO CAUSE HARDWARE DETECTABLE ERRORS ON THE KL10, ALMOST ALL THE ERRORS CAN BE CAUSED ENTIRELY BY SOFTWARE MEANS. A FEW CASES REQUIRE SOME MANUAL INTERVENTION IN THE FORM OF SWITCH THROWING TO SWITCH HARDWARE OFF LINE IN ORDER TO CAUSE THE ERROR. THESE CASES HAVE BEEN KEPT TO A MINIMUM.

1. DATA PARITY

THE DATA PARITY LOGIC CAN BE TESTED BY WRITING EVEN PARITY FROM THE AR.

EVEN (BAD) DATA PARITY CAN BE GENERATED ON WRITES TO MEMORY BY EXECUTING A "CONO PI" WITH BIT 19 (WR EVEN DATA PARITY BIT) EQUAL TO 1 BEFORE DOING THE WRITE. ANY WORDS WRITTEN WHILE THIS BIT IS ON WILL BE WRITTEN INTO MEMORY (CACHE) WITH EVEN PARITY. IF THIS BIT IS ON AND ANY AC'S ARE WRITTEN THE DATA IN THE AC'S WILL BE WRITTEN WITH BAD PARITY.

IF THE WORD GOES DIRECTLY TO MEMORY WITHOUT GOING THROUGH THE CACHE THEN AN MB PARITY ERROR WILL BE DETECTED BY THE PARITY CHECKER ON THE MB'S AS THE WORD GOES TO MEMORY. THIS WILL CAUSE THE MB PARITY APR FLAG TO SET IN THE PROCESSOR.

IF THE WORD WITH EVEN PARITY GOES TO THE CACHE THEN IT WILL SIT IN THE CACHE UNTIL A WRITEBACK OCCURS FOR THE QUADWORD CONTAINING THE WORD. THIS CAN BE CAUSED EITHER BY A CACHE SWEEP OR BY REFERENCING THE SAME LINE IN FOUR OTHER PAGES. IN EITHER CASE THE PARITY ERROR WILL BE DETECTED ON THE WRITEBACK INSTEAD OF ON THE ORIGINAL WRITE.

AFTER THE WORD LEAVES THE MB'S IT WILL BE PARITY CHECKED AT THE DMA20 IF ONE EXISTS AND THE ADDRESS IS IN THE CORRECT RANGE. IF THE DMA20 DETECTS BAD PARITY IT WILL SEND SBUS ERROR TO THE PROCESSOR AND THIS WILL CAUSE THE SBUS APR FLAG TO SET.

IF BAD PARITY WAS WRITTEN INTO ONE OF THE AC'S THEN THE EBOX'S CLOCK WILL STOP IF THE AC IS READ AFTER THE BAD PARITY WAS WRITTEN.

PARITY ON READS CAN BE CHECKED BY WRITING A WORD WITH BAD PARITY INTO MEMORY AND THEN READING IT BACK. ON THE WAY TO THE PROCESSOR THE WORD WILL BE PARITY CHECKED AT THE DMA20, MB'S, AND IN THE AR OR ARX. THIS CHECKING WILL CAUSE THE SBUS AND MB PARITY APR FLAGS AND AN AR OR ARX PARITY PAGE FAIL.

2. PAGE TABLE PARITY (KI-MODE PAGING)

THE PAGE TABLE PARITY LOGIC CAN BE CHECKED BY WRITING BAD PARITY INTO A WORD OF THE CORE COPY OF A JOBS PAGE TABLE. NEXT THE JOB SHOULD BE STARTED AND IT SHOULD REFERENCE THE TWO PAGES FOR WHICH THE CORE PAGE TABLE HAS A WORD WITH BAD PARITY. THE REFERENCES SHOULD IMMEDIATELY PAGE FAIL WITH A CODE OF 25. BOTH HALF WORD ENTRIES SHOULD BE REFERENCED TO MAKE SRE THAT AL OF THE PAGE TABLE PARITY LOGIC IS WORKING.

3. PAGE TABLE PARITY (KL-MODE PAGING)

IN KL PAGING MODE, BAD PARITY CAN BE WRITTEN INTO THE PAGE TABLE BY CAUSING A REFILL WHILE "WR EVEN DATA PARITY" IS SET.

4. CHANNEL DATA AND COMMAND WORD PARITY

CHANNEL DATA AND COMMAND WORD PARITY CAN BE CHECKED BY WRITING DATA OR COMMAND WORDS WITH BAD PARITY INTO MEMORY FROM THE PROCESSOR. NEXT A CHANNEL WHICH WILL USE THE BAD DATA OR COMMAND WORDS CAN BE STARTED.

IF A COMMAND WORD WITH BAD PARITY IS PICKED UP THE CHANNEL WILL SHUT DOWN AND STORE STATUS INFORMATION IN ITS LOGOUT AREA IN THE EXEC PROCESS TABLE.

IF A DATA WORD WITH BAD PARITY IS FETCHED THE ACTION OF THE CHANNEL IS DEPENDENT ON WHAT ERROR CHECKING IS ENABLED IN RH20. THIS ACTION CAN RANGE FROM TERMINATING THE TRANSFER AND INTERRUPTING THE PROCESSOR TO IGNORING THE ERROR. IF THE TRANSFER IS TERMINATED THE CHANNEL WILL STORE STATUS IN THE EXEC PROCESS TABLE.

IN BOTH CASES THE MB PARITY APR FLAG WILL BE SET WHEN THE BAD WORDS ARE TAKEN OUT OF THE MB'S BY THE CHANNEL.

DATA GOING FROM THE DEVICE TO MEMORY CAN BE CHECKED BY WRITING A WORD WITH BAD PARITY INTO MEMORY, WRITING IT OUT TO THE DEVICE WITH DATA PARITY CHECKING DISABLED IN THE RH20, AND THEN READING IT BACK WITH THE PARITY CHECKING ENABLED.

5. AR AND ARX PARITY

THE AR PARITY LOGIC CAN BE CHECKED BY WRITING A WORD WITH BAD PARITY INTO CORE AND THEN DOING

```
MOVE AC,MEM
```

TO BRING THE DATA WORD INTO THE AR. THIS INSTRUCTION WILL IMMEDIATELY PAGE FAIL WITH A CODE OF 26.

THE ARX PARITY LOGIC CAN BE CHECKED BY WRITING A WORD WITH BAD PARITY INTO CORE AND THEN DOING

```
MOVEI AC,@MEM
```

TO BRING THE INDIRECT WORD INTO THE ARX. THIS INSTRUCTION WILL IMMEDIATELY PAGE FAIL WITH A CODE OF 27.

THE ABILITY TO WRITE WORDS WITH BAD PARITY INTO MEMORY SHOULD BE TESTED BEFORE THE AR AND ARX PARITY PAGE FAIL LOGIC IS TESTED.

6. SBUS ADDRESS PARITY

THE SBUS ADDRESS PARITY LOGIC CAN BE CHECKED BY MAKING A MEMORY REFERENCE WHILE EVEN (BAD) ADDRESS PARITY IS BEING FORCED. BAD ADDRESS PARITY CAN BE FORCED BY DOING A "CONO PI" WITH BIT 18 (WRITE EVEN ADDRESS PARITY BIT) EQUAL TO 1. THIS WILL CAUSE THE NEXT MEMORY REFERENCE FROM EITHER THE CHANNELS OR THE PROCESSOR TO BE MADE WITH BAD ADDRESS PARITY. AS SOON AS THE APR ADDRESS PARITY FLAG IN THE PROCESSOR SETS THE WRITING OF BAD ADDRESS PARITY WILL BE INHIBITED.

THE ADDRESS PARITY NOTIFICATION IS GUARANTEED TO ARRIVE AT THE PROCESSOR BEFORE THE NEXT MEMORY REFERENCE IS INITIATED, HOWEVER, THE APR FLAG WILL NOT SET UNTIL THE EBOX CLOCK TICKS. THIS MEANS THAT IF IT IS DESIRED TO WRITE BAD PARITY FOR A SINGLE MEMORY REFERENCE THE PROGRAM THAT WRITES BAD ADDRESS PARITY MUST RUN OUT OF THE AC'S SO THAT THE EBOX CLOCK DOES NOT STOP TICKING.

7. CACHE DIRECTORY PARITY

THE CACHE DIRECTORY PARITY LOGIC CAN BE CHECKED BY WRITING A WORD INTO THE CACHE WHILE THE WR EVEN CACHE DIRECTORY PARITY BIT IS SET. THE WRITE EVEN CACHE DIRECTORY PARITY BIT CAN BE SET BY DOING A "CONO PI" WITH BIT 20 EQUAL TO 1. EVEN CACHE DIRECTORY PARITY WILL BE WRITTEN FOR ALL DATA THAT GOES INTO THE CACHE WHILE THIS BIT IS SET.

IF THE WORD IN THE CACHE IS SUBSEQUENTLY REFERENCED FOR ANY REASON (EBOX READ, CHANNEL READ OR WRITE, PAGE REFILL, EBOX WRITE, CACHE SWEEP, CACHE WRITEBACK) THE CACHE DIRECTORY PARITY ERROR APR FLAG WILL SET AFTER THE CURRENT REFERENCE COMPLETES. AS LONG AS THIS FLAG IS SET THE CACHE WILL BE TURNED OFF AND ALL REFERENCES WILL GO DIRECTLY TO CORE.

8. NONEXISTENT MEMORY REFERENCES (PROCESSOR)

THE NXM RECOVERY LOGIC CAN BE TESTED BY HAVING THE PROCESSOR MAKE A REFERENCE TO A LOCATION THAT DOESN(T EXIST IN THE MEMORY ADDRESS SPACE.

IF A SYSTEM HAS 4 MILLION WORDS OF MEMORY THEN PART OF THE MEMORY WILL HAVE TO BE SWITCHED OFF LINE IN ORDER TO VERIFY THAT THE NXM LOGIC IS WORKING.

9. NONEXISTENT MEMORY REFERENCES (DMA20)

THE NXM RECOVERY LOGIC IN THE DMA20 CAN BE TESTED BY HAVING THE PROCESSOR MAKE A QUADWORD REFERENCE TO 4-WAY INTERLEAVED MEMORY WHICH HAS ONE OF THE 4 MEMORY BOXES SWITCHED OFF LINE. THE BOX THAT IS REFERENCED FIRST WHOULD NOT BE THE ONE THAT IS SWITCHED OFF LINE.

A QUADWORD READ REFERENCE CAN BE INITIATED BY DOING A READ FROM A CHCHE THAT HAS JUST HAD ALL ITS DATA INVALIDATED. THIS WILL CAUSE THE CACHE TO DO A FOUR WORD READ TO BRING THE QUADWORD INTO THE CACHE.

A QUADWORD WRITE REFERENCE CAN BE INITIATED BY DOING THE FOLLOWING:

1. SWEEP THE CACHE TO INVALIDATE ALL DATA (VALIDATE CORE IF NECESSARY).

2. WRITE INTO THE FOUR LOCATIONS OF THE QUADWORD THAT IS TO BE SENT TO MEMORY.

3. REFERENCE THE SAME LINE AS THE ONE CONTAINING THE QUADWORD THAT IS TO BE SENT TO MEMORY IN FOUR OTHER PAGES THAT ARE IN EXISTENT MEMORY.

THE LAST REFERENCE IN STEP 3 WILL CAUSE THE QUADWORD IN STEP TWO TO BE WRITTEN BACK TO MEMORY.

10. I/O PAGE FAIL

THE I?O PAGE FAIL LOGIC CAN BE TESTED BY HAVING THE 10-PROCESSOR WRITE A WORD WITH EVEN (BAD) PARITY INTO MEMORY. NEXT THE 10 SHOULD SIGNAL THE 11 TO DO AN EXAMINE OF THE LOCATION IN 10 MEMORY WITH THE BAD PARITY. THE 10 WILL GET AN I/O PAGE FAIL ERROR WHEN THE 11 EXAMINES THIS LOCATION.

11. EBUS PARITY 10 TO 11

THE 10 CAN FORCE BAD EBUS PARITY ON A TRANSFER TO THE 11 BY MOVING A BYTE OF GREATER THAN 16 BITS WITH AN ODD NUMBER OF ONE BITS IN THE LEFT HALF TO THE 11. THIS WILL IN EFFECT COMPLEMENT THE PARITY BIT SETN OVER THE EBUS TO THE 11 BECAUSE THE 11 COMPUTES PARITY ON ONLY THE RIGHT 16 BITS OF THE 36 BIT WORD SENT FROM THE 10.

12. DRAM AND CRAM PARITY

DRAM AND CRAM PARITY MUST BE TESTED FROM THE 11. PAERITY ERRORS CAN BE INSERTED INTO EITHER OF THE RAMS BY HAVING THE 11 CHANGE A BIT IN ONE OF THE RAM LOCATIONS.

13. FAST MEMORY PARITY

SEE THE PARAGRAPH ENTITLED "DATA PARITY".

[END OF CH1S06,SPC]

1080,2040,2060 ENGINEERING FUNCTIONAL SPEC - CHAP 2.2

TO: KL10 LIST, J. PARSLOW (200 FILE)

TITLE: USER INTERFACE TO EXTENDED ADDRESSING - CHAP 2.2

STATUS: THIS SPECIFICATION IS THE RESULT OF THE WORK OF THE EXTENDED ADDRESSING DESIGN GROUP (SEE BELOW) FORMED TO INVESTIGATE THE SIX PROPOSALS FOR EXTENDED ADDRESSING. THIS SPECIFICATION IS THE RESULT OF THAT WORK. IT WILL BE REVIEWED WIDELY ON WEDNESDAY, 5 MARCH 75, FROM 10-12 AM, IN THE LEPRECHAUN LOUNGE IN MARLBORO. WARNING: SINCE THIS CAPABILITY HAS NOT BEEN ANNOUNCED, IT MUST BE TREATED AS COMPANY CONFIDENTIAL.

FILE: [EFS]CH2S02.SPC

PDM #: 200-200-026-08

DATE: 17 FEB 76

SUPERSEDED MEMOS: ALL PREVIOUS REVISIONS OF PROPOSAL # 4; AS WELL AS PROPOSALS 1,2,3; EXTENDED ADDRESSING FOR KL10, D. MURPHY AND A. KOTOK, 22 FEB 73.

SUPERSEDED SPECS: ORIGINAL CHAP 2.2 DATED 11 SEPT 74 WITH PDM # 200-200-020-00.

ENGINEER: EXTENDED ADDRESSING DESIGN GROUP (SEE BELOW)

APPROVED:

EDITOR: T. HASTINGS

TYPIST: J. MCCARTHY

REVIEWED:

DISTRIBUTED:

ABSTRACT

THIS CHAPTER CONCENTRATES ON THE USER INTERFACE TO EXTENDED ADDRESSING. THE SPECIFICATION PERMITS A VIRTUAL ADDRESS SPACE OF UP TO 1 BILLION WORDS (30-BIT) ADDRESS. THIS IS ACCOMPLISHED BY PROVIDING UP TO 4096 256K ADDRESS SPACES CALLED SECTIONS. THE KL10 WILL IMPLEMENT ONLY 32 USER SECTIONS (8 MILLION WORDS). THE PROGRAMMER MAY CONSIDER EACH SECTION TO BE SEPARATE OR CONTIGUOUS. A POSITIVE NON-ZERO LH OF AN XR IS USED TO SPECIFY A SECTION NUMBER. A NEGATIVE OR ZERO LH OF AN XR IS USED FOR LOCAL REFERENCES. A SECOND FORM OF INDIRECT WORD PERMITS INDIRECTION, INDEXING, AND A 30-BIT ADDRESS.

REVISION HISTORY

REV	DESCRIPTION	CHG NO	ORIG	DATE	APPD BY	DATE
0	EXCT SCHEME			11 SEP 74		
5	PROPOSAL #4			25 FEB 75		
6	PROPOSAL #4 - TYPOS			26 FEB 75		
7	IMPLEMENTATION DETAILS			28 JAN 76		
8	MINUTES OF REVIEW			17 FEB 76		

PREFACE

THIS SPECIFICATION REPRESENTS THE WORK OF A NUMBER OF PEOPLE OVER THE LAST TWO YEARS. THE EXTENDED ADDRESSING DESIGN GROUP IS L. DICKMAN, T. EGGERS, L. FEHSHENS, T. HASTINGS, L. HUGHES, A. KOTOK, D. LEWINE, P. LIPMAN, D. MURPHY, R. REID, D. RODGERS, G. STEIL, R. STEWART, W. STRECKER. IN ADDITION D. MOON AND G. BENEDICT MADE SIGNIFICANT CONTRIBUTIONS. THE DESIGN GROUP CONSIDERED 6 DIFFERENT PROPOSALS IN THE PROCESS OF ITS WORK [16, 17, 18]. WE BELIEVE THIS PROPOSAL TO BE SUPERIOR TO THE OTHERS. HOWEVER SINCE THIS ADDRESSING WILL APPEAR IN EVERY INSTRUCTION OF FUTURE PROGRAMS, IT IS IMPORTANT THAT THIS SPECIFICATION HAVE A CAREFUL AND WIDE REVIEW TO MAKE SURE THAT WE CAN LIVE WITH IT FOR ALL FUTURE MACHINES. WE HAVE ALSO LEARNED THAT ADDRESSING IS A BASIC PART OF A SYSTEM ARCHITECTURE AND SO TAKES A LOT OF THOUGHT, BUT IS WORTH THE EFFORT.

1. SUMMARY

THE KI10 AND ITS PREDECESSORS IMPLEMENTED A 256K VIRTUAL ADDRESS SPACE WITH EACH INSTRUCTION SPECIFYING AN 18-BIT ADDRESS. THIS EXTENDED ADDRESSING SPECIFICATION INCREASES THE USER VIRTUAL ADDRESS SPACE TO 1 BILLION WORDS BY ALLOWING THE USER PROGRAM TO ADDRESS UP TO 4096 256K ADDRESS SPACES, CALLED SECTIONS. EACH INSTRUCTION SPECIFIES A 30-BIT ADDRESS, IMPLICITLY OR EXPLICITLY. THE KL10 WILL IMPLEMENT ONLY 32 SECTIONS (8 MILLION WORDS).

THE PROGRAMMER MAY USE THE SECTIONS AS SEPARATE LOGICAL ENTITIES OR AS A SINGLE CONTIGUOUS ADDRESSING SPACE. ARRAYS, STRINGS, AND PUSH DOWN STACKS MAY BE ARBITRARILY LONG AND CROSS SECTION BOUNDARIES. HOWEVER CODE MUST TRANSFER CONTROL BETWEEN SECTIONS EXPLICITLY AND CANNOT FLOW ACROSS SECTION BOUNDARIES.

IN ORDER TO SPECIFY A 30-BIT ADDRESS, THE INDEXING AND INDIRECTION OF THE MACHINE HAVE BEEN MODIFIED WHEN THE PC IS IN A NON-ZERO SECTION. HOWEVER THE MODIFICATION HAS BEEN MADE SUCH THAT NEW SUBROUTINES CAN BE WRITTEN SO THAT THEY CAN RUN ON A KI OR A KL. IN ADDITION, SECTION 0 IS CALLED THE KI COMPATIBLE SECTION. WHEN THE PC IS IN SECTION 0, INDEXING AND

INDIRECTION WORK THE SAME AS ON THE KI. IMMEDIATE MODE INSTRUCTIONS CONTINUE TO USE 18-BIT OPERANDS IN ALL SECTIONS.

THE LEFT HALF (BITS 6-17) OF A 30-BIT ADDRESS SPECIFIES THE 12-BIT SECTION NUMBER AND THE RIGHT HALF (BITS 18-35) SPECIFIES AN 18-BIT ADDRESS-WITHIN-SECTION. INSTRUCTIONS WHICH SPECIFY NO INDEXING OR INDIRECTION ARE CALLED LOCAL REFERENCES BECAUSE THEY REFERENCE THE PC SECTION IMPLICITLY. FURTHERMORE INDEXING IN WHICH THE LEFT HALF IS NEGATIVE OR 0 IS CALLED LOCAL INDEXING BECAUSE IT ALSO REFERENCES THE PC SECTION IMPLICITLY. LOCAL INDEXING IS PROVIDED FOR POSITIVE OR NEGATIVE INDEXING OF ARRAYS IN THE CURRENT SECTION AND FOR USING THE AOBJN LOOP INSTRUCTION. LOCAL INDEXING NEVER CROSSES SECTION BOUNDARIES.

INDEXING IN WHICH THE LEFT HALF IS POSITIVE AND NON-ZERO IS CALLED GLOBAL INDEXING BECAUSE THE LH IS INTERPRETED AS AN EXPLICIT SECTION NUMBER. THE 18-BIT RH OF THE INSTRUCTION (OR INDIRECT WORD) IS SIGN EXTENDED AND ADDED TO THE INDEX REGISTER. THEREFORE THE FINAL SECTION NUMBER MAY BE THE SAME AS THE LH OF THE INDEX REGISTER, 1 GREATER, OR 1 LESS. THUS THE RH OF THE INSTRUCTION CAN BE A POSITIVE OR NEGATIVE OFFSET (MAX. MAGNITUDE OF 2 TO THE 17) TO THE GLOBAL ADDRESS IN THE INDEX REGISTER. GLOBAL INDEXING CROSSES SECTION BOUNDARIES.

IN ADDITION TO THE KI STYLE INDIRECT WORDS (CALLED INSTRUCTION FORMAT INDIRECT WORDS - IFIW), AN EXTENDED FORMAT INDIRECT WORD (EFIW) IS PROVIDED. BIT 0 = 1 MEANS IFIW, 0 MEANS EFIW. IN AN EFIW, THE INDIRECT AND INDEX FIELDS ARE IN BITS 1 AND 2-5, INSTEAD OF BITS 13 AND 14-17. BITS 6-17 OF THE EFIW SPECIFY AN EXPLICIT SECTION NUMBER. FURTHERMORE THE INDEXING IS ALWAYS GLOBAL SINCE THE 30-BIT SUM OF THE EFIW AND THE INDEX REGISTER IS USED (WITH CARRIES OUT OF BIT 18 PROPAGATED TO BIT 17).

THUS THE INDEX REGISTER CAN CONTAIN A GLOBAL ADDRESS AND THE EFIW A (POSSIBLY LARGE) SIGNED OFFSET OR VICE VERSA. CONCEPTUALLY IN AN EFIW, BITS 6-35 OF THE INDEX REGISTER AND INDIRECT WORD ARE TREATED IN THE SAME WAY AS BITS 18-35 OF THE INDEX REGISTER AND INDIRECT WORD ON A KI.

IN ADDITION TO THE ABOVE MODIFICATIONS TO INDEXING AND INDIRECTION, SEVERAL INSTRUCTIONS HAVE ALSO BEEN MODIFIED (WHEN PC IS IN A NON-ZERO SECTION). THE STACK POINTER CAN BE GLOBAL OR LOCAL DEPENDING ON WHETHER ITS LEFT HALF IS GREATER THAN 0 OR NOT. PC STORING INSTRUCTIONS STORE A GLOBAL ADDRESS WITH NO FLAGS. BYTE POINTERS ARE SINGLE-WORD IF BIT 12=0, OR DOUBLE WORD IF BIT 12=1, WITH THE SECOND WORD CONTAINING A GLOBAL ADDRESS. LUUOS CAN STORE AND DISPATCH TO GLOBAL ADDRESSES. A NEW INSTRUCTION, MOVE EXTENDED ADDRESS (XMOVEI), IS PROVIDED AS A 30-BIT ANALOG TO MOVEI. A FLAGS AND PC DOUBLE-WORD IS DEFINED MAINLY FOR MONITOR USE.

2. GOALS

1. BE SOMETHING WITH WHICH WE WILL WANT ALL FUTURE MACHINES TO BE UPWARD COMPATIBLE.

2. BE ABLE TO RUN KI CODE WITHOUT MODIFICATION (IN SECTION 0 ONLY).

3. BE ABLE TO WRITE NEW SUBROUTINES WHICH WILL WORK IN EXTENDED SECTIONS, THE KI COMPATIBLE SECTION AND THE KI ITSELF.

4. MINIMIZE THE SPACE AND TIME OVERHEAD FOR EXTENDED ADRESSING SUFFICIENTLY SO THAT ALL FUTURE SUBROUTINES COULD REALISTICALLY MEET GOAL 3.

5. PRESERVE THE INTENT AND USAGE OF EACH OF THE INSTRUCTIONS IN THE KI ORDER CODE.

6. ALLOW THE PROGRAMMER TO CHOOSE WHETHER TO DIVIDE HIS ADDRESS SPACE INTO SEPARATE LOGICAL DIVISIONS OR TO CONSIDER SECTIONS TO BE LOGICALLY CONTIGUOUS. THUS ARRAYS, STRINGS, AND PUSHDOWN STACKS CAN CROSS SECTION BOUNDARIES AND CAN BE MANY SECTIONS LONG.

7. MAKE INDEX REGISTERS, INDIRECT WORDS, BYTE POINTERS, THE PC, AND STACK POINTERS BE AS SIMILAR TO EACH OTHER AS POSSIBLE.

3. NON-GOALS

1. THE USER CANNOT FREELY MIX EXISTING KI CODE WITH EXTENDED ADDRESSING. SECTION 0 CANNOT CALL OTHER SECTIONS EXCEPT VIA THE MONITOR.

2. THIS SCHEME DOES NOT SOLVE THE PROTECTION PROBLEMS OF A DOMAIN ARCHITECTURE. THAT SHOULD BE CONSIDERED FOR A FUTURE MACHINE.

3. CODE CANNOT FLOW ACROSS SECTION BOUNDARIES. THE PROGRAM MUST EXPLICITLY TRANSFER CONTROL FROM ONE SECTION TO ANOTHER.

4. TERMINOLOGY

LOCAL ADDRESS - AN 18-BIT QUANTITY SPECIFYING AN ADDRESS WITHIN THE SECTION IN WHICH THE QUANTITY IS STORED.

GLOBAL ADDRESS - A 30-BIT QUANTITY SPECIFYING A 12-BIT SECTION NUMBER AND AN 18-BIT ADDRESS WITHIN SECTION.

INSTRUCTION FORMAT INDIRECT WORD (IFIW) - A FORM OF INDIRECT WORD (BIT 0-1 = 10) IN WHICH THE INDIRECT AND INDEX FIELDS ARE SPECIFIED AND HANDLED THE SAME AS IN AN INSTRUCTION, I.E., BITS 13 AND 14-17. THE INDEX REGISTER CONTAINS A LOCAL ADDRESS (OR HALF WORD OFFSET) IF THE LH (XR) IS NEGATIVE OR ZERO. IT CONTAINS A GLOBAL ADDRESS IF THE LH IS POSITIVE AND NON-ZERO.

EXTENDED FORMAT INDIRECT WORD (EFIW) - AN ALTERNATE FORM OF INDIRECT WORD (BIT 0 = 0) IN WHICH THE INDIRECT AND INDEX FIELDS ARE SPECIFIED BY BITS 1 AND 2-5. BITS 6-17 SPECIFY A SECTION NUMBER. THE INDEX REGISTER CAN CONTAIN A 30-BIT GLOBAL ADDRESS AND THE EFIW A 30-BIT OFFSET OR VICE-VERSA. THE TWO 30-BIT QUANTITIES ARE ADDED TOGETHER (WITH CARRIES OUT OF BIT 18 INTO BIT 17).

LOCAL INDEXING - INDEXING WHICH REMAINS WITHIN THE SECTION IN WHICH THE WORD SPECIFYING THE INDEX FIELD RESIDES. LOCAL INDEXING IS PERFORMED IN SECTION 0 (ALWAYS) AND IN NON-ZERO SECTIONS (IN INSTRUCTIONS OR INSTRUCTION FORMAT INDIRECT WORDS) WHEN THE LEFT HALF OF THE INDEX REGISTER IS NEGATIVE OR 0.

GLOBAL INDEXING - INDEXING IN WHICH THE INDEX REGISTER MAY SPECIFY A SECTION NUMBER IN ITS LEFT HALF. INDEXING IN INSTRUCTIONS AND IFIWS IS GLOBAL IF THE LH OF THE INDEX REGISTER IS POSTIVE AND NON-ZERO. INDEXING IN EFIWS IS ALWAYS GLOBAL.

GLOBAL PC - THE ONE-WORD PROGRAM COUNTER CONTAINING A GLOBAL ADDRESS. BITS 0-5 ARE ZERO. NO FLAGS ARE INCLUDED.

LOCAL STACK POINTER - A ONE WORD POINTER TO THE END OF THE STACK IN THE CURRENT PC SECTION. THE LH HAS A NEGATIVE COUNT OF THE NUMBER OF WORDS LEFT UNTIL OVERFLOW AND IS IN LOCAL INDEXING FORMAT.

GLOBAL STACK POINTER - A ONE WORD POINTER TO THE END OF THE STACK WHICH MAY BE IN ANY SECTION. THE LH IS GREATER THAN 0, AND IS A GLOBAL ADDRESS.

LOCAL BYTE POINTER - A ONE WORD BYTE POINTER AS ON THE KI10 WITH THE ADDITION THAT BIT 12 = 0. INDEXING AND INDIRECTION FOLLOW THE RULES FOR INSTRUCTIONS.

GLOBAL BYTE POINTER - A TWO WORD BYTE POINTER IN WHICH BIT 12 = 1. THE SECOND WORD CONTAINS A GLOBAL ADDRESS.

5. GENERAL SPECIFICATIONS

THE EXISTING PDP-10 SERIES OF MACHINES IMPLEMENT AN 18-BIT, 256K-WORD VIRTUAL ADDRESS SPACE. ALL INSTRUCTIONS WHICH USE A MEMORY ADDRESS MAY BE GIVEN AN 18-BIT ADDRESS. THE KI10 IS THE REFERENCE POINT FOR THE DISCUSSION IN THIS DOCUMENT; INSTRUCTIONS ARE INTERPRETED AS IMPLEMENTED ON THE KI10 UNLESS OTHERWISE SPECIFIED.

THIS DOCUMENT IS NOT RELATED TO AND DOES NOT REFERENCE ANY PREVIOUS DOCUMENT ON THE SUBJECT OF EXTENDED ADDRESSING INSOFAR AS THE SPECIFICATION OF INSTRUCTION INTERPRETATION IS CONCERNED. IT IS INTENDED TO BE A COMPLETE DOCUMENT. THE BODY OF THIS SPECIFICATION CONTAINS ALL OF THE INFORMATION WHICH SHOULD EVENTUALLY APPEAR IN THE HARDWARE REFERENCE MANUAL, INCLUDING RATIONALE FOR MANY OF THE DESIGN DECISIONS. THE APPENDIX CONTAINS EXPLANATIONS FOR REJECTED ALTERNATIVES FLAGGED IN THE TEXT WITH BRACKETS.

5.1 THE VIRTUAL ADDRESS SPACE

THIS DOCUMENT PROVIDES A SPECIFICATION FOR INSTRUCTION INTERPRETATION SUCH AS TO IMPLEMENT A 30-BIT (1-BILLION WORD) ADDRESS SPACE. THE VIRTUAL ADDRESS CONSISTS OF TWO PARTS, THE HIGH-ORDER 12 BITS CALLED THE "SECTION", AND THE LOW-ORDER 18-BITS CALLED THE "WORD-WITHIN-SECTION" (OR JUST "WORD"). THE USER AND MONITOR ADDRESS SPACES ARE INDEPENDENT, EACH CONSISTING OF 4096 SECTIONS OF 256K WORDS[1].

ON THE KL10 PROCESSOR, ONLY 5 BITS OF SECTION NUMBER WILL BE IMPLEMENTED, THUS RESTRICTING USE OF THE ADDRESS SPACE TO THE FIRST 32 SECTIONS. THE KL10 HARDWARE WILL TRAP ANY REFERENCE TO A LOCATION OUTSIDE OF THE FIRST 32 SECTIONS.

THIS DESIGN IS BASED ON THE PREMISE THAT NO CLEAN AND EFFICIENT IMPLEMENTATION OF EXTENDED ADDRESSING IS POSSIBLE WHICH ALLOWS EXISTING KI10 CODE TO BE FREELY INTERMIXED WITH CODE EMPLOYING EXTENDED ADDRESSES. THEREFORE, A DISTINCTION IS MADE BETWEEN SECTION 0 AND ALL OTHER SECTIONS AS FOLLOWS:

> CODE BEING RUN IN EXEC SECTION 0 OR USER SECTION 0 WILL BE INTERPRETED EXACTLY AS IT IS ON THE KI10, USER CODE IN USER SECTION 0 WILL NOT BE ABLE TO ACCESS OTHER SECTIONS EXCEPT VIA NEWLY DEFINED MONITOR CALLS. MONITOR CODE IN EXEC SECTION 0 WILL BE ABLE TO ACCESS ANY SECTION IN ITS CALLER USING PXCT. PXCT IS DOCUMENTED ELSEWHERE. CODE BEING RUN IN SECTIONS OTHER THAN 0 WILL BE INTERPRETED IN CERTAIN WAYS WHICH ARE INCOMPATIBLE WITH THE KI10 INTERPRETATION BUT WHICH ALLOW CONVENIENT USE OF THE EXTENDED ADDRESS SPACE. SECTION 0 (EXEC AND USER) IS CALLED THE KI COMPATIBLE SECTION. NON-ZERO SECTIONS ARE CALLED EXTENDED SECTIONS.
>
> HOWEVER, USER CODE IN NON-ZERO SECTIONS AND MONITOR CODE IN ANY EXEC SECTION WILL BE ABLE TO REFERENCE AND TRANSFER CONTROL TO SECTION 0. ONCE THE EFFECTIVE ADDRESS COMPUTATION ENTERS SECTION 0, FURTHER INDEXING AND INDIRECTION WILL CONTINUE FOLLOWING THE RULES FOR SECTION 0. THUS PROGRAMS (MONITOR ESPECIALLY) DO NOT NEED TO KNOW WHICH SECTION CALLED.

ALTHOUGH EXISTING KI CODE CANNOT BE FREELY INTERMIXED WITH CODE EMPLOYING EXTENDED ADDRESSES, IT IS POSSIBLE TO WRITE NEW SUBROUTINES EMPLOYING EXTENDED ADDRESSING WHICH WILL ALSO RUN ON THE KI10. THUS THE SUPPORT COSTS FOR EXTENDED ADDRESSING WILL BE GREATLY REDUCED.

5.2 OVERVIEW OF SOFTWARE USE OF EXTENDED ADDRESSING

THE SECTION ARCHITECTURE BEARS SOME SIMILARITY TO EXTENDED ADDRESS SCHEMES, SEGMENTATION SCHEMES, AND DOMAIN ARCHITECTURE SCHEMES OF OTHER MACHINES. HOWEVER, IT DOES NOT PROPERLY REPRESENT SEGMENTATION, NOR TRUE DOMAIN ARCHITECTURE [20]. IT INCREASES THE ADDRESS SPACE TO 8M WORDS ON THE KL10. HOWEVER EXPANSION TO A 30-BIT ADDRESS (1 BILLION WORDS) WILL BE POSSIBLE IN FUTURE MACHINES WITH NO SOFTWARE COMPATIBILITY PROBLEMS. BECAUSE THE KL10 IS ONLY IMPLEMENTING 32 SECTIONS PER PROCESS, IT IS NOT POSSIBLE TO PUT EVERY PROCEDURE AND DATA ARRAY IN A SEPARATE SECTION LIKE SOME SEGMENTATION SCHEMES. WE WILL CONTINUE TO USE LINK10 TO RELOCATABLY LOAD AND LINK PROCEDURES AND DATA INTO ONE OR A SMALL NUMBER OF SECTIONS. PROGRAMS CAN BE WRITTEN TO PUT A NUMBER OF DATA ARRAYS IN DIFFERENT SECTIONS. THIS WILL HELP THE DYNAMIC ALLOCATION PROBLEM WHEN THE MAXIMUM SIZE OF THE ARRAYS IS UNKNOWN. UNLIKE MOST SEGMENTATION SCHEMES, AN ARRAY CAN SPAN ACROSS SECTIONS AND CAN BE LARGER THAN 256K. PROGRAMS CAN DEPEND ON THE OPERATING SYSTEM'S VIRTUAL MEMORY CAPABILITY INSTEAD OF DOING EXPLICIT I/O. COMMONLY USED ROUTINES WILL BE LINKED TOGETHER TO FORM A SMALL NUMBER OF POTENTIALLY LARGE LIBRARY SECTIONS. THESE PROCEDURES WILL FOLLOW THE PROCEDURE STANDARDS CURRENTLY BEING DEVELOPED. RUN TIME SYSTEMS, COMPATIBILITY PACKAGES, COMMAND LANGUAGE EXECS, AND DEBUGGERS ARE ALSO EXAMPLES OF PROGRAMS WHICH CAN USE THE SECTION ARCHITECTURE. FINALLY VERY LARGE PROGRAMS CAN BE WRITTEN TO RUN IN MULTIPLE SECTIONS WITHOUT OVERLAYS.

NO PROTECTION IS PROVIDED FOR INTER-SECTION CALLS OR REFERENCES BEYOND WHAT IS AVAILABLE WITHIN A SECTION. THUS ANY SECTION CAN REFERENCE ANY OTHER SECTION. PUBLIC PROGRAMS CANNOT REFERENCE PRIVATE PAGES. PUBLIC PROGRAMS MUST TRANSFER CONTROL TO PRIVATE PAGES THROUGH PORTAL INSTRUCTIONS. PROTECTION IS ASSOCIATED WITH THE PAGE TABLE. THUS EACH USER PROCESS HAS ITS OWN PAGE TABLE FOR EACH SECTION. HOWEVER A SECTION WILL TYPICALLY BE MADE UP OF A NUMBER OF STORAGE CLASSES AS IN A DOMAIN ARCHITECTURE. THESE WILL INCLUDE READ-ONLY CODE AND DATA, PROCESS OWN STORAGE (PUSHDOWN STACK), INCARNATION OWN STORAGE (COPY ON WRITE), AND DOMAIN OWN STORAGE (FILES MAPPED INTO THE ADDRESS SPACE) [20].

SECTION NUMBERS WILL NOT BE ASSIGNED AT CODING OR COMPILE TIME. THEY WILL BE ASSIGNED AT RUN TIME FOR MOST APPLICATIONS SUCH AS LIBRARIES AND AT LINK TIME FOR LARGE OWN ARRAYS DECLARED TO BE IN A SECTION OTHER THAN THE CODE SECTION. PROCEDURES WILL BE WRITTEN AS SECTION-INDEPENDENT CODE. THE SAME SHARED COPY OF A PROCEDURE CAN BE ASSIGNED TO DIFFERENT SECTION NUMBERS IN TWO PROCESSES. ALL INTER-SECTION LINKAGE POINTERS BETWEEN PROCEDURES MUST BE STORED IN PER-PROCESS STORAGE.

6. EFFECTIVE ADDRESS COMPUTATION

ALL PDP-10 INSTRUCTIONS COMPUTE AN EFFECTIVE ADDRESS. THIS CHAPTER DESCRIBES THE EFFECTIVE ADDRESS COMPUTATION PERFORMED WHEN THE PC IS IN A NON-0 SECTION. THE EFFECTIVE ADDRESS COMPUTATION CONSISTS OF (OPTIONAL) INDEXING AND (OPTIONAL) INDIRECTING IN THAT ORDER, WITH THE POTENTIAL OF ITERATING THESE TWO STEPS AN ARBITRARY NUMBER OF TIMES. IF NO INDEXING OR INDIRECTION IS SPECIFIED, THE EFFECTIVE ADDRESS IS ASSUMED TO BE IN THE CURRENT SECTION AND IS CALLED A LOCAL REFERENCE. THERE ARE TWO FORMS OF INDEXING, (1) INSTRUCTION FORMAT INDEXING AND (2) EXTENDED FORMAT INDEXING. SEE ATTACHED WORD FORMAT PICTURES.

EXAMPLE - LOCAL REFERENCE (NO INDEXING OR INDIRECTION)

IN SECTION 22:

```
        MOVE    T,1000
```

MOVES LOCATION 22,,1000 TO T, I.E., LOCATION 1000 IN THE CURRENT SECTION WHICH IS 22.

6.1 INSTRUCTION FORMAT INDEXING

INSTRUCTION FORMAT INDEXING OCCURS IN INSTRUCTIONS AND IN INSTRUCTION FORMAT INDIRECT WORDS (IFIW). IFIWS ARE KI-STYLE INDIRECT WORDS WITH BITS 0-1 = 10 [15]. WHEN THE INDEX FIELD OF AN INSTRUCTION(*) IS NON-0, THE CONTENTS OF THE DESIGNATED REGISTER IS EXAMINED.

1. LOCAL INDEXING

   IF THE LEFT-HALF IS NEGATIVE OR 0, LOCAL INDEXING IS PERFORMED. THIS MEANS THAT THE RIGHT HALF OF THE INDEX IS ADDED TO THE RIGHT HALF OF THE INSTRUCTION(*), AND THE SECTION NUMBER FROM WHICH THE INSTRUCTION(*) WAS FETCHED IS INSERTED AS THE SECTION NUMBER OF THE RESULTING INTERMEDIATE ADDRESS. THUS SMALL INDICES (MAGNITUDE LESS THAN 2 TO THE 18TH) AND AOBJN POINTERS CAN BE USED FOR ARRAYS IN THE CURRENT SECTION. LOCAL INDEXING ALWAYS STAYS IN THE SAME SECTION. THUS ARRAYS ADDRESSED WITH LOCAL INDEXING CANNOT CROSS SECTION BOUNDARIES.

2. GLOBAL INDEXING

   IF THE LEFT-HALF IS POSITIVE AND NON-0, THE ENTIRE INDEX IS ADDED TO THE RIGHT HALF OF THE INSTRUCTION(*) WITH BIT 18 OF THE INSTRUCTION SIGN EXTENDED TO BITS 0-17. THE LOW-ORDER 30-BITS OF THE RESULT ARE TAKEN AS THE INTERMEDIATE ADDRESS. THE CONTENTS OF THE INDEX REGISTER CAN BE THOUGHT OF AS A GLOBAL ADDRESS (SECTION IN LH, ADDRESS WITHIN SECTION IN RH), WHILE BITS 18-35 OF THE INSTRUCTION(*) ARE A POSITIVE OR NEGATIVE OFFSET (MAXIMUM MAGNITUDE OF 2 TO THE 17TH). THUS THE RESULTING SECTION NUMBER MAY BE THE SAME AS THE LH OF THE XR, 1 GREATER, OR 1 LESS. GLOBAL INDEXING IGNORES SECTION BOUNDARIES AND NEVER WRAPS AROUND. CONTRAST THIS WITH LOCAL INDEXING WHICH OBSERVES SECTION BOUNDARIES BY WRAPPING AROUND.

+------------+

(*) INSTRUCTION OR INSTRUCTION FORMAT INDIRECT WORD (IFIW)

EXAMPLE - LOCAL INDEXING

IN SECTION 22:

```
        MOVEI   I,100
        MOVE    T1,1000(I)
```

MOVES 22,,1100 TO T1, IE,, 100TH ENTRY IN ARRAY STARTING AT 1000 IN CURRENT SECTION (22).

EXAMPLE - NEGATIVE INDEXING

IN SECTION 22:

```
        MOVNI   I,100
LOOP:   ADD     T,1000(I)
        AOJL    I,LOOP
```

ADDS LOCATIONS 22,,700 THROUGH 22,,777 IN THE CURRENT SECTION, IE,, SECTION 22.

EXAMPLE - AOBJN LOOP

IN SECTION 22:

```
        MOVSI   I,-LENGTH
LOOP:   ADD     T,1000(I)
        AOBJN   I,LOOP
```

ADDS ALL LOCATIONS IN ARRAY STARTING AT 1000 IN CURRENT SECTION (22). NOTE THAT ARRAY CANNOT CROSS SECTION BOUNDARIES SINCE LOCAL INDEXING IS USED.

EXAMPLE - GLOBAL INDEXING

IN ANY (NON-0) SECTION:

```
MOVE    T,[22,,1000]
ADD     T1,100(T)
```

ADDS THE 100TH LOCATION OF CONTROL BLOCK STARTING AT 22,1000, IE,, LOCATION 22,,1100.

EXAMPLE - GLOBAL INDEXING WITH NEGATIVE OFFSET

IN ANY (NON-0) SECTION:

```
MOVE    T,[22,,1000]
ADD     T1,-100(T)
```

ADDS THE -100TH LOCATION OF CONTROL BLOCK STARTING AT 22,,1000, IE,, LOCATION 22,,700. NOTE THAT IN THE GLOBAL INDEXING CASES, THE CONTROL BLOCK CAN CROSS A SECTION BOUNDARY SINCE CARRIES ARE NOT SUPPRESSED OUT OF BIT 18.

6,2 INDIRECTION

IF THE INDIRECT BIT (BIT 13) OF THE INSTRUCTION IS A ONE, THE INTERMEDIATE ADDRESS (CALCULATED FROM INDEXING IF ANY) IS USED TO FETCH AN INDIRECT WORD WHICH IS INTERPRETED IN ONE OF FOUR WAYS DEPENDING ON A TWO BIT CODE IN BITS 0-1 OF THE INDIRECT WORD[11].

CODE	INDIRECT WORD TYPE
10	INSTRUCTION FORMAT INDIRECT WORD (IFIW)

NO EXPLICIT SECTION NUMBER APPEARS. INSTEAD THE SECTION NUMBER IS THE SAME AS THAT CONTAINING THE INDIRECT WORD. THE I, X, AND Y FIELDS ARE INTERPRETED AS THEY ARE IN AN INSTRUCTION, AND ADDITIONAL INDEXING AND/OR INDIRECTION MAY OCCUR. INDEXING IN AN IFIW FOLLOWS THE SAME RULES AS FOR INSTRUCTIONS, NAMELY LOCAL AND GLOBAL INDEXING ARE PERMITTED. THE SOFTWARE IS FREE TO STORE INFORMATION IN BITS 2-12. FUTURE HARDWARE WILL NOT USE BITS 2-12 IN IFIWS. THE DEC PROCEDURE CALLING STANDARD DESCRIBES THE SOFTWARE USE OF THESE BITS IN ARGUMENT LISTS FOR DEFINING ARGUMENT TYPES.

00,01	EXTENDED FORMAT INDIRECT WORD (EFIW)

THIS FORM IS PROVIDED FOR THE CONVENIENCE OF ADDRESSING LARGE ARRAYS AND FOR REPRESENTING A 30-BIT GLOBAL ADDRESS IN A WORD. BIT 1 IS INTERPRETED AS AN INDIRECT BIT. BITS 2-5 ARE INTERPRETED AS AN INDEX FIELD. BITS 6-17 ARE INTERPRETED AS AN EXPLICIT SECTION NUMBER. THE ENTIRE INDEX REGISTER (POSITIVE OR NEGATIVE) IS ADDED TO THE INDIRECT WORD TO FORM A 30-BIT SUM. THUS THE INDEX REGISTER CAN CONTAIN A GLOBAL ADDRESS AND THE EFIW A (POSSIBLY LARGER THAN A SECTION) SIGNED OFFSET OR VICE VERSA. CONCEPTUALLY IN AN EFIW, BITS 6-35 OF AN INDEX REGISTER WORD ARE TREATED THE SAME AS BITS 18-35 OF AN INDEX REGISTER AND INDIRECT WORD ON THE KI. FURTHER INDIRECTION CAN OCCUR. EFIWS ARE USED FOR HANDLING ARRAYS WHICH ARE BIGGER THAN A SECTION [19].

11 RESERVED FOR FUTURE

BITS 2-35 MAY BE USED BY FUTURE HARDWARE FOR ANY PURPOSE. THE SOFTWARE MUST NOT USE THIS TYPE OF INDIRECT WORD FOR ANYTHING. THE HARDWARE WILL TRAP (PAGE FAIL, CODE 24) WHENEVER THIS COMBINATION IS ENCOUNTERED IN AN EFFECTIVE ADDRESS COMPUTATION IN A NON-ZERO SECTION. THE MONITOR CANNOT PERFORM ANY USER SERVICE AS A RESULT OF THIS TRAP, INCLUDING TRAPPING TO THE USER. OTHERWISE THE HARDWARE COULD NOT USE THIS FORMAT IN A FUTURE MACHINE WITHOUT COMPATIBILITY PROBLEMS.

NOTE THAT THE SENSE OF THE SIGN BIT CORRESPONDS TO THE SIGN BIT OF AN INDEX REGISTER, IE., 0 MEANS GLOBAL AND 1 MEANS LOCAL (15). THUS INDIRECT WORDS CAN BE USED AS INDEXES AND VICE VERSA IN MANY CASES. THE TWO TYPES OF INDIRECT WORDS CAN BE MIXED IN AN INDIRECT CHAIN. IN FACT, AN ARGUMENT LIST CONTAINING A POINTER TO ANOTHER SECTION WILL DO SO WITH AN IFIW POINTING TO AN EFIW.

THE FOREGOING MEANS THAT IF AN INSTRUCTION SPECIFIES NO INDEXING OR INDIRECTION, THEN THE SECTION NUMBER OF THE EFFECTIVE ADDRESS WILL BE DEFAULTED TO THE SECTION FROM WHICH THE INSTRUCTION WAS FETCHED. IF AN INSTRUCTION FORMAT INDIRECT WORD SPECIFIES NO INDEXING OR INDIRECTION, THEN THE SECTION NUMBER OF THE EFFECTIVE ADDRESS WILL BE DEFAULTED TO THE SECTION FROM WHICH THE INDIRECT WORD WAS FETCHED. IN GENERAL, WHENEVER A DEFAULT SECTION IS REQUIRED DURING AN EFFECTIVE ADDRESS COMPUTATION, IT IS TAKEN TO BE THE SECTION FROM WHICH THE MOST RECENT INSTRUCTION OR INDIRECT WORD WAS TAKEN. IF THE LAST CYCLE OF AN EFFECTIVE ADDRESS COMPUTATION NEEDS A DEFAULT SECTION NUMBER, THE RESULTING EFFECTIVE ADDRESS IS SAID TO BE LOCAL, RATHER THAN GLOBAL.

EXAMPLE - EXTENDED FORMAT INDIRECTION

IN ANY (NON-0) SECTION:

```
        MOVE    T,@[30,,1000]
```

LOADS T WITH THE CONTENTS OF LOCATION 1000 IN SECTION 30.

EXAMPLE - EXTENDED FORMAT INDIRECTION WITH INDEXING

```
        MOVEI   J,100
        MOVE    T,@[EFIW 30,1000(J)]
```

LOADS T WITH THE CONTENTS OF LOCATION 1100 IN SECTION 30. NOTE THAT THE EFIW PSEUDO-OP CAUSES THE ASSEMBLER TO SWAP THE EXPRESSION INSIDE PARENTHESES AROUND BIT 6 INSTEAD OF BIT 18. THUS J APPEARS IN BITS 2-5 OF THE EFIW, INSTEAD OF 14-17.

EXAMPLE - ARRAY IN ANOTHER SECTION

```
        MOVE    C,[2000000-1]   ;2 SECTIONS WORTH
LOOP:   ADD     T,@[EFIW 30,1000(C)]
        SOJGE   C,LOOP
```

ADDS THE 512K ARRAY FROM 30,,1000 THROUGH 32,,777. NOTE THAT EVEN IF THE ARRAY HAD BEEN LESS THAN 2 TO THE 17TH WORDS LONG AND DIDN'T CROSS ANY SECTION BOUNDARIES, IT WOULD STILL NOT BE POSSIBLE TO USE AOBJN FOR THE LOOP. THIS IS BECAUSE THE ENTIRE INDEX REGISTER IS ALWAYS ADDED IN AN EFIW[12].

EXAMPLE - NEGATIVE INDEXING A LARGE ARRAY

```
        MOVE    C,[-2000000+1] ;2 SECTIONS WORTH
LOOP:   ADD     T,@[EFIW 32,1000(C)]
        AOJLE   C,LOOP
```

ADDS THE 512K ARRAY FROM 30,,1001 THROUGH 32,,1000.

SEE SECTION 8,3 FOR EXAMPLE OF INSTRUCTION FORMAT INDIRECT WORD USED WITH EXTENDED PUSHDOWN STACK.

6,3 FLOW CHART OF EFFECTIVE ADDRESS COMPUTATION

THE ATTACHED FLOWCHART DESCRIBES THE EFFECTIVE ADDRESS COMPUTATION ALGORITHM. IT DOES NOT PURPORT TO SHOW HOW ANY ACTUAL COMPUTER WOULD IMPLEMENT IT. THE VARIABLES USED ARE:

VMA	VIRTUAL MEMORY ADDRESS; 30-BIT REGISTER USED TO HOLD THE ADDRESS FOR A MEMORY REFERENCE.
PC	PROGRAM COUNTER; 30-BITS
BREG	A WORKING REGISTER; 36-BITS
OP, AC, I, X	REGISTERS FOR HOLDING OPCODE, AC ADDRESS, INDIRECT BIT, INDEX ADDRESS RESPECTIVELY.
P,S	REGISTERS FOR HOLDING BYTE POSITION AND SIZE RESPECTIVELY.
L	LOCAL FLAG; IS A ONE IF THE RESULT OF THE E COMPUTATION IS A LOCAL ADDRESS, ZERO IF GLOBAL. [IT IS USED SOLELY TO HELP DETERMINE WHETHER A 30-BIT ADDRESS IS SPECIFYING THE HARDWARE ACS OR NOT.]
D	DOUBLE-WORD BYTE POINTER. IF A 1, THE BYTE POINTER IS A DOUBLE WORD BYTE POINTER, ELSE IT IS A SINGLE WORD BYTE POINTER. D IS SET FROM BIT 12 OF THE BYTE POINTER.

7. GENERAL OPERATION OF INSTRUCTIONS

ALL INSTRUCTIONS COMPUTE A 30-BIT LOCAL OR GLOBAL EFFECTIVE ADDRESS AS DESCRIBED ABOVE.

1. FOR INSTRUCTIONS WHICH REFERENCE MEMORY USING THE EFFECTIVE ADDRESS, THE MEMORY OPERAND WILL BE TAKEN FROM THE SAME SECTION AS THE INSTRUCTION IF NO INDEXING OR INDIRECTION IS SPECIFIED. IF INDEXING AND/OR INDIRECTION IS SPECIFIED, THE MEMORY OPERAND WILL BE TAKEN FROM THE SECTION DETERMINED BY THE EFFECTIVE ADDRESS CALCULATION.

2. AS ON THE KI, IMMEDIATE-MODE INSTRUCTIONS USE ONLY THE LOW-ORDER 18 BITS OF THE EFFECTIVE ADDRESS FOR THE OPERAND REGARDLESS OF WHETHER INDEXING AND/OR INDIRECTION WAS SPECIFIED.

3. THE EFFECTIVE ADDRESS COMPUTATION ALWAYS RESULTS IN A 30-BIT ADDRESS. HOWEVER, IT MAY BE A LOCAL OR A GLOBAL ADDRESS. AN EFFECTIVE ADDRESS COMPUTATION IS SAID TO BE LOCAL IF THE LAST MEMORY CYCLE SPECIFIED NO INDEXING OR SPECIFIED LOCAL INDEXING. SEE ATTACHED FLOW CHART FOR CONDITIONS UNDER WHICH THE LOCAL (L) FLAG IS SET OR CLEARED. IF THE RESULTING EFFECTIVE ADDRESS IS LOCAL, THE SECTION NUMBER IS IMPLIED BY THE SECTION NUMBER OF THE PC OR THE LAST EXPLICIT GLOBAL ADDRESS ENCOUNTERED DURING THE EFFECTIVE ADDRESS COMPUTATION.

4. IF AN INSTRUCTION READS OR WRITES THE WORD AT ITS EFFECTIVE ADDRESS, THE REFERENCE WILL BE MADE TO AN AC IF BITS 18-31 OF THE EFFECTIVE ADDRESS ARE 0 AND:

   A. THE EFFECTIVE ADDRESS IS A LOCAL ADDRESS (L=1),

      OR

   B. THE SECTION FIELD (BITS 6-17) OF THE EFFECTIVE ADDRESS IS 0 OR 1 [3].

   THIS MEANS THAT AN ORDINARY DIRECT REFERENCE TO A LOW ADDRESS (E.G., MOVE 1,3) WILL REFERENCE THE HARDWARE ACS, AS WILL A REFERENCE THROUGH AN EXTENDED FORMAT INDIRECT WORD CONTAINING A GLOBAL ADDRESS IN THE RANGE 0-17(OCTAL). IF THE ABOVE CONDITIONS FOR AN AC REFERENCE ARE NOT MET, THE REFERENCE IS MADE TO MEMORY IN THE DESIGNATED SECTION. THIS ALLOWS REFERENCES TO MEMORY LOCATIONS 0-17(OCTAL) IN ANY SECTION, EXCEPT 0.

5. ALL PC FETCHES ARE MADE AS LOCAL REFERENCES. THUS THE HARDWARE ACS ARE ALWAYS REFERENCED WHEN PC BITS 18-31=0, NO MATTER WHAT SECTION NUMBER IS IN THE LEFT HALF OF THE PC. CONCEPTUALLY THIS MEANS THAT THE ACS APPEAR AS THE FIRST 20 LOCATION OF EACH SECTION WHILE EXECUTING IN THAT SECTION. GLOBAL REFERENCES TO ANY SECTION (EXCEPT 0 OR 1) WILL REFERENCE MEMORY INSTEAD OF THE ACS.

THE EXISTENCE OF EXTENDED ADDRESSING HAS NO OTHER EFFECT ON MOST OF THE DEFINED PDP-10 INSTRUCTIONS, E.G., MOVE, ADD. THOSE INSTRUCTIONS FOR WHICH THERE ARE OTHER CONSIDERATIONS ARE DESCRIBED BELOW.

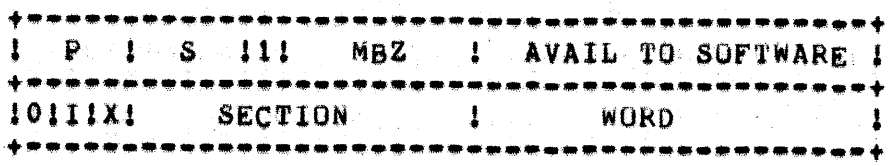
8. SPECIAL-CASE INSTRUCTIONS

8.1 PC-STORING INSTRUCTIONS; PUSHJ, JSP[4], JSR, POPJ.

ON THE KI, AND WHEN THE PC IS IN A KI COMPATIBLE SECTION (EXEC SECTION 0 AND USER SECTION 0), THESE INSTRUCTIONS STORE 13 BITS OF PROCESSOR FLAGS IN THE WORD WITH THE 18 BITS OF PC. IN ORDER TO ACCOMMODATE THE 30-BIT ADDRESS, THESE INSTRUCTIONS WILL STORE A 30-BIT PC WITHOUT FLAGS WHEN THE PC IS IN A NON-ZERO SECTION. NEW INSTRUCTIONS (SEE BELOW) ARE AVAILABLE TO PROVIDE ACCESS TO THE PROCESSOR FLAGS. WHEN THE PC IS IN A NON-ZERO SECTION, POPJ WILL RESTORE THE 30-BIT PC FROM THE STACK WORD. THUS MACHINE INDEPENDENT SUBROUTINES CAN BE WRITTEN WHICH RUN ON THE KI, SECTION 0, AND NON-ZERO SECTIONS.

8,2 BYTE INSTRUCTIONS [5]

TO REPRESENT THE P AND S FIELDS AND THE FULL MEMORY ADDRESS REQUIRES MORE THAN 36 BITS OF BYTE POINTER, THEREFORE, A BYTE POINTER WILL BE TAKEN AS A TWO-WORD QUANTITY IN THE FOLLOWING FORMAT IF BIT 12 OF THE FIRST WORD IS A ONE.



THE ADDRESS OF THE WORD CONTAINING THE BYTE IS COMPUTED FROM THE SECOND WORD AS AN INDIRECT WORD.

IN THE USUAL CASE, BITS 13-35 OF THE FIRST WORD WOULD BE 0 AND THE SECOND WORD WOULD SPECIFY THE ENTIRE ADDRESS. INCREMENTING A BYTE POINTER OF THIS FORMAT WILL, WHEN INCREMENTING THE WORD ADDRESS, INCREMENT THE SECOND WORD ONLY. CARRIES FROM THE RH OF THE SECOND WORD WILL PROPAGATE INTO THE LH SO THAT STRINGS CAN CROSS SECTION BOUNDARIES.

FOR CONVENIENCE, BIT 12 OF THE FIRST WORD MAY BE SET TO ZERO. IN THIS CASE THE SECOND WORD NEED NOT BE PRESENT. THE BYTE REFERENCE WILL BE TO THE SECTION SPECIFIED BY THE NORMAL EFFECTIVE ADDRESS. INCREMENTING A BYTE POINTER WITH BIT 12=0, WILL INCREMENT THE RH OF THE FIRST WORD AS ON THE KI WITH NO CARRY OUT OF BIT 18.

8,3 STACK INSTRUCTIONS, PUSH, PUSHJ, POP, POPJ, ADJSP

THE PRESENT FORMAT OF THE STACK POINTER (HALF-WORD COUNT, HALF-WORD ADDRESS) IS INSUFFICIENT TO HOLD A FULL ADDRESS BUT IS CONVENIENT WHEN THE STACK IS LOCAL AND SMALL. THEREFORE, IN NON-ZERO SECTIONS THE STACK INSTRUCTIONS WILL RECOGNIZE EITHER OF TWO FORMS OF STACK POINTER:

1. LOCAL STACK POINTER

   IF THE LEFT HALF OF THE STACK POINTER IS NEGATIVE OR 0 BEFORE INCREMENTING OR DECREMENTING, THE RIGHT HALF WILL BE TAKEN AS AN ADDRESS WITHIN THE SECTION SPECIFIED BY THE PC. INCREMENTING AND DECREMENTING THE STACK POINTER WILL MODIFY BOTH HALVES OF THE POINTER AS ON THE KI10. AS ON THE KI, ANY CARRY OUT OF BIT 18 IS SUPPRESSED, SO THAT LOCAL STACKS WILL NOT CROSS SECTION BOUNDARIES.

2. GLOBAL STACK POINTER

   IF THE LEFT HALF OF THE STACK POINTER IS POSITIVE AND NON-0 BEFORE INCREMENTING OR DECREMENTING, THE ENTIRE POINTER WILL BE TAKEN AS A 30-BIT STACK ADDRESS WITH NO COUNT FIELD. THE POINTER WILL BE INCREMENTED AND DECREMENTED AS A SINGLE QUANTITY. STACK OVERFLOW AND UNDERFLOW DETECTION IS EXPECTED TO BE PROGRAMMED BY SETTING A RESTRICTED ACCESS ON THE PAGES AT EITHER END OF THE STACK, SINCE THE ABSENCE OF A COUNT PREVENTS AN EXPLICIT HARDWARE CHECK. CARRIES OUT OF BIT 18 ARE NOT SUPRESSED. THUS A STACK CAN CROSS SECTION BOUNDARIES. THIS FORMAT IS EXPECTED TO BE USED AS THE STANDARD IN EXTENDED SECTIONS.

THUS MACHINE INDEPENDENT SUBROUTINES CAN BE WRITTEN WHICH RUN ON THE KI, IN SECTION 0 AND NON-ZERO SECTIONS ON THE KL. ONLY THE CODE WHICH INITIALIZES THE STACK POINTER NEEDS TO KNOW THE ENVIRONMENT. NOTE THAT THE ABOVE TWO FORMATS ARE THE SAME AS THE INDEX REGISTER FORMATS AND BEHAVE IN AN ANALOGOUS MANNER. WARNING: PUSHING ON A LOCAL STACK WHICH HAS PREVIOUSLY OVERFLOWED (I.E., 0,,N BEFORE PUSH) WILL RESULT IN STORING IN SECTION 1 (I.E., 1,,N+1).

EXAMPLES - PUSHDOWN STACK POINTER BEFORE AND AFTER

```
INSTR,  STACK POINTER AC
        BEFORE          AFTER

PUSH    -6,,100         -5,,101 ;STACK IN PC SECT,
PUSH    -1,,105         0,,106  ;OVERFLOW
PUSH    32,,100         32,,101 ;STACK IN SECT,32
```

EXAMPLE - SUBROUTINE ARGUMENT LIST MECHANISM (IFIWS)

IN SECTION 22:

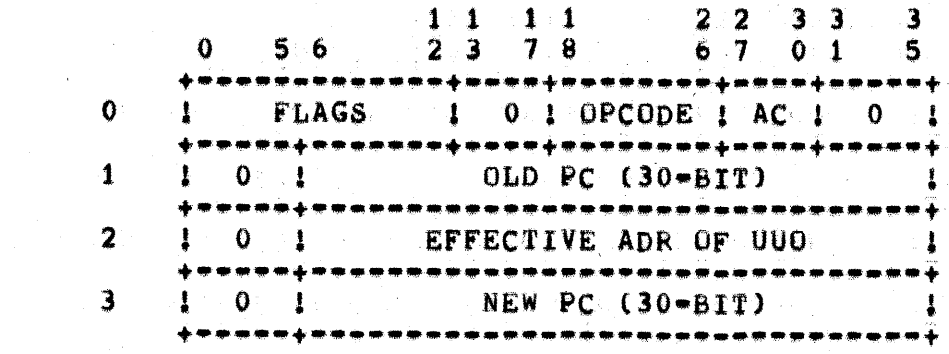
```
        MOVE    P,[10,,2000]    ;START PUSHDOWN LIST IN
                                ; 3RD PAGE OF SECT, 10
        MOVE    CF,P            ;SETUP CURRENT STK FRAME
        ..
        XMOVEI  AP,ARGLST       ;SET LH=22,RH=ARGLIST
        PUSHJ   P,@[30,,SUBR]   ;CALL SUBR IN SECT, 30

        -2,,0                   ;ARG COUNT
ARGLST: 400000,,1000            ;ARG IN LOC 22,,1000
        400000,,50(PF)          ;ARG IN LOC 50 IN
                                ; TRANSIENT STORAGE
```

IN SECTION 30:

```
SUBR:   PUSH    P,PF            ;SAVE OLD PREV STK FRAME
        MOVE    PF,CF           ;SETUP PREVIOUS STACK FRAME
        ..
        MOVE    T,@0(AP)        ;GET 0ST ARG [FROM 22,,1000]
        ..
        MOVE    T,@1(AP)        ;GET 1ST ARG [FROM 50 IN
                                ; CALLER'S STACK FRAME]
        ..
        POP     P,PF            ;RESTORE PREVIOUS FRAME
        POPJ    P,              ;RETURN [TO SECTION 22]
```

8,4 LUUO (OPCODES 1-37)

IT IS DESIRABLE TO BE ABLE TO EXECUTE LUUOS IN ANY SECTION AND INVOKE COMMON CODE. THEREFORE, WHEN THE PC IS IN A NON-ZERO SECTION, ALL LUUOS WILL TRAP TO THE SAME PLACE [6]. WORD 420 IN THE USER PROCESS TABLE (UPT) WILL CONTAIN THE ADDRESS OF A 4-WORD BLOCK FOR LUUO INFORMATION. SEE CHAPTER 2,9, EXEC AND USER PROCESS TABLES (EPT,UPT). THE INFORMATION IS FORMATTED AS FOLLOWS:



HENCE, EXECUTION OF AN LUUO WILL CAUSE THE PROCESSOR PC AND FLAGS TO BE STORED, THE OPCODE, AC, AND EFFECTIVE ADDRESS OF THE LUUO TO BE STORED, AND THE PROCESSOR TO BEGIN OPERATION AT THE LOCATION SPECIFIED BY THE "NEW PC" THE PROCESSOR FLAGS WILL NOT BE CHANGED. IN SECTION 0 THE LUUO MECHANISM WILL WORK AS ON THE KI AND SO WILL INVOKE A SEPARATE LUUO HANDLER WHICH MUST BE IN SECTION 0.

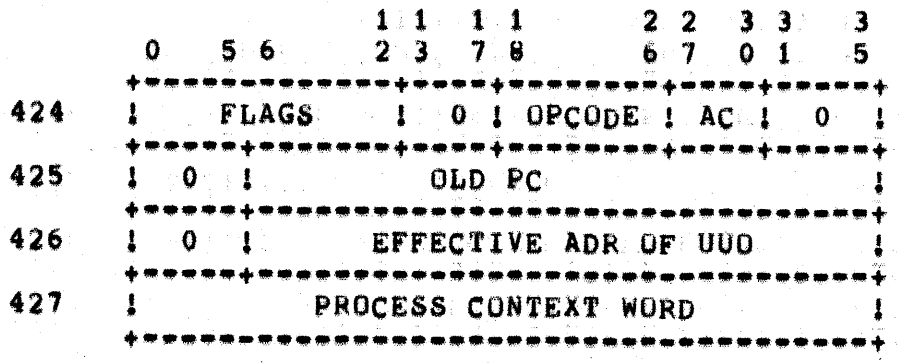
WARNING: THE USE OF LUUOS BY ONE PROGRAMMER WILL PROBABLY PREVENT HIM FROM INTERFACING WITH OTHER CODE WHICH ALSO USES LUUOS, UNLESS THERE IS PRIOR AGREEMENT BETWEEN THE TWO PROGRAMMERS.

THUS LUUOS SHOULD BE USED FOR SELF-CONTAINED APPLICATIONS ONLY. THEREFORE THE USE OF LUUOS WILL NOT BE ALLOWED UNDER THE DEC PROCEDURE CALLING STANDARD. FURTHERMORE, IF ROUTINES INITIATED BY SOFTWARE INTERRUPTS ALSO USE LUUOS, THEY MUST BE CAREFUL TO SAVE AND RESTORE WORDS 0-2 OF THE LUUO BLOCK. THIS WARNING WILL APPEAR IN THE HARDWARE REFERENCE MANUAL.

IN EXEC MODE AN LUUO IN A NON-ZERO SECTION WILL DO AN MUUO. MONITORS DO NOT TYPICALLY USE LUUOS.

8,5 MUUO (OPCODES 0, 40-77, ALL UNDEFINED OPCODES)

EXECUTION OF AN MUUO WILL CAUSE THE UUO INFORMATION TO BE STORED IN A 4-WORD BLOCK BEGINNING AT LOCATION 424 OF THE UPT.



THE NEW PC WORD WILL BE SELECTED FROM A BLOCK OF 8 WORDS AT UPT+430 ACCORDING TO THE CONTEXT IN WHICH THE MUUO WAS EXECUTED JUST AS ON THE KI10. THE NEW PC WORD WILL BE TAKEN AS A 30-BIT GLOBAL ADDRESS WITH NO FLAGS. THE HARDWARE/MICROCODE WILL COMPUTE THE PROPER SETTINGS OF PCU AND PCP ("PREVIOUS CONTEXT USER" AND "PREVIOUS CONTEXT PUBLIC"), AND WILL CLEAR THE REST OF THE PROCESSOR FLAGS.

IN ORDER TO FACILITATE USE OF AC OPERANDS WITH LUUOS AND MUUOS, THE FOLLOWING RULES GOVERN THE EFFECTIVE ADDRESS WORD STORED IN THE UUO BLOCK [3]:

1. IF BITS 18-31 OF THE EFFECTIVE ADDRESS OF THE UUO ARE ZERO AND A LOCAL ADDRESS BUT BITS 6-17 ARE NON-ZERO, STORE 1 IN THE SECTION FIELD OF WORD 2, AND STORE BITS 18-35 IN THE RH OF WORD 2. THIS CONVERTS THE HARDWARE AC ADDRESS TO THE SECTION-INDEPENDENT FORM.

2. OTHERWISE, STORE THE FULL 30-BIT EFFECTIVE ADDRESS IN WORD 2.

THIS IS THE SAME SET OF RULES AS GOVERNS XMOVEI.

8,6 BLT

THE PRESENT FORMAT OF BLT OPERANDS IS INSUFFICIENT TO SPECIFY THREE FULL ADDRESSES, THEREFORE A NEW INSTRUCTION, XBLT, WILL BE SPECIFIED (SEE BELOW). HOWEVER, THE EXISTING BLT INSTRUCTION IS USEFUL FOR INTRA-SECTION DATA MOVES, AND IS SPECIFIED TO WORK AS FOLLOWS:

1. THE SOURCE ADDRESS IS TAKEN TO BE THE LEFT HALF OF THE CONTENTS OF AC IN THE SAME SECTION AS THE EFFECTIVE ADDRESS (WHICH CAN SPECIFY ANY SECTION).

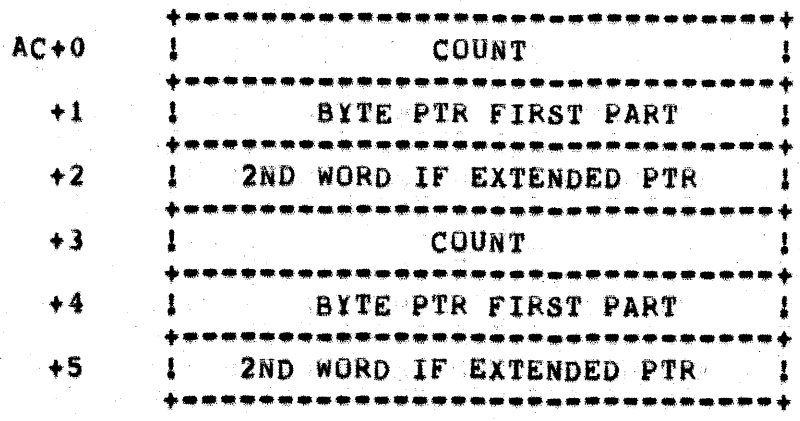
2. THE DESTINATION ADDRESS IS TAKEN TO BE THE RIGHT HALF OF THE CONTENTS OF AC IN THE SAME SECTION AS THE EFFECTIVE ADDRESS (WHICH CAN SPECIFY ANY SECTION).

3. DATA IS MOVED UNTIL THE DESTINATION ADDRESS IS EQUAL TO THE EFFECTIVE ADDRESS. CARRIES OUT OF BIT 18 ARE SUPPRESSED, SO THAT THE BLT STAYS IN THE SAME SECTION BY WRAPPING AROUND.

HENCE THE FORMAT OF THE AC OPERAND OF BLT IS NOT CHANGED FROM THE KI10 IMPLEMENTATION. THE SOURCE AND DESTINATION SECTIONS ARE ALWAYS THE SAME AS SPECIFIED BY E WHICH CAN BE DIFFERENT FROM THE PC. REFERENCES BY BLT TO ADDRESSES IN WHICH 18-31 ARE 0 WILL ALWAYS BE AC REFERENCES.

8,7 EXTEND-STRING OPERATIONS

THE EXTEND INSTRUCTION IS DOCUMENTED IN CHAPTER 2,10, THE EXTEND INSTRUCTION. TO SUPPORT EXTENDED ADDRESSING, A 6-WORD BLOCK WILL BE USED IN ALL SECTIONS CONSISTING OF TWO 3-AC BLOCKS AS FOLLOWS:



BYTE POINTERS ARE IN THE ONE- OR TWO-WORD FORMAT DESCRIBED ABOVE. IF BIT 12 OF THE POINTER IN AC+1 (AC+4) IS 0, AC+2 (AC+5) IS IGNORED.

8.8 AOBJN

THE TWO HALF-WORD FORMAT OF THE AOBJN POINTER IS UNSUFFICIENT TO SPECIFY A GLOBAL ADDRESS. HOWEVER, THE FORMAT MAY BE USED FOR INDEXING AS DESCRIBED ABOVE BECAUSE THE LEFT HALF IS NORMALLY NEGATIVE. IT IS THEREFORE USEFUL FOR SCANNING LOCAL TABLES (WITHIN THE SAME SECTION) AND WILL BE RETAINED WITHOUT MODIFICATION. FOR SCANNING TABLES IN AN ARBITRARY SECTION, THE PROGRAMMER WILL TYPICALLY USE AN INDEX CONTAINING A GLOBAL ADDRESS AND, IF NECESSARY, A SECOND WORD FOR THE COUNT, AND WILL NOT EMPLOY AOBJN IN THIS CASE.

8.9 JSA, JRA

THESE INSTRUCTIONS USE A FORMAT WHICH DOES NOT ALLOW THE STORING OR SPECIFICATION OF A GLOBAL ADDRESS. SINCE THEY ARE ALSO CONSIDERED AN OBSOLETE AND UNRECOMMENDED METHOD FOR SUBROUTINE CALLING, THEY WILL WORK THE SAME AS ON THE KI. NOTE THAT THEY WILL WORK IN NON-ZERO SECTIONS, BUT WILL BE USEFUL ONLY FOR INTRA-SECTION CALLS, SINCE ONLY AN 18-BIT PC IS STORED [7].

8.10 BLKI, BLKO

THESE INSTRUCTIONS USE A POINTER FORMAT WHICH DOES NOT ALLOW THE SPECIFICATION OF A GLOBAL ADDRESS. THEY ALSO APPEAR TO HAVE NO LIKELY USE ON 20-SERIES MACHINES. FOR DIAGNOSTIC COMPATABILITY HOWEVER, THE KL10 WILL SUPPORT THESE INSTRUCTIONS BY DEFINING THAT THE POINTER ADDRESS ALWAYS REFERS TO THE PC SECTION. SEE SECTION 9 FOR BLKI IN PI LOCATION.

8.11 XCT

THE DEFAULT SECTION FOR THE OBJECT INSTRUCTION SHALL BE THE SECTION OF THE EFFECTIVE ADDRESS OF THE XCT. SEE ATTACHED FLOWCHART. HOWEVER PC STORING INSTRUCTIONS WILL STORE THE PC SECTION, RATHER THAN THE SECTION SPECIFIED BY THE EFFECTIVE ADDRESS OF THE XCT. THIS MAINTAINS COMPATIBILITY WITH THE KI (WHICH STORES THE PC+1) [8]. LOCAL STACK POINTERS WILL ALSO ASSUME THE PC SECTION, RATHER THAN THE SECTION SPECIFIED BY THE EFFECTIVE ADDRESS OF THE XCT [10]. NOTE THAT THE HARDWARE PERFORMS ALL INSTRUCTIONS WITHOUT ANY SPECIAL CHECK FOR BEING EXECUTED UNDER XCT OR NOT.

EXAMPLE OF XCT OF CODE IN ANOTHER SECTION:

IN SECTION 22:

```
        XCT     @[30,,1000]
```

LOCATION 1000 IN SECTION 30:

```
        MOVE    T,2000
```

WILL LOAD T WITH CONTENTS OF LOCATION 2000 IN SECTION 30, NOT SECTION 22.

EXAMPLE - STACK AND PC STORING UNDER XCT

IN SECTION 22, LOCATION 100:

```
        XCT     @[30,,1000]
```

LOCATION 1000 IN SECTION 30:

```
        PUSHJ   P,SUBR
```

TRANSFER TO SUBROUTINE SUBR IN SECTION 30, NOT 22. IF C(P) IS LOCAL, STACK IS ASSUMED TO BE IN SECTION 22, NOT 30. THE PC STORED ON STACK IS 22,,101, NOT 30,,1001.

9. PI HANDLING

INITIATION OF A PI CYCLE WILL CAUSE THE EXECUTION OF AN INSTRUCTION IN THE EPT AS IS NOW DONE ON THE KI10. FOR EXTENDED ADDRESSING SUPPORT, THE RECOMMENDED INSTRUCTION IS SRFP (SAVE THEN RESTORE FLAGS AND PROGRAM-COUNTER) DEFINED BELOW. THIS INSTRUCTION SAVES THE CURRENT FLAGS AND 30-BIT PC AND ESTABLISHES NEW FLAGS AND PC. THE INTERRUPT IS DISMISSED BY EXECUTION OF ANOTHER NEW INSTRUCTION, RFPD, (RESTORE FLAGS AND PROGRAM-COUNTER) WHICH RESTORES THE GLOBAL PC AND FLAGS.

WHEN AN INSTRUCTION IS BEING EXECUTED AS A PI INSTRUCTION, THE DEFAULT SECTION FOR COMPUTING THE EFFECTIVE ADDRESS IS TAKEN TO BE EXEC SECTION 0, RATHER THAN THE ACTUAL PC SECTION. THEREFORE IF A BLKI, BLKO, OR JSR IS EXECUTED AS A PI INSTRUCTION, IT WILL WORK AS ON THE KI10 FOR PROGRAMS (E.G., DIAGNOSTICS) NOT USING EXTENDED ADDRESSING.

10. NEW INSTRUCTIONS

THE FOLLOWING NEW INSTRUCTIONS ARE REQUIRED TO PROPERLY HANDLE EXTENDED ADDRESSING.

10.1 XMOVEI - MOVE EXTENDED ADDRESS (OPCODE = SETMI [9])

THIS INSTRUCTION MOVES ITS ENTIRE EFFECTIVE ADDRESS INTO THE DESIGNATED AC. IT IS GENERALLY USED TO FIND THE EFFECTIVE ADDRESS OF A POINTER CHAIN AND MAKE IT AVAILABLE FOR INDEXING. IT IS THE ONLY "IMMEDIATE MODE" INSTRUCTION WHICH HAS AN OPERAND WHICH IS GREATER THAN 18 BITS. BITS 0-5 ARE ALWAYS ZERO. IF THE EFFECTIVE ADDRESS SPECIFIES A HARDWARE AC, THE EFFECTIVE ADDRESS WILL BE CONVERTED TO THE SECTION-INDEPENDENT FORM, IE., 1,,AC. THUS THE RESULT OF AN XMOVEI CAN BE STORED OR MOVED TO ANOTHER SECTION AND STILL MEAN THE SAME ADDRESS. THIS IS ANALOGOUS TO THE EFFECTIVE ADDRESS STORED BY MUUOS AND LUUOS. NOTE THAT IF THIS INSTRUCTION IS EXECUTED WITHOUT INDEXING OR INDIRECTION, E.G., XMOVEI 1,20, IT WILL MOVE THE CURRENT SECTION INTO THE LEFT HALF OF THE DESIGNATED AC (E MUST BE 20 OR GREATER). XMOVEI ON A KI WILL WORK CORRECTLY (SINCE OPCODE IS SETMI), SO THAT THE SAME SUBROUTINES WILL RUN ON THE KI AND KL. MOVXA IS ALSO USED TO TEST FOR KI COMPATIBLE SECTION (SEE SECTION 11.1).

10,2 XBLT - EXTENDED BLOCK TRANSFER (EXTEND OPCODE 020)

XBLT IS A MEMBER OF THE EXTENDED INSTRUCTION SET, USED UNDER EXTENDED ADDRESSING. XBLT TRAPS AS AN MUUO IN SECTION 0, BUT OTHERWISE MOVES DATA FROM ANY VIRTUAL ADDRESS TO ANY OTHER. THE NUMBER OF WORDS TRANSFERRED IS SPECIFIED BY AC, THE ADDRESS OF THE SOURCE BLOCK IS GIVEN BY AC+1, AND THE ADDRESS OF THE DESTINATION BLOCK IS IN AC+2. BOTH ADDRESSES ARE ALWAYS GLOBAL.

IF AC IS POSITIVE, THE BLOCK ADDRESSES IN AC+1 AND AC+2 ARE THE LOWEST ADDRESSES OF EACH BLOCK, AND IDENTIFY THE WORDS WHICH ARE TRANSFERRED FIRST IN TIME. THE TRANSFER PROCEEDS BY INCREMENTING THE ADDRESSES IN AC+1 AND AC+2 AFTER EACH TRANSFER, AND DECREMENTING AC UNTIL IT REACHES ZERO.

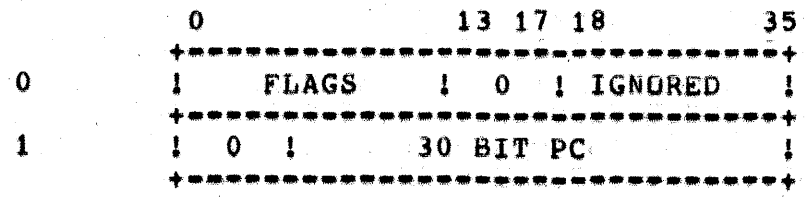
IF AC IS NEGATIVE, THE BLOCK ADDRESSES IN AC+1 AND AC+2 ARE GREATER, BY 1, THAN THE HIGHEST ADDRESSES OF EACH BLOCK. THE TRANSFER PROCEEDS BY DECREMENTING THE ADDRESSES IN AC+1 AND AC+2 BEFORE EACH TRANSFER, AND INCREMENTING AC UNTIL IT REACHES ZERO.

SINCE XBLT IS INTERRUPTABLE, RESULTS ARE INDETERMINATE IF AC, AC+1, OR AC+2 IS IN EITHER SOURCE OR DESTINATION BLOCK. OTHERWISE THE EFFECT OF BLT ON AC'S IS EQUIVALENT TO:

```
ADD AC+1,AC
ADD AC+2,AC
SETZM AC
```

10.3 XJRSTF - RESTORE FLAGS AND PROGRAM-COUNTER (JRST 5,)

THIS INSTRUCTION RESTORES THE FLAGS AND PC DOUBLE-WORD FROM E AND E+1 . THE DOUBLE-WORD FORMAT IS:



FLAGS AND PC DOUBLE-WORD

THIS FORMAT IS USED BY LUUOS, MUUOS, AND ADDITIONAL INSTRUCTIONS DEFINED BELOW. NOTE THAT, UNLIKE JRSTF, NO INDIRECTION IS NEEDED, BITS 13-17 MUST BE ZERO, BECAUSE THAY ARE RESERVED FOR FUTURE HARDWARE. BOTH WORDS ARE FETCHED BEFORE THE FLAGS TAKE EFFECT. XJRSTF HALTS IN KI EXEC MODE, TRAPS IN KI USER MODE, WORKS IN ALL KL MODES AND SECTIONS, XJRSTF WILL BE IN FUTURE MACHINES WHICH HAVE EXTENDED ADDRESSING.

EXAMPLE - CPU INDEPENDENT FLAG CODE

```
XMOVI   T,20            ;GET SECTION NO.
HLLZM   T,SECTNO        ;SAVE FOR TEST
MOVSI   17,ACBLK        ;RESTORE ALL ACS
BLT     17,17           ;INCLUDING 17
SKIPN   SECTNO          ;IS THIS KL EXT. SECT.?
JRSTF   @FLGPC          ;NO, RESTORE FLAGS AND PC
XJRSTF  FLGPC           ;YES,RESTORE FLAGS AND PC
```

NOTE THAT THE ABOVE CODE WORKS ON ALL CPUS, INCLUDING PDP-6, KA, KI, KL SECTION 0, AND KL EXTENDED SECTIONS.

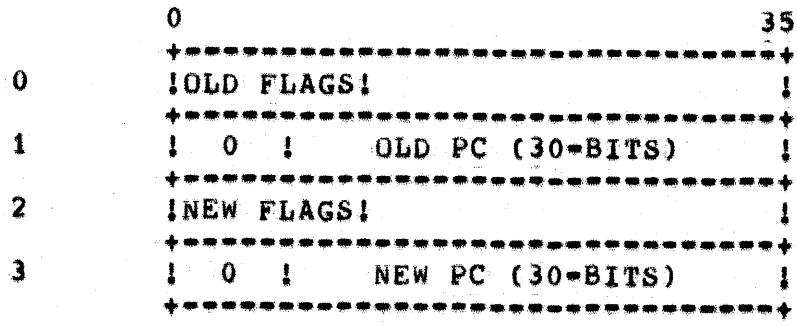
10,4 XJEN - RESTORE FLAGS AND PROGRAM-COUNTER AND DISMISS (JRST 6,)

THIS PRIVILEGED INSTRUCTION PERFORMS ALL THE FUNCTIONS OF XJRSTF AND IN ADDITION DISMISSES THE CURRENT PI LEVEL. IT IS INTENDED TO BE USED IN PLACE OF JEN @E UNDER EXTENDED ADDRESSING.

XJEN HALTS IN KI EXEC MODE, TRAPS IN KI USER MODE, WORKS IN EXEC KI COMPATIBLE SECTION, TRAPS IN USER KI COMPATIBLE SECTION (EXCEPT USER IOT MODE), AND TRAPS IN USER EXTENDED SECTIONS (EXCEPT USER IOT MODE). XJEN MAY NOT BE PRESENT IN FUTURE MACHINES.

10,5 XPCW - SAVE THEN RESTORE FLAGS AND PROGRAM-COUNTER (JRST 7,)

THIS PRIVILEGED INSTRUCTION IS INTENDEDTO BE USED IN INTERRUPT LOCATIONS. IT REFERENCES A 4-WORD BLOCK AT ITS EFFECTIVE ADDRESS FORMATTED AS FOLLOWS:



THE CURRENT FLAGS AND PC ARE STORED IN THE FIRST DOUBLEWORD, AND NEW FLAGS AND PC ARE ESTABLISHED FROM THE SECOND DOUBLEWORD. DISMISSING AN INTERRUPT INITIATED WITH XPCW WOULD TYPICALLY BE DONE WITH XJRSTF ADDRESSING THE SAME BLOCK. NOTE THAT THE 4-WORD BLOCK MUST BE IN SECTION 0, SINCE THE DEFAULT SECTION IS 0 FOR INSTRUCTIONS EXECUTED IN AN INTERRUPT LOCATION. XPCW HALTS IN KI EXEC MODE, TRAPS IN KI USER MODE, WORKS IN EXEC KI COMPATIBLE SECTION, TRAPS IN USER KI COMPATIBLE SECTION (EXCEPT USER IOT MODE), WORKS IN EXTENDED EXEC SECTIONS, AND TRAPS IN EXTENDED USER SECTIONS (EXCEPT USER IOT MODE). XPCW MAY NOT BE IN FUTURE MACHINES.

10,6 XSFM - SAVE FLAGS IN MEMORY (JRST 14,)

THIS INSTRUCTION SAVES THE FLAGS IN BITS 0-12 OF E IN THE SAME FORMAT AS THE FLAGS IN THE FLAGS AND PC DOUBLE WORD. THE INSTRUCTION WORKS IN EXEC AND USER MODE. XSFM HALTS IN KI EXEC MODE, TRAPS IN KI USER MODE, WORKS IN EXEC KI COMPATIBLE SECTION, TRAPS IN USER KI COMPATIBLE SECTION (EXCEPT USER IOT MODE), AND WORKS IN EXEC AND USER EXTENDED SECTIONS. XSFM WILL BE IN FUTURE MACHINES WHICH HAVE EXTENDED ADDRESSING.

THE 6 ARITHMETIC FLAGS: OVERFLOW (0), CARRY 0 (1), CARRY 1 (2), FLOATING OVERFLOW (3), FLOATING UNDERFLOW (11), AND NO DIVIDE (12), WILL BE PRESERVED IN FUTURE MACHINES. THE REMAINING FLAGS: FIRST PART DONE (4), USER (5), USER IN-OUT (6), PUBLIC (7), ADDRESS FAILURE INHIBIT (8), TRAP 2 (9), TRAP 1 (10) ARE CPU DEPENDENT AND MAY BE CHANGED OR DISAPPEAR IN FUTURE MACHINES.

EXAMPLE - CPU INDEPENDENT FLAG TEST CODE

```
MOVEM   T,SAVEAC        ;SAVE AC
XMOVEI  T,20            ;GET SECTION #, OR 0 LH
TLNN    T,-1            ;IS THIS KL EXTENDED SECT?
JSP     T,.+2           ;NO, -6, KA, KI, KL SECT 0
XSFM    T               ;YES, SAVE FLAGS IN T
HERE WITH FLAGS IN LH OF T
```

11. COMPATIBILITY SUMMARY

THE FOLLOWING TABLE SHOWS HOW EACH OF THE NEW FEATURES WORKS IN EXEC AND USER MODE ON THE KI, IN THE KI COMPATIBLE SECTION AND IN EXTENDED ADDRESSING SECTIONS. THE CRITERIA FOR COMPATIBILITY ARE AS FOLLOWS IN DECREASING IMPORTANCE:

1. USER CODE WHICH RUNS ON THE KI MUST RUN IN THE KL COMPATIBLE SECTION, INCLUDING SAVED CORE IMAGES. (TRAP ON BIT 12 OF BYTE POINTERS DOES NOT VIOLATE THIS, IF MONITOR CONTINUES PROGRAM.).

2. IT SHOULD BE EASY AND NATURAL TO WRITE SUBROUTINES FOLLOWING A STANDARD WHICH CAN RUN ON KIS, IN THE COMPATIBLE KI SECTION AND IN EXTENDED SECTIONS. THE LOADER CAN TAKE CARE OF ANY DIFFERENCES, SO THAT A SINGLE REL FILE WORKS FOR ALL THREE CASES.

3. CODE IN USER KI COMPATIBLE SECTION CAN USE NEW INSTRUCTIONS EXCEPT THOSE ADDED SOLELY FOR EXTENDED ADRESSING. HOWEVER CODE CANNOT REFERENCE OR TRANSFER TO OTHER SECTIONS, EXCEPT THROUGH MONITOR CALLS.

FEATURE	KI EXEC	KI USER	KL EXEC SECT=0	KL USER SECT=0	KL EXEC SECT>0	KL USER SECT>0
INDEXING	KI	KI	KI	KI	NEW	NEW
INDIRECTION	KI	KI	KI	KI	NEW	NEW
HARDWARE ACS	KI	KI	KI	KI	NEW	NEW
PUSH,PUSHJ,ETC (GLOBAL IF LH OF AC GREATER THAN 0)	KI	KI	KI	KI	YES	YES
LUUO (EXTENDED PC)	KI	KI	KI	KI	MUUO	YES
BLT (REFERENCE SECTION SPECIFIED BY E?)	KI	KI	KI	KI	YES	YES

FEATURE	KI EXEC	KI USER	KL EXEC SECT=0	KL USER SECT=0	KL EXEC SECT>0	KL USER SECT>0
2-WORD BYTE POINTER (USE 2ND WORD IF BIT 12 = 1?)	IGNORED	IGNORED	IGNORED	IGNORED	YES	YES
EXTEND-STRING (REFERENCE OTHER SECTIONS?)	TRAP	TRAP	NO	NO	YES	YES
AOBJN (TABLES IN CURRENT SECTION?)	KI	KI	KI	KI	YES	YES
JSA,JRA (18 BIT PC ONLY)	KI	KI	KI	KI	YES	YES
BLKI,BLKO (CURRENT PC SECTION)	KI	KI	KI	KI	YES	YES
XCT (DEFAULT SECTION FROM E OF XCT?)	KI	KI	KI	KI	YES	YES
XMOVEI (LH)	0	0	0	0	SECTION	SECTION
XBLT (REFERENCE ANY SECTION?)	TRAP	TRAP	TRAP	TRAP	YES	YES
XJRSTF (TRANSFER TO ANY SECTION?)	HALT	TRAP	YES	YES	YES	YES
XJEN (TRANSFER TO ANY SECTION?)	HALT	TRAP	YES	TRAP	YES	TRAP
XPCW (TRANSFER TO ANY SECTION?)	HALT	TRAP	YES	TRAP	YES	TRAP
XSFM (READ FLAGS?)	HALT	TRAP	YES	TRAP	YES	YES
JRSTF (TRANSFER CONTROL?)	YES	YES	YES	YES	TRAP	TRAP
PC SETTING (TRANSFER TO ANY SECTION INCLUDING 0?)	NO	NO	0 ONLY	0 ONLY	YES	YES

11.1 TESTING FOR KI COMPATIBLE SECTION

THE CODE TO DISTINGUISH KI COMPATIBLE SECTION (AND KI MACHINE) FROM KL EXTENDED SECTIONS IS:

```
        XMOVEI  T,20            ;GET CURRENT SECTION NUMBER
        TLNN    T,777777        ;NON-ZERO SECTION?
        HERE IF KI COMPATIBLE SECTION OR KI,KA, OR PDP-6
        HERE IF KL EXTENDED SECTION
```

12. OLD INSTRUCTIONS

12.1 JRSTF - JUMP AND RESTORE FLAGS

IN NON-ZERO SECTIONS, JRSTF WILL GIVE AN ILLEGAL INSTRUCTION TRAP. THIS IS BECAUSE JRSTF IS USUALLY USED WITH AN INDIRECT WORD WHICH CONTAINS PC FLAGS IN THE LEFT HALF. THESE FLAGS MIGHT MISTAKENLY APPEAR TO BE AN EFIW (IF BIT 0=0).

12.2 JRST X,E

THE AC FIELD OF JRST IS BEING USED TO ENCODE NEW OPCODES WHICH DO NOT NEED AN AC FIELD. UNUSED BIT COMBINATIONS WILL TRAP. THE FOLLOWING AC BIT COMBINATIONS ARE DEFINED:

AC	OPCODE	AC	OPCODE
0	JRST	10	JUMP AND RESTORE INT.
1	PORTAL	11	ILLEGAL
2	JRSTF	12	JEN
3	ILLEGAL	13	ILLEGAL
4	HALT	14	XSFM
5	XJRSTF	15	ILLEGAL
6	XJEN	16	ILLEGAL
7	XPCW	17	ILLEGAL

THE NEW OPCODES WERE SELECTED BECAUSE THEY HAVE THE HALT BIT ON AND SO ARE LEAST LIKELY TO BE USED IN EXISTING EXEC AND USER CODE.

13. SPECIAL CONSIDERATION FOR ACS

THE ABOVE SPECIFICATION STATES THAT THE HARDWARE ACS APPEAR AS THE FIRST 20 LOCATION OF ANY SECTION REFERENCED WITH A LOCAL EFFECTIVE ADDRESS. INSTRUCTION FETCHES SPECIFIED BY THE PC ARE ALWAYS LOCAL, EVEN IF A TRANSFER INSTRUCTION JUMPED TO A GLOBAL ADDRESS.

EXAMPLE - JUMPING TO SHADOW ACS [14]

```
JRST    @[30,,2]
```

JUMPS TO SECTION 30 LOCATION 2. HOWEVER, THE PC FETCH WILL COME FROM AC 2 (SINCE PC FETCH IS ALWAYS LOCAL). THIS SHOULD NOT BE A PROBLEM, SINCE THE LOADER WILL LOAD CODE STARTING AT 20 IN EACH SECTION, RATHER THAN 0.

EXAMPLE - JSR TO SHADOW ACS

```
JSR     @[30,,2]
```

STORES THE PC IN MEMORY IN 30,,2 AND CHANGES THE PC TO 30,,3. THE NEXT INSTRUCTION IS FETCHED FROM AC 3, NOT MEMORY. THIS SHOULD NOT BE A PROBLEM SINCE THE LOADER WILL LOAD CODE STARTING AT 20 IN EACH SECTION, RATHER THAN 0.

EXAMPLE - XCTING SHADOW ACS

```
XCT     @[30,,2]
```

WILL EXECUTE THE INSTRUCTION IN MEMORY AT 30,,2, NOT AC 2. THIS IS DESIRABLE SINCE TABLES OF INSTRUCTIONS ARE EXECUTED AND THIS "DATA" SHOULD BE ABLE TO BE ANYWHERE IN MEMORY, JUST LIKE ANY OTHER KIND OF DATA. HOWEVER, AN INTERPRETER RUNNING IN A SEPARATE SECTION MUST CHECK FOR THE PC GETTING INTO THE ACS. THIS IS EASILY DONE WITH XMOVEI.

EXAMPLE - SUBROUTINE CALLING FROM ACS

```
2/      PUSHJ   P,SUBR
3/      EXP     ARGLIST
4/      RETURN
```

WOULD NOT WORK CORRECTLY, IF THE FOLLOWING STRAIGHT FORWARD CODE WAS PERFORMED:

```
SUBR:   MOVE    AP,@(P)          ;FETCH MEMORY 20,,3
```

UNLESS SUBR PICKED UP THE WORD FOLLOWING THE PUSHJ:

```
SUBR:   XMOVEI  AP,@(P)          ;GET ADDRESS 0,,3 TO AP
        MOVE    AP,0(AP)         ;GET ARGLIST FROM AC 3
```

HOWEVER THIS IS A RARELY USED CALLING TECHNIQUE AND CALLS TO SUBROUTINES ARE NOT USUALLY MADE FROM THE ACS.

14. TRAPPING FOR DYNAMIC LINKING

THE MONITOR WILL ASSIGN SECTION NUMBERS TO A USER PROCESS AS THEY ARE NEEDED, INSTEAD OF HAVING THEM BUILT INTO USER PROGRAMS OR ASSIGNED BY THE LOADER. THEN THE SYSTEM CAN HANDLE MORE CODE AND DATA THAN CAN FIT IN A SINGLE USER PROCESS, EVEN THOUGH NO SINGLE USER CAN USE ALL OF IT. THE MONITOR WOULD NOT ATTEMPT TO UNALLOCATE A SECTION NUMBER ONCE IT HAS BEEN ASSIGNED. DEC WILL NOT HAVE TO MAINTAIN A REGISTER (FOR DEC AND ITS CUSTOMERS) OF SECTION NUMBERS ASSIGNMENTS. WE WILL NOT NEED TO RESOLVE DUPLICATE ASSIGNMENTS, ETC. IN ORDER TO DO THIS, THE MONITOR WILL NEVER ASSIGN SECTION 7777. THUS A NOT ASSIGNED TRAP INVOLVING SECTION 7777 WILL BE INTERPRETED BY THE MONITOR AS A REQUEST TO ASSIGN A SECTION NUMBER. THE RH OF THE WORD WILL POINT TO ANY INFORMATION NEEDED, INCLUDING THE PROPER RH. EXTENDED FORMAT INDIRECT WORDS WITH SECTION 7777 WILL BE KEPT IN A PER-PROCESS COPY-ON-WRITE PAGE.

APPENDIX

THE FOLLOWING DESIGN DECISION EXPLANATIONS ARE INCLUDED HERE SO THAT WE WILL NOT FORGET THE REASONING WHICH LED TO THE DECISIONS. EACH POINT IS DEFINED BY A QUESTION IN THE FORM OF AN ALTERNATE PROPOSAL. THEN THE PROS AND CONS OF THE ALTERNATIVE ARE INDICATED BY THE LABELED PARAGRAPHS. THE LAST PARAGRAPH IS A BRIEF DESCRIPTION OF WHAT IS IN THE SPECIFICATION.

1. WHY NOT HAVE EXEC AND USER ADDRESS SPACE BE A SINGLE ADDRESS SPACE BY USING THE HIGH ORDER BIT OF THE SECTION FIELD TO INDICATE WHICH?

PRO: SIMPLIFIES THE HARDWARE SINCE ONLY ONE MAP IS NEEDED PER PROCESS INSTEAD OF TWO. ALSO MAY SIMPLIFY THE METHOD THE MONITOR USES TO REFERENCE THE CALLER (PXCT). THE AC FIELD BITS OF PXCT WOULD ONLY BE USED TO INDICATE WHETHER TO USE CURRENT ACS OR PREVIOUS ACS AND WHETHER TO GUARANTEE THAT THE USER BIT IS ON IN THE SECTION NUMBER OR NOT. ALSO RECURSIVE MONITOR CALLS COULD PASS USER ADDRESSES SINCE USER BIT IS IN ADDRESS (EACH LEVEL WOULD HAVE TO VERIFY THAT USER BIT WAS ON BY DOING A PXCT).

CON: IT IS NOT CLEAR THAT IT SIMPLIFIES THE HARDWARE SUFFICIENTLY TO PUT IT IN THE KL. IT MAY GO IN A FUTURE MACHINE.

SPEC: THE ADDRESS SPACES ARE SEPARATE AS ON THE KI. THE AC BITS ALSO CONTROL EFFECTIVE ADDRESS AND DATA ACCESSES. SEE PXCT SPEC FOR MONITOR USE.

2. WHY NOT SUPPRESS CARRIES OUT OF BIT 18 IN INSTRUCTIONS (AND IFIWS)?

PRO: INSTRUCTIONS (AND IFIWS) WORK CONSISTENTLY NO MATTER WHAT IS IN THE INDEX REGISTER.

CON: CAN'T CROSS A SECTION BOUNDARY.

SPEC: THE SPEC SAYS THAT WHENEVER GLOBAL INDEXING IS INVOLVED, SECTION BOUNDARIES ARE NOT OBSERVED. TRUE IN INSTRUCTIONS, IFIWS, AND EFIWS.

3. WHY NOT ELIMINATE THE ALTERNATE, SECTION-INDEPENDENT FORM OF AC ADDRESS, I.E., VMA[6-31]=0?

PRO: THEN LH=0 NOT CONFUSED WITH SECTION 0. ALSO CAN REFERENCE SHADOW ACS IN SECTION 0 JUST LIKE ANY OTHER SECTION. 0 IN LH IS USED FOR 3 THINGS, WHICH ARE NOT NECESSARILY INDEPENDENT:

A. SECTION 0 (IN EFIWS)

B. LOCAL INDEXING (IN LH OF XR)

C. HARDWARE ACS (IN EFIWS IF BITS 18-31=0).

CON: SOFTWARE NEEDS A SECTION INDEPENDENT WAY OF ADDRESSING THE ACS. THEN AN ADDRESS CAN BE PASSED, A SUBROUTINE CAN COPY, OR A UUO CAN STORE AN ADDRESS AND HAVE IT STILL REFER TO THE PROPER LOCATION. ALSO AN EFIW CAN SPECIFY ALL POSSIBLE LOCATIONS INCLUDING THE ACS.

SPEC: THE SPEC HAS TWO FORMS OF ADDRESSING THE ACS:

A. LOCAL ADDRESS LESS THAN 20

B. GLOBAL ADDRESS LESS THAN 20.

4. WHY NOT KEEP JSP AND JSR THE SAME AS ON THE KI? I.E., STORE FLAGS AND 18-BIT PC.

PRO: DON'T NEED TO FIND A NEW WAY TO GET PC FLAGS. JSP AND JSR AREN'T USED VERY MUCH FOR SUBROUTINES.

CON: WOULDN'T BE ABLE TO USE TO GO TO ANOTHER SECTION SINCE ONLY 18-BIT PC. WOULDN'T BE ABLE TO PUT SETUP ROUTINES FOR CALLING STANDARD IN A SEPARATE SECTION. CODE VERY SELDOM DOES ANYTHING WITH THE FLAGS. NO SPECIAL HARDWARE IS NEEDED TO MAKE JSP AND JSR WORK WITH GLOBAL PC.

SPEC: JSP AND JSR STORE A GLOBAL PC IF LH OF PC IS NON-ZERO.

5. WHY NOT MAKE ILDB AND IDPB WORK "CORRECTLY" WITH BYTE POINTERS WHICH HAVE AN INDIRECT BIT ON? I.E., INCREMENT LAST WORD FOUND IN INDIRECT CHAIN, INSTEAD OF BYTE POINTER ITSELF.

PRO: THEN A BYTE POINTER COULD BE USED TO SPECIFY BYTE DATA IN A DIFFERENT SECTION. WOULD NOT NEED TO INVENT THE DOUBLE WORD BYTE POINTER FORMAT. THEN EXTEND WOULD ONLY NEED 4 ACS.

CON: MANY SUBROUTINES WANT TO MAKE A COPY OF A BYTE POINTER FOR EFFICIENCY, INSTEAD OF REFERENCING INDIRECTLY THROUGH AN ARGUMENT LIST. IT WOULD BE HARD FOR THE CALLER TO STORE BACK THE UPDATED P AND S FIELDS IN THE BYTE POINTER AND THE ADDRESS IN THE LAST INDIRECT WORD.

SPEC: IN NON-ZERO SECTION, SPEC HAS ONE AND TWO WORD BYTE POINTERS DEPENDING ON BIT 12.

6. WHY NOT HAVE LUUOS TRAP TO CURRENT SECTION 40 AND 41?

PRO: FORMAT FOR 40 AND 41 IN EACH SECTION WOULD BE THE SAME AS ON KI. ALSO TWO PIECES OF SOFTWARE WHICH USE LUUOS COULD CO-EXIST IN THE SAME PROCESS, PROVIDED THEY WERE LOADED INTO DIFFERENT SECTIONS.

CON: HAVING "WIRED-IN" ADDRESSES IN HARDWARE IS A BAD IDEA AND RESTRICTS SOFTWARE. ALSO A SINGLE PROCESS-WIDE LUUO HANDLER WOULD BE MORE DIFFICULT.

SPEC: IN NON-ZERO SECTIONS, LUUO INFORMATION IS STORED IN A BLOCK OF 4 WORDS SPECIFIED BY THE CONTENTS OF A WORD IN THE UPT. THE GLOBAL PC AND GLOBAL EFFECTIVE ADDRESS IS STORED ALONG WITH THE FLAGS AND OPCODE.

7. WHY NOT TRAP JSA AND JRA IN NON-ZERO SECTIONS?

PRO: USER MAY BE MAKING A MISTAKE (IF EFFECTIVE ADDRESS IS IN A DIFFERENT SECTION) SINCE ONLY 18-BIT PC CAN BE STORED.

CON: EVEN THOUGH JSA AND JRA ARE RARELY USED, LEAVE FOR USE IN INTRA-SECTION CALLS SINCE COSTS NO EXTRA HARDWARE.

SPEC: JSA AND JRA CONTINUE TO STORE ONLY AN 18-BIT PC.

8. WHY CAN'T INSTRUCTIONS EXECUTED BY XCT WORK AS IF THE INSTRUCTION HAD BEEN COPIED TO THE LOCATION OF THE XCT?

PRO: SIMPLE RULE. IT IS A NATURAL EXTENSION OF KI SPEC.

CON: IF A SUBROUTINE IS EXECUTING A TABLE IN ANOTHER SECTION, SOME ENTRIES WOULD WANT TO HAVE LOCAL REFERENCES REFER TO THE LOCAL SECTION INSTEAD OF THE PC SECTION. ALSO IT WOULD BE HARD TO WRITE AN INTERPRETER WHICH RUNS IN A DIFFERENT SECTION.

SPEC: COMPROMISES: PC IS STORED AS LOCATION OF XCT+1 TO BE COMPATIBLE WITH KI. LOCAL STACK COMES FROM PC SECTION BECAUSE IT PROBABLY IS WHAT A DISPATCHER WANTS. ALSO IT IS EASIER IN HARDWARE. EFFECTIVE ADDRESS OF OBJECT INSTRUCTION STARTS AT SECTION OF OBJECT INSTRUCTION (NOT SECTION OF XCT). IF CODE PERFORMING XCT WISHES TO START EFFECTIVE ADDRESS IN PC SECTION, IT CAN FIRST PICKUP THE INSTRUCTION INTO AC AND THEN DO XCT AC.

9. WHY NOT USE A NEW OPCODE FOR XMOVEI, INSTEAD OF SETMI?

PRO: WOULD NOT PERTURB THE SYMMETRY OF THE BOOLEAN SET.

CON: COULD NOT WRITE CODE WHICH RUNS ON KI ALSO. FURTHERMORE, SETMI IS USED VERY INFREQUENTLY.

SPEC: MOVXA (SETMI) MOVES THE SECTION NUMBER OF THE EFFECTIVE ADDRESS TO THE LH OF THE AC, AS WELL AS THE LOW ORDER PART TO THE RH.

10. WHY ISN'T DEFAULT SECTION OF AN INSTRUCTION EXECUTED UNDER XCT ALWAYS THE SECTION WHERE THE INSTRUCTION IS?

PRO: THEN THERE WOULD BE A CONSISTENT RULE FOR THE PROGRAMMER FOR THE 4 CASES IN WHICH DEFAULT SECTIONS ARE USED: (1) PC STORING INSTRUCTIONS, (2) LOCAL STACK POINTERS, AND (3) LOCAL EFFECTIVE ADDRESS.

CON: IT IS MORE IMPORTANT TO DO WHAT IS MOST USEFUL FOR SOFTWARE, EVEN IF IT IS INCONSISTENT. (THOREAU WOULD BE PROUD). PC STORING WOULD BE INCOMPATIBLE WITH KI, WHICH STORES XCT+1. LOCAL STACKS PROBABLY WANT TO BE IN XCT SECTION. ALSO EASIER IN HARDWARE, SINCE STACK INSTRUCTIONS WORK THE SAME WHETHER UNDER XCT OR NOT. EFFECTIVE ADDRESS OF OBJECT INSTRUCTION DOES START IN SECTION WHERE THE INSTRUCTION IS.

SPEC: PC STORED IS XCT+1, LOCAL STACK POINTER ASSUMES PC SECTION, AND EFFECTIVE ADDRESS STARTS IN SECTION WHERE INSTRUCION IS.

11. WHY NOT HAVE JUST ONE FORM OF INDIRECT WORD?

PRO: SIMPLER IN HARDWARE AND SOFTWARE.

CON: NEED BOTH FORMS:

THE INSTRUCTION FORMAT INDIRECT WORD IS NEEDED:

A. SOFTWARE NEEDS BIT IN ARGUMENT LISTS FOR ARGUMENT TYPE INFORMATION.

B. NEED TO BE ABLE TO HAVE ARGUMENT LISTS WHICH WILL WORK ON KI, KI COMPATIBLE SECTION AND IN EXTENDED SECTIONS. WHILE IT WOULD BE POSSIBLE FOR THE LINKER TO CONVERT FROM ONE FORMAT TO ANOTHER, IT SEEMS A PROBLEM.

THE EXTENDED FORMAT INDIRECT WORD IS NEEDED TO REFERENCE LARGE ARRAYS. THE EQUIVALENT CODE IS:

```
MOVE    B,[EFIW SECTION,TABLE]
ADD     B,I
FMP     T1,(B)
```

INSTEAD OF:

```
FMP     T1,@[EFIW SECTION,TABLE(I)]
```

WHERE I CONTAINS THE INDEX INTO THE ARRAY. THIS RESULTS IN 3 MEMORY CYCLES INSTEAD OF 5 AND TAKES 2 WORDS INSTEAD OF 4. (ACTUALLY SINCE THE LITERALS IN BOTH EXAMPLES WOULD BE POOLED, THE ACTUAL COMPARISON IS 1+ WORDS COMPARED TO 3+.

SPEC: HAS BOTH IFIWS AND EFIWS.

12. WHY NOT HAVE A LOCAL FORM OF INDEXING IN EXTENDED FORMAT INDIRECT WORDS, IE., IF LH IS NEGATIVE OR 0, ADD ONLY RH OF XR WITH SIGN EXTENDED?

THE RIGHT HALF OF FLOW CHART WOULD BECOME:

IF LH(XR) > 0, XR + VMA<6:35> => VMA<6:35>
ELSE XR<SIGN EXTENDED> + VMA<6:35> => VMA<6:35>

INSTEAD OF JUST:

XR + VMA<6:35> => VMA<6:35>

PRO: THIS WOULD ALLOW AOBJN TO BE USED FOR SMALL ARRAYS IN OTHER SECTIONS, AS SHOWN BELOW:

```
        MOVSI   I,-LENGTH
LOOP:   ADD     T,@(EFIW SECTION,TABLE(I))
        ..
        AOBJN   I,LOOP
```

CON: LARGE ARRAYS (GREATER THAN A SECTION) COULD NOT BE REFERENCED NEGATIVELY.

SPEC: BITS 6-35 OF INDEX REGISTER AND EFIW ARE TREATED THE SAME WAY AS BITS 18-35 OF INDEX REGISTER AND INDIRECT WORDS IN THE KI.

13. WHY NOT HAVE A LOCAL FORM OF INDEXING IN EXTENDED FORMAT INDIRECT WORDS, IE., IF LH OF XR IS NEGATIVE OR 0, ADD ONLY RH OF XR WITH NO SIGN EXTENSION. THIS QUESTION IS LIKE QUESTION 12 EXCEPT NO SIGN EXTENSION.

THE RIGHT HALF OF FLOW CHART WOULD BECOME:

IF LH(XR) > 0, XR<0:35> + VMA<6:35> => VMA<6:35>
ELSE XR + VMA => VMA<18:35>

PRO: THEN INDEX REGISTERS WOULD WORK THE SAME IN ALL FORMATS OF INSTRUCTIONS AND INDIRECT WORDS. EITHER LOCAL OR GLOBAL INDEXING COULD BE SPECIFIED ANYWHERE INDEXING IS DONE.

CON: IT WOULD VIOLATE THE OBJECTIVE OF IGNORING SECTION BOUNDARIES IN EFIWS, WHEN THE INDEX WAS SMALL. SECONDLY IT WOULD PREVENT LARGE ARRAYS (GREATER THAN A SECTION) FROM BEING REFERENCED NEGATIVELY (SEE 12 ABOVE).

SPEC: INDEXING IN EFIWS JUST DOES A 30-BIT ADD TO 30-BIT EFIW.

14. WHY NOT TRAP IF A USER ATTEMPTS TO JUMP INTO THE ACS WITH THE SECTION NUMBER NON-ZERO?

PRO: THE ANOMALIES IN SECTION 13 OF THIS SPECIFICATION WOULD NOT HAVE TO BE EXPLAINED IN THE MANUAL, UNDERSTOOD BY THE PROGRAMMER. ALSO FUTURE MACHINES WOULD NOT BE CONSTRAINED TO IMPLEMENT ACS IN THIS WAY.

CON: IT IS USEFUL, EVEN THOUGH NO FASTER, TO BE ABLE TO PUT CODE IN THE ACS TO MAKE A LOOP WHICH IS DEPENDENT ON DATA. THEN THE PROGRAM DOES NOT NEED TO STORE THE LOOP IN TRANSIENT STORAGE ON THE STACK AND JUMP TO IT.

SPEC: ALLOWS JUMPING TO THE ACS. THE USER MUST BE AWARE THAT THE ACS ARE A LITTLE "DIFFERENT" FROM MEMORY. SEE SECTION 13.

15. WHY NOT MAKE CODE IN BITS 0-1 FOR IFIW BE 00, INSTEAD OF 10?

PRO: MORE COMPATIBLE WITH EXISTING SUBROUTINE ARGUMENT LIST IN WHICH BITS 0-8 ARE ZERO.

CON: THEN THE SENSE OF THE SIGN BIT WOULD BE THE OPPOSITE OF ITS SENSE IN XRS, IE., 1 IF LOCAL, 0 IF GLOBAL. THIS MEANS THAT AN INDEX REGISTER QUANTITY COULD NOT BE INDIRECTED THROUGH AND VICE VERSA.

SPEC: THE SPECIFICATION CALLS FOR BITS 0-1 TO BE 10 IN IFIWS. EXISTING CODE WILL NOT BE RUNNING NON-ZERO SECTIONS WITHOUT SOME CHANGES ANYWAY.

16. WHY NOT HAVE A SPECIAL EXECUTE INSTRUCTION INDICATE THAT THE EFFECTIVE ADDRESS COMPUTATION IS TO BE PERFORMED WITH LARGE ADDRESSES (PROPOSAL #1 - EXCT)?

PRO: THEN EXISTING KI CODE COULD BE PATCHED TO TAKE ADVANTAGE OF EXTENDED ADDRESSING.

CON: TOO MUCH OVERHEAD. AN EXTRA WORD OF SPACE AND AN EXTRA MEMORY CYCLE IS REQUIRED FOR EACH EXTENDED REFERENCE.

SPEC: USE EXTENDED ADDRESSING WHEN NOT IN SECTION 0.

17. WHY NOT EXTEND THE INDEX REGISTERS TO THE LEFT OF BIT 0 AND PROVIDE HARDWARE EXTENSION REGISTERS (PROPOSAL #3)?

PRO: ALL INSTRUCTIONS WORK WITH EXTENDED ADDRESSING IN AN OBVIOUS WAY.

CON: ADDS EXTRA CONTEXT TO A PROCESS AND A SUBROUTINE. MAYBE ERROR PRONE, SINCE INDEXING COULD HAVE SIDE EFFECTS. TOO RADICAL A CHANGE TO THE -10 ARCHITECTURE, WHERE ACS ARE JUST LIKE MEMORY.

SPEC: THE PROGRAMMER PUTS THE SECTION NUMBER IN THE LEFT HALF OF THE INDEX REGISTER.

18. WHY NOT HAVE DOUBLE WORD INDIRECT WORDS, IF BIT 12=1 (PROPOSAL #3)?

PRO: THEN BIT 12 IS CHECKED THE SAME IN INDIRECT WORDS AS IN BYTE POINTERS. IN A FUTURE MACHINE, BITS 0-11 OF AN INDIRECT WORD COULD MEAN P AND S. THEN INDIRECTION IS JUST A SPECIAL CASE OF BYTE OPERATION AND VICE VERSA.

CON: AN EXTRA WORD OF SPACE AND TIME WOULD BE REQUIRED FOR EACH INDIRECT REFERENCE. SOFTWARE NEEDS SOME BITS FOR ARGUMENT TYPE IN ARGUMENT LIST.

SPEC: THE SECTION NUMBER APPEARS IN THE SAME WORD. THE INDIRECT WORD WITH BITS 0-1=11 IS RESERVED FOR THE FUTURE.

19. WHY NOT MAKE A SECTION # OF 0 IN AN EFIW MEAN LOCAL REFERENCE, INSTEAD OF SECTION 0?

PRO: THEN INDEX AND INDIRECT FORMATS ARE EVEN MORE COMPATIBLE.

CON: DON'T NEED IT, SINCE CAN ALWAYS USE IFIW WHEN LOCAL ADDRESSING IS WANTED. ALSO IT WOULD MAKE IT IMPOSSIBLE FOR SPECIAL CODE LIKE DUBUGGER, MONITOR, OR COMPATIBILITY PACKAGE TO REFERENCE SECTION 0. THE KI COMPATIBILITY SECTION WOULD HAVE TO BE MOVED TO ANOTHER SECTION.

SPEC: A 0 IN AN EFIW EXPLICITLY MEANS SECTION 0.

20. WHY NOT IMPLEMENT A TRUE DOMAIN ARCHITECTURE?

PRO: MORE RELIABLE SOFTWARE, SINCE HARDWARE WOULD PROVIDE PROTECTION.

CON: WE DON'T UNDERSTAND IT COMPLETELY YET. HOWEVER, WE SHOULD STUDY IT BEFORE NEXT MACHINE. IT WILL REQUIRE MORE EFFORT THAN EXTENDED ADDRESSING HAS. SEE "AN EXPERIMENTAL KERNAL/DOMAIN ARCHITECTURE", BY M. SPIER, T. HASTINGS, AND D. CUTLER, ACM OPERATING SYSTEM REVIEW, VOL 7 #4, OCT 1973.

SPEC: THE PROTECTION FOR INTER-SECTION REFERENCES IS THE SAME AS FOR INTRA-SECTION REFERENCES.

[END OF CH2S02.SPC]

1080,2040,2060 ENGINEERING FUNCTIONAL SPEC - CHAP 2,3

TO: KL10 SPEC ANNOUNCEMENT LIST

TITLE: MONITOR CALLING (MUUO, PXCT) - REV 2

STATUS: THIS CHAPTER BRINGS TOGETHER THE MEMOS ON PXCT. IT FLOW CHARTS THE INSTRUCTION EXPLICITLY, INCLUDING INTERACTIONS WITH EXTENDED ADDRESSING. IT IS MUCH SIMPLER THAN THE ORIGINAL SPEC. IN THIS SPEC, ONLY THE EFFECTIVE ADDRESS CALCULATION AND OPERAND FETCH IS MODIFIED WHEN UNDER PXCT. THIS CHAPTER WILL BE REVIEWED TUES, 11 MAR, 10-NOON, IN THE LEPRECHAUN LOUNGE. WARNING: BECAUSE THIS SPEC CONTAINS UNANNOUNCED CAPABILITIES, THIS SPEC IS COMPANY CONFIDENTIAL.

FILE: [EFS]CH2S03.SPC

PDM #: 200-200-011-03

DATE: 3 FEB 76

SUPERSEDED MEMOS: PXCT DEFINITION, R. REID, 22 OCT 73; AC BITS IN PXCT, J. LEONARD, 21 MAY 73; PREVIOUS CONTEXT EXECUTE, D. MURPHY, 1 MAY 73.

SUPERSEDED SPECS: MONITOR CALLING (MUUO, PXCT) - REV 0, 13 MAR 74, PXCT.SPC - REV 1, 18 FEB 75

ENGINEER: D. MURPHY, J. LEONARD, R. REID, T. HASTINGS

APPROVED:

EDITOR: T. HASTINGS

TYPIST: J. MC CARTHY

REVIEWED:

DISTRIBUTED:

ABSTRACT

WHEN THE MONITOR IS CALLED VIA A TRAP, THE ADDRESSING SPACE OF THE CALLER IS USUALLY DIFFERENT FROM THAT OF THE MONITOR. THE PREVIOUS-CONTEXT EXECUTE (PXCT) INSTRUCTION PROVIDES A MECHANISM FOR THE MONITOR TO REFERENCE THE CALLING ADDRESS SPACE, INCLUDING HARDWARE ACS. ON THE KL10, PXCT IS RESTRICTED TO USE IN EXEC MODE. THEREFORE, THE XCT OPCODE IS USED, AS ON THE KI. HOWEVER, IT HAS BEEN ENHANCED TO WORK WITH EXTENDED ADDRESSING. ALSO THE USE OF PREVIOUS ACS HAS BEEN GENERALIZED OVER THE KI.

REVISION HISTORY

REV	DESCRIPTION	CHG NO	ORIG	DATE	APPD BY	DATE
0	ORIGINAL WITH CSWX,SXCT			13 MAR 74		
1	PXCT MEMO - CSWX,SXCT REMOVED			18 FEB 75		
2	UPDATE FROM 26 FEB REVIEW			10 MAR 75		
3	IMPLEMENTATION DETAILS, CLARIFICATION			3 FEB 76		

0. CONTENTS

1. INTRODUCTION

WHEN THE MONITOR IS CALLED VIA MUUO TRAP (INCLUDING JSYS), THE ADDRESS SPACE OF THE CALLER IS USUALLY DIFFERENT FROM THAT OF THE MONITOR. IN ADDITION, ON THE KL10 THE EXEC AND USER ADDRESS SPACES HAVE EACH BEEN EXTENDED TO PERMIT 32 256K SEPARATE ADDRESS SPACES CALLED SECTIONS. THE PREVIOUS-CONTEXT EXECUTE (PXCT) INSTRUCTION PROVIDES A PROTECTED MECHANISM FOR THE MONITOR TO REFERENCE THE CALLING CONTEXT NO MATTER WHICH OF THE 64 EXEC OR USER SECTIONS IS CALLING THE MONITOR. FUTURE MACHINES MAY EXTEND THE NUMBER OF SECTION FROM 64 TO 8192. FOR A COMPLETE DESCRIPTION OF EXTENDED ADDRESSING SEE CHAP 2.2, USER INTERFACE TO EXTENDED ADDRESSING.

THIS PROPOSAL IS INTENDED TO REPLACE ALL PREVIOUS DOCUMENTS ON PXCT AND MUUOS. THE REASONS FOR SOME OF THE DESIGN DECISIONS ARE FLAGGED WITH NUMBERS IN BRACKETS AND ARE EXPLAINED IN THE APPENDIX.

2. BACKGROUND AND MOTIVATION

AN EXECUTE INSTRUCTION WITH NON-0 AC FIELD EXECUTED IN EXEC MODE ON THE KI10 HAS GENERALLY BEEN KNOWN AS "MAPPED EXECUTE" OR "EXECUTE PAGED". IT IS NOW UNDERSTOOD THAT THIS FUNCTION IS MORE ACCURATELY DESCRIBED AS "PREVIOUS CONTEXT EXECUTE", I.E., IT EXECUTES AN INSTRUCTION WHILE FORCING SELECTED MEMORY REFERENCES TO HAVE THE SAME EFFECT AS THEY WOULD HAVE HAD IF DONE BY THE PROGRAM WHICH CALLED THE CURRENT ROUTINE. ON THE KL10 THE CURRENT ROUTINE WILL BE RESTRICTED TO EXEC MODE (1). AS ON THE KI, THE AC FIELD OF THE XCT INSTRUCTION WILL BE INTERPRETED SPECIALLY IN EXEC MODE ONLY (NOT USER OR USER-IOT MODE). FOR CLARITY, THE MNEMONIC PXCT WILL BE USED WHENEVER THE AC FIELD IS NON-ZERO.

FOR EXAMPLE, IF A USER-MODE PROGRAM PERFORMS A MONITOR CALL (MUUO) THEN THE "PREVIOUS" CONTEXT IS THE USER ADDRESS SPACE AND USER AC'S. IF THIS MONITOR ROUTINE ITSELF PERFORMS A MONITOR CALL, THE PREVIOUS CONTEXT WILL BE THE MONITOR ADDRESS SPACE AND THE AC'S IN USE BY THAT ROUTINE, AND THE CURRENT CONTEXT WILL ALSO BE THE MONITOR ADDRESS SPACE, BUT WITH A (POTENTIALLY) DIFFERENT SET OF AC'S. THE MULTIPLE AC BLOCK FEATURE OF THE KL IS USED SO THAT BOTH THE CURRENT AND PREVIOUS ACS ARE IMPLEMENTED IN HARDWARE.

THE EXTENDED ADDRESSING OF THE KL10 PROVIDES 32 256K USER VIRTUAL ADDRESS SPACES, CALLED SECTIONS. EACH CPU HAS 32 EXEC VIRTUAL SECTIONS. THE INSTRUCTION REPERTOIRE AND EFFECTIVE ADDRESS CALCULATION IN EXEC AND USER SECTION 0 IS LIKE THE KI. SECTION 0 IS CALLED THE "KI COMPATIBLE SECTION". THE INSTRUCTION REPETOIRE AND EFFECTIVE ADDRESS CALCULATION IN EXEC AND USER NON-ZERO SECTIONS HAS BEEN EXTENDED. IN ADDITION TO ITS PREVIOUSLY STATED FUNCTIONS, PXCT ALSO PROVIDES A MEANS TO FORCE THE EFFECTIVE ADDRESS COMPUTATION AND/OR OPERAND REFERENCES TO USE THE PREVIOUS CONTEXT SECTION, THAT IS, THE SECTION IN WHICH THE CALLING PROGRAM WAS RUNNING. THUS PXCT PROVIDES A MEANS FOR PASSING ARGUMENTS AND DATA TO AND FROM THE MONITOR. ONCE AN EFFECTIVE ADDRESS COMPUTATION ENTERS (EXEC OR USER) SECTION 0, IT CAN NEVER LEAVE, WHETHER UNDER PXCT OR NOT. THUS, IN ORDER TO SUPPORT EXTENDED ADDRESSING, THE PART OF THE MONITOR (THE JSYS OR UUO LEVEL) WHICH USES PXCT MUST BE MOVED TO A NON-ZERO SECTION.

NOTE THAT IT IS NOT INTENDED FOR PXCT TO BE USED BY THE MONITOR FOR UNSOLICITED REFERENCES TO THE USER PROGRAM. INSTEAD, SUCH REFERENCES WOULD BE TYPICALLY DONE BY THE MONITOR MAPPING THE PAGE TEMPORARILY INTO ITS ADDRESS SPACE.

3. PXCT SPECIFICATION

IN EXEC MODE THE AC FIELD OF PXCT (SAME OPCODE AS XCT) IS USED TO SPECIFY WHICH OF THE MEMORY REFERENCES GENERATED BY THE OBJECT INSTRUCTION ARE TO USE THE PREVIOUS CONTEXT AND WHICH ARE TO USE THE CURRENT CONTEXT. THE AC OPERAND OF THE OBJECT INSTRUCTION ALWAYS USES THE CURRENT AC'S AS SPECIFIED BY BITS 9-12 OF THE OBJECT INSTRUCTION; ONLY THOSE AC AND MEMORY REFERENCES GENERATED DURING THE EFFECTIVE ADDRESS COMPUTATION AND THE DATA READ/WRITE REFERENCES ARE CONTROLLABLE BY THESE BITS.

3.1 PXCT, AC FIELD ENCODING

THE ENCODING SHALL BE AS FOLLOWS; WHERE A 1 IN THE SPECIFIED BIT POSITION CAUSES REFERENCES OF THE STATED TYPE TO BE DONE IN THE PREVIOUS CONTEXT, AND A 0 CAUSES THEM TO BE DONE IN THE CURRENT CONTEXT.

B9 (E1) EFFECTIVE ADDRESS COMPUTATION, E, OF THE OBJECT INSTRUCTION.

B10(D1) MEMORY OPERAND REFERENCES, BOTH FETCH AND STORE, SPECIFIED BY E. THEREFORE: SOURCE FOR PUSH; DESTINATION FOR POP, AND BLT; SOURCE BYTE DATA FETCH IN EXTEND; DESTINATION WORD DATA STORE IN EXTEND-BLT.

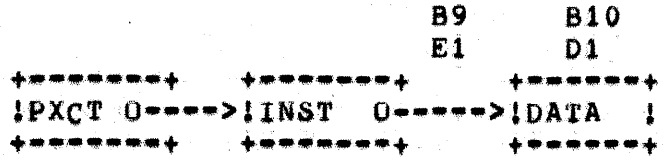
B11(E2) EFFECTIVE ADDRESS COMPUTATION OF BYTE POINTER; EFFECTIVE ADDRESS COMPUTATION SPECIFIED BY DESTINATION BYTE POINTER IN EXTEND.

B12(D2) BYTE DATA FETCH OR STORE; STACK WORD IN PUSH, POP; SOURCE DATA IN BLT; DESTINATION BYTE DATA FETCH/STORE IN EXTEND; SOURCE WORD DATA FETCH IN EXTEND-BLT.

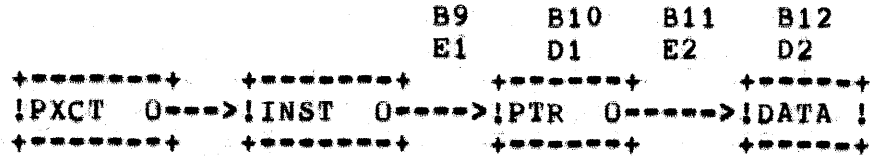
THE EFFECTIVE ADDRESS COMPUTATION MAY INVOLVE ARBITRARY INDEXING AND/OR INDIRECTION WHICH FOLLOW THE RULES FOR EXTENDED ADDRESSING, I.E., VMA<6-17> IS TESTED AT EVERY STEP TO SEE WHETHER TO USE KI STYLE (SECTION 0) OR EXTENDED ADDRESSING.

IF BIT 9=1, MEANING COMPUTE EFFECTIVE ADDRESS IN CALLER'S CONTEXT, THE SECTION NUMBER OF THE CALLER IS COPIED INTO VMA<6-17> BEFORE THE EFFECTIVE ADDRESS OF THE OBJECT INSTRUCTION IS COMPUTED. THE SECTION NUMBER OF THE CALLER IS KEPT IN A 5-BIT REGISTER CALLED PREVIOUS CONTEXT SECTION, PCS.

THE EFFECT OF EACH BIT CAN BE SEEN IN THE FOLLOWING PICTURE FOR THE TYPICAL INSTRUCTION:



FOR A BYTE INSTRUCTION:



3.1.1 PXCT AC BIT COMBINATIONS

NOT ALL OF THE 16 POSSIBLE BIT COMBINATIONS ARE USEFUL. IF THE EFFECTIVE ADDRESS COMPUTATION BIT (BIT 9 OR 11) SPECIFIES PREVIOUS CONTEXT, IT ONLY MAKES SENSE FOR THE CORRESPONDING DATA ACCESS BIT TO BE PREVIOUS ALSO (BIT 10 OR 12). THE HARDWARE IS FREE TO DO WHAT IS EASIEST FOR THE "NOT APPLICABLE" CASES SINCE THE SOFTWARE WILL NOT USE THEM. FURTHERMORE, UNTIL CUSTOMERS START WRITING THEIR OWN OPERATING SYSTEMS IN LARGE NUMBERS, THERE IS NO NEED FOR THE HARDWARE TO TRAP THE "NOT APPLICABLE" CASES. THE FOLLOWING TABLE SHOWS THE LEGAL COMBINATIONS:

INST.	E1	D1	E2	D2	WHAT'S IN PREVIOUS CONTEXT
MOVE	0	0	0	0	ALL CURRENT CONTEXT
	0	1	0	0	E CURRENT, DATA PREV
	1	1	0	0	E PREV, DATA PREV
BYTE	0	0	0	0	ALL CURRENT CONTEXT
	0	0	0	1	BYTE DATA PREV
	0	0	1	1	E(DATA) AND DATA PREV
	0	1	1	1	C(E), E(DATA), DATA PREV
	1	1	1	1	E,C(E), E(DATA), DATA PREV
BLT	0	0	0	0	ALL CURRENT CONTEXT
	0	0	0	1	SOURCE PREV.
	0	1	0	0	DEST. PREV
	0	1	0	1	SOURCE PREV, DEST. PREV
STACK	0	0	0	0	ALL CURRENT
	0	0	0	1	STACK PREV
	0	1	0	0	C(E) PREV
	0	1	0	1	C(E) AND STACK PREV
	1	1	0	0	E, C(E) PREV
	1	1	0	1	E, C(E) AND STACK PREV
EXTEND	0	0	0	0	ALL CURRENT
	0	0	0	1	DEST. DATA PREV
	1	0	0	1	DEST E, DEST DATA PREV
	0	0	1	0	SOURCE DATA PREV.
	0	0	1	1	SOURCE DATA PREV, DEST DATA PREV
	1	0	1	0	SOURCE E AND DATA PREV
	1	0	1	1	SOURCE E AND DATA PREV, DEST E AND DATA PREV
XBLT	0	0	0	0	ALL CURRENT
	0	0	0	1	SOURCE DATA PREV
	0	0	1	0	DEST. DATA PREV
	0	0	1	1	SOURCE AND DEST DATA PREV

3.2 HARDWARE AC BLOCKS

THE FOLLOWING CHANGE FROM THE KI IS SPECIFIED FOR ALL ORDINARY INSTRUCTIONS:

1. THE 2-BIT QUANTITY DEFINED AS "USER FAST MEMORY BLOCK" ON THE KI10 IS RENAMED "PREVIOUS-CONTEXT AC" (PAC) AND IS USED ONLY DURING PXCT AS DESCRIBED BELOW. IT IS INCREASED TO 3 BITS IN ORDER TO PROVIDE 8 AC BLOCKS.

2. THE NEW 3-BIT REGISTER IS DEFINED AS THE "CURRENT-CONTEXT AC BLOCK" (CAC), AND SHALL DESIGNATE THE AC BLOCK TO BE USED FOR ALL AC REFERENCES WHETHER THE PROCESSOR IS IN USER OR EXEC MODE.

NOTE: FUTURE MACHINES WILL NOT NECESSARILY HAVE MORE THAN ONE SET OF HARDWARE ACS. HOWEVER THEY WILL IMPLEMENT THE FUNCTIONALITY OF COMPUTING THE EFFECTIVE ADDRESS, INCLUDING INDEXING AND INDIRECTION, USING AC VALUES THAT ARE DIFFERENT FROM THE MONITORS BECAUSE THEY ARE THE CALLER'S AC VALUES.

FOR AC REFERENCES UNDER PXCT, THE FOLLOWING APPLIES:

1. THE AC REFERENCE(S) FOR THE AC OPERAND (IF ANY) OF THE OBJECT INSTRUCTION (BITS 9-12 OF OBJECT INSTRUCTION) ALWAYS USE THE AC'S IN THE "CURRENT AC BLOCK", AS SPECIFIED BY THE CAC REGISTER.

2. OTHER AC REFERENCES, MEANING REFERENCES FOR INDEXING DURING AN EFFECTIVE ADDRESS COMPUTATION AND ALL OTHER ADDRESSES WITH BITS 18-31 = 0 GENERATED DURING THE EFFECTIVE ADDRESS COMPUTATION OR DURING THE EXECUTION OF THE OBJECT INSTRUCTION, ARE CONTROLLED BY ONE OF THE BITS OF THE AC FIELD OF THE PXCT INSTRUCTION AS DEFINED ABOVE. WHEN THE CONTROL BIT IS A 1 (INDICATING PREVIOUS CONTEXT REFERENCE), THE REFERENCE WILL USE THE AC(S) IN THE AC BLOCK SPECIFIED BY THE "PREVIOUS-CONTEXT AC BLOCK" (PAC). WHEN THE CONTROL BIT IS A 0, THE REFERENCE WILL USE THE AC(S) IN THE AC BLOCK SPECIFIED BY THE "CURRENT-CONTEXT AC BLOCK" (CAC).

NOTE THAT, UNLIKE THE KI10, AC REFERENCES UNDER PXCT ARE NOT AFFECTED AT ALL BY PCU, THEY ALWAYS USE THE PREVIOUS OR CURRENT AC BLOCK AS DIRECTED BY THE CONTROL BITS OF PXCT. THE AC STACK NOTION IN THE KI PROVED DIFFICULT TO IMPLEMENT WHEN INDEXING IN THE PREVIOUS CONTEXT WAS CALLED FOR. AS A RESULT, ON THE KL MONITOR CALLS FROM THE MONITOR ARE MORE SIMILAR TO MONITOR CALLS FROM THE USER, ON A MONITOR TO MONITOR CALL, THE MONITOR CAN EITHER COPY THE ACS OR CHANGE THE CONTENTS OF CAC AND PAC APPROPRIATELY.

3.3 PROCESS CONTEXT REGISTERS FOR PXCT

IN ADDITION TO THE 3-BIT REGISTERS CURRENT-CONTEXT AC BLOCK (CAC) AND PREVIOUS-CONTEXT AC BLOCK (PAC), THE FOLLOWING HARDWARE PROCESS STATE VARIABLES ARE DEFINED FOR USE UNDER PXCT.

NAME	# BITS	FUNCTION
PCS	12 (KL=5)	PREVIOUS CONTEXT SECTION [NEW]
PCU	1	PREVIOUS CONTEXT USER-MODE [ON KI]
PCP	1	PREVIOUS CONTEXT PUBLIC [ON KI]

THESE PROCESS STATE VARIABLES ARE USED ONLY WHEN THE PROCESSOR IS IN EXEC MODE. AS ON THE KI10, PCU IS PC BIT 6 AND PCP IS PC BIT 0 WHEN THE PROCESSOR IS IN EXEC MODE. IN USER MODE, THESE BITS HAVE THEIR TRADITIONAL MEANINGS (BIT 0 = OVERFLOW, BIT 6 = USER IO).

3.3.1 PREVIOUS CONTEXT SECTION (PCS)

THE PREVIOUS CONTEXT SECTION REGISTER IS A POTENTIALLY 12-BIT (5 BITS ON KL) PROCESS STATE VARIABLE WHICH CONTAINS THE SECTION NUMBER FROM WHICH THE MOST RECENT MONITOR CALL WAS MADE. SINCE MONITOR CALLS CAN BE MADE FROM BOTH EXEC AND USER MODE, PCU DISTINGUISHES EXEC AND USER SECTION NUMBERS (SEE 3.3.2).

IF THE PXCT CONTROL BIT E1 IS 1 (COMPUTE EFFECTIVE ADDRESS IN PREVIOUS CONTEXT), VMA BITS 6-17 ARE REPLACED BY PCS BEFORE THE EFFECTIVE ADDRESS OF THE OBJECT INSTRUCTION IS COMPUTED. SEE ATTACHED FLOW CHART. THUS PCS WILL AFFECT WHETHER KI OR EXTENDED ADDRESSING RULES ARE USED FOR THE EFFECTIVE ADDRESS COMPUTATION. SEE CHAPTER 2.2, EXTENDED ADDRESSING.

PCS IS IMPLEMENTED AS AN INTERNAL REGISTER AND IS NOT PART OF THE NEW FLAGS AND PC DOUBLE WORD [2]. INSTEAD, IT IS SETTABLE AND READABLE BY THE MONITOR USING A SINGLE DATAO AND DATAI. SEE SECTION 4, MONITOR UUO'S BELOW.

3,3,2 PREVIOUS CONTEXT USER-MODE

PREVIOUS CONTEXT USER-MODE IS A 1-BIT PROCESS STATE VARIABLE WHICH, IF 1, SPECIFIES THAT THE MOST RECENT MONITOR CALL WAS FROM USER-MODE. IF A 1, PCU CAUSES ALL NON-AC PREVIOUS CONTEXT REFERENCES GENERATED UNDER PXCT TO USE THE USER VIRTUAL ADDRESS SPACE AS SPECIFIED BY THE C(UBR). IF A 0, PCU CAUSES ALL NON-AC PREVIOUS CONTEXT REFERENCES GENERATED UNDER PXCT TO USE THE EXEC VIRTUAL ADDRESS SPACE AS SPECIFIED BY THE C(EBR). THIS BIT CAN BE CONSIDERED TO EXTEND THE PREVIOUS CONTEXT SECTION NUMBER BY 1 BIT TO GIVE A PROCESS-WIDE SECTION NUMBER, I.E., 6-BITS ON KL, 13-BITS ON A FUTURE MACHINE. A FUTURE MACHINE MAY CHOOSE TO IMPLEMENT THE USER BIT AS THE HIGH ORDER BIT OF THE SECTION NUMBER FIELD THROUGHOUT, IN WHICH CASE THE PROCESS-WIDE SECTION NUMBER WOULD BE 12-BITS.

AS ON THE KI, PCU IS IMPLEMENTED AS BIT 6 OF THE PC FLAGS WHEN THE CPU IS IN EXEC-MODE. BIT 6 IS CALLED USER-IO MODE WHEN THE CPU IS IN USER-MODE.

3,3,3 PREVIOUS CONTEXT PUBLIC (PCP)

PREVIOUS CONTEXT PUBLIC IS A 1-BIT PROCESS STATE VARIABLE WHICH, IF 1, SPECIFIES THAT THE MOST RECENT MONITOR CALL WAS FROM A PUBLIC PAGE. IF A 1, PCP CAUSES ALL PREVIOUS CONTEXT REFERENCES TO CONCEALED PAGES TO TRAP. THUS A MALICIOUS USER CANNOT PASS THE MONITOR AN ADDRESS IN A CONCEALED PAGE UNLESS THE CALLER IS CONCEALED. THIS BIT IS SET FROM THE PAGE SPECIFIED BY THE PC OF THE MUUO OR JSYS, NOT PC+1. THUS AN MUUO CAN BE DONE FROM THE LAST LOCATION IN A PUBLIC PAGE WHICH ABUTS A CONCEALED PAGE.

AS ON THE KI, PCP IS IMPLEMENTED AS BIT 0 OF THE PC FLAGS WHEN THE CPU IS IN EXEC-MODE. BIT 0 IS OVERFLOW WHEN THE CPU IS IN USER-MODE.

4. MONITOR UUOS (MUUO)

PCS IS READABLE/SETTABLE BY DATAI/DATAO PAG,. DATAO PAG, PROVIDES INDIVIDUAL LOAD BITS, SO THAT PCS CAN BE CHANGED WITHOUT CHANGING ANY OTHER STATES OF THE HARDWARE [2]. PCU CHANGES ONLY DURING A MONITOR CALL OR RETURN. IT SHALL BE THE RESPONSIBILITY OF THE HARDWARE/MICROCODE WHILE PERFORMING ANY MUUO TO STORE THE VALUE OF THESE VARIABLES IN MEMORY AND TO SET THEIR NEW VALUES. LOCATION UPTPCW (USER PROCESS TABLE - PROCESS CONTEXT WORD) IN THE UPT WILL BE ASSIGNED TO RECEIVE PCS IN THE SAME FORMAT AS USED BY THE DATAO PAG. THE ONLY REASON FOR SAVING THE OLD CONTENTS OF PCS, IS IN CASE THE CALLER IS THE MONITOR. THEN PCS IS SET TO THE SECTION FROM WHICH THE CALLER CALLED (BEFORE THE PC IS INCREMENTED).

FLOW FOR MUUO (MUUO AC, E): (NOTE: THE LOCATIONS IN THE UPT ARE GIVEN SYMBOLICALLY. SEE CHAP 2.9 - EXEC AND USER PROCESS TABLES - EPT, UPT FOR VALUE OF SYMBOLS.

```
BEGIN MUUO
COMPUTE EFFECTIVE ADDRESS, VMA<6-35>;
STORE PC FLAGS IN UPTUUO<0-12> ALONG WITH
   OPCODE IR<0-8> AND AC<9-12> IN UPTUUO<18-30>;
STORE PC<13-35> GLOBAL ADDRESS IN UPTUUO+1;
STORE EFFECTIVE ADDRESS VMA<13-35> IN UPTUUO+2;
STORE CURRENT AC BLOCK REGISTER (CAC) [3],
   PREVIOUS AC BLOCK (PAC) [3],
   CALLER'S PREVIOUS CONTEXT SECTION (PCS) [5 IN KL],
   IN UPTPCW IN DATAO PAG, FORMAT;
COPY CALLER'S CURRENT SECTION, PC<13-17> TO PCS;
CLEAR ALL FLAGS AND SET PCU AND PCP APPROPRIATELY
   IN PROCESSOR STATUS DEPENDING WHERE CALLER CAME
   FROM;
FETCH NEW GLOBAL PC FROM C(UPT+D)<13-35>
   WHERE D = UPTNMP + 4*USER + 2*(PUBLIC) + TRAP
   CYCLE(SAME AS KI);
END (OF FLOW FOR MUUO)
```

IT IS THE RESPONSIBILITY OF THE MONITOR TO RESTORE PCS BEFORE RETURNING TO A CALLER. FOR EFFICIENCY, THIS IS DONE IN A SINGLE INSTRUCTION. THE TYPICAL SEQUENCE WOULD BE:

```
DATAO   PAG,UPTPCW      ;RESTORE PREV CONTEXT,
                        ; INCLUDING PCS IN CASE MONITOR
                        ; IS CALLING
XJRSTF  UPTUUO          ;LOAD FLAGS AND 30-BIT PC
```

5. APPLICABLE AND NON APPLICABLE INSTRUCTIONS UNDER PXCT

THE FOLLOWING TABLE SHOWS WHICH INSTRUCTIONS WILL BE USED BY THE MONITOR UNDER PXCT AND WHICH WILL NOT. NO ATTEMPT WILL BE MADE TO MAKE THE "NOT APPLICABLE" CATEGORY WORK COMPATIBLY ON FUTURE MACHINES. FURTHERMORE SINCE VERY FEW CUSTOMERS WRITE THEIR OWN MONITORS, NO ATTEMPT WILL BE MADE TO TRAP THE NOT APPLICABLE CASES.

APPLICABLE	NOT APPLICABLE
ADJSP	LUUOS, MUUOS, JSYS
FLOATING POINT	
DMOVE, DMOVN, DMOVEM, DMOVNM	
IBP, ADJBP, ILDB, LDB, IDPB DPB	AOBJN, AOBJP, JRSTXX, XCT, PXCT
EXTEND	PUSHJ, POPJ, JSR, JSP
XBLT	JSA, JRA
MOVXX CLASS	JUMPXX, AOJXX, SOJXX
XMOVEI	I/O
INTEGER ARITHMETIC	
EXCH	
BLT	
PUSH, POP	
CAIXX, CAMXX	
SKIPXX	
AOSXX, SOSXX	
BOOLEAN	
HALF WORD	
TEST	
SHIFT, ROTATE	
MAP	

6. SPECIAL-CASE INSTRUCTIONS

THIS SECTION DESCRIBES THE OPERATION OF THE APPLICABLE INSTRUCTIONS. IT PARALLELS THE ORDER OF THE DESCRIPTIONS OF THE INSTRUCTIONS DESCRIBED IN CHAPTER 2,2, EXTENDED ADDRESSING.

6.1 PC-STORING INSTRUCTIONS: PUSHJ, JSP, JSR, POPJ

NOT APPLICABLE.

6.2 BYTE INSTRUCTIONS LDB, ILDB, DPB, IDPB, IBP, ADJSP

THE BYTE INSTRUCTIONS USE ALL 4 PXCT AC BITS AS FOLLOWS:

- 9(E1) EFFECTIVE ADDRESS COMPUTATION TO FETCH BYTE POINTER.
- 10(D1) FETCH OF BYTE POINTER.
- 11(E2) EFFECTIVE ADDRESS COMPUTATION OF BYTE DATA.
- 12(D2) FETCH OR STORE OF BYTE DATA.

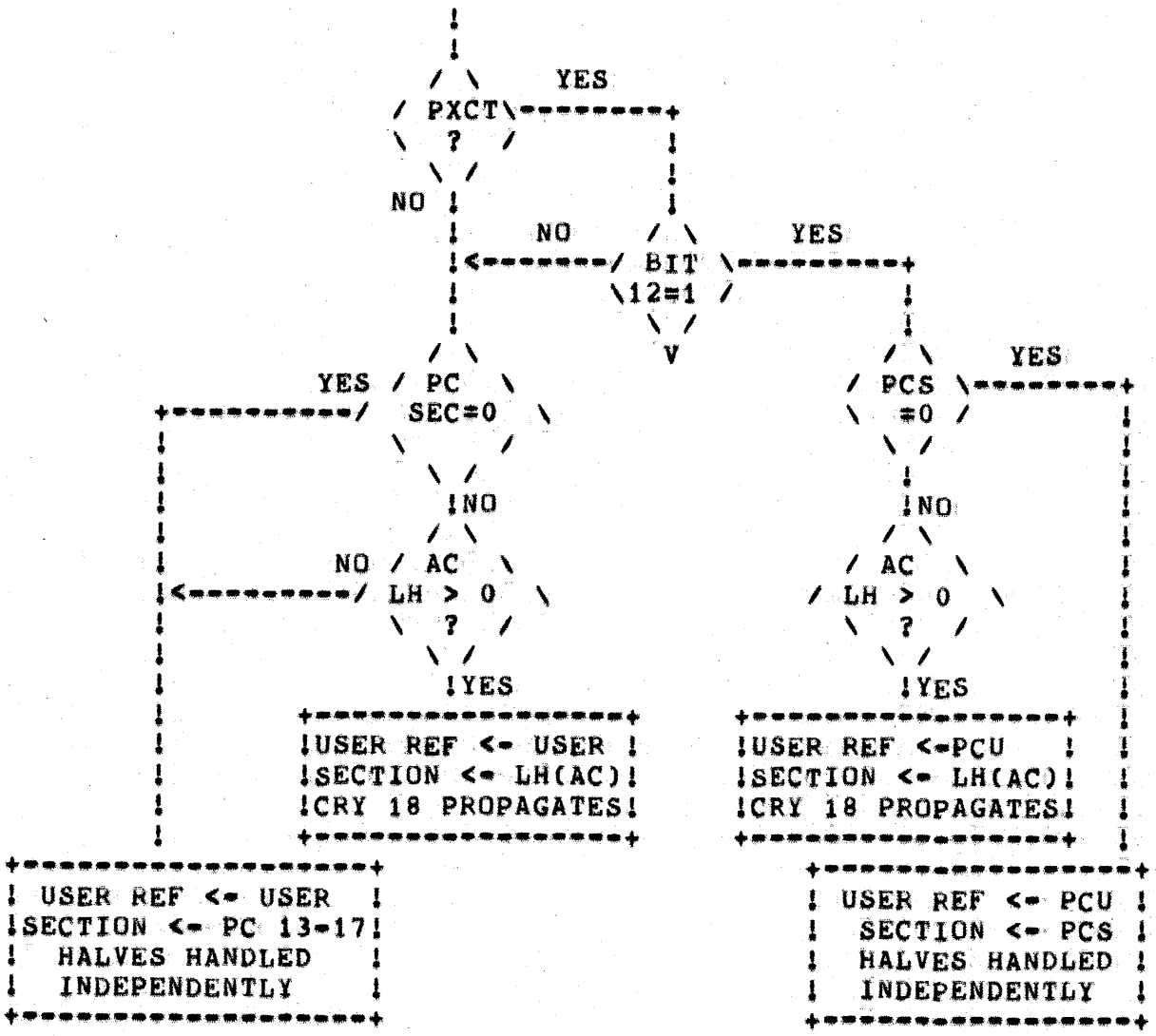
6.3 STACK INSTRUCTIONS: PUSH, POP, ADJSP

THE STACK INSTRUCTIONS USE THE FOLLOWING PCXT AC BITS:

- 9(E1) EFFECTIVE ADDRESS COMPUTATION.
- 10(D1) DATA OPERAND FETCH OR STORE (PUSH, POP).
- 12(D2) STACK DATA FETCH OR STORE

THE STACK POINTER IS ALWAYS IN THE CURRENT AC, SINCE THE AC OF ALL OBJECT INSTRUCTIONS ARE TAKEN FROM THE CURRENT ACS. HOWEVER, THE DATA FETCH OR STORE IN THE STACK IS CONTROLLED BY BIT 12. IN DETERMINING WHICH SECTION IS MEANT BY THE LH OF THE STACK POINTER, PCS TAKES THE ROLE OF PC [6-17] WHEN BIT 12 = 1.

FLOW CHART FOR STACK POINTER UPDATE



6,4 BLT

BLT IS PROVIDED UNDER PXCT SO THAT THE CODING IS THE SAME FOR MONITORS WHICH RUN ON KIS, 1080S AND/OR KL WITHOUT SUPPORTING EXTENDED ADDRESSING,

BLT USES PXCT AC BITS:

9(E1) DESTINATION AND TERMINATION EFFECTIVE ADDRESS,

10(D1) DESTINATION DATA STORE,

12(D2) SOURCE DATA FETCH,

6,5 EXTEND

EXTEND USES THE 4 PXCT AC BITS MAINLY FOR THE SOURCE AND DESTINATION
EFFECTIVE ADDRESS AND OPERAND ACCESS, THE DATA FETCH FOR THE EXTENDED
OPCODE WORD C(E) IS CONTROLLED BY B10, HOWEVER BIT 9(E1) DOES AFFECT
THE EFFECTIVE ADDRESS COMPUTATION TO OBTAIN THE EXTENDED OPCODE WORD,
AS WITH E OF ALL INSTRUCTIONS, THEREFORE THE MONITOR PROGRAMMER MUST
MAKE SURE THAT E1=0, IN THE UNLIKELY CASE THAT E HAS INDIRECTION OR
INDEXING,

EXTEND USES PXCT AC BITS:

9(E1) EFFECTIVE ADDRESS COMPUTATIONS OF E (EXTEND OP);
ALSO OF SOURCE POINTER IF BIT 11 IS SET; AND OF DESTINATION POINTER IF BIT 12 IS SET,

10(D1) EXTEND OP

11(E2) SOURCE BYTE DATA FETCH (AND SOURCE POINTER CALCULATION IF B9 IS SET),

12(D2) DESTINATION BYTE DATA STORE (AND DESTINATION POINTER
CALCULATION IF B9 IS SET),

THE AC'S USED BY EXTEND OPERATIONS ARE ALWAYS CURRENT CONTEXT, AND
BIT 12 OF BYTE POINTERS USED BY EXTEND IS INTERPRETED OR IGNORED
ACCORDING TO THE CURRENT SECTION,

6,6 XCT

PXCT [XCT] IS NOT APPLICABLE, HOWEVER XCT [PXCT E] WILL WORK THE
SAME AS PXCT E,

6,7 XMOVEI

NO SPECIAL ACTION IS TAKEN, ONLY BIT 9 (E1) AFFECTS THE INSTRUCTION,
SEE CHAPTER 2,2, EXTENDED ADDRESSING,

6,8 XBLT

XBLT USES THE SAME BITS AS EXTEND (RATHER THAN BLT) AND HAS THE SAME SOURCE, DESTINATION RELATIONSHIP, AS WITH ALL EXTEND OPCODES, THE DATA FETCH OF THE EXTENDED OPCODE IS CONTROLLED BY BIT 10(D1), BECAUSE BIT 9 (E1) AFFECTS THE EFFECTIVE ADDRESS COMPUTATION IN THE SAME WAY FOR ALL INSTRUCTIONS, IT SHOULD BE 0,

10(D1) DESTINATION WORD STORES

12(D2) SOURCE WORD FETCHES

7. PROGRAMMING EXAMPLES

THE FOLLOWING EXAMPLES SHOW AN MUUO CALLING SEQUENCE USING ARGUMENT
LISTS. THIS IS NOT NECESSARILY THE WAY THE MONITOR WILL BE CALLED.
HOWEVER, SUCH A CALLING SEQUENCE IS SUFFICIENTLY GENERAL TO ILLUSTRATE ALL THE USES OF PXCT BY THE MONITOR.

ASSUME AN ARGUMENT LIST CONSISTING OF POINTERS TO ARGUMENTS. IF
THE USER PROGRAM DOES AN MUUO FROM SECTION 0, THESE POINTERS CAN BE:

1. A NUMBER LESS THAN 20 SPECIFYING AN ARGUMENT IN AN AC (PREVIOUS CONTEXT AC).

2. AN 18 BIT ADDRESS (IN SECTION 0).

3. 18 BIT ADDRESS + INDEX FIELD AND/OR INDIRECTION (INDEXING
IGNORES LEFT HALF OF INDEX REGISTER).

IF THE USER PROGRM DOES AN MUUO FROM A NON-ZERO SECTION, THESE POINTERS
CAN BE:

1. A NUMBER LESS THAN 20 SPECIFYING AN ARGUMENT IN AN AC (PREVIOUS CONTEXT AC).

2. AN 18 BIT ADDRESS (IN CALLING SECTION), IE., AN INSTRUCTION
FORMAT INDIRECT WORD (IFIW).

3. AN 18 BIT ADDRESS (IN CALLING SECTION) + INDEX AND/OR INDIRECTION, IE., AN INSTRUCTION FORMAT INDIRECT WORD (IFIW)
(INDEXING MAY USE LH OF INDEX REGISTER AS A GLOBAL SECTION NUMBER IN CALLING ADDRESS SPACE).

4. A 30-BIT ADDRESS ANYWHERE IN THE ADDRESS SPACE OF THE CALLER, IE., AN EXTENDED FORMAT INDIRECT WORD (EFIW). AN EFIW MAY HAVE INDIRECTION AND/OR INDEXING.

EXAMPLE - IN SECTION 0:

```
        MOVEI   16,[EXP 3        ;ARG IN AN AC
                    EXP 100      ;ARG IN LOC 100
                    EXP 1000(5) ;ARG IN LOC 1000 + C(5)
                    EXP @2000] ;ARG IN LOC C(2000)
        MUUO
```

EXAMPLE - IN NON-ZERO SECTION:

```
        XMOVEI  16,[EXP 3        ;ARG IN AC
                   EXP 1B0+100  ;ARG IN LOC 100
                   EXP 1B0+1000(5) ;ARG IN LOC 1000 + C(5)
                   EXP 1B0+@2000   ;ARG IN LOC C(2000)
                                   ;2000 MAY CONTAIN EFIW OR
                                   ;IFIW
                   XWD 30,3000] ;ARG IN LOC 3000 OF
                                ; SECTION 30
        MUUO
```

THE FOLLOWING MONITOR CODE WILL WORK IN A NON-ZERO SECTION, NO MATTER WHETHER THE CALLER IS IN SECTION 0 OR NOT: (AFTER THE PREVIOUS AC BLOCK REGISTER (PAC) CONTAINS THE AC BLOCK NUMBER OF THE CALLER'S ACS).

7.1 TO FETCH THE ARGUMENT POINTER (CALLER'S AC 16):

```
        PXCT    14,[MOVE T1,16] ;PXCT 4, IS EQUIVALENT
```

7.2 TO SETUP THE ADDRESS OF THE FIRST ARGUMENT POINTER (SO TO STEP THROUGH ARGUMENTS IN A LOOP, SAY):

```
        PXCT    14,[XMOVEI T1,@16]
```

NOTE THAT THIS IS ALWAYS A 30-BIT ADDRESS IN THE PREVIOUS CONTEXT WHETHER THE CALLER CALLED FROM SECTION 0 OR NOT. NOTE: MOVEI COULD NOT BE USED TO GET THE SECTION NUMBER, SINCE ALL IMMEDIATE MODE INSTRUCTIONS (EXCEPT XMOVEI) USE ONLY 18 BITS, EVEN UNDER EXTENDED ADDRESSING.

7,3 TO FETCH THE NTH ARGUMENT:

```
        PXCT    14,[MOVE T1,@N(16)]
```

7,4 TO FETCH THE ADDRESS OF THE NTH ARGUMENT:

```
        PXCT    14,[XMOVEI T1,@N(16)]
```

NOTE THAT THIS IS ALWAYS A 30-BIT ADDRESS IN THE PREVIOUS CONTEXT WHETHER THE CALLER CALLED FROM SECTION 0 OR NOT.

7,5 TO ITERATE THROUGH AN ARRAY:

NOTE THE USE OF 04, INSTEAD OF 14 IN PXCT TO COMPUTE EFFECTIVE ADDRESS IN EXEC MODE AND ONLY GET DATA FROM PREVIOUS CONTEXT.

```
        PXCT    14,[XMOVEI T1,@1(16)]   ;FETCH ARRAY ADDRESS
        MOVEI   T2,N                    ;SET INTERATION COUNT
                                        ; TO N
        MOVEI   T3,0                    ;SET SUM TO 0
LOOP:   PXCT    04,[ADD T3,(T1)]        ;ADD NEXT ENTRY
        ADDI    T1,1                    ;INCREMENT 30-BIT
                                        ; ADDRESS, PXCT CROSSES
                                        ; SECT BOUNDARIES ONLY
                                        ; CALLED FROM SECT > 0.
        SOJG    T2,LOOP                 ;LOOP UNTIL DONE
```

APPENDIX - EXPLANATION OF DESIGN DECISIONS

THIS APPENDIX CONTAINS EXPLANATIONS OF DESIGN DECISIONS WHICH ARE FLAGGED IN THE TEXT WITH FOOTNOTE NUMBERS IN BRACKETS. THUS NEW READER'S OF THE SPEC WILL NOT HAVE TO ASK, "WHY DIDN'T YOU DO...?" ALSO THEY CAN SEE IF THEY AGREE WITH OUR REASONING. FINALLY IT FORCES US TO JUSTIFY OUR DECISIONS WITH GOOD REASONS.

1. WHY NOT HAVE PXCT WORK IN USER MODE?

PRO: IT COULD BE USED TO MAKE A FAST CALLING STANDARD USING 2 AC BLOCKS. SEE CAL PROPOSAL BY T. HASTINGS.

CON: CALLING STANDARD SHOULD BE SIMILAR ON KA, KI, AND KL. WE ARE STILL TRYING TO DEFINE THE STANDARD. PXCT IS NOT UNDERSTOOD WELL ENOUGH TO PUT IT IN USER-MODE YET.

SPEC: PXCT WORKS ONLY IN EXEC MODE.

2. WHY NOT KEEP PCS IN NEW FLAGS AND PC DOUBLE WORD SINCE IT IS A PART OF THE PROCESS CONTEXT?

PRO: LOGICALLY IT SHOULD GO THERE, SINCE THE PREVIOUS CONTEXT SECTION NUMBER IS A PART OF THE PROCESS CONTEXT. IT WOULD SAVE THE MONITOR HAVING TO DO A DATAO ON EACH CALL.

CON: THE OPCODE AND AC FIELD STORED ON AN MUUO WOULD HAVE TO BE MOVED INTO ANOTHER WORD. ALSO PCS IS ONLY USED IN EXEC MODE, THEREFORE IT IS NOT REALLY A FULL PROCESS STATE VARIABLE.

SPEC: MONITOR MUST DO A DATAO TO SET PCS ON EACH MONITOR CALL AND CONTEXT SWITCH.

LOOSE ENDS

1. IS IT PROPER THAT PXCT NOT WORK IN USER-IOT MODE?

2. HOW DOES NEW CODE CALL MONITOR FROM NON-ZERO SECTION WITH OLD (18-BIT) JSYSS?

3. HOW CAN THE MONITOR (RUNNING IN A NON-ZERO SECTION) SETUP A POINTER TO USER SECTION 0? 0 IN THE LH OF A MONITOR INDEX REGISTER MEANS LOCAL, NOT SECTION 0. EXAMPLE 7.5 FAILS.

4. WHERE SHOULD PCS GET INVOLVED IN BYTE OPERATIONS IF E1=0?

[END OF PXCT.SPC]

1080,2040,2060 ENGINEERING FUNCTIONAL SPEC - CHAPTER 2.6

TO: KL10 LIST

TITLE: KL10 INTERNAL DEVICES (APR, PI, PAG)

STATUS: THIS CHAPTER REFLECTS THE DESIGN INTENTION FOR PROTOTYPE-LEVEL MACHINES. PARAGRAPH 7.4 (CACHE SWEEP) HAS BEEN PROPOSED BUT NOT ACCEPTED.

FILE: [EFS]CH2S06.SPC

PDM #: 200-200-019-00

DATE: 7 AUGUST 74

SUPERSEDED MEMOS:

ENGINEER: J. LEONARD

APPROVED:

EDITOR: M. MILLER

TYPIST: K. PAPPAS

REVIEWED:

ABSTRACT

BIT ASSIGNMENTS AND FUNCTIONAL DEFINITIONS ARE GIVEN FOR INTERNAL IO DEVICES APR, PI, AND PAG. THE INTERNAL DEVICE TIM/MTR AND THE SBUS DIAGNOSTIC INSTRUCTION ARE COVERED IN OTHER CHAPTERS.

REVISION HISTORY

REV	DESCRIPTION	CHG NO	ORIG	DATE	APPD BY	DATE

0. CONTENTS

1. SUMMARY

THE INTERNAL "DEVICES" APR, PI, AND PAG PROVIDE MEANS FOR THE MONITOR TO TEST AND CONTROL VARIOUS ASPECTS OF THE MACHINE'S BEHAVIOR, AS NEEDED TO PERFORM TIMESHARING. IN ADDITION, THESE DEVICES PROVIDE INFORMATION ABOUT HARDWARE FAILURES, SO THE MONITOR CAN ATTEMPT RECOVERY, AND SO DIAGNOSTICS CAN NARROW DOWN FAILURES AS AN AID TO FIELD SERVICE.

2. TERMINOLOGY

2.1 SBUS

STORAGE BUS, THE INTERNAL COMMUNICATION PATH BY WHICH THE MEMORY CONTROL PORTION OF THE CPU CONTROLS AND PASSES DATA TO AND FROM THE INTERNAL MEMORIES AND THE DMA20 EXTERNAL MEMORY ADAPTER.

2.2 IO BUS RESET

A SIGNAL WHICH RESETS ALL IO DEVICES AND CONTROLLERS.

2.3 POWER FAIL FLAG

A SIGNAL ASSERTED BY THE PROCESSOR POWER SUPPLY IF ANY PHASE OF THE POWER LINE DROPS BELOW TOLERANCE. LOGIC POWER WILL REMAIN IN TOLERANCE FOR AT LEAST 15 MILLISECONDS.

2.4 CACHE REFILL ALGORITHM

A TABLE LOOKUP MECHANISM BY WHICH THE CACHE CONTROL KEEPS TRACK OF THE USE HISTORY OF EACH WORD IN THE CACHE, IN ORDER TO DISCARD THE LEAST RECENTLY USED WORDS AS NEW DATA IS LOADED INTO THE CACHE.

2.5 MBZ

MBZ IS AN ABBRIEVATION FOR MUST BE ZERO. IT IS USED IN INSTRUCTION FORMATS FOR UNUSED BITS AND FIELDS. SOFTWARE MUST ENSURE THAT THESE BITS AND FIELDS ARE ZERO BECAUSE THEY ARE RESERVED TO DIGITAL FOR FUTURE USE BY MICRO-CODE OR HARDWARE ON THE KL10 OR FUTURE MACHINES.

3. GOALS

1. KI10 COMPATIBILITY WHERE IT IS RELEVANT.

2. MINIMUM NUMBER OF INSTRUCTIONS FOR THE FREQUENTLY OCCURRING FUNCTIONS.

3. MINIMUM NUMBER OF DEVICE CODES USED.

4. ALL INTERNAL DEVICE CODES USED SHOULD BE LESS THAN 34 OCTAL (I.E., 034) FOR EASE OF BUILDING THE HARDWARE.

4. NON-GOALS

1. NO MAINTENANCE FEATURES INCLUDED IN THE INTERNAL DEVICES: THE PDP-11 WILL PERFORM THAT PORTION OF MAINTENANCE.

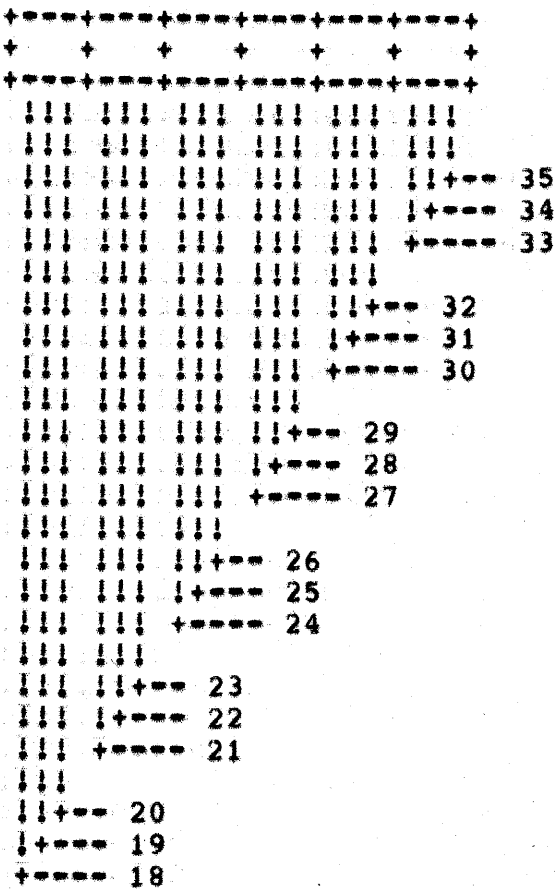
2. WE DO NOT NEED THE EQUIVALENCE OF DATAO AND BLKO, OR OF DATAI AND BLKI, AS IN EARLIER MACHINES BECAUSE INTERNAL DEVICES DO NOT TRANSFER BLOCKS OF DATA.

5. DEVICE APR (ADDRESS 000 OCTAL)

5.1 PROGRAMMABLE PROCESSOR INTERRUPT REGISTER (CONO, CONI-APR)

CONDITIONS OUT (CONO) AND CONDITIONS IN (CONI) APR HAVE BEEN REARRANGED FROM WHAT THEY WERE IN THE KI10. THE FOLLOWING FUNCTIONS HAVE BEEN DELETED. THEY ARE NOT NEEDED IN THE KL10 BECAUSE THE PDP-11 PERFORMS THEM: THE TIMER, AUTO RESTART, THE LINE FREQUENCY CLOCK, AND THE SENSE SWITCHES. THE FOLLOWING FUNCTIONS HAVE BEEN ADDED: FLAGS FOR SBUS ERROR, CACHE SWEEP BUSY, AND CACHE SWEEP DONE.

CONO APR IS AN IMMEDIATE INSTRUCTION. CONI APR RETURNS FLAGS TO THE WORD AT E. BITS OF E ARE INTERPRETED AS FOLLOWS:



0-5		CONI:	0
6	SEE	CONI:	SBUS ERROR ENABLED
7	NXME	CONI:	NON-EXISTENT MEMORY ENABLED
8	MPEE	CONI:	MEMORY PARITY ERROR ENABLED
9	IPFE	CONI:	IO PAGE FAIL ENABLED

10		CONI: MBZ
11	PFE	CONI: POWER FAIL ENABLED
12		CONI: MBZ
13	CSDE	CONI: CACHE SWEEP DONE ENABLED
14-18		CONI: 0
19	IBR	CONO: IO BUS RESET - A 1 CAUSES IO BUS RESET (EQUIVALENT TO FUNCTION IT PERFORMED ON KI10)
	CSIP	CONI: CACHE SWEEP IN PROGRESS - A 1 IF A CACHE SWEEP IS IN PROGRESS AS INITIATED BY

BITS 20 - 23 OF CONO DETERMINE A FUNCTION TO BE PERFORMED ON SELECTED FLAGS, AS SHOWN BELOW. BITS 20 - 23 OF CONI ARE 0.

BIT	SYMBOL	DESCRIPTION
20	ESI	CONO: ENABLE SELECTED INTERRUPTS - ONES SPECIFY THAT THE SELECTED FLAG(S) SHALL BE ENABLED FOR INTERRUPT.
21	DSI	CONO: DISABLE SELECTED INTERRUPTS - ONES SPECIFY THAT THE SELECTED FLAG(S) SHALL BE DISABLED FOR INTERRUPT.
22	CSDF	CONO: CLEAR SELECTED DONE FLAG(S) - ONES SPECIFY THAT THE SELECTED FLAG(S) SHALL BE SET.

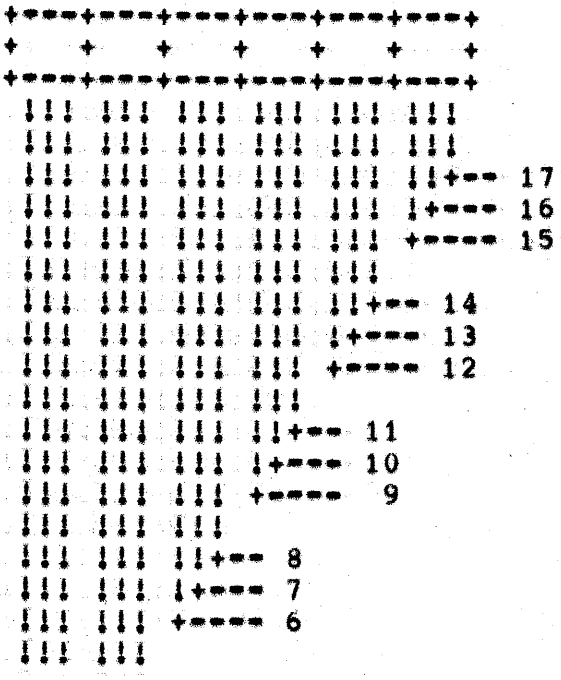
BITS 24 - 31 OF CONO ARE THE FLAGS SELECTED. BITS 24 - 31 OF CONI INDICATE WHICH FLAGS ARE SET.

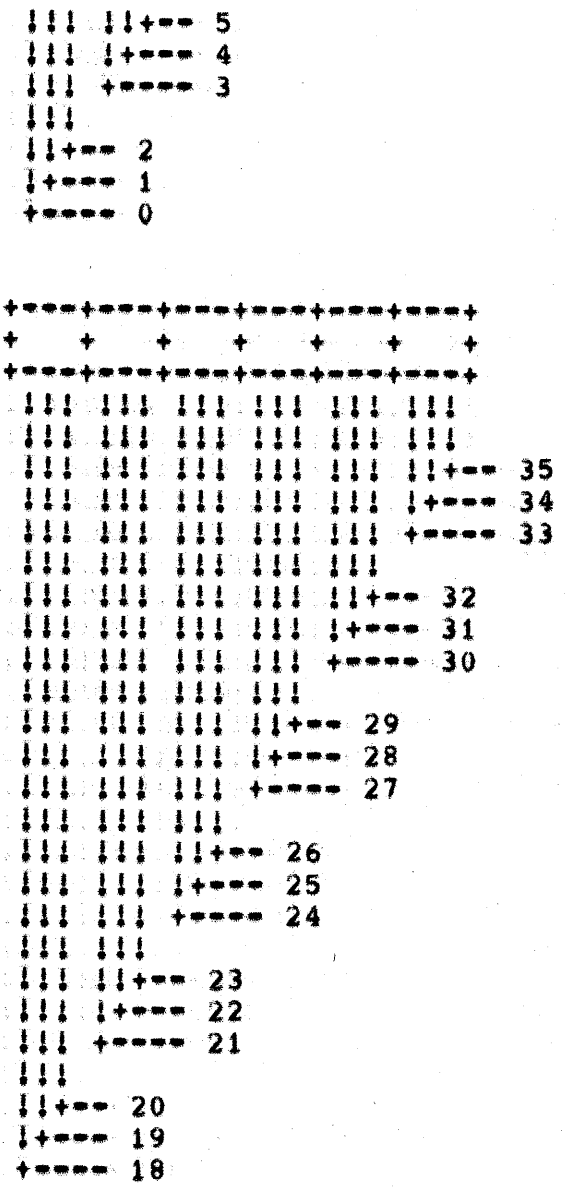
BIT	SYMBOL	DESCRIPTION
24	SE	CONO OR CONI: SBUS ERROR FLAG.
25	NXM	CONO OR CONI: NON-EXISTENT MEMORY ERROR FLAG.
26	MPE	CONO OR CONI: MEMORY PARITY ERROR FLAG.
27	IPF	CONO OR CONI: IO PAGE FAIL FLAG.
28		CONO: UNDEFINED (STET 0).
		CONI: 0

29	PF	CONO OR CONI: POWER FAIL FLAG.
30		CONO: UNDEFINED (MUST BE 0). CONI: 0
31	CSD	CONO OR CONI: CACHE SWEEP DONE FLAG.
32		CONO: MUST BE 0.
	AI	CONI: APR INTERRUPT A 1 IF ANY FLAG IS ENABLED AND SET, AND IS THEREFORE REQUESTING AN INTERRUPT. OTHERWISE, CONI IS 0.
33-35	PIA	CONO: PRIORITY INTERRUPT (PI) ASSIGNMENT FOR THE DEVICE APR CONI: RETURN THE PI ASSIGNMENT OF THE DEVICE APR.

5.2 ADDRESS BREAK (DATAO APR)

IN THE KI10, DATAO APR SETS THE MARGINS. SINCE KL10 DOES NOT HAVE MARGINS, THE KL DATAO APR, LOADS THE ADDRESS BREAK REGISTER. ADDRESS BREAK CAN BE TURNED OFF BY SETTING BREAK ENABLE BITS 9 - 11 TO ALL ZEROES OR BY USING AN ADDRESS LESS THAN 20 OCTAL. THE WORD AT E IS INTERPRETED AS FOLLOWS:





BIT	SYMBOL	DESCRIPTION
0 - 8		MUST BE 0.
9	ABIF	DATAO: ADDRESS BREAK ON INSTRUCTION FETCH - A 1 ENABLES ADDRESS BREAK ON AN INSTRUCTION FETCH.
10	ABDR	DATAO: ADDRESS BREAK ON DATA READ- A 1 ENABLES ADDRESS BREAK ON A DATA READ REFERENCE.
11	ABDW	DATAO: ADDRESS BREAK IN USER MODE- A 1 ENABLES ADDRESS BREAK ON A DATA WRITE REFERENCE.

12	ABUM	DATAO: ADDRESS BREAK ON DATA WRITE - A 1 ENABLES ADDRESS BREAK IN A USER PROGRAM. A 0 ENABLES ADDRESS BREAK IN AN EXECUTIVE PROGRAM.
13-35	ABA	DATAO: ADDRESS BREAK ADDRESS - THE 23-BIT ADDRESS WHICH MUST MATCH THAT GENERATED BY THE PROGRAM IN ORDER FOR A BREAK TO OCCUR. (A MATCH AND THEREFORE AN ADDRESS BREAK CAN OCCUR WHEN A PROGRAM MEMORY REFERENCE HAS THE SAME VIRTUAL ADDRESS AS THAT GIVEN BY BITS 13 - 35.)

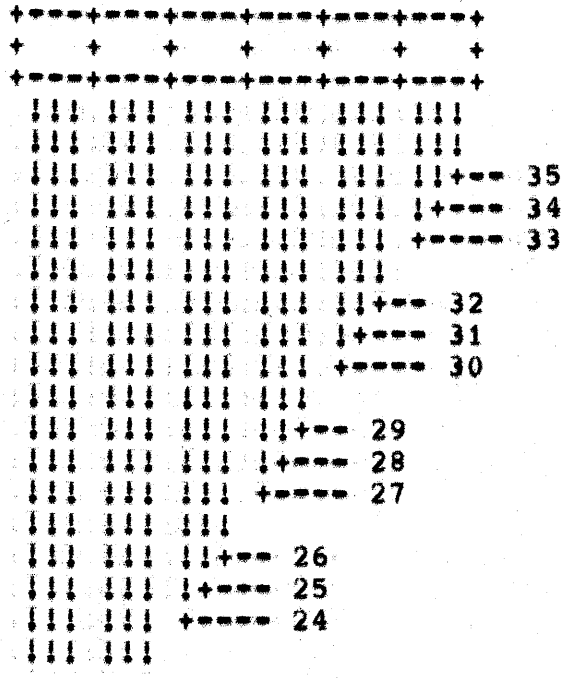
ADDRESS BREAK APPEARS TO THE PROGRAM AS A PAGE FAULT (CODE 023). THE ADDRESS BREAK HANDLER CAN RESTART THE PAGE FAULTED PROGRAM BY SETTING ADDRESS BREAK INHIBIT (BIT 8) IN THE PC WORD. THIS PERMITS THE FAULTING PROGRAM TO CONTINUE PAST THE INSTRUCTION WHICH CAUSE THE ADDRESS BREAK, BUT SUBSEQUENT REFERENCES WHICH MATCH THE ADDRESS BREAK REGISTER WILL CAUSED LATER BREAKS.

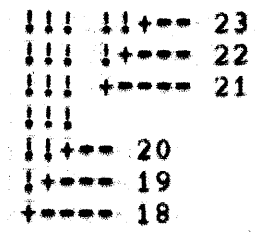
5.3 DATA IN (DATAI APR)

THIS FUNCTION IS UNUSED IN THE KL10.

5.4 CACHE REFILL ALGORITHM (BLKO APR)

BLKO APR, IS AN IMMEDIATE MODE INSTRUCTION. THE HALF-WORD E IS INTERPRETED AS FOLLOWS:



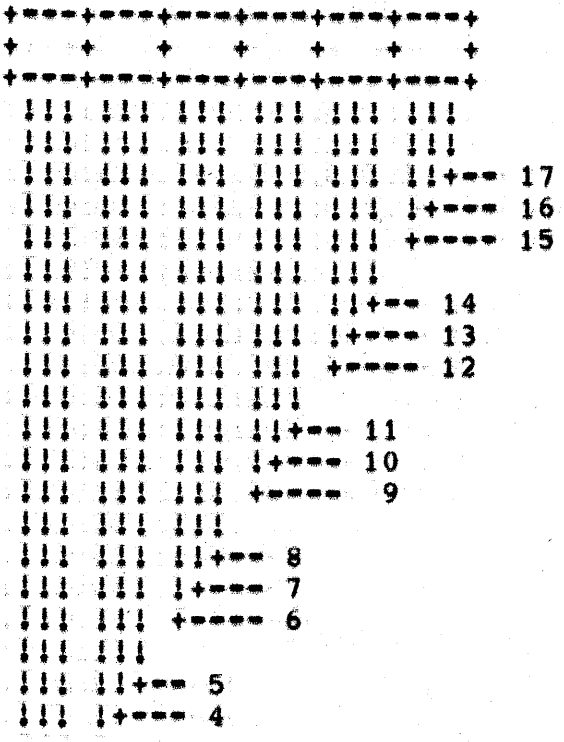


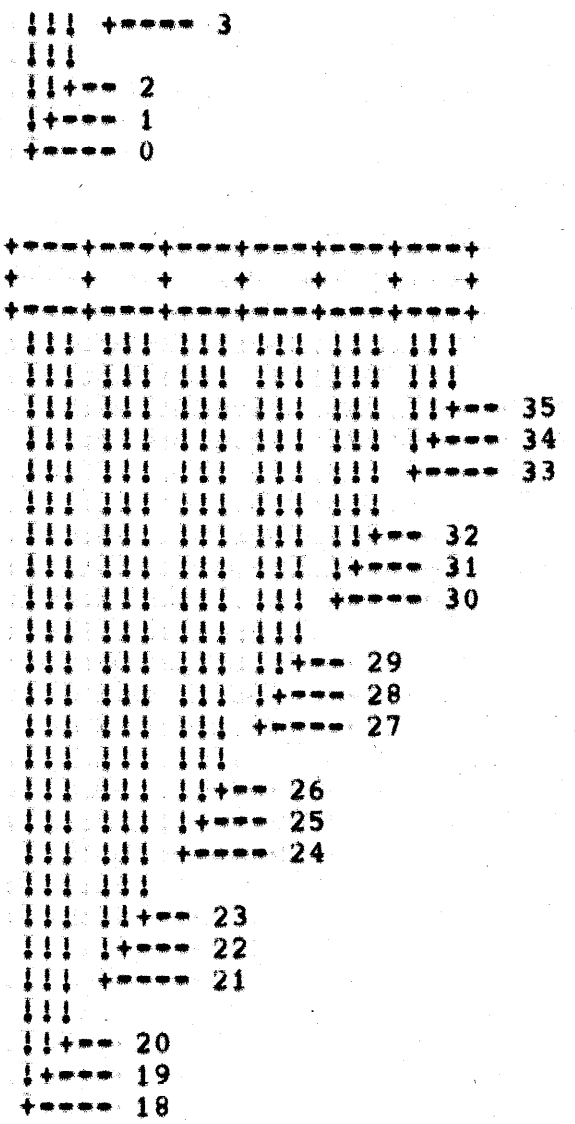
BIT	SYMBOL	DESCRIPTION
18-20		DATA TO BE STORED IN THE REFILL ALGORITHM RAM IN THE MBOX.
21-26		MUST BE 0.
27-33		THE ADDRESS IN THE REFILL ALGORITHM WHERE BITS 18 - 20 ARE TO BE STORED.
34-35		MUST BE 0.

ONE EXECUTION OF THIS INSTRUCTION CAUSES ONE WORD (3 BITS) OF THE REFILL ALGORITHM RAM TO BE WRITTEN. IN ORDER TO LOAD THE ENTIRE RAM, 128 BLKO APR'S MUST BE EXECUTED.

5.5 PROCESSOR OPTIONS REGISTER (BLKI APR,)

BLKI APR RETURNS A WORD AT E WITH THE FOLLOWING BIT IDENTIFICATION:



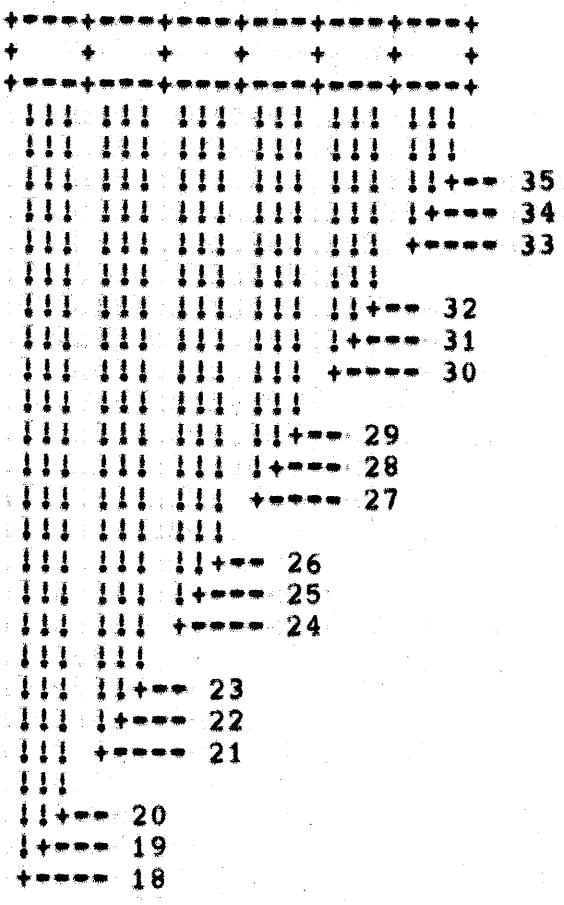


BIT	SYMBOL	DESCRIPTION
0-8		FLAG BITS INDICATING EXISTENCE OF CERTAIN MICRO-CODE OPTIONS.
9-17		VERSION NUMBER OF MICRO-CODE RUNNING.
18-23		INDICATES WHAT HARDWARE OPTIONS ARE PRESENT ON THE MACHINE.
24-35		PROCESSOR SERIAL NUMBER IN BINARY. SERIAL NUMBERS START AT 2000 OCTAL, CORRESPONDING TO SERIAL NUMBER 1024.

6. DEVICE PI (ADDRESS 004 OCTAL)

6.1 PI SYSTEM (CONO PI)

CONO PI, IS AN IMMEDIATE MODE INSTRUCTION. THE HALF WORD E IS INTERPRETED AS FOLLOWS:



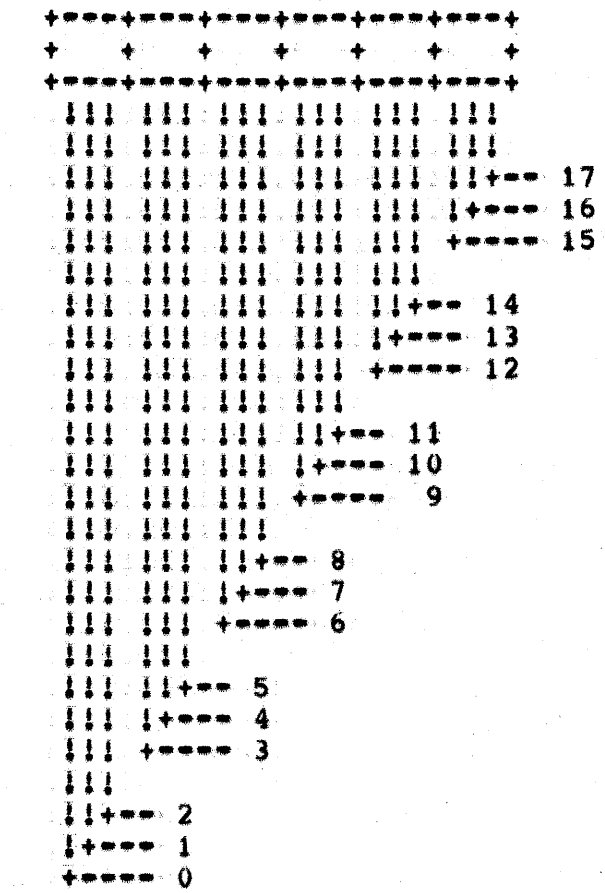
BITS	SYMBOL	DESCRIPTION
18	WEAP	CONO: WRITE EVEN ADDRESS PARITY - A 1 CAUSES ADDRESS PARITY ON WRITE TO MEMORY TO BE EVEN. THE MEMORY CONSIDERS EVEN ADDRESS PARITY TO BE AN ERROR, SO THIS FUNCTION IS PROVIDED TO TEST THE ERROR HANDLING OF THE SYSTEM FOR DIAGNOSTICS AND THE TIME SHARING MONITOR. A 0 CAUSES ADDRESS PARITY TO BE ODD.
19	WEDP	CONO: WRITE EVEN DATA PARITY - CAUSES DATA PARITY ON WRITE TO MEMORY TO BE EVEN. THE MEMORY CONSIDERS EVEN PARITY

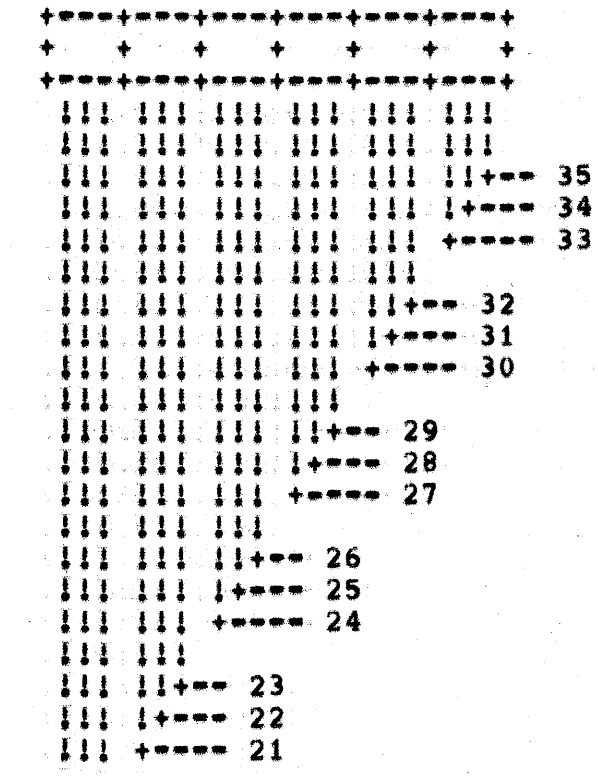
		TO BE AN ERROR. SO THIS FUNCTION IS PROVIDED TO TEST THE ERROR HANDLING OF THE SYSTEM FOR DIAGNOSTICS AND THE TIME SHARING MONITOR. A 0 CAUSES DATA PARITY TO BE EVEN.
20-21		MUST BE 0.
22	DISC	CONO: DROP INTERRUPTS ON SELECTED CHANNELS - A 1 DROPS INTERRUPTS REQUESTED BY THE PROGRAM ON CHANNELS SELECTED BY BITS 29 - 35.
23	CPS	CONO: CLEAR PI SYSTEM - A 1 CLEARS THE PI SYSTEM (I.E., TURNS OFF ALL CHANNELS, TURNS OFF THE PI SYSTEM, EXCEPT LEVEL 0 AND DROPS ALL PROGRAM REQUESTS). SEE CHAPTER 4.3, PDP-10/11 INTERFACE (DTE20) FOR A DESCRIPTION OF PI 0.
24	RISC	CONO: REQUEST INTERRUPTS ON SELECTED CHANNELS - A 1 REQUEST AN INTERRUPT ON CHANNELS SELECTED BY BITS 29 - 35.
25	TNSC	CONO: TURN ON SELECTED CHANNELS - A 1 TURNS ON CHANNELS SELECTED BY BITS 29 - 35.
26	TFSC	CONO: TURN OFF SELECTED CHANNELS - A 1 TURNS OFF CHANNELS SELECTED BY BITS 29 - 35.
27	TFPS	CONO: TURN OFF PI SYSTEM - A 1 TURNS OFF THE ENTIRE PI SYSTEM (EXCEPT LEVEL 0).
28	TNPS	CONO: TURN ON PI SYSTEM - A 1 TURNS ON THE ENTIRE PI SYSTEM.
29-35		CONO: ONES SPECIFY THAT CHANNELS 1 - 7, RESPECTIVELY, ARE SELECTED.

THE EFFECT OF SPECIFYING CONFLICTING CONDITIONS IS INDETERMINATE. DEC RESERVES THE RIGHT TO CHANGE THE EFFECT OF CONFLICTING SPECIFICATIONS IN THIS AND FUTURE MACHINES. PROGRAMS MUST AVOID USING THEM. CONFLICTING REQUESTS ARE: DISC=RISC=1, TNSC=TFSC=1, TFPS=TNPS=1.

6.2 PI SYSTEM (CONI PI)

CONI PI RETURNS A WORD TO E WITH THE FOLLOWING BIT CONFIGURATION:


17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0


35
34
33
32
31
30
29
28
27
26
25
24
23
22
21

```
!!!
!!+-- 20
!+--- 19
+---- 18
```

BIT	SYMBOL	DESCRIPTION
0-10		CONI: 0
11-17	PRC	CONI: PROGRAM REQUESTS ON CHANNELS CONTAIN 1'S FOR EACH CHANNEL (I.E., EACH PI LEVEL) ON WHICH THERE EXISTS A PROGRAMMED REQUEST. BIT 11 CORRESPONDS TO PI LEVEL 1 AND BIT 17 CORRESPONDS TO PI LEVEL 7.
18	WEAP	CONI: REFLECTS THE CURRENT STATE OF THE WRITE EVEN ADDRESS PARITY FLAG.
19	WEDP	CONI: REFLECTS THE CURRENT STATE OF THE WRITE EVEN DATA PARITY FLAG.
20		MUST BE 0.
21-27	IPC	CONI: INTERRUPTS IN PROGRESS ON CHANNEL - CORRESPOND TO THE PI LEVELS CURRENTLY IN PROGRESS. BIT 21 CORRESPONDS TO PI LEVEL 1 AND BIT 27 CORRESPONDS TO PI LEVEL 7.
28	PA	CONI: PI ACTIVE - A 1 IF THE PI SYSTEM IS CURRENTLY TURNED ON. A 0 IF IT IS TURNED OFF.
29-35	CO	CONI: CHANNELS ON - CORRESPOND TO THE PI CHANNELS WHICH ARE CURRENTLY TURNED ON. BIT 29 CORRESPONDS TO PI CHANNEL 1 AND BIT 35 CORRESPONDS TO PI CHANNEL 7. PI CHANNEL 0 IS ALWAYS TURNED ON WHETHER THE PI SYSTEM IS ON OR OFF, NO MATTER WHAT REQUESTS ARE MADE FOR TURNING SPECIFIC CHANNELS ON OR OFF. SEE CHAPTER 4.3 PDP-10/11 INTERFACE (DTE20) FOR A DESCRIPTION OF PI 0.

6.3 DATA OUT, DATA IN (DATAO, DATAI PI,)

THESE FUNCTIONS ARE UNUSED IN THE KL10.

6.4 MEMORY ADDRESS ERROR REGISTER (BLKO PI,)

BLKO PI RETURNS AT E THE CONTENTS OF THE ERROR ADDRESS (ERA) REGISTER IN THE MBOX. THE ERA REGISTER IS LOADED BY THE MBOX WHEN CERTAIN ERRORS OCCUR ON THE SBUS OR IN THE MBOX. BITS 14

- 35 OF THIS REGISTER CONTAIN THE PHYSICAL ADDRESS AT WHICH THE FIRST SUCH ERROR OCCURED.

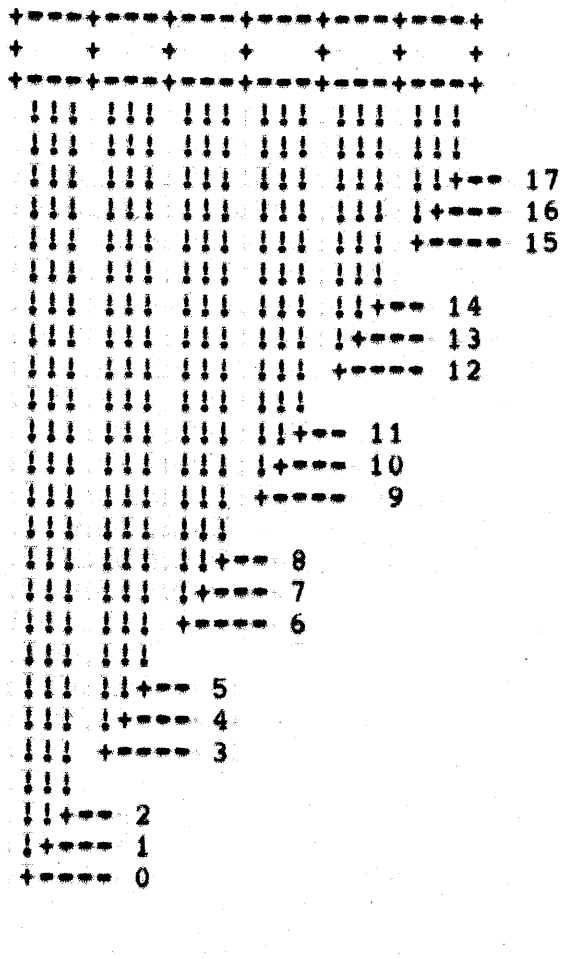
6.5 SBUS DIAGNOSTIC CYCLE (BLKI PI,)

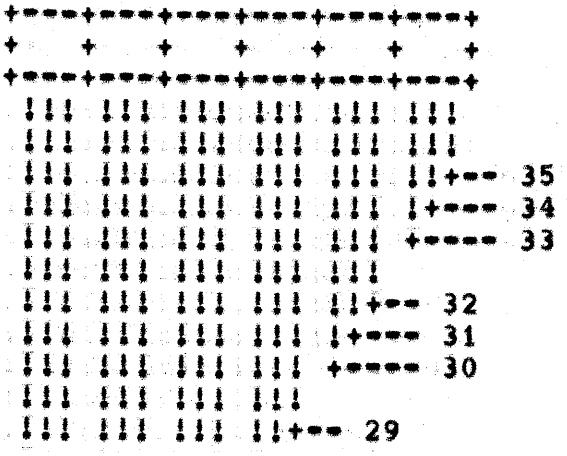
BLKI PI IS THE SBUS DIAGNOSTIC INSTRUCTION. IT TAKES A WORD FROM E AS AN SBUS DIAGNOSTIC INPUT WORD AND RETURNS A WORD TO E + 1 AS THE SBUS DIAGNOSTIC OUTPUT WORD (SEE CHAPTER 3,7, SBUS DIAGNOSTIC CYCLE, FOR COMPLETE DETAILS).

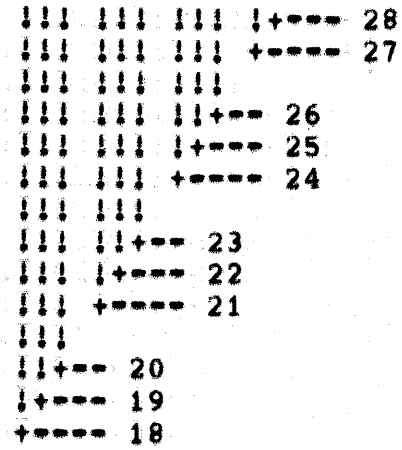
7, DEVICE PAG (ADDRESS 010 OCTAL)

7,1 EXECUTIVE BASE REGISTER (CONO, CONI PAG,)

CONO PAG, IS AN IMMEDIATE INSTRUCTION, IT ALWAYS CLEARS THE HARDWARE PAGE TABLE, CONI PAG RETURNS AT E PRECISELY THE SAME DATA THAT WAS WRITTEN BY CONO PAG, BIT INTERPRETATION IS AS FOLLOWS:







BIT	SYMBOL	DESCRIPTION

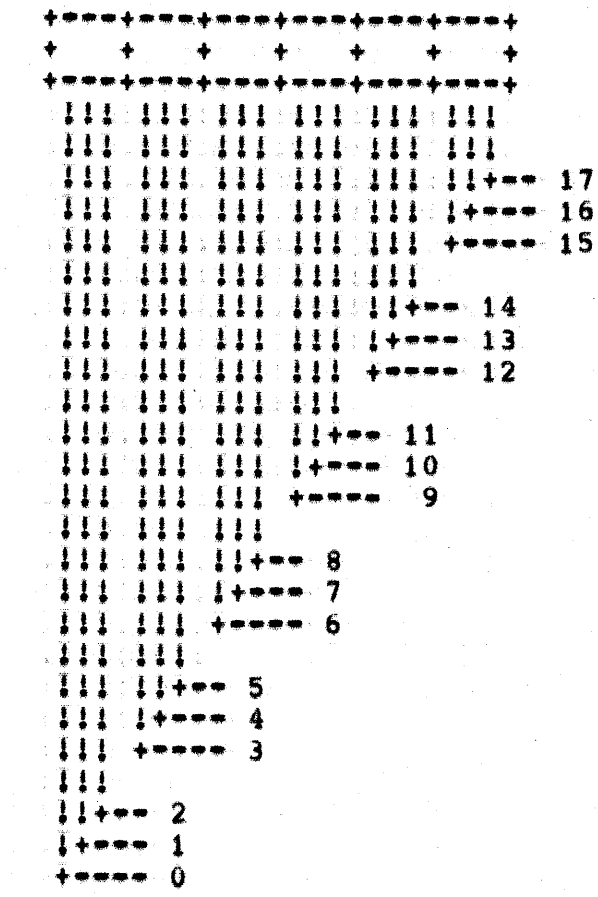
BITS 18 AND 19 DEFINE THE CACHE STRATEGY TO BE USED BY THE PROCESSOR. USUALLY THE OPERATING SYSTEM EITHER SET OR CLEAR THESE BITS - AT SYSTEM STARTUP AND LEAVES THEM ALONE IN ORDER TO USE THE CACHE OR NOT. TURNING OFF BITS 18 AND/OR 19 WITHOUT PERFORMING A CACHE SWEEP MAY HAVE INDETERMINATE RESULTS. THESE BITS ARE INTENDED PRIMARILY FOR DIAGNOSTIC PURPOSES AND FOR RUNNING THE MACHINE WITH OR WITHOUT THE CACHE. THEY ARE NOT INTENDED TO BE CHANGED DURING ORDINARY OPERATION.

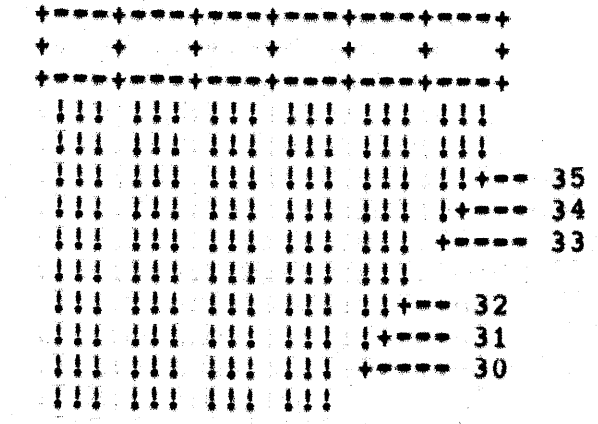
18	CSK	CONO: CACHE STRATEGY LOOK - A 1 PERMITS A CACHE MATCH WHICH INDICATES IT IS POSSIBLE FOR THE PROCESSOR TO FIND A WORD IN THE CACHE.
19	CSD	CONO: CACHE STRATEGY LOAD - A 1 CONTROLS CACHE LOADING WHICH SPECIFIES IT IS POSSIBLE FOR THE PROCESSOR TO INSERT A WORD (OR WORDS) IN THE CACHE.
20		MUST BE 0.
21	KP	CONO: KI PAGING - A 1 SPECIFIES KI PAGING.
		A 0 SPECIFIES THAT THE PAGING PROCESS IS DETERMINED BY THE MICRO-CODE.
22	TE	CONO: TRAP ENABLE - A 1 TURNS ON TRAPS AND PAGING. WHEN 0, NO TRAPS ARE PERMITTED AND ALL ADDRESSES ARE PHYSICAL.
23-35	EBR	CONO: EXECUTIVE BASE REGISTER - SET

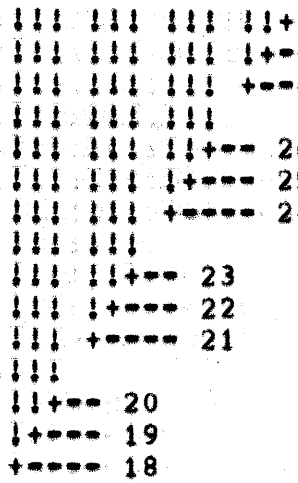
THE EXECUTIVE BASE REGISTER (HIGHER-ORDER 13 BITS OF PHYSICAL ADDRESS OF EXECUTIVE PROCESS TABLE).

7.2 USER BASE REGISTER (DATAO, DATAI PAG,)

DATAO PAG, SETS UP THE PAGING HARDWARE ACCORDING TO THE CONTENTS OF E. DATAI PAG READS THE STATUS OF THE PAGING HARDWARE INTO E. THE BIT CONFIGURATION IS AS FOLLOWS:







BIT	SYMBOL	DESCRIPTION
0	LAB	DATAO: LOAD AC BLOCKS - ENABLES LOADING CURRENT AND PREVIOUS AC BLOCKS (BITS 6 - 11). DATAI: 1 - A 1 IS READ SO THAT THE MONITOR CAN SAVE THE USER STATUS IN THE UPT ON A MUUO AND RESTORE IN ON RETURN WITH A SINGLE DATAO AND DATAO.
1	LPCC	DATAO: LOAD PREVIOUS CONTEXT AND CSWX - ENABLES LOADING THE PREVIOUS CONTEXT SECTION AND CWSX (CALLED WITH SPECIAL EXECUTE) FLAG. DATAI: 1
2	LUBR	DATAO: LOAD USER BASE REGISTER - ENABLES LOADING THE USER BASE REGISTER AND CLEARING THE PROCESS TABLE. DATAI: 1
3-5		MUST BE 0.

ON DATAO, BITS 6 -11 ARE ONLY LOADED WHEN BIT 0 (LAB) IS A 1.

6-8	CAB	DATAO AND DATAI: CURRENT AC BLOCK - THE "CURRENT" AC BLOCK TO BE USED BY ALL AC REFERENCES, INDEX REGISTER REFERENCES, AND EFFECTIVE ADDRESS REFERENCES LESS THAN 20 BY THE CURRENT PROGRAM, EXCEPT UNDER CERTAIN CONDITIONS OF PREVIOUS CONTEXT EXECUTE (PXCT). SEE CHAPTER 2.3, MONITOR CALLING (MUUO AND PXCT).

9-11	PAB	DATAO AND DATAI: PREVIOUS AC BLOCK - SPECIFY THE "PREVIOUS" AC BLOCK NUMBER TO BE USED UNDER THOSE CONDITIONS OF PREVIOUS CONTEXT (PXCT).

ON DATAO, BITS 12 - 17 ARE ONLY LOADED IF BIT 1 (LPCC) OF THE WORD IS A 1.

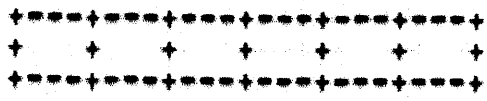
12	CWSX	DATAO AND DATAI: CALLED WITH SPECIAL EXECUTE - CONTAINS CWSX FLAG, SPECIFIES THAT PREVIOUS CONTEXT REFERENCES SHALL USE SECTION EXECUTE ADDRESSING.
13-17	PCS	DATAO AND DATAI: PREVIOUS CONTEXT SECTION - SPECIFY THE SECTION IN WHICH PREVIOUS CONTEXT REFERENCES SHALL BE COMPUTED.
18	ISAM	DATAO: INHIBIT STORING ACCOUNTING METER - A 1 IN BIT 18 INHIBITS STORING THE ACCOUNTING METERS IN THE USER PROCESS TABLE. THIS IS USEFUL AT SYSTEM STARTUP WHEN THERE IS NO OLD USER PROCESS TABLE. IF BIT 18 IS A 0, THE ACCOUNTING METERS WILL BE STORED IN THE USER PROCESS TABLE BEFORE MODIFYING THE USER BASE REGISTER (UBR). UPON MODIFICATION OF THE UBR, NEW VALUES FOR THE ACCOUNTING METERS WILL BE LOADED FROM THE NEW USER PROCESS TABLE. DATAI: 0.
19-22		MUST BE 0'S.

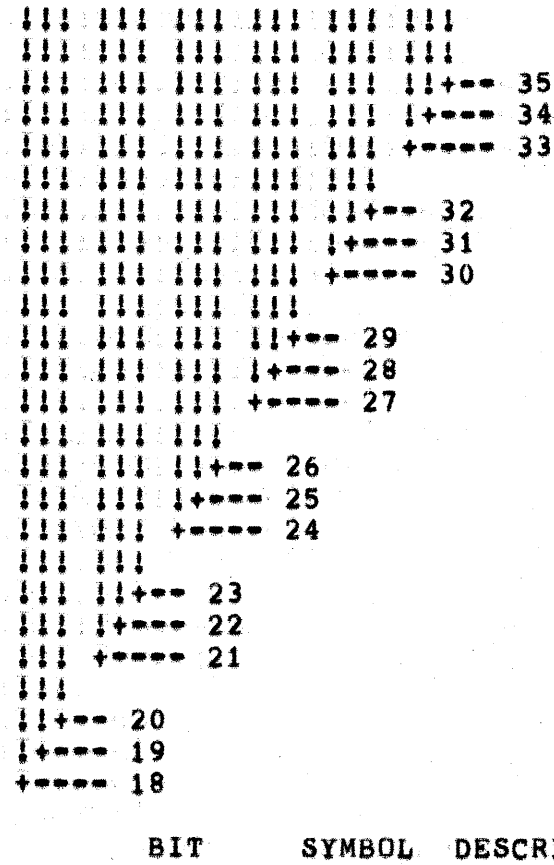
ON DATAO, BITS 23 - 35 ARE LOADED ONLY IF BIT 2 (LUBR) IS A 1.

23-35	UBR	DATAO AND DATAI: USER BASE REGISTER - USER BASE REGISTER (THE 13-BIT PHYSICAL PAGE NUMBER IN WHICH THE PROCESSOR WILL FIND THE USER PROCESS TABLE).

7.3 HARDWARE PAGE TABLE INVALIDATE (BLKO PAG,)

BLKO PAG, IS AN IMMEDIATE INSTRUCTION. THE HALF-WORD E IS INTERPRETED AS FOLLOWS:

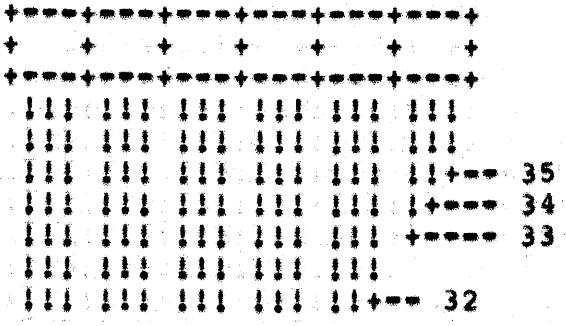


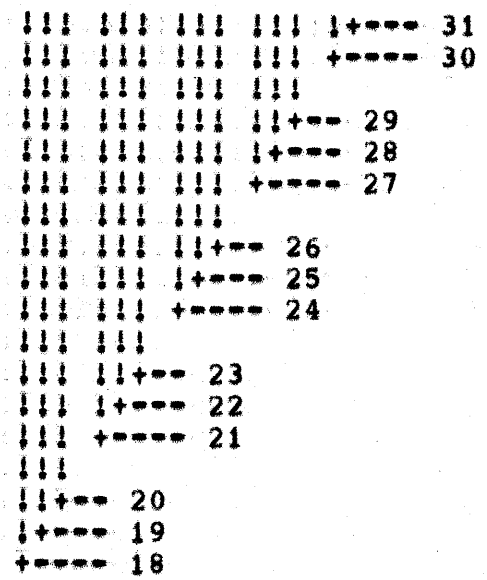


BIT	SYMBOL	DESCRIPTION
18	UPI	BLKO: USER PAGE INVALIDATE - A 1 SPECIFIES A USER PAGE, A 0 SPECIFIES AN EXECUTIVE PAGE.
19-21		MUST BE 0.
22-35	VPN	VIRTUAL PAGE NUMBER - SPECIFY VIRTUAL PAGE NUMBER WHOSE ENTRY IN THE PAGE TABLE SHALL BE INVALIDATED.

7.4 CACHE SWEEP (BLKI PAG,)

BLKI PAG IS AN IMMEDIATE INSTRUCTION. THE HALF-WORD E IS INTERPRETED AS FOLLOWS:





BIT	SYMBOL	DESCRIPTION
18	SOP	BLKI: SWEEP ONE PAGE - A 1 SPECIFIES THAT ONLY ONE PAGE WILL BE SWEPT AS SPECIFIED BY BITS 23 - 35. A 0 SPECIFIES THAT THE ENTIRE CACHE WILL BE SWEPT.
19	VC	BLKI: VALIDATE CORE - A 1 SPECIFIES THAT CORE SHALL BE VALIDATED (I.E., ANY WORD IN THE CACHE WITH ITS WRITTEN BIT ON SHALL BE WRITTEN BACK TO MEMORY). A 0 SPECIFIES THAT NO WORDS WILL BE WRITTEN BACK TO MEMORY.
20	IC	BLKI: INVALIDATE CACHE - A 1 SPECIFIES THAT THE CACHE SHALL BE INVALIDATED (I.E., THE VALID BIT IN THE CACHE SHALL BE TURNED OFF SO THAT THIS WORD WILL NOT BE USED BY THE PROCESSOR FOR ANY MEMORY REFERENCE). A 0 SPECIFIES THAT THE VALID BITS WILL NOT BE CHANGED.
21-22		MUST BE 0.
23-35	PPN	BLKI: PHYSICAL PAGE NUMBER - SPECIFY THE PHYSICAL PAGE NUMBER TO BE USED IN THE SWEEP ONE-PAGE CASES. FOR CASES WHERE ALL PAGES ARE TO BE SWEPT (BIT 18 = 0), BITS 23-35 ARE IGNORED.

8. APPENDIX A - LOOSE ENDS

   A. APPROVAL FOR PARAGRAPH 7.4 (CACHE SWEEP)

   B. CONO/I APR: CACHE DIRECTORY PARITY ERROR FLAG?

   C. POWER FAIL WARNING TIME: 15MS?

   D. BIT ASSIGNMENTS FOR PROCESSOR OPTIONS IN BLKI APR.

9. INDEX

[END OF CH2S06.SPC]

1080,2040,2060 ENGINEERING FUNCTIONAL SPEC - CHAP 2.7

TO: KL10 LIST, J. PARSLOW (200 FILE)

TITLE: KL10 PROGRAM CLOCKS (TIM, MTR) - REV 3

STATUS: THIS CHAPTER REFLECTS THE HARDWARE AS OF INITIAL LAYOUT AND PLANNED MICRO-CODE. SOME CHANGES HAVE OCCURRED SINCE APRIL 30 REVIEW IN ORDER TO FIT THE LOGIC ON A SINGLE BOARD. THIS HAS BEEN REVISED SINCE THE 16 JULY VERSION. WARNING: THE CHANNELS MENTIONED IN THIS SPECIFICATION HAVE NOT BEEN ANNOUNCED. THEREFORE THAT PART MUST BE TREATED AS COMPANY CONFIDENTIAL.

FILE: [EFS]CH2S07.SPC

PDM #: 200-200-016-02

DATE: 25 APR 75

SUPERSEDED MEMOS: PROPOSAL FOR DK20 CLOCK, T. HASTINGS, B. BRUCKERT, 19 APRIL 74; KL10 CLOCKS PROPOSAL, H. STEADMAN, T. HASTINGS, 5 SEPT 73; RESULTS OF DESIGN REVIEW OF KL10 CLOCKS PROPOSAL, H. STEADMAN, 26 SEPT 73; KL10 CLOCKS, H. STEADMAN, 4 OCT 73.

ENGINEER: J. LEONARD, T. HASTINGS, B. BRUCKERT

APPROVED:

EDITOR: T. HASTINGS, M. MILLER

TYPIST:

REVIEWED: 30 APRIL 74 (REV 0)

ABSTRACT

THE KL10 HAS A NUMBER OF BUILT-IN CLOCKS, EACH PROVIDING DIFFERENT TIMING AND COUNTING FUNCTIONS. THESE FUNCTIONS ARE (1) AN INTERVAL TIMER AS A PROGRAMMABLE SOURCE OF INTERRUPTS FROM 10 MICROSECONDS TO 40.96 MILLISECONDS; (2) A ONE MICRO-SECOND TIME BASE; (3) TWO ACCOUNTING METERS, ONE FOR RECORDING "EBOX BUSY TIME" AND ONE FOR COUNTING "MBOX REFERENCES"; AND (4) A PERFORMANCE ANALYSIS COUNTER. THE CLOCKS ARE IMPLEMENTED USING THE TWO INTERNAL DEVICE CODES, TIM AND MTR, ALONG WITH PAG. THE CLOCKS CAN BE READ IN EXEC MODE ONLY. MONITOR CALLS PROVIDE USER ACCESS. THE FRONT END IS EXPECTED TO PROVIDE A LONG-TERM POWER LINE FREQUENCY TIME BASE.

REVISION HISTORY

REV	DESCRIPTION	CHG NO	ORIG	DATE	APPD BY	DATE
0	ORIGINAL			16 JULY 74		
1	CHANGES TO FIT BOARD			21 MAR 75		
2	CHANGES FOUND DURING TESTING			2 APR 75		

0. CONTENTS

1. SUMMARY

THE BUILT-IN KL10 CLOCKS PROVIDE A NUMBER OF TIMING AND COUNTING FUNCTIONS. THEY ARE:

1. INTERVAL TIMER
2. TIME BASE
3. ACCOUNTING METERS (EBOX BUSY TIME AND MBOX REFERENCES)
4. PERFORMANCE ANALYSIS COUNTER

ALL OF THE OPERATIONS ON THESE CLOCKS ARE DONE BY MEANS OF I/O INSTRUCTIONS TO THE INTERNAL DEVICES, TIM, MTR AND PAG, WHICH HAVE INTERNAL DEVICE CODES 020, 024, AND 010 RESPECTIVELY. MANY OF THESE FUNCTIONS USE A MICRO-SECOND SOURCE OF PULSES WHICH IS DOWN COUNTED FROM THE BASIC MACHINE CLOCK WHICH HAS LESS THAN FIVE SECONDS DRIFT OVER 24 HOURS. NOTE: IT HAS BEEN AGREED THAT IF THE BASIC MACHINE CLOCK RATE MUST BE SLOWED DOWN, IT WILL BE DONE TO A SMALLER WHOLE INTEGER MHZ SO IT CAN BE DIVIDED DOWN TO MICRO-SECONDS, I.E. 31 MHZ, 30 MHZ, ETC. IT WILL NOT BE SLOWED DOWN TO A FRACTION OF A MHZ.

THE INTERVAL TIMER CONSISTS OF A 12-BIT LOADABLE PERIOD REGISTER AND A 12-BIT READABLE INTERVAL COUNTER. THE INTERVAL TIMER CAUSES A VECTOR INTERRUPT WHEN THE INTERVAL COUNTER MATCHES THE PERIOD REGISTER. THE OTHER FOUR "CLOCKS" DIFFER FROM THE INTERVAL TIMER BUT ARE SIMILAR TO EACH OTHER. EACH OF THESE "CLOCKS" CONSISTS OF A 59-BIT UNSIGNED DOUBLE PRECISION INTEGER WITH THE BINARY POINT BETWEEN BITS 23 AND 24 OF THE SECOND WORD. THE CLOCKS ARE READABLE BY SPECIAL INSTRUCTIONS ON INTERNAL DEVICES TIM OR MTR. THUS THE SOFTWARE CAN ADD A DOUBLE PRECISION BASE WITH ONE DOUBLE PRECISION INTEGER ADD (DADD). IN ORDER TO SAVE HARDWARE, ONLY THE LOW ORDER 16 BITS OF THE 59-BIT QUANTITIES ARE IMPLEMENTED IN HARDWARE. THE REST OF THE INTEGER PART IS STORED IN MEMORY IN DOUBLE WORDS IN THE EPT AND UPT. THE LOW ORDER 12 BITS ARE ALWAYS ZERO AND ARE RESERVED FOR A FUTURE MACHINE WHICH MAY WANT TO COUNT AT A HIGHER RATE THAN THE KL10. THESE 16-BIT REGISTERS ARE CLEARABLE BUT CANNOT BE DIRECTLY LOADED. UNLIKE THE INTERVAL TIMER, THESE "CLOCKS" NEVER GENERATE SOFTWARE INTERRUPTS.

2. TERMINOLOGY

2.1 PROGRAM INTERRUPT

AN INTERRUPT OF THE SEQUENCE OF CPU INSTRUCTIONS (AS OPPOSED TO MICRO-CODE INSTRUCTIONS) IN RESPONSE TO AN EXTERNAL EVENT, USUALLY AN IO DEVICE. PROGRAM INTERRUPTS ARE DISTINGUISHED FROM IOP FUNCTIONS IN THAT THE PROGRAM IS UNAWARE OF THE OCCURRENCE OF THE LATTER.

2.2 PI 1-7 INTERRUPT REQUEST

A SIGNAL THAT AN EXTERNAL DEVICE REQUIRES SERVICE. THE ACTION TAKEN BY THE EBOX MAY BE EITHER AN IOP FUNCTION OR A PROGRAM INTERRUPT. THE SOFTWARE CAN ENABLE AND DISABLE THESE SIGNALS.

2.3 PI 0 INTERRUPT REQUEST

A HIGH PRIORITY SIGNAL THAT AN EXTERNAL DEVICE (DTE20) REQUIRES SERVICE (HIGHER THAN PI 1-7). THE ACTION TAKEN BY THE EBOX CAN ONLY BE AN IOP DATAI, DATAO, OR BYTE TRANSFER.

2.4 METER UPDATE REQUEST

A HIGH PRIORITY SIGNAL THAT AN INTERNAL DEVICE (MTR, TIM) REQUIRES SERVICE (HIGHER THAN PI 0-7). THE ACTION TAKEN BY THE EBOX IS TO DETERMINE WHICH COUNTER HAS OVERFLOWED, READ AND CLEAR IT, AND ADD ITS CONTENTS TO THE APPROPRIATE EPT OR UPT DOUBLE WORD.

2.5 MBZ

MBZ IS AN ABBREVIATION FOR MUST BE ZERO. IT IS USED IN INSTRUCTION FORMATS FOR UNUSED FIELDS. THESE FIELDS MUST BE ZERO BECAUSE THEY ARE RESERVED TO DIGITAL FOR FUTURE USE BY MICRO-CODE OR HARDWARE ON THE KL10 OR FUTURE MACHINES.

2,6 EBOX BUSY TIME

WHILE THE EBOX IS WORKING, THE ACCOUNTING METER INCREMENTS AT HALF THE SYSTEM CLOCK RATE. HOWEVER, IF THE EBOX CLOCK IS SUSPENDED, TO WAIT FOR COMPLETION OF A MEMORY REFERENCE (EITHER CACHE OR CORE MEMORY), THE ACCOUNTING METER STOPS RECORDING EBOX BUSY TIME. IN ADDITION, NO BUSY TIME IS RECORDED DURING PAGE REFILLS, OR PI CYCLES. THUS, EBOX BUSY TIME IS AN ACCOUNTING MEASURE OF THE USEFUL WORK A USER OBTAINED FROM THE EBOX I.E., ALL USER MODE AND MONITOR CALL TIME. THE MONITOR MAY CHOOSE TO INCLUDE OR EXCLUDE (1) EXEC MODE PROGRAM INTERRUPT TIME AND/OR (2) EXEC MODE TIME WITH NO INTERRUPTS IN PROGRESS.

2,7 MBOX REFERENCE COUNT

EACH REFERENCE BY THE EBOX TO THE MBOX FOR A MEMORY WORD (CACHE HIT OR MISS) IS A SEPARATE MBOX REFERENCE. AC REFERENCES DO NOT GO THROUGH THE MBOX AND SO DO NOT CONTRIBUTE TO THE MBOX REFERENCE COUNT. THE MBOX MAY REFERENCE CORE MEMORY 1, 2, 3, OR 4 WORDS AT A TIME IN RESPONSE TO A SINGLE EBOX REQUEST. THE MBOX REFERENCE COUNT IS ONLY COUNTED ONCE FOR EACH EBOX REQUEST. WRITE BACKS OR CHANNEL REFERENCES ARE NOT PART OF THE MBOX REFERENCE COUNT EITHER. THUS THE MBOX REFERENCE COUNT IS A REPEATABLE MEASURE OF THE USEFUL WORK DONE BY THE MBOX FOR THE CURRENT PROGRAM.

2,8 CHANNEL BUSY

A CHANNEL IS BUSY IF IT IS ACTUALLY TRANSFERRING DATA OR WAITING FOR LATENCY. IT IS NOT BUSY IF ALL THE ATTACHED UNITS ARE IDLE OR JUST SEEKING. CHANNEL BUSY IS MEASURED WHEN THE CHANNEL REGISTER IS FULL.

3. GOALS

1. PROVIDE THE MONITOR WITH A PROGRAMMABLE PERIODIC SOURCE OF INTERRUPTS FOR REAL TIME SCHEDULING AND PAGE MANAGEMENT.

2. PROVIDE THE MONITOR WITH A 1 MICROSECOND RESOLUTION TIME BASE WITH DRIFT OF LESS THAN 5 SECONDS IN 24 HOURS. THUS CORRECTION USING POWER LINE FREQUENCY CLOCK IN FRONT END PDP-11 IS NEEDED ONLY AT MIDNIGHT.

3. PROVIDE THE MONITOR WITH A REPRODUCIBLE MEASURE OF A USER PROGRAM EXECUTION. THIS MEASURE MUST CLOSELY REFLECT AVERAGE RESOURCES USED BUT BE INDEPENDENT OF CACHE MISSES, MONITOR INTERRUPTS, HARDWARE PAGE TABLE RELOADS, MONITOR CALLS (IF DESIRED), PAGING (IF DESIRED) AND SYSTEM LOAD. IT SHOULD NOT BE POSSIBLE FOR A MALICIOUS USER TO ESCAPE CHARGES. A 1% VARIATION IS TOLERABLE OVER THE MAXIMUM RANGE OF SYSTEM LOADS.

4. THE ACCOUNTING MEASURE MUST BE EASILY COMPARABLE TO OTHER SYSTEMS (KI AND COMPETITION) FOR PURPOSES OF BENCHMARKS AND COMPARISONS.

5. PROVIDE FOR PERFORMANCE ANALYSIS MEASURES FOR PURPOSES OF IDENTIFYING HARDWARE AND SOFTWARE BOTTLENECKS NOT EASILY OBTAINED STRICTLY WITH SOFTWARE. SPECIFICALLY CACHE PERFORMANCE, I/O OVERLAP, AND INTERRUPT TIME SHOULD BE MEASURABLE. SUCH PROVISIONS WILL HELP THE CUSTOMER MAKE THE BEST UPGRADE DECISIONS AS HIS LOAD INCREASES.

7. USE A MINIMUM OF DEVICE CODES (ONLY TWO). USE ONLY INTERNAL DEVICE CODES AS OPPOSED TO USER DEVICE CODES SO I/O OPCODE SPACE CAN BE RECLAIMED IN A FUTURE MACHINE.

8. TO HELP DIAGNOSING TO THE BOARD LEVEL, THE KL10 CLOCKS BOARD AND PI BOARD WILL BE KEPT SEPARATE, EVEN AT THE COST OF EXTRA DIPS AND EBUS LOADING. HOWEVER THE SEPARATION OBJECTIVE WILL BE SACRIFICED IF NECESSARY IN ORDER TO FIT THE KL10 CLOCKS AND PI SYSTEM ON TWO BOARDS TOTAL.

9. PLAN FOR FUTURE MACHINES WHERE CLOCK RATES MAY BE FASTER.

4. NON-GOALS

1. THE MICRO-CODE WILL NOT BE ABLE TO AFFECT THE USER ACCOUNTING METERS EXCEPT TO TURN THEM ON AND OFF. THUS FUTURE VERSIONS OF THE MICRO-CODE OR FUTURE MACHINES WILL GENERALLY GENERATE DIFFERENT CHARGES FOR INSTRUCTIONS.

2. LONG TERM POWER LINE CLOCK IS NOT PROVIDED. THE FRONT END WILL PROVIDE.

3. DO NOT PROVIDE PERFORMANCE ANALYSIS INFORMATION WHICH CAN BE GATHERED EASILY BY PURELY SOFTWARE TECHNIQUES.

4. WE DISCUSSED AN INTERMEDIATE 4-BIT REGISTER TO ACCUMULATE ACCOUNTING COUNTS IN ORDER TO PREVENT DOUBLE ACCOUNTING OF INTERRUPTED INSTRUCTIONS. WE HAVE SINCE DECIDED NOT TO HAVE THE 4-BIT REGISTER SINCE WE ESTIMATE THAT THE NUMBER OF INTERRUPTS WILL BE LESS THAN 5000 PER SECOND. THUS INTERRUPTS WILL CAUSE VARIATIONS, BUT WE THINK IT WILL BE WITHIN 1% GOAL.

5. ONLY 1 SET OF KL10 CLOCKS CAN BE PROVIDED PER CPU.

6. THE KL10 CLOCKS ARE REQUIRED AND CANNOT BE OPTIONALLY LEFT OUT OF EACH CPU ON 1080, 2040, OR 2020. HENCE THE OPTION DESIGNATION DK20 IS NOT USED.

7. THE CLOCK DIVIDER FROM 32 MHZ (OR WHATEVER THE FINAL CLOCK RATE IS) TO 1 MHZ WILL NOT BE PROGRAMMABLE. THUS IF MACHINE IS SLOWED DOWN AT A PARTICULAR INSTALLATION, ALL OF THE CLOCKS ARE SLOWED DOWN TOO.

8. MAINTENANCE OF TIME BASE OVER POWER FAILURE BY USE OF BACKUP POWER SUPPLY WILL NOT BE PROVIDED. ALSO THE TIME BASE MAY STOP IF KL10 HALTS OR EBOX CLOCK STOPS.

9. THE KL10 CLOCKS WILL NOT PROVIDE THE SYNCHRONIZATION BETWEEN THE CPUS IN A MULTI-PROCESSING SYSTEM. INSTEAD EACH CPU SHOULD BE ABLE TO RECEIVE ITS BASIC MACHINE FREQUENCY FROM A MASTER CPU (UNDER PROGRAM CONTROL).

10. PROBE INPUT FROM TTL PART OF MACHINE WILL NOT BE PROVIDED, SINCE IT WOULD REQUIRE SYNCHRONIZERS WHICH WOULD BE TOO MANY DIPS INCLUDING TTL POWER.

11. PERFORMANCE ANALYSIS OF MULTIPLE EVENTS WILL NOT BE VERY ACCURATE, BECAUSE THE TRIGGER CIRCUIT IS PUT ON THE BOOLEAN EXPRESSION OUTPUT INSTEAD OF EACH OF THE 22 INPUTS. THIS WOULD HAVE COST 11 DIPS.

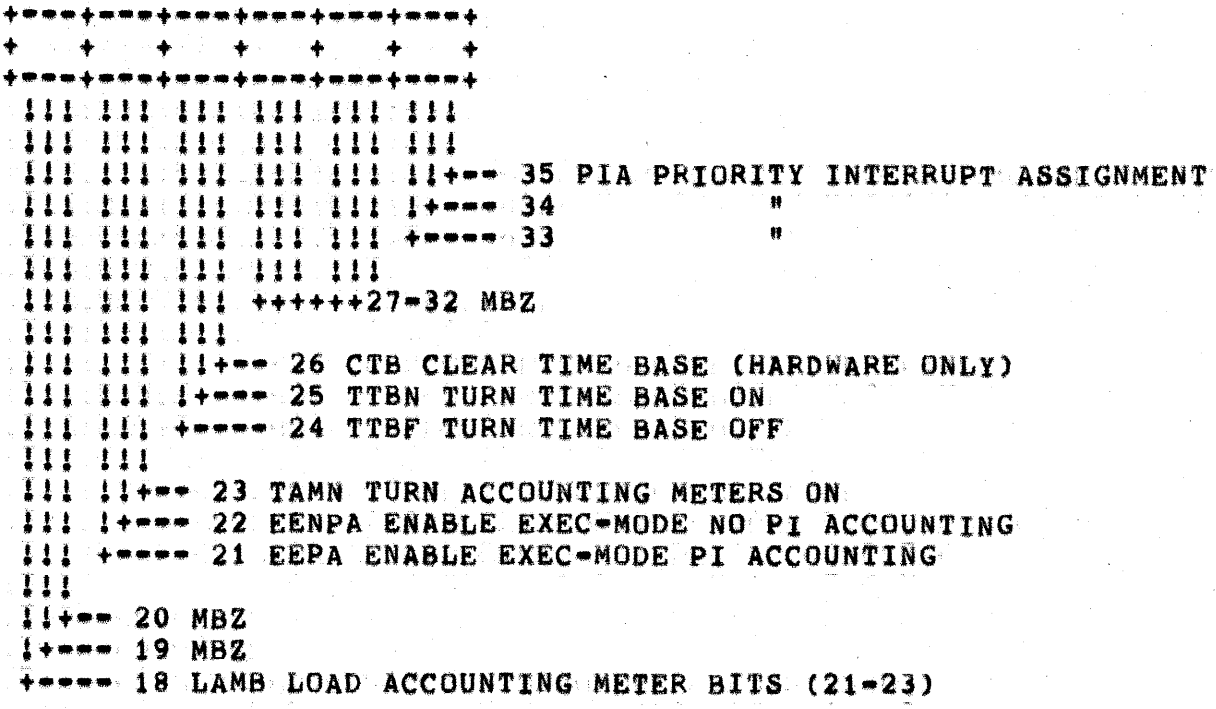
12. A USER INSTRUCTION WILL NOT BE PROVIDED TO READ ANY CLOCKS DIRECTLY.

5. PROGRAMMING COMMON TO ALL 5 CLOCKS

MOST OF THE TRADITIONAL CONTROL FLAGS ARE CONCENTRATED IN THE MTR REGISTER FOR ALL OF THE 5 CLOCKS.

5.1 SET KL10 CLOCK ENABLES

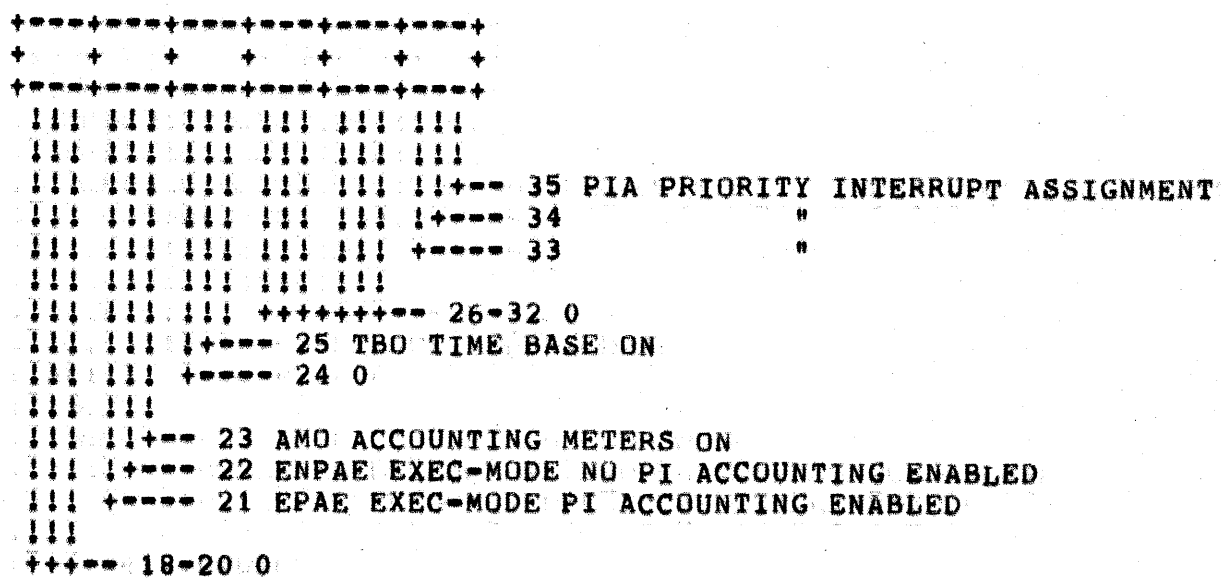
CONO MTR,E



5,2 READ KL10 CLOCK ENABLES

CONI MTR, E (LH) = 0

CONI MTR, E (RH)



5.3 DETAILED DESCRIPTION OF CONO/CONI MTR.

BIT	SYMBOL	DESCRIPTION
0-17		CONI: 0
18	LAMB	CONO: LOAD ACCOUNTING METER BITS 21-23 - 1 MEANS CHANGE THE STATE OF THE ACCOUNTING METER ENABLES TO THAT SPECIFIED BY CONO BITS 21-23. A 0 MEANS DO NOT CHANGE THE STATE OF THE ACCOUNTING METER ENABLES.
		CONI: 0
19-20		CONO: MBZ; CONI: 0
21	EEPA	CONO: ENABLE EXEC-MODE PI ACCOUNTING - IF LAMB=1, A 1 MEANS ENABLE ACCOUNTING WHEN ANY PI IS IN PROGRESS IN EXEC MODE (IN ADDITION TO ACCOUNTING IN USER MODE WHETHER A PI IS IN PROGRESS OR NOT).
	EPAE	CONI: EXEC-MODE PI ACCOUNTING ENABLED - A 1 MEANS ACCOUNTING IS ENABLED WHEN ANY PI IS IN PROGRESS IN EXEC MODE (IN ADDITION TO USER MODE WHETHER A PI IS IN PROGRESS OR NOT). HOWEVER THE ACCOUNTING METERS MUST ALSO BE TURNED ON (AMO=1) IN ORDER TO COUNT. A 0 MEANS ACCOUNTING IS DISABLED WHEN ANY PI IS IN PROGRESS IN EXEC MODE.
22	EENPA	CONO: ENABLE EXEC-MODE NO PI ACCOUNTING. IF LAMB = 1, A 1 MEANS ENABLE ACCOUNTING WHEN NO PI IS IN PROGRESS AND CPU IS IN EXEC MODE.
	ENPAE	CONI: EXEC-MODE NO PI ACCOUNTING ENABLE. A 1 MEANS ACCOUNTING IS ENABLED IN EXEC-MODE WHEN THERE ARE NO PIS IN PROGRESS.
23	TAMN	CONO: TURN ACCOUNTING METERS ON - A 1 MEANS TURN ON THE ACCOUNTING METERS. THE ACCOUNTING METERS WILL COUNT (1) WHEN THE CPU IS IN USER MODE WHETHER PIS ARE IN PROGRESS OR NOT AND (2), IF ENPAE=1, WHEN THE CPU IS IN EXEC MODE WITH NO PIS IN PROGRESS AND (3) IF EPAE=1, WHEN IN EXEC MODE WITH PIS IN PROGRESS. A 0 MEANS TURN OFF THE ACCOUNTING METER NO MATTER THE MODE OF THE CPU.
	AMO	CONI: ACCOUNTING METERS ON - A 1 MEANS THE ACCOUNTING METERS ARE ON. THE ACCOUNTING METER WILL NOT COUNT UNLESS IT IS TURNED ON. A 0

MEANS THE ACCOUNTING METER IS OFF.

24	TTBF	CONO: TURN TIME BASE OFF - A 1 MEANS TURN TIME BASE OFF. IF TTBF AND TTBN ARE 0, DON'T AFFECT THE STATE OF THE TIME BASE NOTE: TFTB AND TNTB MAY NOT BOTH BE 1.
		CONI: 0
25	TTBN	CONO: TURN TIME BASE ON - A 1 MEANS TURN TIME BASE ON. IF TTBF AND TTTBN ARE 0, DON'T AFFECT THE STATE OF THE TIME BASE. NOTE: TTBF AND TTBN MAY NOT BOTH BE 1.
	TBO	CONI: TIME BASE ON - A 1 MEANS THE TIME BASE IS ON. A 0 MEANS THE TIME BASE IS OFF.
26	CTB	CONO: CLEAR TIME BASE - A 1 MEANS CLEAR TIME BASE. ONLY THE HARDWARE REGISTER IS CLEARED. THE SOFTWARE MUST ALSO CLEAR THE CORE LOCATIONS IN THE EPT. THIS MUST BE DONE WHEN THE TIME BASE IS TURNED OFF TO BE RACE FREE. THIS SHOULD NOT BE A PROBLEM SINCE THE TIME BASE WILL BE CLEARED AT SYSTEM STARTUP ONLY.
		CONI: 0
27-32		CONO: MBZ; CONI: 0
33-35	PIA	CONO: PRIORITY INTERRUPT ASSIGNMENT FOR PROGRAM INTERRUPTS FOR THE INTERVAL TIMER. A 0 MEANS THAT NO INTERRUPT ASSIGNMENT HAS BEEN GIVEN, SO THAT NO PROGRAM INTERRUPTS ARE GENERATED. HOWEVER ALL TURNED ON CLOCKS WILL CONTINUE TO INCREMENT EVEN IF PIA=0.
	PIA	CONI: PRIORITY INTERRUPT ASSIGNMENT - READS THE PRIORITY INTERRUPT ASSIGNMENT SET BY THE LAST CONO MTR.

6. INTERVAL TIMER

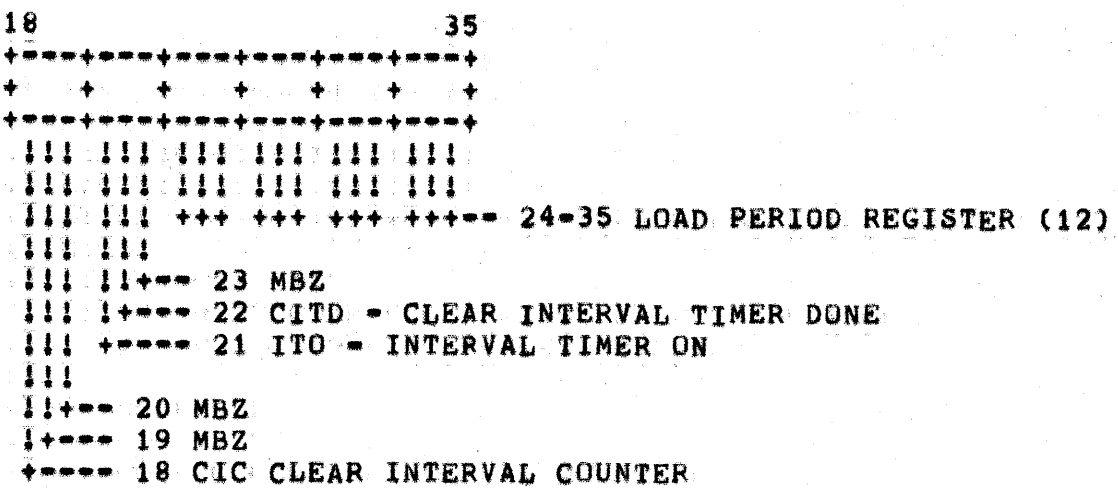
6.1 INTERVAL TIMER SUMMARY

THE INTERVAL TIMER PROVIDES A PROGRAMMABLE SOURCE OF INTERRUPTS FROM 10 MICROSECONDS TO 40.96 MILLISECONDS AND IS SIMILAR TO THE DK10. IT IS USED FOR REAL TIME APPLICATIONS AND FOR PAGE MANAGEMENT BY THE OPERATING SYSTEM. IT IS DESIGNED SO THAT A REAL TIME DEADLINE SCHEDULER WITH VARYING DEADLINES CAN BE IMPLEMENTED. THE NEW DEADLINE IS SET WITH RESPECT TO THE TIME OF THE PREVIOUS INTERRUPT REQUEST AND IS INDEPENDENT OF HOW LONG THE INTERRUPT ROUTINE TAKES TO RESET THE PERIOD REGISTER. THE INTERVAL TIMER IS TURNED ON BY CONO TIM, ITO (INTERVAL TIMER ON). RESET AND CONO TIM, WITH ITO=0, TURN OFF THE INTERVAL TIMER SO THAT IT NO LONGER INCREMENTS (OR INTERRUPTS). PROGRAM INTERRUPTS MAY BE DISABLED WHILE THE CLOCK CONTINUES TO COUNT BY LOADING 0 IN BITS 33-35 OF CONO MTR,. IT IS THE ONLY KL10 CLOCK FUNCTION WHICH GENERATES PROGRAM INTERRUPTS. A CONO TIM, E LOADS BIT 24-35 OF E INTO THE 12-BIT PERIOD REGISTER. A 12-BIT INTERVAL COUNTER IS INCREMENTED EVERY MICROSECOND. WHEN THE INTERVAL COUNTER COMPARES EQUAL TO THE PERIOD REGISTER, THE INTERVAL TIMER RESETS THE INTERVAL COUNTER TO 0, SETS INTERVAL TIMER DONE (ITD), AND REQUESTS A VECTOR INTERRUPT TO EPT LOCATION EPTITV (EXEC PROCESS TABLE - INTERVAL TIMER VECTOR = 514). SEE CHAPTER 2.9, EXEC AND USER PROCESS TABLE (EPT, UPT). THE SOFTWARE MUST CLEAR INTERVAL TIMER DONE, BY A CONO TIM, CITD. THE SOFTWARE MAY OPTIONALLY CLEAR THE INTERVAL COUNTER WHEN THE PERIOD REGISTER IS LOADED BY THE CONO TIM, E IF THE CIC (CLEAR INTERVAL COUNTER) BIT IS SET. A CONI TIM, E READS THE 12-BIT INTERVAL COUNTER AT ANY TIME WITH BITS 0-5 ZERO FOR EASY COMPARING. FOR DIAGNOSTIC PURPOSES, THE RH OF CONI MTR, E READS THE 12-BIT PERIOD REGISTER.

TO HELP THE SOFTWARE DETECT THE PROGRAMMING ERROR OF LOADING THE PERIOD REGISTER WITH A NUMBER WHICH IS LESS THAN OR EQUAL TO THE CURRENT VALUE OF THE INTERVAL COUNTER, AN INTERVAL COUNTER OVERFLOW (ICO) INTERRUPT IS PROVIDED WHEN A CARRY OCCURS OUT OF THE MOST SIGNIFICANT BIT. CLEARING THE INTERVAL TIMER DONE FLAG (CONO TIM, CITD) CLEARS BOTH BITS.

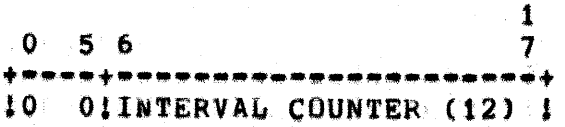
6,2 SET INTERVAL TIMER PERIOD REGISTER

CONO TIM,E

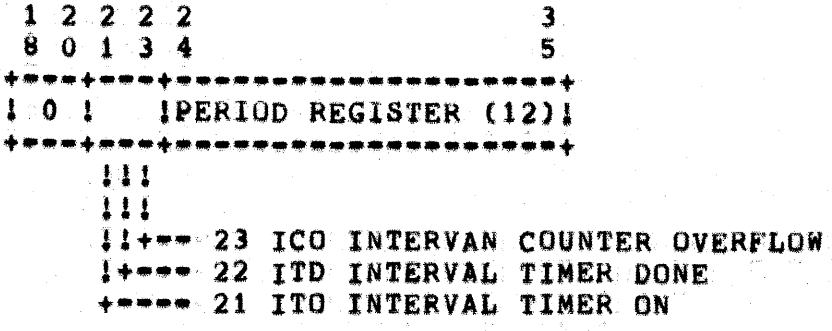


6,3 READ INTERVAL TIMER

CONI TIM,E (LH)



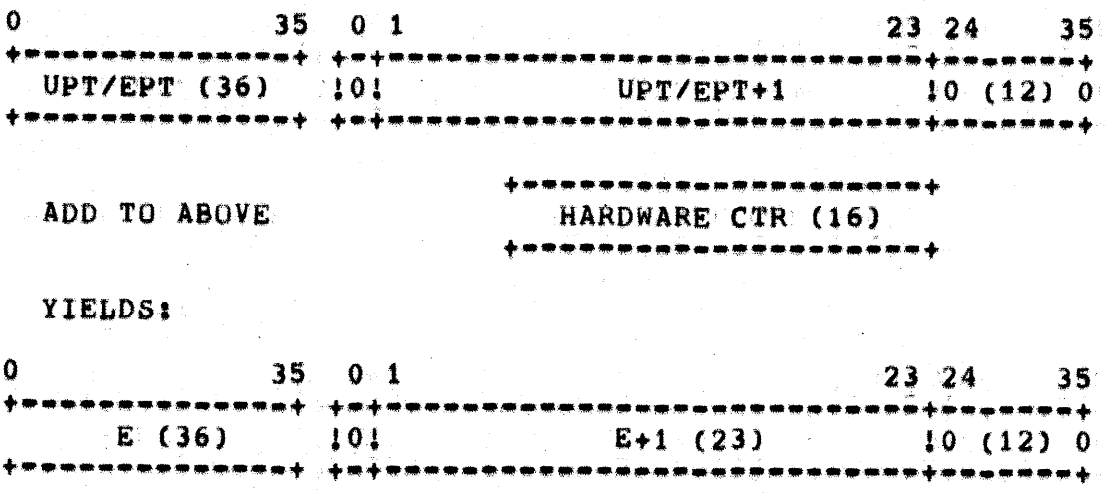
CONI TIM,E (RH)



7. FUNCTIONS COMMON TO THE FOUR 59-BIT CLOCKS

THE FOUR 59-BIT CLOCKS, (TIME BASE, EBOX BUSY, MBOX REFERENCE AND PERFORMANCE ANALYSIS) ALL HAVE THE FOLLOWING FUNCTIONAL DESCRIPTION IN COMMON. EACH OF THE CLOCKS IS READABLE BY A SPECIAL DOUBLE PRECISION INSTRUCTION. THE BINARY POINT OF THE UNSIGNED DOUBLE PRECISION INTEGER IS BETWEEN BITS 59 AND 60 I.E., BITS 23 AND 24 OF THE SECOND WORD. THE FORMAT IS THE SAME AS THAT FOR DOUBLE-PRECISION INTEGERS, I.E., SIGN BIT, (=0) PLUS 70 BITS, WITH BIT 0 OF SECOND WORD ALWAYS BEING 0. THUS A FUTURE MACHINE CAN COUNT AT A FASTER RATE AND PROVIDE A COMPATIBLE CLOCK FORMAT.

IN ORDER TO SAVE HARDWARE, ONLY THE LOW ORDER 16 BITS OF EACH CLOCK ARE MAINTAINED IN HARDWARE. EACH HAS A CORRESPONDING DOUBLE WORD IN A PROCESS TABLE, WHICH CAN BE REGARDED AS THE VALUE OF THE FULL CLOCK AT THE MOST RECENT TIME THE HARDWARE COUNTER HAD 0. THE TIME BASE AND PERFORMANCE ANALYSIS COUNTER ARE PER-SYSTEM QUANTITIES AND SO ARE STORED IN THE EPT, WHILE THE EBOX BUSY AND MBOX REFERENCE ARE PER-PROCESS QUANTITIES AND SO ARE STORED IN THE UPT. THE INSTRUCTIONS TO READ EACH CLOCK READ THE 16-BIT HARDWARE COUNTER, ADD THE DOUBLEWORD FROM A PROCESS TABLE, AND RETURN THE SUM AT E AND E+1, AS A DOUBLE PRECISION INTEGER WITH THE BINARY POINT BETWEEN BITS 23 AND 24 OF THE SECOND WORD.



WHEN THE 16-BIT REGISTER IS "HALF FULL" (IE, THE HIGH-ORDER BIT IS 1), IT POSTS A REQUEST FOR SERVICE BY THE MICROCODE. AT ITS NEXT OPPORTUNITY, THE MICRO-CODE READS AND CLEARS THAT REGISTER, ADDING ITS CONTENTS TO THE APPROPRIATE DOUBLE WORD IN THE PROCESS TABLE. THUS NO PI LEVEL IS ASSIGNED TO THE INCREMENT FUNCTION. THE APPROPRIATE CLOCK MEMORY WORD IS ALWAYS UPDATED (IF THE CLOCK IS TURNED ON) NO MATTER WHAT THE STATE OR LEVEL OF THE PI SYSTEM. THUS THE SOFTWARE NEVER HAS TO WORRY ABOUT THE RACE CONDITION WHEN READING A 60-BIT CLOCK.

8. TIME BASE

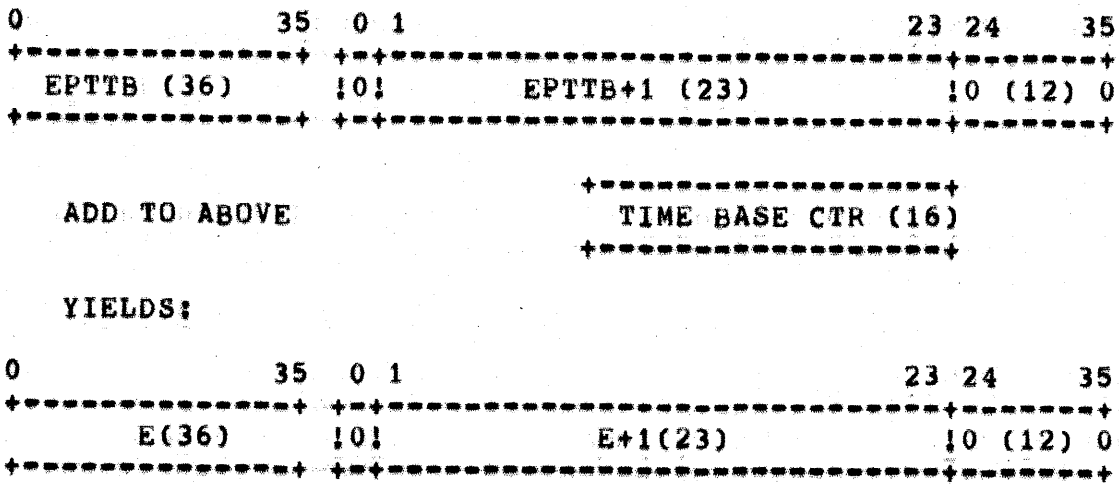
8.1 TIME BASE SUMMARY

THE TIME BASE PROVIDES A 59-BIT MICROSECOND SOURCE OF ELAPSED TIME. A RDTIME E READS THE 59-BIT RELATIVE TIME OF DAY INTO E AND E+1. THE HIGH ORDER 36 BITS ARE KEPT IN WORD EPTTB (EXEC PROCESS TABLE - TIME BASE = 510) OF THE EPT. THE LOW ORDER 23 BITS ARE KEPT IN BITS 1-23 OF WORD EPTTB+1 OF THE EPT. THE TIME BASE IS TURNED ON BY A CONO MTR, TTBN. THE TIME BASE IS TURNED OFF BY RESET AND BY CONO MTR, TTBF.

8.2 READ TIME BASE

RDTIME E

READS 59-BIT TIME BASE IN MICRO SECONDS INTO E AND E+1 AS A DOUBLE PRECISION UNSIGNED INTEGER WITH BINARY POINT BETWEEN BITS 23 AND 24 OF THE SECOND WORD.



9. ACCOUNTING METERS

9.1 ACCOUNTING METERS SUMMARY

THE ACCOUNTING METERS PROVIDE AN ACCURATE AND REPRODUCIBLE MEASURE OF THE AMOUNT OF PROCESSOR RESOURCE USED INDEPENDENT OF THE VARIATION CAUSED BY THE CACHE.

THE EBOX BUSY COUNTER COUNTS ALTERNATE SYSTEM CLOCK TICKS FOR WHICH THE EBOX IS BUSY. WHEN THE BUSY COUNTER IS HALF FULL, A REQUEST IS MADE TO ADD ITS CONTENTS TO A DOUBLE WORD IN MEMORY AND CLEAR THE COUNTER. INSTEAD OF COUNTING EVERY CLOCK TICK, ONLY EVERY OTHER ONE IS COUNTED BECAUSE OF THE DIFFICULTY OF COUNTING AT 32 MHZ. A RDEACT E READS THE 59-BIT QUANTITY INTO E AND E+1.

THE MEMORY REFERENCE (CYCLE) COUNTER KEEPS A 59-BIT COUNT OF THE NUMBER OF LOGICAL MEMORY REFERENCES MADE BY THE EBOX, WHETHER A CACHE HIT OR A MISS. ONLY REQUESTS MADE BY THE EBOX BY THE CURRENTLY RUNNING PROGRAM ARE COUNTED. THE ACCOUNTING METERS DO NOT COUNT DURING PI CYCLES (DTE DEX REQUESTS - DTE20 EXAMINE, DEPOSIT REQUESTS; DTE20 BYTE TRANSFERS, INTERRUPTS WHICH EXECUTE A SINGLE INSTRUCTION), CHANNEL MEMORY REFERENCES ARE NOT COUNTED. AC REFERENCES ARE NEVER COUNTED. WRITE-BACKS AND MULTIPLE WORD FETCHES ARE NOT COUNTED SINCE THEY ARE INITIATED BY THE MBOX RATHER THAN THE EBOX. A RDMACT E READS THE 59-BIT QUANTITY INTO E AND E+1. THE ACCOUNTING METERS ARE TURNED ON FOR THE SYSTEM BY A CONO MTR, LAMB+TAMN, RESET AND CONO MTR, LAMB+AMO WITH AMO=0 TURN THE ACCOUNTING CLOCKS OFF.

IN ORDER TO GIVE MEANINGFUL CHARGES, USERS WILL BE GIVEN A PAIR OF NUMBERS: EBOX CYCLES (E) AND MBOX CYCLES (M). IN ORDER TO REPORT SECONDS OF CPU TIME USED, THE SOFTWARE IS EXPECTED TO USE THE TWO NUMBERS, E AND M, IN THE FOLLOWING FORMULA:

SECONDS KL10 TIME = X*E + A*M

WHERE A IS THE AVERAGE FRACTION OF A SECOND REQUIRED TO MAKE A SINGLE LOGICAL MEMORY REFERENCE AND X IS TWICE THE PERIOD OF THE SYSTEM CLOCK.

X = 1/16 000 000
A = 1/ 2 500 000 (APPROX)

"A" WILL BE DETERMINED EMPIRICALLY WHEN THE MACHINE IS RUNNING USING THE PERFORMANCE ANALYSIS COUNTER ON THE DEVELOPMENT MACHINE.

WHEN THE ACCOUNTING METERS ARE TURNED ON, THEY ARE ALWAYS ENABLED TO COUNT WHEN THE CPU IS IN USER MODE EVEN IF PIS ARE IN PROGRESS. THE OPERATING SYSTEM CAN ENABLE THE ACCOUNTING METERS TO COUNT WHEN THE CPU HAS PI LEVELS IN PROGRESS AND/OR TO ACCOUNT IN EXEC MODE WITH NO PI'S IN PROGRESS.

NOTE THAT PI 0 AND PI CYCLES (E.G., THE INSTRUCTION AT 40+2N) ARE NOT INCLUDED.

AS FAR AS THE ACCOUNTING METERS ARE CONCERNED, THE CPU IS IN ONE OF THE FOLLOWING 3 STATES (X,Y,Z):

PI IN PROGRESS	EXEC MODE	USER MODE
1-7	Y	X
NONE	Z	X

THE ACCOUNTING METERS COUNT THE STATES MARKED X WHENEVER THEY ARE TURNED ON. THE SOFTWARE CAN SELECTIVELY ENABLE FOR Y AND/OR Z BY A CONO MTR, DATAO PAG, [LUBR] (BIT 2 = 1) LOADS THE USER BASE REGISTER, AND UPDATES THE ACCOUNTING CLOCKS IN THE OLD UPT, CLEARING THE HARDWARE COUNTERS. THUS THE CONTEXT SWITCH IS PERFORMED WITH ONE INSTRUCTION. WHEN THE SYSTEM IS FIRST STARTED UP, THE SOFTWARE MUST SET INHIBIT STORING ACCOUNTING METER (DATAO PAG, ISAM)(BIT 18 = 1) ON THE FIRST DATAO PAG WHICH LOADS THE UBR, SO THAT EXEC LOCATIONS UPTEBM, UPTEBM+1, UPTMRM, UPTMRM+1 (504-507) DO NOT GET WRITTEN INTO.

9.2 ENABLING THE ACCOUNTING METERS

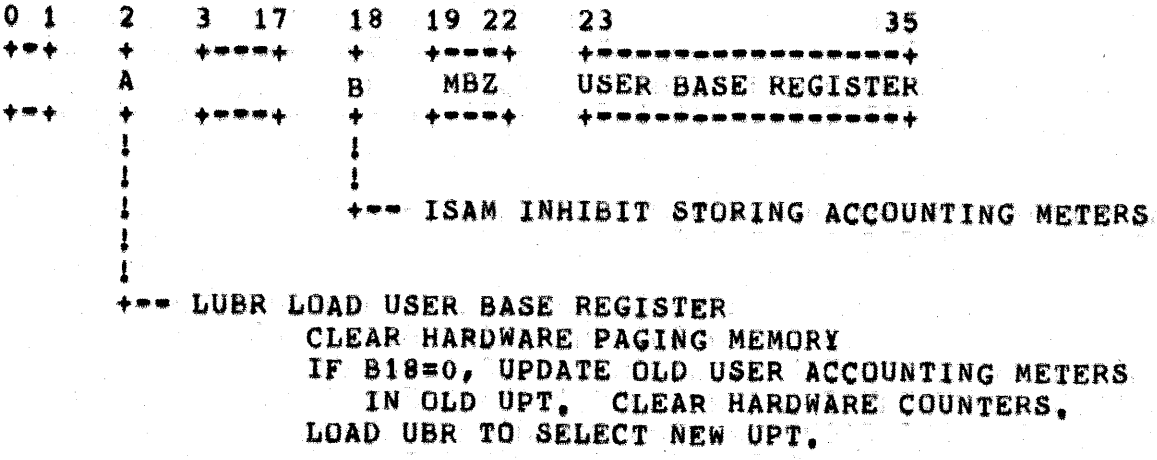
SEE SECTION 5 WHERE CONO MTR, IS DESCRIBED.

9.3 SET (AND SAVE) ACCOUNTING METERS

THE 16-BIT REGISTER PARTS OF BOTH OF THE ACCOUNTING METERS ARE SAVED FOR THE OLD USER IN HIS UPT AND CLEARED FOR THE NEW USER AS PART OF THE CONTEXT SWITCH INSTRUCTION, DATAO PAG, E. THE OLD CONTENTS OF THE 16-BIT REGISTERS ARE SAVED BY ADDING THEM TO THE APPROPRIATE DOUBLE WORD IN THE UPT AND STORING THE RESULT BACK. THE EBOX CLOCK TICK METER IS SAVED IN UPTEBM (USER PROCESS TABLE - EBOX BUSY METER = 504) AND UPTEBM+1. THE MBOX REFERENCE METER IS SAVED IN UPTMRM (USER PROCESS TABLE - MBOX REFERENCE METER = 506) AND UPTMRM+1.

```
        DATAO   PAG,E
```

(SEE CHAPTER 2.6, INTERNAL DEVICES (APR, PI, PAG) FOR REST OF SPECIFICATION FOR DATAO PAG,,)



9.4 DETAILED DESCRIPTION OF ACCOUNTING METERS (DATAO PAG,E)

BIT	SYMBOL	DESCRIPTION
0-1		SEE CHAPTER 2.6, INTERNAL DEVICES.
2	LUBR	DATAO: LOAD USER BASE REGISTER - IF 1 PERFORMS A COMPLETE CONTEXT SWITCH OF THE FOLLOWING QUANTITIES: 1. STORE OLD USER ACCOUNTING METERS IN OLD UPT LOCATIONS UPTEBM, UPTEBM+1, UPTMRM,

UPTMRM+1 (UNLESS INHIBITED BY ISAM)

2. LOAD USER BASE REGISTER (ADDRESS OF UPT)

3. CLEAR HARDWARE PAGE TABLE

4. CLEAR HARDWARE ACCOUNTING METERS FOR NEW USER.

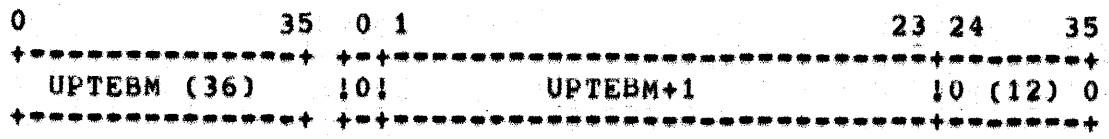
IF 0, NONE OF THE ABOVE CONTEXT SWITCH FUNCTIONS ARE PERFORMED.

3-17		SEE CHAPTER 2.6, INTERNAL DEVICES
18	ISAM	DATAO: INHIBIT STORING ACCOUNTING METERS - IF 1, DO NOT STORE THE ACCOUNTING METERS IN THE OLD UPT. THIS FUNCTION IS PROVIDED MAINLY FOR SYSTEM STARTING BEFORE THERE IS AN OLD UPT.
19-35		SEE CHAPTER 2.6, INTERVAL DEVICES

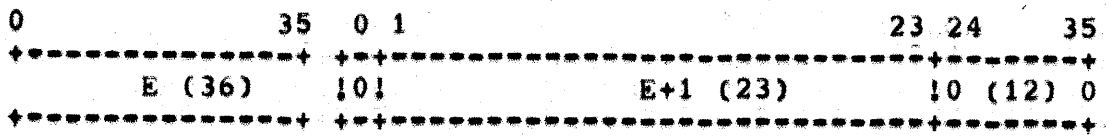
9,5 READ EBOX ACCOUNT

```
        RDEACT E
```

READ 59-BIT EBOX BUSY METER INTO E AND E+1 AS A DOUBLE PRECISION UNSIGNED INTEGER WITH BINARY POINT BETWEEN BITS 23 AND 24 OF THE SECOND WORD.



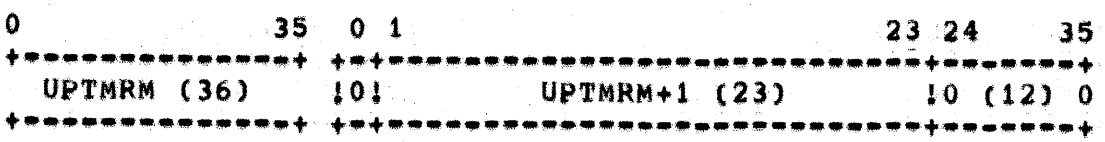
YIELDS:



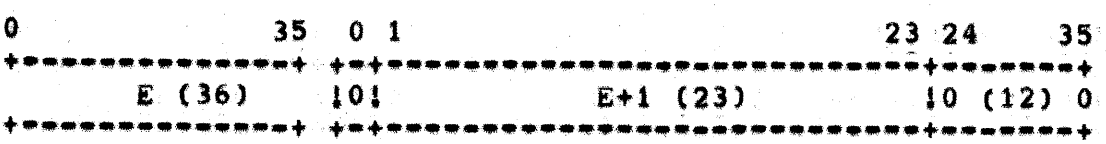
9,6 READ MBOX ACCOUNT

```
        RDMACT E
```

READ 59-BIT MBOX REFERENCE METER INTO E AND E+1 AS A DOUBLE PRECISION UNSIGNED INTEGER WITH BINARY POINT BETWEEN BITS 23 AND 24 OF THE SECOND WORD.



YIELDS



10. PERFORMANCE ANALYSIS COUNTER

10.1 PERFORMANCE ANALYSIS COUNTER SUMMARY

THE PERFORMANCE ANALYSIS COUNTER PROVIDES A TOOL FOR STUDYING HARDWARE AND SOFTWARE PERFORMANCE OF THE SYSTEM. IT WILL POINT TO HARDWARE AND/OR SOFTWARE BOTTLENECKS. IT WILL HELP THE CUSTOMER DECIDE WHAT PIECE OF EQUIPMENT TO ORDER NEXT AS HIS LOAD
INCREASES. REPORTS FROM THE FIELD WILL HELP HARDWARE ENGINEERS DECIDE WHAT NEW COMPONENTS TO DESIGN. OTHER REPORTS FROM THE FIELD
WILL HELP SOFTWARE ENGINEERS IMPROVE THE PERFORMANCE OF FUTURE SOFTWARE RELEASES. A RDPAC E (READ PERFORMANCE ANALYSIS CLOCKS -
BLKI TIM,E) READS THE 60-BIT
PERFORMANCE ANALYSIS COUNTER INTO E AND E + 1. THE 60-BIT QUANTITY
AT LAST 0 IS KEPT IN EPT LOCATIONS EPTPAC (EXEC PROCESS TABLE - PERFORMANCE ANALYSIS COUNTER = 512) AND EPTPAC+1.

THE PERFORMANCE ANALYSIS COUNTER HAS TWO MODES, DURATION MODE AND EVENT MODE. TWENTY-THREE BOOLEAN STATE SIGNAL LINES OF THE MACHINE ARE BROUGHT TOGETHER AND INPUT INTO A PROGRAMMABLE BOOLEAN EXPRESSION. EACH STATE SIGNAL IS EITHER TRUE OR FALSE. IN DURATION MODE, THE COUNTER COUNTS AT HALF OF THE BASIC MACHINE CLOCK RATE WHENEVER THE PROGRAMMABLE BOOLEAN
EXPRESSION ENABLE CONDITION IS TRUE. IN EVENT MODE THE PERFORMANCE ANALYSIS COUNTER INCREMENTS EVERY TIME THE BOOLEAN EXPRESSION CHANGES FROM FALSE TO TRUE. THUS GREAT CARE MUST BE USED IN INTERPRETING EVENT COUNTS WHERE MORE THAN ONE STATE SIGNAL
IS ENABLED, SINCE TWO OVERLAPPING STATE SIGNALS WILL COUNT ONCE INSTEAD OF TWICE. EVENT MODE IS INDICATED BY THE EVENT MODE (EM) BIT IN C(LOC) IN WRPAE LOC (WRITE PERFORMANCE ANALYSIS ENABLES
- BLKO TIM,LOC) A 1 MEANS EVENT MODE, A
0 MEANS DURATION MODE. THE PERFORMANCE ANALYSIS COUNTER CAN BE CLEARED USING CLEAR PERFORMANCE ANALYSIS CLOCK BIT (WRPAE [CPAC]).

THE FOLLOWING 23 STATE SIGNALS EXIST:

1. USER MODE (1)
2. PI LEVEL N (8)
3. CACHE REFILL IN PROGRESS (1)
4. CACHE WRITE BACK IN PROGRESS FOR EBOX (1)
5. SWEEP WRITE BACK IN PROGRESS (1)
6. EBOX TO MBOX REQUEST IN PROGRESS (1)
7. MICRO-CODE EVENT IN PROGRESS (1)
8. CHANNEL REGISTER J FULL (BUSY) (8)
9. ECL PROBE INPUT EVENT (1)

TWO MICRO-CODE SPECIAL FUNCTIONS ALLOW THE MICRO-CODE TO DEFINE ONE ARBITRARY STATE. ONE OF THE MICRO-CODE FUNCTIONS SETS THE MICRO-CODE STATE LINE TRUE AND THE OTHER SETS THE MICRO-CODE STATE LINE FALSE. THE PERFORMANCE ANALYSIS COUNTER CAN BE TURNED OFF BY DISABLING TERMS OF THE PERFORMANCE ALALYSIS EXPRESSION WITH WRPAE LOC. THE RECOMMENDED PROCEDURE IS WRPAE[0]. IT IS INTENDED THAT THE SOFTWARE WILL DYNAMICALLY PATCH THE MONITOR VIA A PRIVILEGED MONITOR CALL IN ORDER TO DEFINE A PARTICULAR SOFTWARE STATE SUCH AS NULL JOB RUNNING.

THE PERFORMANCE ANALYSIS COUNTER COUNTS WHEN THE FOLLOWING BOOLEAN EXPRESSION CHANGES TO TRUE (EVENT MODE) OR COUNTS AT HALF OF THE BASIC MACHINE CLOCK RATE WHEN THE FOLLOWING BOOLEAN EXPRESSION IS TRUE. NOTE THAT THE BOOLEAN EXPRESSION IS MADE UP OF THE AND OF 7 BOOLEAN SUBEXPRESSIONS. THUS EACH SUBEXPRESSION MUST BE TRUE IN ORDER FOR THE ENTIRE EXPRESSION TO BE TRUE. THE SOFTWARE INDICATES WHICH OF THE 23 STATE LINES IT IS INTERESTED IN BY SETTING A CORRESPONDING BIT IN A WRPAE LOC. IF THE MEASUREMENT BEING MADE DOES NOT INVOLVE ONE OR MORE OF THE SUBEXPRESSIONS, THE SOFTWARE MUST ENSURE THAT THE SUBEXPRESSION IS ALWAYS TRUE BY SETTING ALL OF THE BITS IN THE SUBEXPRESSION OR AT LEAST THE DON'T CARE BIT IN THE SUBEXPRESSION. A SOFTWARE DON'T CARE BIT IS NOT PROVIDED SINCE THE SOFTWARE STATE SIGNAL IS PROVIDED BY THE SOFTWARE ALTERNATELY TURNING THE PERFORMANCE ANALYSIS CLOCK ON AND OFF. WHEN CHANGING THE PATCHES, THE MONITORING PROGRAM SHOULD FIRST DISABLE THEPERFORMANCE ANALYSIS CLOCK BY DOING A BLKO TIM,[0].

(EXEC MODE OR USER MODE OR DONT CARE)

AND

(PI 0 OR PI 1 OR... OR PI 7 OR NO PIS IN PROGRESS)

AND

(CACHE MISS IN PROGRESS OR EBOX WRITE BACK IN PROGRESS OR EBOX TO MBOX REQUEST IN PROGRESS OR SWEEP WRITEBACK IN PROGRESS OR CACHE DON'T CARE)

AND

(CHANNEL 0 COMMAND REGISTER FULL OR...CHANNEL 7 COMMAND REGISTER FULL OR CHANNEL DON'T CARE)

AND

(MICRO-CODE STATE OR MICRO-CODE STATE DON'T CARE

AND

(ECL PROBE INPUT STATE HIGH OR ECL PROBE INPUT STATE LOW OR PROBE DON'T CARE

10.2 PERFORMANCE ANALYSIS APPLICATIONS - EVENT MODE

THE PERFORMANCE ANALYSIS COUNTER WILL ALLOW PERFORMANCE ANALYSIS SOFTWARE TO COLLECT COUNTS OF THE FOLLOWING SYSTEM EVENTS. NOTE: SINCE THERE IS ONLY ONE COUNTER, ONLY ONE OF THE FOLLOWING EVENTS CAN BE COUNTED ON A DATA COLLECTION EXPERIMENT:

1. NO. OF TRANSITIONS TO EXEC MODE
2. NO. OF (UNNESTED) PI INTERRUPTS ON ANY CHANNEL
3. NO. OF INTERRUPTS ON PI CHANNEL N.
4. NO. OF CACHE MISSES
5. NO. OF CACHE WRITE BACKS
6. NO. OF CACHE MISSES AND IN USER MODE
7. NO. OF EBOX TO MBOX REQUESTS
8. NO. OF CACHE WRITE BACKS AND IN USER MODE
9. NO. OF TIMES A PARTICULAR INSTRUCTION OR SET OF INSTRUCTIONS IS USED (REQUIRES MODIFIED MICRO-CODE)
10. NO. OF TIMES A SOFTWARE EVENT OCCURS (REQUIRES THE MONITOR TO BE PATCHED)
11. ETC.

10.3 PERFORMANCE ANALYSIS APPLICATIONS - DURATION MODE

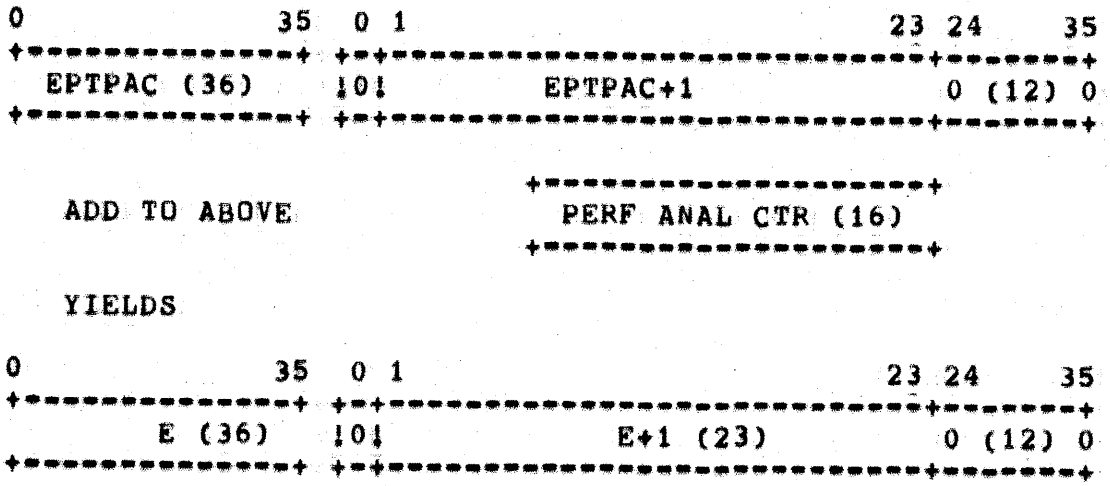
THE PERFORMANCE ANALYSIS COUNTER WILL ALLOW PERFORMANCE ANALYSIS SOFTWARE TO MEASURE THE PERCENT OF ELAPSED TIME THE SYSTEM SPENDS IN THE FOLLOWING STATES. NOTE: SINCE THERE IS ONLY ONE COUNTER, ONLY ONE OF THE FOLLOWING STATES CAN BE COUNTED ON A DATA COLLECTION EXPERIMENT. HOWEVER STATES, UNLIKE EVENTS, CAN BE THE AND OF SEVERAL OF THE 6 SUBEXPRESSIONS:

1. % TIME IN USER MODE
2. % TIME IN EXEC MODE
3. % TIME AT ANY PI LEVEL
4. % TIME AT PI LEVEL
5. % TIME DOING CACHE MISSES
6. % TIME DOING CACHE WRITE BACKS
7. % TIME EBOX TO MBOX REQUEST IN PROGRESS
8. % TIME IN NULL JOB
9. % TIME IN NULL JOB AND ANY CHANNEL BUSY
10. % TIME IN NULL JOB AND CHANNEL N BUSY
11. % TIME DOING CACHE MISSES IN EXEC MODE
12. % TIME AT PI 0 (DTE20 BYTE TRANSFERS OR EXAMINE/DEPOSIT)
13. % TIME DOING STRING INSTRUCTION (REQUIRES MODIFIED MICRO-CODE)
14. % TIME WITH PROBE INPUT HIGH
15. ETC.

10,4 READ PERFORMANCE ANALYSIS COUNTER

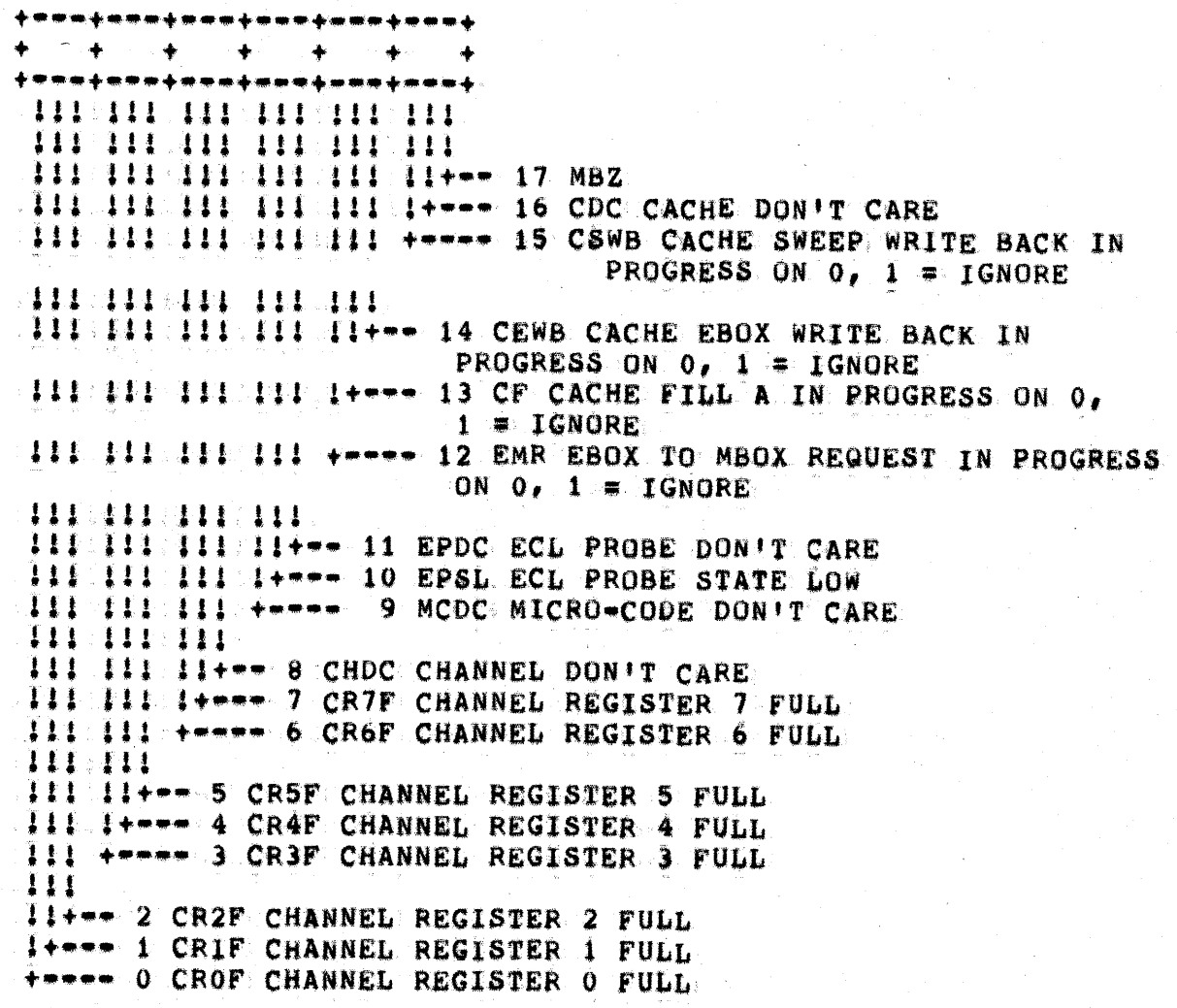
RDPERF E

READS 59-BIT PERFORMANCE ANALYSIS COUNTER INTO E AND E+1 AS A DOUBLE PRECISION SIGNED (POSITIVE) INTEGER WITH BINARY POINT BETWEEN BITS 23 AND 24 OF THE SECOND WORD.



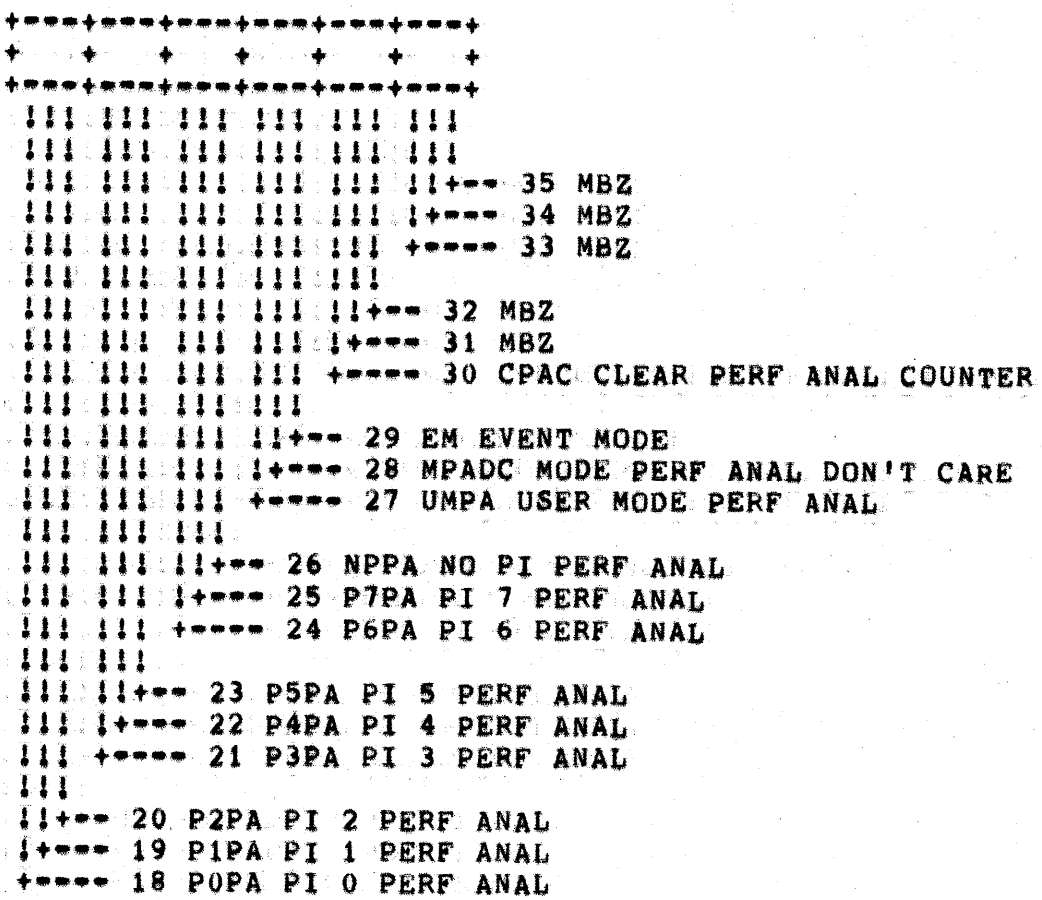
10.5 WRITE PERFORMANCE ANALYSIS ENABLES

WRPAE E (LH)



NOTE: THE SENSE OF CSWB, CEWB, CF, EMR BITS ARE REVERSED FROM THE OTHER BITS. FOR THESE BITS, A 1 (RATHER THAN A 0) MEANS IGNORE AND A 1 (RATHER THAN 0) MEANS INCLUDE THIS TERM.

WRPAE E (RH)


35 MBZ
34 MBZ
33 MBZ
32 MBZ
31 MBZ
30 CPAC CLEAR PERF ANAL COUNTER
29 EM EVENT MODE
28 MPADC MODE PERF ANAL DON'T CARE
27 UMPA USER MODE PERF ANAL
26 NPPA NO PI PERF ANAL
25 P7PA PI 7 PERF ANAL
24 P6PA PI 6 PERF ANAL
23 P5PA PI 5 PERF ANAL
22 P4PA PI 4 PERF ANAL
21 P3PA PI 3 PERF ANAL
20 P2PA PI 2 PERF ANAL
19 P1PA PI 1 PERF ANAL
18 P0PA PI 0 PERF ANAL

10,6 DETAILED DESCRIPTION OF PERFORMANCE ANALYSIS METER WRPAE LOC (BLKO TIM,LOC)

BIT SYMBOL DESCRIPTION

BITS 0-8 COMPRISE THE CHANNEL TERM. AT LEAST ONE BIT IN 0-8 MUST BE SET IN ORDER TO COUNT ANYTHING. CHANNEL REGISTER FULL IS DECODED FROM THE FOLLOWING SIGNALS: CRC SEL 4, CRC SEL 2, CRC SEL 1, CBUS READY.

BIT	SYMBOL	DESCRIPTION
0	CR0F	BLKO: CHANNEL REGISTER 0 FULL - IF A 1, THE PERFORMANCE ANALYSIS COUNTER WILL COUNT WHEN CHANNEL 0 IS BUSY (WAITING FOR LATENCY OR TRANSFERRING DATA) AND OTHER BOOLEAN TERMS ARE TRUE. IF A 0, THE STATE OF CHANNEL IS IGNORED.
1	CR1F	BLKO: CHANNEL REGISTER 1 FULL - IF A 1, THE PERFORMANCE ANALYSIS COUNTER WILL COUNT WHEN CHANNEL 1 IS BUSY (WAITING FOR LATENCY OR TRANSFERRING DATA) AND THE OTHER BOOLEAN TERMS ARE TRUE.
2	CR2F	BLKO: CHANNEL REGISTER 2 FULL - IF A 2, THE PERFORMANCE ANALYSIS COUNTER WILL COUNT WHEN CHANNEL 2 IS BUSY (WAITING FOR LATENCY OR TRANSFERRING DATA) AND THE OTHER BOOLEAN TERMS ARE TRUE.
3	CR3F	BLKO: CHANNEL REGISTER 3 FULL - IF A 3, THE PERFORMANCE ANALYSIS COUNTER WILL COUNT WHEN CHANNEL 3 IS BUSY (WAITING FOR LATENCY OR TRANSFERRING DATA) AND THE OTHER BOOLEAN TERMS ARE TRUE.
4	CR4F	BLKO: CHANNEL REGISTER 4 FULL - IF A 4, THE PERFORMANCE ANALYSIS COUNTER WILL COUNT WHEN CHANNEL 4 IS BUSY (WAITING FOR LATENCY OR TRANSFERRING DATA) AND THE OTHER BOOLEAN TERMS ARE TRUE.

5	CR5F	BLKO: CHANNEL REGISTER 5 FULL - IF A 5, THE PERFORMANCE ANALYSIS COUNTER WILL COUNT WHEN CHANNEL 5 IS BUSY (WAITING FOR LATENCY OR TRANSFERRING DATA) AND THE OTHER BOOLEAN TERMS ARE TRUE.
6	CR6F	BLKO: CHANNEL REGISTER 6 FULL - IF A 6, THE PERFORMANCE ANALYSIS COUNTER WILL COUNT WHEN CHANNEL 6 IS BUSY (WAITING FOR LATENCY OR TRANSFERRING DATA) AND THE OTHER BOOLEAN TERMS ARE TRUE.
7	CR7F	BLKO: CHANNEL REGISTER 7 FULL - IF A 7, THE PERFORMANCE ANALYSIS COUNTER WILL COUNT WHEN CHANNEL 7 IS BUSY (WAITING FOR LATENCY OR TRANSFERRING DATA) AND THE OTHER BOOLEAN TERMS ARE TRUE.
8	CHDC	BLKO: CHANNEL DON'T CARE - IF A 1, THE PERFORMANCE ANALYSIS COUNTER CONSIDERS THE CHANNEL TERM TO BE ALWAYS TRUE AND WILL COUNT NO MATTER WHAT THE STATE THE CHANNELS ARE IN IF ALL THE OTHER BOOLEAN TERMS ARE TRUE. HENCE THE NAME CHANNEL DON'T CARE. IF A 0, BITS 0-7 SPECIFY WHICH CHANNELS ARE OF INTEREST.

BIT 9 IS THE MICRO CODE STATE TERM. IT IS THE ONLY TERM WHICH CAN BE MADE ALWAYS TRUE WITHOUT ANY BITS.

9	MCDC	BLKO: MICRO CODE DON'T CARE - IF A 0, THE PERFORMANCE ANALYSIS COUNTER WILL COUNT WHEN THE MICRO CODE HAS SET THE MICRO-CODE STATE SIGNAL TIME AND THE OTHER BOOLEAN TERMS ARE TRUE. [THE INTERNAL SIGNAL USED IS: CON UCODE STATE 01] IF A 1, THE HARDWARE TREATS THE MICRO-CODE STATE TERM AS IF IT WERE ALWAYS TRUE.

BITS 10-11 ARE THE ECL PROBE STATE TERM.

10	EPSL	BLKO: ECL PROBE STATE LOW - IF A 1, THE PERFORMANCE ANALYSIS COUNTER WILL COUNT WHEN THE ECL PROBE INPUT SIGNAL IS LOW AND THE OTHER BOOLEAN TERMS ARE TRUE. [THE INTERNAL SIGNAL USED IS: PROBE] IF A 0, THE PERFORMANCE ANALYSIS COUNTER WILL COUNT WHEN THE ECL PROBE INPUT SIGNAL IS HIGH AND THE OTHER BOOLEAN TERMS ARE TRUE.
11	EPDC	BLKO: ECL PROBE STATE DON'T CARE - IF A 1, THE PERFORMANCE ANALYSIS COUNTER WILL COUNT WHENEVER THE STATE OF THE OTHER BOOLEAN TERMS IS TRUE. IF A 0, THE PERFORMANCE ANALYSIS COUNTER WILL DEPEND ON THE ECL PROBE AND EPSL.

BITS 12-16 ARE THE MBOX STATE TERM. EXCEPT FOR DON'T CARE, ALL THESE BITS ARE CONDITIONAL ON EBOX WAITING FOR MBOX.

NOTE: THE SENSE EMR, CF, CEWB, AND CSWB BITS ARE REVERSED FROM THE OTHER BITS. FOR THESE BITS, A 1 (RATHER THAN 0) MEANS IGNORE AND 0 (RATHER THAN 1) MEANS INCLUDE THIS TERM.

12	EMR	BLKO: EBOX TO MBOX REQUEST IN PROGRESS - IF A 0, THE PERFORMANCE ANALYSIS COUNTER WILL COUNT WHEN THE EBOX IS WAITING FOR A CYCLE REQUESTED FROM THE MBOX AND THE OTHER BOOLEAN TERMS ARE TRUE. THIS INCLUDES TIME WHEN THE EBOX IS WAITING FOR THE MBOX IT DOES NOT INCLUDE ADDITIONAL READS WHICH THE MBOX DOES IN ORDER TO GET 4 WORDS. IT DOES INCLUDE ANY WRITE-BACK WHICH THE MBOX MAY HAVE TO DO IN ORDER TO SATISFY THE EBOX REQUEST. IT DOES NOT INCLUDE WRITE-BACKS DUE TO SWEEP OPERATIONS. [THE INTERNAL SIGNAL USED IS: EBOX SYNC AND MB WAIT] IF A 1, THE STATE OF EBOX TO MBOX REQUEST IN PROGRESS IS IGNORED.

13	CF	BLKO: CACHE FILL IN PROGRESS - IF A 0, THE PERFORMANCE ANALYSIS COUNTER WILL COUNT WHEN THE EBOX MAKES A REFERENCE TO THE MBOX AND THE DATA MUST BE FETCHED FROM MEMORY AND THE OTHER BOOLEAN TERMS ARE TRUE. THIS INCLUDES DTE20 EBOX REFERENCES BUT DOES NOT INCLUDE CHANNEL REQUESTS. IT ALSO INCLUDES REFERENCES TO MEMORY BECAUSE THE C BIT IN THE PAGE TABLE IS OFF (OPERATING SYSTEM HAS SAID NOT TO CACHE THAT PAGE). IT DOES NOT INCLUDE ANY TIME SPENT DOING WRITE-BACKS. IT DOES NOT INCLUDE CACHE SWEEP OPERATIONS. IT INCLUDES ALL MEMORY READ REFERENCES TO PAGE MAPS WHETHER MADE BY THE EBOX OR THE MBOX. [THE INTERNAL SIGNAL USED IS: CSH FILL CACHE RD] IF A 1, THE STATE OF CACHE FILL IN PROGRESS IS IGNORED.
14	CEWB	BLKO: CACHE EBOX WRITE-BACK IN PROGRESS - IF A 0, THE PERFORMANCE ANALYSIS COUNTER WILL COUNT WHEN THE MBOX MUST PERFORM A WRITE-BACK TO MEMORY AS THE RESULT OF AN EBOX REQUEST. THIS INCLUDES DTE20 REFERENCES BUT DOES NOT INCLUDE CHANNEL REFERENCES. IT ALSO INCLUDES WRITES TO MEMORY BECAUSE THE C BIT IN THE PAGE TABLE IS OFF (OPERATING SYSTEM HAS SAID NOT TO CACHE THAT PAGE.) IT DOES NOT INCLUDE CACHE SWEEP OPERATIONS. IT INCLUDES ALL MEMORY REFERENCES TO PAGE MAPS WHETHER MADE BY THE EBOX OR MBOX. THUS CEWB IS DISJOINT FROM CL. [THE INTERNAL SIGNAL USED IS: CSH E WRITEBACK]. IF A 1, THE STATE OF CACHE EBOX WRITE-BACK IN PROGRESS IS IGNORED.
15	CSWB	BLKO: CACHE SWEEP WRITE BACK IN PROGRESS. IF A 0, THE PERFORMANCE ANALYSIS COUNTER WILL COUNT WHEN THE MBOX IS PERFORMING WRITE-BACKS DUE TO CACHE SWEEPS. THIS DOES NOT INCLUDE WRITE-BACKS DUE TO ANY OTHER CAUSE. [THE INTERNAL SIGNAL USED IS: CSH CCA WRITEBACK]. IF A 1, THE STATE OF CACHE SWEEP WRITE-BACK IN PROGRESS IS IGNORED.

16	CDC	BLKO: CACHE DON'T CARE - IF A 1, THE PERFORMANCE ANALYSIS COUNTER CONSIDERS THE CACHE TERM TO BE ALWAYS TRUE AND WILL COUNT NO MATTER WHAT THE CACHE IS DOING IF ALL THE OTHER BOOLEAN TERMS ARE TRUE. HENCE THE NAME CACHE DON'T CARE. IF A 0, BITS 14-16 SPECIFY WHICH CACHE STATE SIGNALS ARE OF INTEREST.
17		MBZ MUST BE ZERO.

BITS 18-26 COMPRISE THE PI LEVEL TERM. AT LEAST 1 BIT MUST BE SET IN ORDER TO COUNT ANYTHING. THE MODE (EXEC/USER) DOES NOT MATTER FOR THIS TERM TO BE TRUE. SINCE THE MODE (EXEC/USER) IS A SEPARATE TERM IN THE BOOLEAN EXPRESSION, THE SOFTWARE CAN ENABLE FOR ALL POSSIBLE COMBINATIONS OF MODE AND PI LEVEL. (DIFFERENT FROM ACCOUNTING METER WHICH ALWAYS COUNTS IN USER MODE).

INTERNAL SIGNALS USED: THE CURRENT LEVEL OF THE PROCESSOR IS DETERMINED FROM 7 SIGNALS. IF MCL PI CYCLE IS TRUE, THEN PI2 PI4 A, PI2 PI2 A, AND PI2 PI1A REPRESENT THE CURRENT PI LEVEL. IF MCL PI CYCLE IS FALSE, THEN PI2 HOLD 4, PI2 HOLD2, AND PI2 HOLD1 REPRESENT THE CURRENT STATE.

18	P0PA	BLKO: PI 0 PERF ANAL - IF A 1, THE PERFORMANCE ANALYSIS COUNTER WILL COUNT WHENEVER THE CPU IS AT PI LEVEL 0 (DTE20 EXAMINE, DEPOSIT, OR BYTE TRANSFER) AND THE OTHER BOOLEAN TERMS ARE TRUE. IF A 0, PI LEVEL 0 IS IGNORED.
19	P1PA	BLKO: PI 1 PERF ANAL - IF A 1, THE PERFORMANCE ANALYSIS COUNTER WILL COUNT WHENEVER THE CPU IS AT PI LEVEL 1 AND THE OTHER BOOLEAN TERMS ARE TRUE. IF A 0, PI LEVEL 1 IS IGNORED.

20	P2PA	BLKO: PI2 - PERF ANAL
21	P3PA	BLKO: PI3 - PERF ANAL
22	P4PA	BLKO: PI4 - PERF ANAL
23	P5PA	BLKO: PI5 - PERF ANAL
24	P6PA	BLKO: PI6 - PERF ANAL
25	P7PA	BLKO: PI7 - PERF ANAL
26	NPPA	BLKO: NO PI PERF ANAL - IF A 1, THE PERFORMANCE ANALYSIS COUNTER WILL COUNT WHENEVER THE CPU HAS NO PIS IN PROGRESS AND THE OTHER BOOLEAN TERMS ARE TRUE. IF A 0, THE STATE OF NO PIS IN PROGRESS IS IGNORED.

BITS 27-28 COMPRISE THE CPU MODE TERM.

27	UMPA	BLKO: USER MODE PERFORMANCE ANALYSIS - IF A 1, THE PERFORMANCE ANALYSIS COUNTER WILL COUNT WHENEVER THE CPU IS IN USER MODE AND THE OTHER BOOLEAN TERMS ARE TRUE. [THE INTERNAL SIGNAL USED IS: SCD USER A]. IF A 0, THE COUNTER WILL COUNT WHENEVER THE CPU IS IN EXEC MODE AND THE OTHER BOOLEAN TERMS ARE TRUE.
28	MPADC	BLKO: MODE PERF ANAL DONT CARE - IF 1, PERFORMANCE ANALYSIS DOES NOT DEPEND ON PROCESSOR MODE.

29	EM	BLKO: EVENT MODE - IF A 1, PUT THE PERFORMANCE ANALYSIS COUNTER INTO EVENT MODE. IN EVENT MODE THE COUNTER COUNTS ON EVERY 0 TO 1 TRANSITION OF THE AND OF ALL OF THE SIGNALS CONDITIONED BY THE BOOLEAN EXPRESSION, SEE SUMMARY ABOVE (SECTION 10,1). IF A 0, BUT THE PERFORMANCE ANALYSIS COUNTER INTO DURATION MODE. IN DURATION MODE THE COUNTER IS COUNTED AT HALF OF THE BASIC MACHINE CLOCK RATE WHENEVER THE SIGNALS CONDITIONED BY THE BOOLEAN EXPRESSION ARE TRUE. NOTE: IN ORDER TO PREVENT THE COUNTER FROM CHANGING WHILE IT IS BEING READ, THE STATE IS FORCED FALSE. THIS HAS THE EFFECT OF CREATING A FALSE EVENT IF THE COUNTER IS IN EVENT MODE AND THE STATE IS TRUE DURING A READ OF THE COUNTER; AND IN DURATION MODE, NO COUNTS ARE TAKEN DURING THE TIME THE COUNTER IS READ.
30	CPAC	BLKO: CLEAR PERFORMANCE ANALYSIS COUNTER - IF A 1, CLEAR THE 16-BIT REGISTER OF THE PERFORMANCE ANALYSIS COUNTER. THE SOFTWARE MUST CLEAR THE EPT LOCS EPTPAC, EPTPAC+1 (512-513). IF A 0, THE COUNTER IS NOT CLEARED. THERE IS NO WAY TO LOAD THE 16-BIT REGISTER OF THE PERFORMANCE ANALYSIS COUNTER.
31-35		BLKO: MBZ

NOTE: THERE IS NO WAY TO READ BACK THE PERFORMANCE ANALYSIS COUNTER ENABLES.

11. RESET

EITHER KL10 RESET OR POWER APPLICATION (POWER "ON") TURNS OFF KL10 CLOCKS DESCRIBED HEREIN, CLEARS THE PI ASSIGNMENT IN MTR.

12. SUMMARY OF REGISTER USE

TWO INTERNAL DEVICE CODES ARE USED, NAMELY TIM AND MTR. THE FOLLOWING TABLE SHOWS THE USE OF THEM. TIM IS USED FOR INTERVAL TIMER, TIME BASE, AND PERFORMANCE ANALYSIS COUNTER. MTR IS USED FOR THE ACCOUNTING METER AND (BECAUSE RUN OUT OF BITS) THE CONTROL BITS FOR ALL THE CLOCKS.

CONO	TIM	SET INTERVAL TIMER PERIOD (12 BITS)
CONI	TIM	READ INTERVAL TIMER (12 BITS), PERIOD (12 BITS), AND INTERVAL TIMER FLAGS
DATAO	TIM	UNUSED
RDTIME		READ TIME BASE (59 BITS)
WRPAE		SET PERFORMANCE ANALYSIS COUNTER ENABLES (30 BITS)
RDPERF		READ PERFORMANCE ANALYSIS COUNTER (59 BITS)
CONO	MTR	SET KL10 CLOCKS CONTROL BITS
CONI	MTR	READ KL10 CLOCKS CONTROL BITS (7 BITS)
DATAO	MTR	UNUSED
RDEACT		READ EBOX BUSY METER (59 BITS)
BLKO	MTR	UNUSED
RDMACT		READ MBOX REFERENCE METER (59 BITS)
DATAO	PAG	STORE ACCOUNTING METERS CLEAR ACCOUNTING METERS

13. MASTER CLOCK ACCURACY (WORST CASE)

THE FREQUENCY TOLERANCE SPECIFICATION OVER TEMPERATURE, INCLUDING LONG TERM DRIFT FOR THE KL10 MASTER CLOCK OSCILLATOR, HAS BEEN SET AT PLUS OR MINUS 25 PARTS PER MILLION MAXIMUM FOR THE FIRST YEAR, INCREASING BY 5 PARTS PER MILLION PER YEAR TO 50 PARTS PER MILLION MAXIMUM FOR THE SIXTH YEAR. IN TERMS OF KEEPING ACCURACY, THIS IS EQUIVALENT TO 16 SECONDS PER WEEK (GAIN OR LOSS) FOR THE FIRST YEAR AND LESS THAN 31 SECONDS PER WEEK FOR THE SIXTH YEAR. THIS PERFORMANCE IS OBTAINED AT LOW COST, WITHOUT ANY FREQUENCY ADJUSTMENTS, AFTER INITIAL PRODUCTION CHECKOUT.

14. APPENDIX A - IMPACT ON REST OF MACHINE

1. INTEGER MHZ.

2. 5 LINES FROM CHANNELS - REGISTER FULL

3. 3 LINES FROM CACHE - CACHE FILL E WRITE BACK, CA WRITE BACK

15. APPENDIX B - LOOSE ENDS

1. HOW SHOULD ACCOUNTING AND PERFORMANCE ANALYSIS HANDLE MBOX REFERENCES MADE BY EBOX FOR PAGING?

16. APPENDIX C - EPT AND UPT SYMBOLS

THIS SPECIFICATION USES THE FOLLOWING EPT AND UPT LOCATIONS. AS A HELP TO THE READER, THE VALUES OF THE SYMBOLS ARE ALSO GIVEN PARENTHETICCALLY IN THE BODY OF THE SPEC. HOWEVER THE VALUES FOR THE SMBOLS MAY CHANGE. THE FINAL AUTHORITY IS LATER EDITIONS OF CHAPTER 2,0, EXEC AND USER PROCESS TABLES (EPT, UPT).

LOC	SYMBOL	NAME
510	EPTTB	EPT - TIME BASE (HIGH)
511	EPTTB+1	(LOW)
512	EPTPAC	EPT - PERFORMANCE ANALYSIS COUNTER (HIGH)
513	EPTPAC+1	(LOW)
514	EPTITV	EPT - INTERVAL TIMER VECTOR
504	UPTEBM	UPT - EBOX BUSY METER (HIGH
505	UPTEBM+1	(LOW)
506	UPTMRM	UPT - MBOX REFERENCE METER (HIGH)
507	UPTMRM+1	(LOW)

[END OF CH2S07,SPC]

1080,2040,2060 ENGINEERING FUNCTIONAL SPEC - CHAP 2.8

TO: KL10 LIST, J. PARSLOW (200 FILE)

TITLE: KL10 PAGING - REV 2

STATUS: FINISHED

FILE: [EFS]CH2S08.SPC

PDM #: 200-200-037-02

DATE: 5 DEC 75

SUPERSEDED MEMOS: KL10 PAGING PROPOSAL, D. MURPHY, 5 JUNE 73
NEW PAGING DESIGN, D. MURPHY, 31 MAY 74

SUPERSEDED SPECS: NONE

ENGINEER: D. MURPHY, J. LEONARD

APPROVED:

EDITOR: D. MURPHY

TYPIST: J. MCCARTHY

REVIEWED: 6 AUG 75

DISTRIBUTED:

ABSTRACT

THIS CHAPTER DESCRIBES THE KL10 PAGING FACILITIES. THESE FACILITIES ARE A CONSIDERABLE ADVANCE OVER KI10 PAGING, ALTHOUGH THE KL10 ALSO PROVIDES A KI10 PAGING MODE. KL10 PAGING PROVIDES FOR EFFICIENT PROGRAM WORKING SET MANAGEMENT AND DEMAND PAGING. IT ALSO PROVIDES FOR EXTENSIVE SHARING OF DATA AND PROGRAMS ON A PAGE-BY-PAGE BASIS.

REVISION HISTORY

REV	DESCRIPTION	CHG NO	ORIG	DATE	APPD BY	DATE
1	REVIEW CHANGES			6 AUG 75		
2	CLEANUP			5 DEC 75		

THE KL10 PAGING FACILITIES ARE INTENDED TO SUPPORT SOPHISTICATED OPERATING SYSTEM FEATURES, INCLUDING IN PARTICULAR:

1. EFFICIENT PROGRAM WORKING SET MANAGEMENT AND DEMAND PAGING.

2. EXTENSIVE SHARING OF DATA AND PROGRAMS ON A PAGE-BY-PAGE BASIS.

MUCH OF THE MECHANISM DESCRIBED HEREIN IS IMPLEMENTED BY KL10 MICROCODE RATHER THAN SPECIFIC HARDWARE. THE COMBINATION OF HARDWARE AND MICROCODE WHICH IMPLEMENTS THIS SPECIFICATION WILL BE REFERRED TO AS THE KL10 PAGER. THE KL10 ALSO SUPPORTS KI10 PAGING AS A SPECIAL MODE. KI10 PAGING IS NOT DESCRIBED IN THIS CHAPTER.

THIS PAGING DESIGN ALSO SUPPORTS THE EXTENDED ADDRESSING FACILITIES OF THE KL10 PROCESSOR, AND IT IS EXTENDABLE BEYOND THE 32-SECTION IMPLEMENTATION OF THE KL10. THE PHYSICAL CORE ADDRESS FIELDS SUPPORT UP TO 27 BITS (134 MILLION WORDS) OF PHYSICAL CORE MEMORY; THE KL10 IMPLEMENTS 22 BITS (4 MILLION WORDS) OF THIS. THERE IS ALSO AN IMPLICIT LIMIT OF 2**23 (8 MILLION) PAGES PER DISK STRUCTURE (4 BILLION WORDS).

N.B. "CORE" IN THIS DISCUSSION IS NOT MEANT TO SPECIFY A TECHNOLOGY, BUT A MEMORY SYSTEM WITH ACCESS TIME AND CAPACITY SIMILAR TO CORE.

ADDRESS SPACES

THE USER ADDRESS SPACE IS HOMOGENEOUS AND CONSISTS OF 32 EQUAL SECTIONS. SECTION 0 IS NOT TREATED IN ANY SPECIAL MANNER BY THE PAGING FACILITIES. THE SECTION TABLE FOR THE USER ADDRESS SPACE RESIDES IN THE UPT AND CONSISTS OF 32 SECTION POINTERS. THE EXEC ADDRESS SPACE ALSO CONSISTS OF 32 EQUAL SECTIONS. ITS SECTION TABLE RESIDES IN THE EPT AND CONSISTS OF 32 SECTION POINTERS. THE MONITOR SOFTWARE CAN EFFECTIVELY DIVIDE THE EXEC ADDRESS SPACE INTO PER-PROCESS AND PER-JOB AREAS THROUGH THE USE OF INDIRECT POINTERS (SEE APPENDIX), HENCE NO SUCH DIVISION NEED BE BUILT INTO THE PAGER. IN A MULTI-PROCESSING SYSTEM, EACH CPU HAS ITS OWN EPT AND HENCE ITS OWN EXEC SECTION TABLE.

SECTION POINTERS

A SECTION POINTER REPRESENTS AN ENTIRE SECTION, I.E., A 256K-WORD ADDRESS SPACE. IT POINTS TO A PAGE TABLE WHICH IN TURN CONTAINS POINTERS REPRESENTING EACH PAGE OF THAT SECTION.

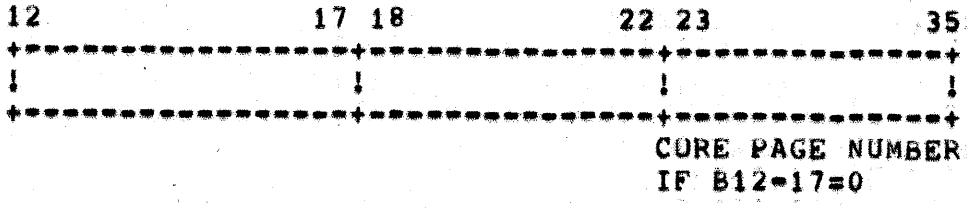
THE SECTION POINTER MAY BE IMMEDIATE, SHARED OR INDIRECT FORMAT; IN ANY CASE, IT MUST YIELD THE PHYSICAL CORE ADDRESS OF A PAGE CONTAINING A PAGE TABLE.

PAGE TABLES

A PAGE TABLE IS A PAGE CONTAINING PAGE POINTERS FOR ONE SECTION. THERE ARE 512 PAGES PER SECTION AND EACH POINTER IS ONE 36-BIT WORD. HENCE A PAGE TABLE IS A FULL PAGE.

STORAGE ADDRESSES (PAGE ADDRESSES)

A STORAGE ADDRESS IDENTIFIES A PAGE OF PHYSICAL STORAGE IN CORE OR ON SOME OTHER MEDIUM. THE FORMAT OF THE STORAGE ADDRESS DETERMINES BOTH THE MEDIUM AND THE ADDRESS WITHIN THAT MEDIUM. STORAGE ADDRESSES ARE FOUND IN PAGE POINTERS AND IN SPT ENTRIES (BELOW), AND ARE 24-BIT QUANTITIES.



IF BITS 12-17 ARE 0, THE ADDRESS REFERS TO CORE MEMORY. BITS 23-35 ARE THE PHYSICAL CORE PAGE NUMBER OF THE PAGE (B18-22 MBZ). ONLY VIRTUAL ADDRESS REFERENCES WHICH TRANSLATE TO PHYSICAL CORE ADDRESSES CAN BE COMPLETED BY THE PROCESSOR.

IF BITS 12-17 ARE NOT 0, THEN THE ADDRESS IS SOMETHING OTHER THAN PHYSICAL CORE (E.G., DISK OR DRUM). A VIRTUAL ADDRESS REFERENCE WHICH TRANSLATES TO A NON-CORE ADDRESS CANNOT BE COMPLETED BY THE PROCESSOR, AND A TRAP TO THE MONITOR MUST BE INITIATED. THE FORMAT OF NON-CORE ADDRESSES IS IRRELEVANT TO THE PAGER BEYOND THE CONVENTION THAT BITS 12-17 ARE NOT 0.

PAGE POINTERS

THERE ARE THREE TYPES OF PAGE POINTERS, IMMEDIATE, SHARED, AND INDIRECT. THE POINTER TYPE IS ENCODED IN BITS 0-2 OF THE POINTER AS FOLLOWS:

0	NO ACCESS
1	IMMEDIATE
2	SHARED
3	INDIRECT

4-7 NOT USED, RESERVED.

EACH PAGE POINTER CONTAINS ACCESS BITS WHICH DETERMINE WHAT TYPES OF REFERENCES MAY BE MADE TO THE PAGE. THE ACCESS BITS ARE HANDLED IN THE SAME WAY REGARDLESS OF WHICH OF THE THREE POINTER TYPES IS BEING INTERPRETED.

P (PUBLIC,B3) IF THIS BIT IS OFF, THE PAGE MAY ONLY BE REFERENCED BY PROGRAMS RUNNING IN CONCEALED OR KERNEL MODE. SEE KI10 REFERENCE MANUAL FOR ADDITIONAL DETAILS.

W (WRITE,B4) IF THIS BIT IS OFF, WRITE REFERENCES MAY NOT BE DONE TO THE PAGE.

C (CACHE,B6) IF THIS BIT IS ON, DATA IN THIS PAGE MAY BE PLACED IN THE CACHE.

B5,7-11 THESE BITS ARE NOT USED BY THE PAGER AND ARE RESRVED FOR FUTURE SPECIFICATION BY DEC.

IMMEDIATE POINTER

AN IMMEDIATE POINTER CONTAINS THE PHYSICAL ADDRESS OF THE ASSOCIATED PAGE IN BITS 12-35. THIS POINTER TYPE IS ALSO KNOWN AS PRIVATE SINCE THE PAGE IS PRIVATE TO THE PAGE TABLE WHICH CONTAINS THE POINTER.

SHARED POINTER

A SHARED POINTER CONTAINS AN INDEX WHICH POINTS INTO THE SPT (SPECIAL/SHARED PAGES TABLE). THE ASSOCIATED SPT ENTRY THEN CONTAINS THE PHYSICAL ADDRESS FOR THE PAGE. THE SPT ENTRY IS FOUND AT THE PHYSICAL CORE ADDRESS GIVEN BY THE SUM OF THE SPT BASE REGISTER AND THE SPT INDEX FROM THE SHARED POINTER. THIS IS KNOWN AS A SHARED POINTER BECAUSE MANY PAGE TABLES MAY CONTAIN SHARED POINTERS TO THE SAME PHYSICAL PAGE. REGARDLESS OF THE NUMBER OF PAGE TABLES HOLDING A PARTICULAR SHARED POINTER, THE PHYSICAL ADDRESS IS RECORDED ONLY ONCE IN THE SPT. HENCE THE MONITOR MAY MOVE THE PAGE WITH ONLY ONE ADDRESS TO UPDATE.

INDIRECT POINTER

THE INDIRECT POINTER IDENTIFIES ANOTHER PAGE TABLE AND A POINTER WITHIN THAT PAGE TABLE. IT CAUSES THE NEW POINTER TO BE FETCHED AND INTERPRETED. HENCE, THE INDIRECT POINTER IS USED TO MAKE A PAGE OF ONE ADDRESS SPACE EXACTLY EQUIVALENT TO A PAGE OF ANOTHER ADDRESS SPACE.

THE INDIRECT POINTER IDENTIFIES THE OBJECT PAGE TABLE USING AN

SPT INDEX. THE PHYSICAL ADDRESS OF THE PAGE TABLE IS FOUND IN THE ASSOCIATED SPT ENTRY JUST AS FOR A SHARED POINTER. THIS IS DONE SO THAT THE PHYSICAL ADDRESS OF A PAGE TABLE MAY BE KEPT IN ONE PLACE AND NEED NOT BE DUPLICATED IN ANY PAGE POINTERS. ONCE THE OBJECT PAGE TABLE HAS BEEN FOUND (AND DETERMINED TO BE IN CORE), THE PAGE NUMBER FIELD OF THE INDIRECT POINTER IS USED AS AN INDEX TO SELECT A NEW POINTER WORD FROM THE PAGE TABLE. THIS NEW POINTER MAY BE OF ANY OF THE THREE POINTER TYPES OR IT MAY BE NO-ACCESS. THE ACCESS BITS OF THE NEW POINTER ARE "AND"ED WITH THOSE OF THE INDIRECT POINTER. THUS ACCESS TO A PAGE IS PROHIBITED IF EITHER POINTER WOULD PROHIBIT IT. THE PAGER WILL INTERPRET INDIRECT POINTERS TO AN ARBITRARY DEPTH BUT MUST BE ABLE TO TERMINATE INDIRECT POINTER INTERPRETATION TO SERVICE A PRIORITY INTERRUPT IN THE CASE OF LONG INDIRECT CHAINS OR INDIRECT LOOPS.

SPT

THE SPT (SPECIAL/SHARED PAGES TABLE) HOLDS PHYSICAL ADDRESSES FOR PAGES WHICH ARE SHARED AMONG MANY PAGE TABLES (PROCESSES) OR WHICH ARE USED IN SOME SPECIAL WAY, E.G., AS PAGE TABLES. A PAGER REGISTER (AC BLOCK 6, WORD 3) HOLDS THE BASE ADDRESS OF THE SPT. THE SPT INDEX FOUND IN POINTERS IS ADDED TO THE SPT BASE ADDRESS TO FORM THE PHYSICAL CORE ADDRESS OF THE ASSOCIATED ENTRY. THE SPT ENTRY CONTAINS A PHYSICAL ADDRESS IN BITS 12-35. BITS 0-11 ARE IGNORED BY THE PAGER AND ARE USED AS A SHARE COUNT BY THE MONITOR.

CORE STATUS TABLE

IN ORDER TO DYNAMICALLY MANAGE CORE IN A VIRTUAL MEMORY ENVIRONMENT, IT IS EXTREMELY IMPORTANT FOR THE MONITOR TO BE ABLE TO OBTAIN INFORMATION ABOUT THE MEMORY REFERENCES GENERATED BY USER JOBS. THE CORE STATUS TABLE (CST) IS USED TO RECORD AND HOLD SUCH INFORMATION.

THE BASE ADDRESS OF THE CST IS HELD IN A PAGER REGISTER (AC BLOCK 6, WORD 2). THIS IS ADDED TO THE PHYSICAL CORE PAGE NUMBER FROM A STORAGE ADDRESS TO FORM THE ADDRESS OF THE ASSOCIATED CST ENTRY. A CST ENTRY IS USED AND UPDATED IN THE FOLLOWING MANNER:

1. FETCH THE CST ENTRY.

2. IF BITS 0-5 ARE 0, THE PAGE IS INACCESSIBLE AND A TRAP TO THE MONITOR IS INITIATED.

3. THE CST ENTRY IS "AND"ED WITH A MASK BEING HELD IN A PAGER REGISTER (CSTMSK, AC BLOCK 6, WORD 0).

4. THE RESULT IS IORED WITH THE QUANTITY IN A SECOND PAGER REGISTER (CSTDATA, AC BLOCK 6, WORD 1).

5. IF THE CURRENT VIRTUAL ADDRESS REFERENCE IS A WRITE REFERENCE (AND THE ACCESS BITS PERMIT A WRITE), A 1 IS IORED INTO BIT 35.

6. THE RESULT IS STORED BACK INTO THE CST.

THIS PROCEDURE ALLOWS THE MONITOR TO SET CERTAIN FIELDS AND MERGE OTHERS. TYPICALLY, ONE FIELD IS SELECTED AS AN "AGE", AND A VALUE REPRESENTING TIME IS SET INTO THIS FIELD. THUS PHYSICAL PAGES MAY BE ORDERED BY TIME OF LAST REFERENCE. ANOTHER FIELD IS TYPICALLY USED TO RECORD WHAT PROCESSES REFERENCE A PAGE. THIS IS DONE BY ASSIGNING A BIT POSITION TO EACH ACTIVE PROCESS AND PLACING A 1 IN THAT BIT POSITION IN THE PAGER DATA WORD. THUS THE CST WORD FOR A PAGE WILL HAVE ONES IN THE BIT POSITIONS OF EACH OF THE PROCESSES WHICH REFERENCED IT. ADDITIONALLY, THE MODIFIED BIT (B35) WILL BE SET IF THE PAGE HAS BEEN MODIFIED. THE MONITOR NEED NOT SWAP OUT PAGES TO WHICH ONLY READ REFERENCES HAVE BEEN DONE.

POINTER INTERPRETATION FLOW

THE FOLLOWING IS A DESCRIPTION OF THE POINTER INTERPRETATION ALGORITHM. IN THE FOLLOWING FLOW, THE TERM "USER" REFERS TO WHETHER AN EXEC OR USER SECTION IS BEING REFERENCED. THIS IS THE SAME AS THE PROCESS STATE BIT, EXCEPT UNDER PXCT (WHERE THE PROCESSOR CAN BE IN EXEC MODE WHILE REFERENCING A USER SECTION). THIS FLOW IS ALSO REPRESENTED BY A FLOW CHART IN THE FIGURES SECTION OF THIS DOCUMENT

1. INITIALIZE LOCAL PAGER VARIABLES P, W, AND C TO 1.

2. FETCH SECTION POINTER. THE SECTION POINTER IS FOUND IN THE USER SECTION TABLE IN THE UPT (LOCATION USECT THROUGH USECT+37) IF THE REFERENCE IS TO A USER SECTION, AND THE EXEC SECTION TABLE IN THE EPT (LOCATION ESECT THROUGH ESECT+37) IF THE REFERENCE IS TO AN EXEC SECTION. (USECT=ESECT=440)

3. TRAP IF A NO-ACCESS SECTION POINTER. A SECTION MAY BE NON-EXISTENT FOR THE RUNNING PROCESS, IN WHICH CASE THE SECTION POINTER WILL BE 0. BITS 0-2 OF THE SECTION POINTER ARE USED AS A CODE FIELD JUST AS WITH PAGE POINTERS. CODE 0 IS NO-ACCESS, CODE 1 IS IMMEDIATE, CODE 2 (SHARE) IS NORMAL SECTION POINTER, CODE 3 (INDIRECT) MAY ALSO BE USED, OTHER CODES ARE NOT DEFINED.

4. UPDATE ACCESS BITS, AND P, W, C (LOCAL PAGER VARIABLES) WITH SECTION POINTER BITS 3, 4, 6 RESPECTIVELY.

5. IF THE CODE IN BITS 0-2 IS 1, BITS 23-35 CONTAIN THE PAGE TABLE ADDRESS. OTHERWISE, FETCH THE PAGE TABLE

PHYSICAL ADDRESS. FETCH FROM THE PHYSICAL CORE ADDRESS GIVEN BY THE SUM OF THE SPT BASE ADDRESS AND THE SPT INDEX. THE SPT INDEX IS BITS 18-35 OF THE LAST POINTER FETCHED (A SECTION POINTER OR AN INDIRECT POINTER).

6. TRAP IF THE PAGE TABLE IS NOT IN CORE. IF THE PHYSICAL ADDRESS OBTAINED IN STEP 5 DOES NOT CONTAIN 0 IN BITS 12-17, THEN THE PHYSICAL ADDRESS IS NOT A CORE ADDRESS AND A TRAP IS INITIATED. IF THE SECTION POINTER WAS INDIRECT, FETCH A NEW SECTION POINTER FROM THIS PAGE TABLE, THE ENTRY GIVEN BY BITS 9-17 OF THE INDIRECT POINTER, AND GO TO 3.

7. CHECK AND UPDATE THE CST FOR THE PAGE TABLE. FETCH FROM THE PHYSICAL CORE ADDRESS GIVEN BY THE SUM OF THE CST BASE ADDRESS AND THE PHYSICAL CORE PAGE NUMBER. TRAP IF BITS 0-5 OF THE CST ENTRY ARE 0. OTHERWISE, AND CSTMSK, IOR CSTDATA, AND STORE THE RESULT BACK INTO CORE. THE MODIFIED BIT SHOULD NOT BE CHANGED HERE.

8. FETCH THE PAGE POINTER. THE PAGE POINTER IS FETCHED FROM THE PHYSICAL CORE ADDRESS CONSISTING OF THE CORE PAGE NUMBER OF THE PAGE TABLE IN BITS 14-26, AND THE CURRENT VIRTUAL PAGE NUMBER IN BITS 27-35. THE CURRENT PAGE NUMBER CAME FROM VMA BITS 18-26 OR FROM BITS 9-17 OF THE LAST INDIRECT POINTER.

9. UPDATE ACCESS BITS. THE P, W, AND C BITS ARE ANDED WITH THE RESPECTIVE PAGER VARIABLES.

10. DISPATCH ON POINTER TYPE.

    IF A NO-ACCESS POINTER (CODE 0), INITIATE A TRAP.

    IF AN INDIRECT POINTER, (CODE 3) TAKE BITS 9-17 AS THE NEW CURRENT PAGE NUMBER, AND TAKE BITS 18-35 AS AN SPT INDEX IDENTIFYING A NEW PAGE TABLE. LOOP BACK TO STEP 5.

    IF A SHARE POINTER, FETCH THE SPT ENTRY GIVEN BY THE SUM OF THE SPT BASE REGISTER AND BITS 18-35 OF THE SHARE POINTER.

    IF AN IMMEDIATE POINTER, CONTINUE TO STEP 11.

11. TRAP IF PAGE NOT IN CORE. IF BITS 12-17 OF THE PHYSICAL ADDRESS ARE NOT 0, THE ASSOCIATED PAGE IS NOT IN CORE AND A TRAP TO THE MONITOR IS INITIATED.

12. CHECK AND UPDATE THE CST FOR THE PAGE. FETCH FROM THE PHYSICAL CORE ADDRESS GIVEN BY THE SUM OF THE CST BASE ADDRESS AND THE PHYSICAL CORE PAGE NUMBER. TRAP IF BITS 0-5 OF THE CST ENTRY ARE 0. OTHERWISE, AND CSTMSK, IOR CSTDATA, AND IOR B35 IF A WRITE REFERENCE

AND W=1. STORE THE RESULT BACK INTO CORE.

13. COMPLETE THE VIRTUAL ADDRESS REFERENCE AND/OR LOAD PAGING MEMORY.

    INITIATE ILLEGAL WRITE TRAP IF WRITE AND NOT-W.

    THE PHYSICAL ADDRESS FOR THE REFERENCE CONSISTS OF THE PHYSICAL CORE PAGE NUMBER IN BITS 14-26, AND VMA BITS 27-35 IN BITS 27-35.

INSTRUCTIONS RELEVANT TO PAGING

SEE CHAP 2,6 OF KL10 FUNCTIONAL SPECS, KL10 INTERNAL IO INSTRUCTIONS, FOR ADDITIONAL INFORMATION.

1. LOAD UBR (DATAO PAG,) LOAD EBR (CONO PAG,).
2. CLEAR PAGING MEMORY (DATAO PAG, CONO PAG,).
3. CLEAR PAGING MEMORY ENTRY FOR MONITOR VIRTUAL ADDRESS E (CLRPT E).
4. PAGING ON/OFF (CONO PAG,).
5. DECLARE SPT BASE ADDRESS; CST BASE ADDRESS; CSTMSK; CSTDATA. (DONE AS DEPOSITS INTO RESERVED AC BLOCK.)

PAGE FAIL DATA

THE FOLLOWING DATA IS STORED WHEN A PAGE FAIL TRAP IS INITIATED:

1. VIRTUAL ADDRESS OF REFERENCE
2. USER BIT, PUBLIC BIT, WRITE REFERENCE BIT
3. PAGE FAIL CODE

SPECIFIC PAGE FAIL CODES ARE GENERATED AND STORED IN THE PAGE FAIL WORD FOR THE FOLLOWING CONDITIONS:

1. PROPRIETARY VIOLATION
2. REFILL ERROR
3. ADDRESS COMPARE
4. PAGE TABLE PARITY ERROR
5. MEMORY DATA PARITY ERROR

ALL OTHER PAGE FAIL CONDITIONS STORE ONLY THE DATA DESCRIBING THE REFERENCE (ADDRESS, USER, PUBLIC, WRITE) AND THE SOFTWARE WILL DETERMINING THE CAUSE OF THE PAGE FAIL AND THE PROPER ACTION.

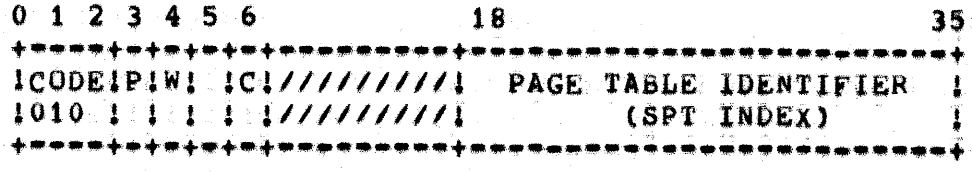
KL10 PAGING - WORD FORMATS

SECTION POINTER

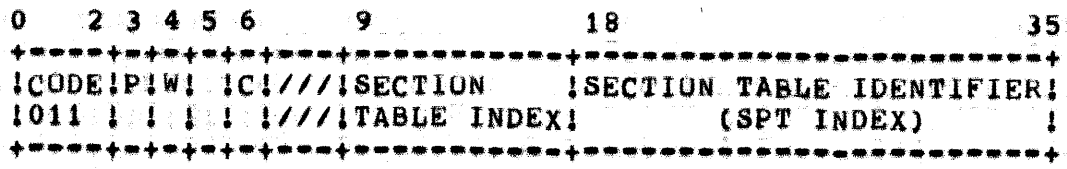
THE SECTION POINTER IS FOUND IN THE USER OR EXEC SECTION TABLE, (PART OF UPT OR EPT.)

SECTION POINTER PROVIDES (VIA THE SPT) THE PHYSICAL ADDRESS OF THE PAGE TABLE FOR THE GIVEN SECTION.

CODE:		
	0	NO-ACCESS (TRAP)
	1	IMMEDIATE
	2	SHARE
	3	INDIRECT
	4-7	UNUSED, RESERVED



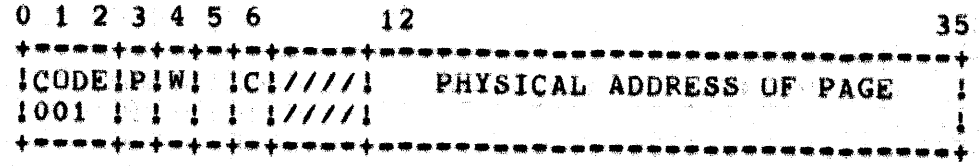
NORMAL SECTION POINTER (CODE = 2)



INDIRECT SECTION POINTER (CODE = 3)

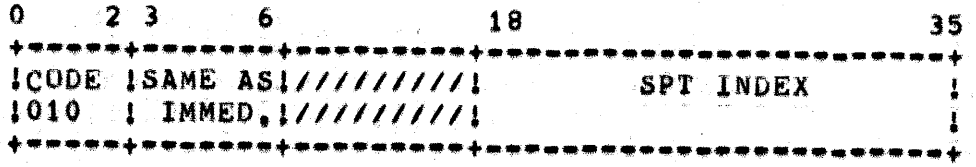
PAGE POINTERS

FOUND IN PAGE TABLES



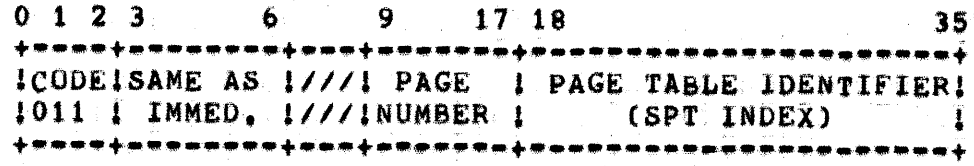
IMMEDIATE POINTER (CODE FIELD = 1)

B12-35 GIVE PHYSICAL ADDRESS OF PAGE
IF B12-17 >< 0, PAGE NOT IN CORE-TRAP
IF B12-17 = 0, B23-35 GIVE CORE PAGE
NUMBER OF PAGE, B18-22 MBZ



SHARED POINTER (CODE FIELD = 2)

B18-35 GIVE SPT INDEX (SPTX). SPTX + SPT BASE
ADDRESS = PHYSICAL CORE ADDRESS OF WORD
HOLDING PHYSICAL ADDRESS OF PAGE.

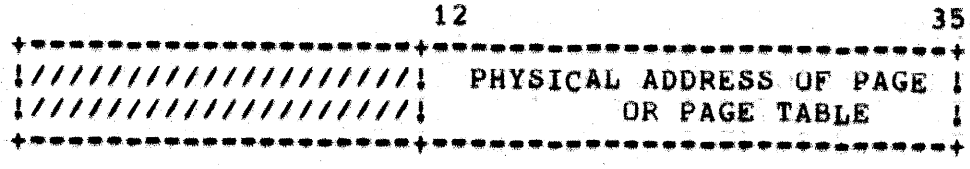


INDIRECT POINTER (CODE FIELD = 3)

THIS POINTER TYPE CAUSES ANOTHER POINTER TO BE FETCHED AND INTERPRETED. THE NEW POINTER IS FOUND IN WORD N (B9-17) OF THE PAGE ADDRESSED BY C(SPT + SPTX).

SPT ENTRY

FOUND IN THE SPT, I.E., WHEN FETCHING C(SPT +SPTX)

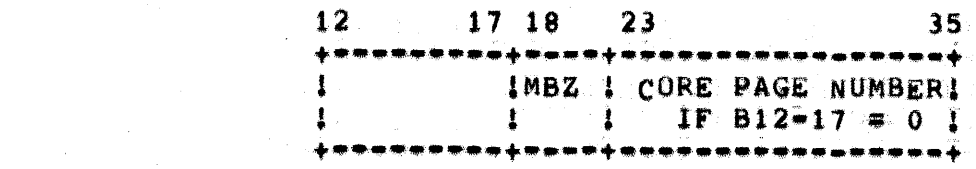


B12-35 GIVE PHYSICAL ADDRESS OF PAGE.

THE BASE ADDRESS (PHYSICAL CORE ADDRESS) OF THE SPT RESIDES IN ONE AC OF THE RESERVED AC BLOCK.

PHYSICAL STORAGE ADDRESS

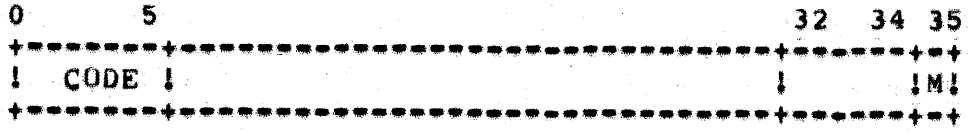
FOUND IN B12-35 OF IMMEDIATE POINTERS AND SPT ENTRIES.



IF B12-17 = 0, THEN B23-35 ARE CORE PAGE NUMBER (I.E., B14-26 OF PHYSICAL CORE ADDRESS) OF PAGE AND B18-22 MBZ. IF B12-17 >< 0, THEN ADDRESS IS NOT CORE AND PAGER TRAPS.

CORE STATUS TABLE ENTRY

FOUND WHEN FETCHING C(CBR + CORE PAGENO)



B0-5 ARE CODE FIELD:

0 - UNAVAILABLE, TRAP

1-77 - AVAILABLE

B32-34 RESERVED FOR FUTURE HARDWARE SPECIFICATION.

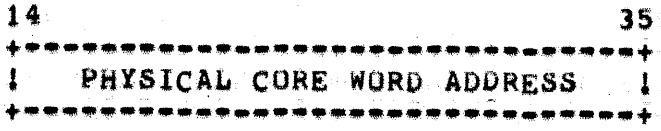
B35 IS "MODIFIED" BIT, SET ON ANY WRITE REF TO PAGE.

WHENEVER A CORE PAGE NUMBER (PHYS. ADR WITH B12-17=0) IS FOUND DURING POINTER INTERPRETATION, THE CST WORD AT CBR + CORE PAGENO IS UPDATED AS FOLLOWS:

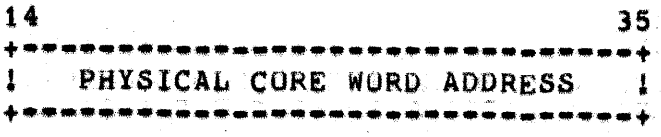
1. FETCH C(CBR+CORE PAGENO)
2. TRAP IF B0-5 =0
3. AND WITH MASK FROM RESERVED AC BLOCK
4. IOR WITH DATA FROM RESERVED AC BLOCK
5. IOR B35 IF WRITE REFERENCE

QUANTITIES IN HARDWARE REGISTERS (RESERVED AC BLOCK)

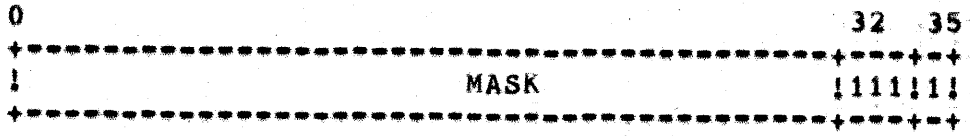
SPT SPT BASE REGISTER



CBR CST BASE REGISTER



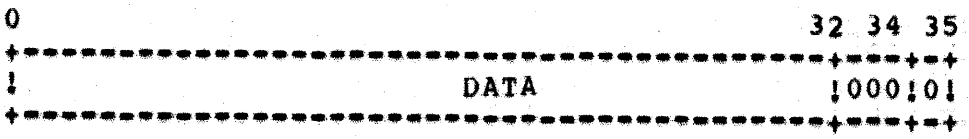
CSTMSK CST UPDATE MASK



ANDED WITH CST WORD DURING UPDATE

(B32-35 MUST BE ALL 1'S TO PRESERVE EXISTING CST INFORMATION)

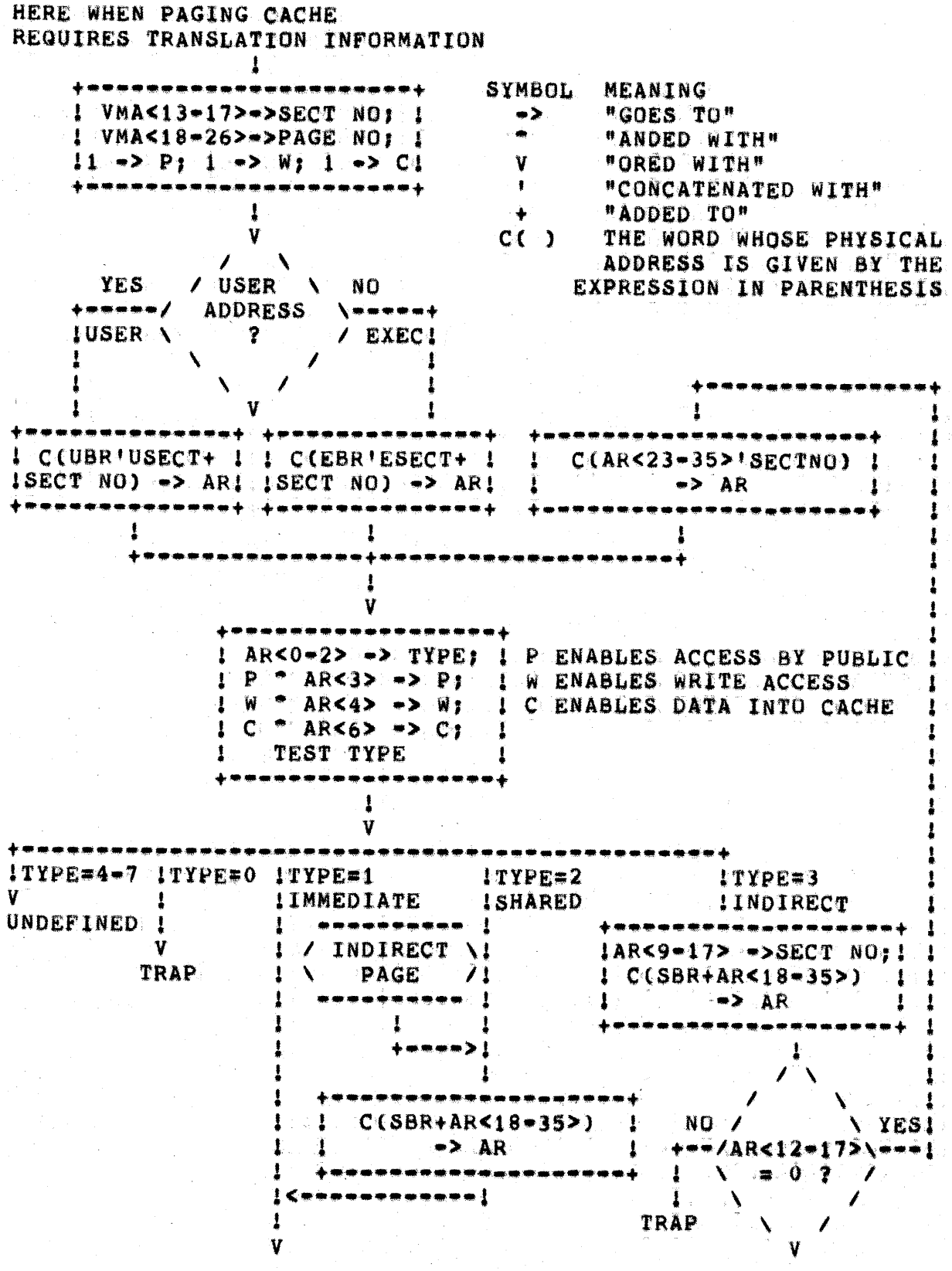
CSTDATA CST UPDATE DATA

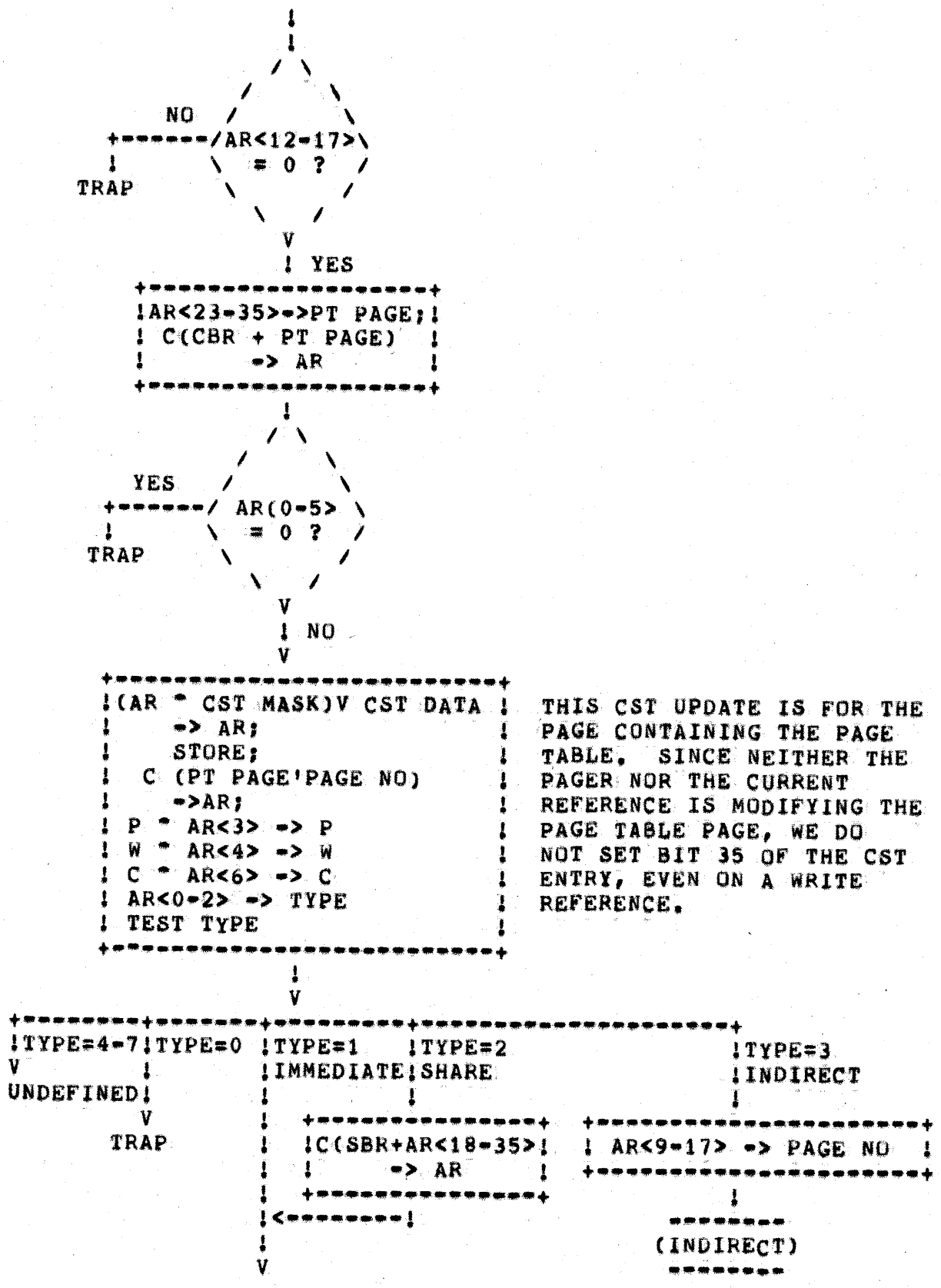


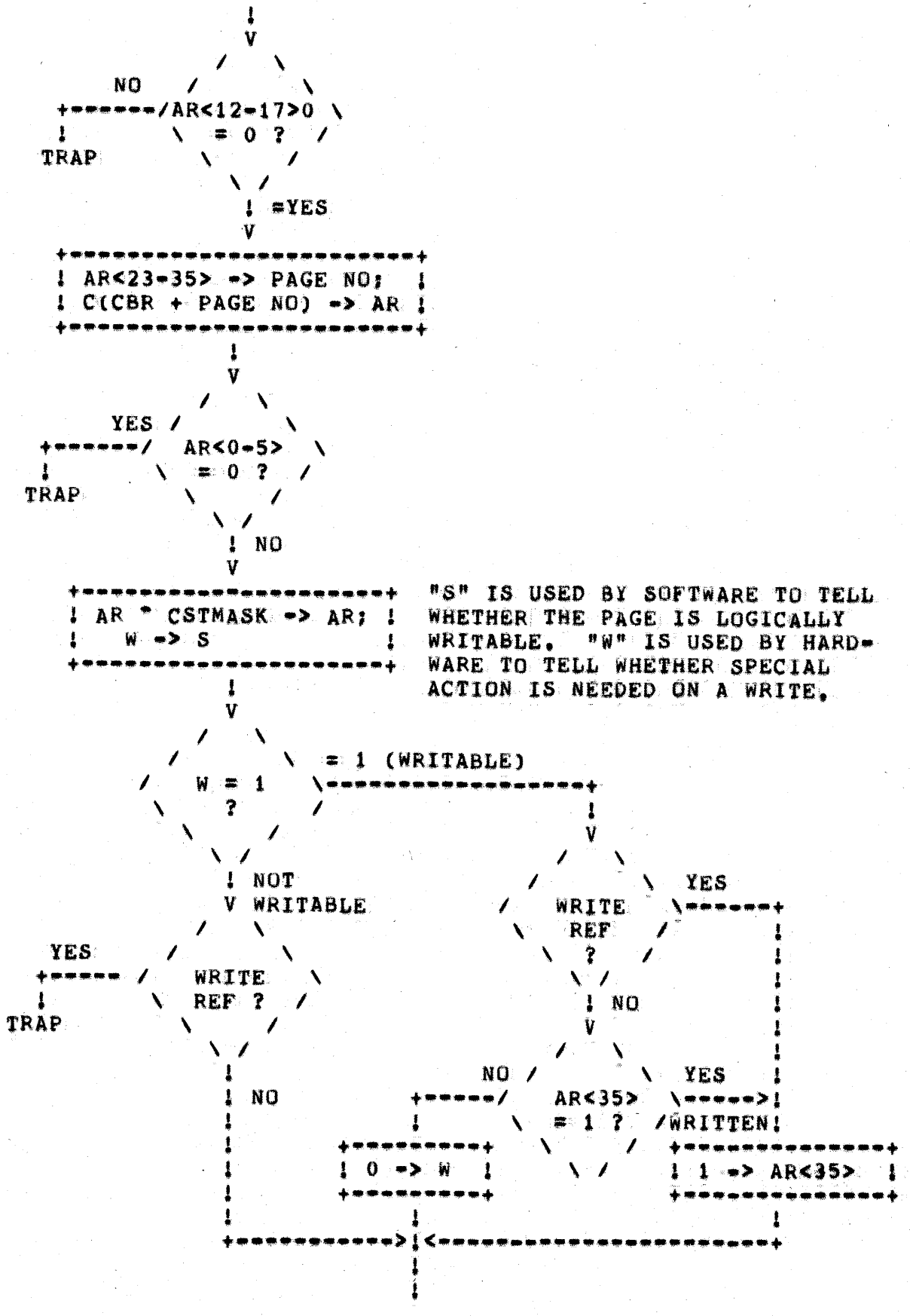
IORED WITH CST WORD DURING UPDATE

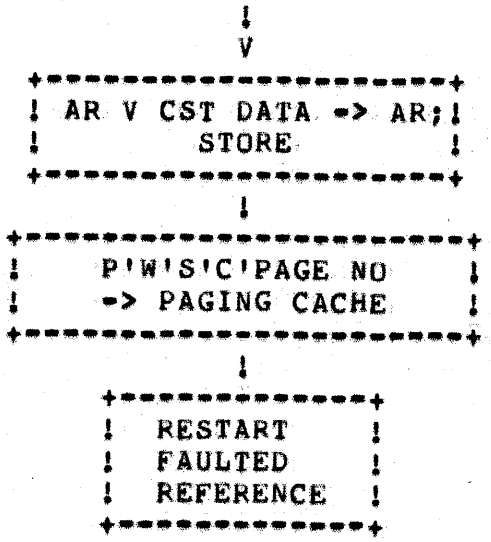
(B32-35 MUST BE ALL 0'S TO PRESERVE EXISTING CST INFORMATION)

ALL UNSPECIFIED BITS AND FIELDS ARE RESERVED FOR FUTURE SPECIFICATION BY DEC.


HERE WHEN PAGING CACHE
REQUIRES TRANSLATION INFORMATION
VMA<13-17>->SECT NO;
VMA<18-26>->PAGE NO;
1 -> P; 1 -> W; 1 -> C
USER ADDRESS ?
YES
NO
USER
EXEC
C(UBR'USECT+ SECT NO) -> AR
C(EBR'ESECT+ SECT NO) -> AR
C(AR<23-35>'SECTNO) -> AR
AR<0-2> -> TYPE;
P ^ AR<3> -> P;
W ^ AR<4> -> W;
C ^ AR<6> -> C;
TEST TYPE
P ENABLES ACCESS BY PUBLIC
W ENABLES WRITE ACCESS
C ENABLES DATA INTO CACHE
TYPE=4-7
UNDEFINED
TRAP
TYPE=0
TYPE=1
IMMEDIATE
INDIRECT PAGE
TYPE=2
SHARED
C(SBR+AR<18-35>) -> AR
TYPE=3
INDIRECT
AR<9-17> ->SECT NO;
C(SBR+AR<18-35>) -> AR
AR<12-17> = 0 ?
NO
YES
TRAP
SYMBOL MEANING
-> "GOES TO"
^ "ANDED WITH"
V "ORED WITH"
' "CONCATENATED WITH"
+ "ADDED TO"
C( ) THE WORD WHOSE PHYSICAL ADDRESS IS GIVEN BY THE EXPRESSION IN PARENTHESIS


NO
AR<12-17> = 0 ?
TRAP
YES
AR<23-35>->PT PAGE;
C(CBR + PT PAGE)
-> AR
YES
AR(0-5> = 0 ?
TRAP
NO
(AR * CST MASK)V CST DATA
-> AR;
STORE;
C (PT PAGE'PAGE NO)
->AR;
P * AR<3> -> P
W * AR<4> -> W
C * AR<6> -> C
AR<0-2> -> TYPE
TEST TYPE
THIS CST UPDATE IS FOR THE PAGE CONTAINING THE PAGE TABLE. SINCE NEITHER THE PAGER NOR THE CURRENT REFERENCE IS MODIFYING THE PAGE TABLE PAGE, WE DO NOT SET BIT 35 OF THE CST ENTRY, EVEN ON A WRITE REFERENCE.
TYPE=4-7
UNDEFINED
TRAP
TYPE=0
TYPE=1
IMMEDIATE
TYPE=2
SHARE
C(SBR+AR<18-35>
-> AR
TYPE=3
INDIRECT
AR<9-17> -> PAGE NO
(INDIRECT)


NO
AR<12-17>0 = 0 ?
TRAP
=YES
AR<23-35> -> PAGE NO;
C(CBR + PAGE NO) -> AR
YES
AR<0-5> = 0 ?
TRAP
NO
AR * CSTMASK -> AR;
W -> S
"S" IS USED BY SOFTWARE TO TELL WHETHER THE PAGE IS LOGICALLY WRITABLE. "W" IS USED BY HARDWARE TO TELL WHETHER SPECIAL ACTION IS NEEDED ON A WRITE.
W = 1 ?
= 1 (WRITABLE)
NOT WRITABLE
YES
WRITE REF ?
TRAP
NO
WRITE REF ?
YES
NO
AR<35> = 1 ?
WRITTEN
0 -> W
1 -> AR<35>


AR V CST DATA => AR;
STORE
P'W'S'C'PAGE NO
=> PAGING CACHE
RESTART
FAULTED
REFERENCE

APPENDIX I - EPT/UPT SYMBOLS USED

ESECT = 440 SECTION TABLE FOR EXEC SECTIONS 0-37

USECT = 440 SECTION TABLE FOR USER SECTIONS 0-37

APPENDIX II

THE FOLLOWING DISCUSSES SOME OF THE DESIGN AND IMPLEMENTATION ISSUES OF THE PAGER.

CACHED PAGING DATA

THE KL10 HAS A "PAGING MEMORY" WHICH HOLDS VIRTUAL-TO-PHYSICAL MAPPING INFORMATION. THIS IS EFFECTIVELY A CACHE OF PAGING DATA WHICH HAS BEEN FETCHED FROM MEMORY AND/OR DETERMINED BY POINTER INTERPRETATION. NOTE, HOWEVER, THAT IT IS TOTALLY DISTINCT FROM THE MEMORY CACHE, USUALLY REFERRED TO AS SIMPLY "THE CACHE". WHEN A VIRTUAL ADDRESS REFERENCE IS REQUESTED, THE PAGING MEMORY IS FIRST CHECKED TO SEE IF THE CORRESPONDING PHYSICAL ADDRESS IS PRESENT. IF IT IS, THE REFERENCE CAN PROCEED WITH NO DELAY. IF IT IS NOT, THE DATA MUST BE OBTAINED FROM CORE.

THE KL10 PAGING MEMORY IS IMPLEMENTED AS A TABLE WITH ONE ENTRY FOR EACH OF THE 512 PAGES OF THE VIRTUAL ADDRESS SPACE. (THE KI10 IMPLEMENTED THE EQUIVALENT FUNCTION WITH ASSOCIATIVE MEMORY.) THE USER AND EXEC ADDRESS SPACES USE THE SAME 512 ENTRIES, BUT THE INDEX IS OFFSET DIFFERENTLY SO AS TO REDUCE CONFLICTS. AT ANY TIME THEN, THE PAGING MEMORY WILL BE HOLDING MAPPING INFORMATION FOR MOST OF THE PAGES ACTIVELY BEING USED BY THE RUNNING PROGRAM. WHEN THE MONITOR TAKES ANY ACTION WHICH WOULD INVALIDATE SOME EXISTING VIRTUAL-TO-PHYSICAL ADDRESS ASSOCIATIONS, THE PAGING MEMORY MUST BE PARTIALLY OR COMPLETELY CLEARED. SUCH CASES INCLUDE:

1. CHANGE OF USER PROCESS - THE ENTIRE USER ADDRESS SPACE CHANGES, SO THE ENTIRE PAGING MEMORY MUST BE CLEARED.

2. REMOVAL OF ONE PAGE FROM CORE OR REMOVAL OF A POINTER FROM THE USER PROCESS PAGE TABLE - THE ENTIRE PAGING MEMORY MUST BE CLEARED SINCE SHARED AND INDIRECT POINTERS MAY HAVE CAUSED THE ONE PHYSICAL PAGE TO APPEAR IN SEVERAL VIRTUAL PAGES.

3. IN SOME CASES, THE MONITOR WILL MAP A PAGE INTO THE EXEC MAP FOR LOCAL USE. WHEN THIS PAGE IS UNMAPPED, ONLY THAT ONE ASSOCIATION NEED BE CLEARED FROM THE PAGING MEMORY. THE PAGER PROVIDES A FUNCTION TO CLEAR THE ASSOCIATION FOR A PARTICULAR VIRTUAL ADDRESS. THIS MAY BE USED IN THIS CASE TO REDUCE SUBSEQUENT RELOAD

OVERHEAD.

IN KL10 PAGING MODE, WHEN THE PAGING MEMORY FAILS TO CONTAIN THE DATA FOR THE REQUESTED VIRTUAL ADDRESS, A SPECIAL TRAP IN THE MICROCODE OCCURS. AFTER SAVING VULNERABLE ACTIVE EBOX DATA, THE MICROCODE INVOKES THE POINTER TRACING AND INTERPRETATION ALGORITHM DESCRIBED ABOVE. IF THE POINTER INTERPRETATION YIELDS A VALID CORE ADDRESS, IT AND THE ACCESS INFORMATION ARE LOADED INTO THE PAGING MEMORY. THEN THE EBOX ACTIVE REGISTERS ARE RESTORED AND THE ORIGINAL MEMORY REFERENCE IS REQUESTED AGAIN.

THE PAGER IS REQUIRED TO MAINTAIN THE MODIFIED BIT IN THE CORE STATUS TABLE. THIS MEANS THAT THE FIRST WRITE REFERENCE TO A PAGE MUST BE DETECTED EVEN IF PREVIOUS READ REFERENCES HAVE BEEN MADE. THE MICROCODE IMPLEMENTS THIS AS FOLLOWS:

> ON A PAGING MEMORY RELOAD, THE WRITE ACCESS (W) BIT IS SET IN THE PAGING MEMORY ONLY IF THE CURRENT MEMORY REFERENCE IS A WRITE (AND WRITE IS LEGAL FOR THE PAGE). THUS IF THE FIRST REFERENCE TO A PAGE IS READ, THE W BIT IN THE CORRESPONDING PAGING MEMORY ENTRY WILL BE SET TO 0, AND A SBSEQUENT WRITE REFERENCE WILL CAUSE ANOTHER TRAP TO THE MICROCODE. ON THIS SECOND TRAP, THE POINTER INTERPRETATION WILL BE REPEATED AND THE PAGING MEMORY RELOADED, THIS TIME WITH THE W BIT SET.

IT HAS BEEN SUGGESTED THAT THE MICROCODE COULD USE THE UNUSED BIT (FORMERLY THE S BIT) IN THE PAGING MEMORY TO RECORD WHETHER OR NOT THE PAGE IS WRITABLE AND THEREBY AVOID THE SECOND POINTER TRACE IN THE WRITE-AFTER-READ CASE. THE PERFORMANCE IMPROVEMENT APPEARS TO BE VERY SMALL HOWEVER AND AT PRESENT DOES NOT SEEM WORTH THE ADDITIONAL MICROCODE WHICH WOULD BE REQUIRED.

CACHE CONSIDERATIONS

IN A ONE-CPU CONFIGURATION, IT DOES NOT MATTER LOGICALLY WHETHER MICROCODE PAGING REFERENCES ARE CACHED. SINCE THE PAGING MEMORY IS AVAILABLE TO HOLD PAGING DATA, IT WOULD SEEM REDUNDANT TO ALSO HOLD PAGE POINTERS IN THE CACHE. ALSO, IT APPEARS UNLIKELY THAT REFERENCES ARE MADE TO PAGE POINTERS VERY OFTEN. ON THE OTHER HAND, THE SECTION 0 SECTION POINTERS (EXEC AND USER) AND THE ASSOCIATED SPT WORDS ARE REFERENCED ON EVERY PAGE MEMORY RELOAD. HENCE, IT IS UNCLEAR WHETHER A PERFORMANCE ADVANTAGE IS OBTAINED BY CACHING OR NOT CACHING PAGE REFILL REFERENCES.

MULTI-CPU CONSIDERATIONS

IN A MULTI-CPU CONFIGURATION USING KL10S, EACH PROCESSOR WOULD HAVE ITS OWN PAGING MEMORY, AND IN GENERAL, ALL OF THEM MUST BE

CLEARED ON ANY EVENT WHICH REQUIRES CLEARING ONE. THEREFORE THE MONITOR MUST HAVE A SIGNAL MECHANISM TO QUICKLY REQUEST THE OTHER PROCESSORS TO EXECUTE THE APPROPRIATE PAGER INSTRUCTIONS.

THE CODE FIELD OF THE CORE STATUS TABLE MAY BE USED TO PREVENT ACCESS TO CERTAIN PAGES BY OTHER PROCESSORS WHEN ONE PROCESSOR BEGINS SOME HOUSEKEEPING FUNCTION ON THOSE PAGES (E.G., SWAPPING OUT TO DISK). IT DOES NOT, HOWEVER, PROVIDE ANY MECHANISM TO LIMIT ACCESS TO EXACTLY ONE PROCESSOR.

IN A MULTI-CPU CONFIGURATION, THE SEVERAL PROCESSORS WILL BE UPDATING THE CORE STATUS TABLE SIMULTANEOUSLY, AND SOME MECHANISM (E.G., READ-PAUSE-WRITE) MUST BE USED TO ENSURE THAT DATA IS NOT LOST BECAUSE OF TWO PROCESSORS SIMULTANEOUSLY UPDATING THE SAME ENTRY. OBVIOUSLY, MICROCODE CST REFERENCES CANNOT BE CACHED, AND THE MONITOR MUST ENSURE THAT PROGRAM REFERENCES TO THE CST ARE ALSO NOT CACHED.

IT IS DESIRABLE THAT ALL MICROCODE PAGING REFERENCES BE UNCACHED IN A MULTI-CPU CONFIGURATION, OTHERWISE THE SOFTWARE MUST DO A CACHE CLEAR ALSO EVERY TIME IT DOES A PAGING MEMORY CLEAR OF ANOTHER PROCESSOR.

DIVISION OF EXEC ADDRESS SPACE

TYPICALLY THE MONITOR USES AREAS OF THE EXEC ADDRESS SPACE IN DISTINCT WAYS:

1. SYSTEM-WIDE CODE AND DATA. E.G., THE MONITOR CODE, SCHEDULER AND SWAPPER DATA BASES, ETC.

2. PER-JOB DATA. E.G., OPEN FILE DATA, FORK STRUCTURE DATA, ETC.

3. PER-PROCESS DATA. E.G., LOCAL STACK, LOCAL VARIABLES, ETC.

IN ADDITION TO PERMANENT CONTENTS, EACH OF THESE AREAS WILL ALSO HAVE FILE OR PROCESS PAGES DYNAMICALLY MAPPED IN AS NEEDED.

THE INDIRECT POINTER MECHANISM MAY BE USED TO CONFIGURE THE EXEC ADDRESS SPACE IN THIS MANNER, AND IT DOES NOT REQUIRE THAT ANY BOUNDARIES BE WIRED INTO HARDWARE (AS ON THE KI10, EXEC PAGES 340-377). TO IMPLEMENT THE DIVISION ABOVE, THE MONITOR WOULD HAVE THREE PARTIAL PAGE TABLES:

1. THE SYSTEM MONITOR MAP RESIDING IN RESIDENT STORAGE;

2. THE JOB MAP RESIDING IN A PAGE SWAPPED WITH THE JOB;

3. THE PROCESS MAP RESIDING IN A PAGE SWAPPED WITH THE PROCESS.

THE THREE MAPS WOULD COVER MUTUALLY EXCLUSIVE AREAS OF THE MONITOR ADDRESS SPACE. EACH WOULD HAVE AN SPT SLOT ASSIGNED TO HOLD ITS PHYSICAL CORE ADDRESS.

ALL POINTER TRACES WOULD FIRST REFERENCE THE SYSTEM MONITOR MAP AFTER HAVING INTERPRETED THE EXEC SECTION 0 POINTER. FOR SYSTEM-WIDE PAGES, THE MAP WOULD CONTAIN A NORMAL PAGE POINTER TO THE APROPRIATE PAGE. FOR PER-JOB PAGES, THE MAP WOULD CONTAIN AN INDIRECT POINTER TO THE JOB MAP, AND FOR PER-PROCESS PAGES, THE MAP WOULD CONTAIN AN INDIRECT POINTER TO THE PROCESS MAP. IN ORDER TO AVOID CHANGING ALL OF THE PER-JOB AND PER-PROCESS POINTERS ON A CONTEXT SWITCH, TWO SPT ENTRIES WOULD BE RESERVED FOR THE "CURRENT" JOB MAP AND PROCESS MAP ADDRESSES, AND THE JOB AND PROCESS POINTERS WOULD ALWAYS USE THESE RESERVED ENTRIES. THAT IS, WHEN ANY PROCESS IS STARTED, ITS JOB MAP CORE ADDRESS IS COPIED INTO THE RESERVED JOB SPT ENTRY, AND ITS PROCESS MAP CORE ADDRESS IS COPIED INTO THE RESERVED PROCESS SPT ENTRY. THESE SPT ENTRIES BECOME IN EFFECT TWO BASE REGISTERS FOR THE TWO MAPS, AND THE MONITOR MUST SET THEM JUST AS THOUGH THEY WERE IMPLEMENTED IN HARDWARE.

[END OF KLPAG.SPC]

1080,2040,2060 ENGINEERING FUNCTIONAL SPEC - CHAP 2,9

TO: KL10 LIST

TITLE: EXEC AND USER PROCESS TABLES (EPT + UPT) - REV 0

STATUS: CURRENT LOCATIONS DEFINED IN EPT + UPT, OTHERS WILL BE ADDED AS NEEDED, SUPERSEDED MEMO WAS REVIEWED SEPT 73, A FEW ADDITIONS AND CHANGES HAVE BEEN MADE, THE NEW PAGING [NOT IN BREADBOARD] IS NOT DESCRIBED HERE, COMMENTS ARE SOLICITED, A SHORT REVIEW WILL BE HELD FRIDAY, 15 MAR, IN LEPRECHAUN LOUNGE.

FILE: [EFS]CH2S09.SPC

PDM #: 200-200-005-00

DATE: 13 MAR 74

SUPERSEDED MEMOS: KL10 PROCESS TABLE, GUGLIELMI, 19 SEPT 73

ENGINEER: P. GUGLIELMI

APPROVED: P. GUGLIELMI

EDITOR: T. HASTINGS

TYPIST: M. PROUTY

REVIEWED:

ABSTRACT

ALL HARDWARE DEFINED ADDRESSES IN THE SYSTEM (EXCEPT THE RH20) ARE DEFINED WITHIN TWO PAGES, CALLED THE EXEC PROCESS TABLE (EPT) AND USER PROCESS TABLE (UPT). EACH CPU IN THE SYSTEM USES A DIFFERENT PAGE FOR THE EPT. EACH PROCESS IN THE SYSTEM USES A DIFFERENT PAGE FOR THE UPT.

REVISION HISTORY

REV	DESCRIPTION	CHG NO	ORIG	DATE	APPD BY	DATE

LAYOUT OF THE EXEC AND USER PROCESS TABLES (EPT + UPT).

0. CONTENTS

1. GOALS

1. NO BUILT-IN HARDWARE PHYSICAL OR VIRTUAL ADDRESSES.

2. ONE INSTRUCTION PROCESS CONTEXT SWITCH, NOT COUNTING AC'S AND PC.

3. HANDLE CLOSELY COUPLED MULTI-PROCESSOR SYSTEMS.

4. BE COMPATIBLE WITH KI, EXCEPT WHERE IMPORTANT.

2, INTRODUCTION

ALL OF THE ADDRESSES DEFINED BY THE HARDWARE (EXCEPT THE RH20) ARE DEFINED AS FIXED OFFSETS WITHIN ONE OF TWO PAGES, CALLED EXEC PROCESS TABLE (EPT) AND USER PROCESS TABLE (UPT). EACH CPU HAS TWO PRIVILEGED BASE REGISTERS, EXEC BASE REGISTER (EBR) AND USER BASE REGISTER (UBR). EACH BASE REGISTER SPECIFIES THE PHYSICAL PAGE NUMBERS (ADDRESS BITS 14-26) OF THE EPT AND UPT FOR THAT CPU. NO DEVICES HAVE BUILT-IN PHYSICAL, EXEC VIRTUAL OR USER VIRTUAL ADDRESSES WITH THE EXCEPTION OF THE USER UUO TRAP ADDRESS. THUS THE HARDWARE IMPOSES NO RESTRICTIONS ON THE EXEC OR USER VIRTUAL ADDRESS SPACES AND PERMITS THE SYSTEM TO RUN WITH ANY COMBINATION OF PHYSICAL MEMORY MODULES. THE RH20 IS AN EXCEPTION ONLY BECAUSE IT WAS DESIGNED BEFORE ALL OF THE OTHER DEVICES. ON THE RH20 THE SOFTWARE LOADS AN INTERRUPT ADDRESS WHICH CAN BE ANYWHERE IN EXEC VIRTUAL PAGE 0. EACH EPT CONTAINS VARIABLES ASSOCIATED WITH EACH CPU IN THE SYSTEM. EACH CPU USUALLY LOADS ITS EBR ONLY ONCE WHEN THE SYSTEM IS STARTED. EACH UPT CONTAINS VARIABLES ASSOCIATED WITH EACH PROCESS IN THE SYSTEM. EACH CPU LOADS ITS UBR EVERY TIME IT RUNS A NEW PROCESS. SEE CHAPTER 2,7, INTERNAL DEVICES (APR, PI, PAG).

3, CHANGES FROM KI10

THE FOLLOWING FIGURES SHOW THE EPT AND UPT WHEN RUNNING WITH EXTENDED ADDRESSING TURNED OFF (SEC = 0) AND WHEN IT IS TURNED ON (SEC = 1). WHEN SEC = 0 THE PROCESS TABLES LOOK VERY MUCH LIKE THE KI10'S PROCESS TABLES. THE USER PROCESS TABLE DIFFERS FROM THE KI10 PROCESS TABLE IN LOCATIONS 420, 426, 427 AND 500. LOCATION 420 NOW RECEIVES THE "USER & EXEC PAGE FAIL WORD" INSTEAD OF CONTAINING THE "USER PAGE FAILURE TRAP INSTRUCTION". LOCATION 426 NOW RECEIVES THE "PC WORD OF THE PAGE FAIL" INSTEAD OF THE "EXEC PAGE FAILURE WORD". LOCATION 427 LOADS THE "PAGE FAIL NEW PC WORD" INSTEAD OF THE "USER PAGE FAILURE WORD". THESE CHANGES REFLECT THE DESIRE TO HANDLE EXEC AND USER PAGE FAULTS SIMILARLY. IT ALSO DOES NOT SEEM NECESSARY TO CONTINUE TO HAVE THE GENERALITY OF ALLOWING ANY TRAP INSTRUCTION TO BE EXECUTED ON A PAGE FAULT. LOCATION 500 RECEIVES THE "PREVIOUS CONTEXT SECTION AND CWSX".

4. CHANNEL LOGOUT AREA

THE EXECUTIVE PROCESS TABLE DIFFERS FROM THE KI10'S IN LOCATIONS 000-037 AND 60-17. LOCATIONS 000-037 COMPRISE A 40 WORD AREA WHICH ARE USED BY THE KL10 BUILT-IN DATA CHANNELS. EACH OF THE 8 DATA CHANNELS WILL HAVE A FOUR WORD BLOCK FROM THIS AREA. CHANNEL 0 IS ASSIGNED LOCATIONS 0-3, CHANNEL 2 LOCATIONS 4-7, ETC. WORD 0 OF EACH 4 WORD CHANNEL BLOCK IS THE INITIAL CHANNEL COMMAND WORD. WORD 1 RECEIVES ERROR STATUS AND THE COMMAND LIST POINTER ON ERRORS. WORD 2 RECEIVES THE LAST WORD COUNT AND ADDRESS. WORD 3 IS RESERVED FOR FUTURE USE BY HARDWARE. SEE CHAPTER 4.1, MASSBUS CONTROLLER (RH20) AND CHANNELS (MBOX).

5. PDP-11 INTERFACE COMMUNICATION AREA (DTE20)

LOCATIONS 60-77 COMPRISE A 20 WORD AREA WHICH IS USED BY THE DTE20 (10/11 INTERFACE). EACH OF THE 4 DTE20'S HAS A BLOCK OF FOUR WORDS FROM THIS AREA ALLOCATED FOR ITS USE. CURRENTLY ONLY THREE OF THE FOUR LOCATIONS ASSIGNED TO A DTE20 ARE BEING USED. THE THREE LOCATIONS HOLD AN "IN BYTE POINTER", "OUT BYTE POINTER", AND AN "INTERRUPT VECTOR". LOCATIONS 100-177 ARE RESERVED FOR FUTURE USE BY HARDWARE. [KI: 60-177 ARE AVAILABLE TO SOFTWARE.] LOCATION 420 IS NO LONGER THE "EXEC PAGE FAIL TRAP INSTRUCTION". IT IS RESERVED FOR FUTURE USE. SEE CHAPTER 4.3, FRONT END INTERFACE (DTE20).

6. HALT STATUS AREA

LOCATIONS 424-427 WILL COMPRISE A PROCESSOR "HALT STATUS AREA". LOCATION 424 WILL CONTAIN THE HALT INSTRUCTION OR ALL ZEROES DEPENDING ON WHETHER A HALT INSTRUCTION WAS EXECUTED OR THE PDP11 TURNED OFF THE PROCESSORS RUN FLOP. LOCATION 425 WILL CONTAIN THE VALUE OF THE PC WORD AT THE SAME TIME THE MACHINE WAS HALTED. LOCATIONS 426-427 WILL CONTAIN TWO WORDS OF PROCESSOR STATUS. SEE CHAPTER 2.5, MISCELLANEOUS OPCODES.

LOCATIONS 501-577 OF THE USER PROCESS TABLE, AND 100-177 OF THE EXEC PROCESS TABLE HAVE BEEN RESERVED FOR FUTURE USE BY THE HARDWARE SO THAT A PLACE WILL BE AVAILABLE TO ADD NEW FEATURES TO THE SYSTEM. THIS WAS DONE BECAUSE LOCATIONS 000-177 AND 400-577 ARE THE ONLY AREAS OF THE PROCESS TABLES THAT THE MICRO-CODE CAN ACCESS DIRECTLY. [KI: 500-577 OF THE UPT ARE AVAILABLE TO SOFTWARE.]

7. PAGING

PAGE TABLE ENTRIES CONTAIN A NEW BIT CALLED THE CACHE BIT. WHEN THIS BIT IS TURNED ON THE PROCESSOR WILL LOAD THE CACHE ON READS AND WRITES. WHEN IT IS TURNED OFF THE PROCESSOR WILL NOT LOAD THE CACHE ON READS AND WRITES UNLESS THE DATA IS ALREADY IN THE CACHE. WHEN SEC = 0 THE PAGE TABLE ENTRIES ARE KI-STYLE HALFWORD ENTRIES. WHEN SEC = 1 THE PAGE TABLE ENTRIES ARE FULL WORD ENTRIES. SEE CHAPTER 2.8, PAGING. [BREADBOARD: ALL PAGETABLE ENTRIES ARE HALFWORD].

WHEN SEC = 1 THE PROCESS TABLES ARE THE SAME AS THE SEC = 0 PROCESS TABLES EXCEPT THAT THE PAGE TABLE ENTRIES ARE MISSING. THE SEC = 1 PROCESS TABLES CONTAIN SECTION TABLE ENTRIES IN LOCATIONS 440-477 OF THE USER PROCESS TABLE. THE FORMAT OF A SECTION TABLE ENTRY IS SHOWN BELOW.

EACH SECTION TABLE ENTRY TAKES ONE PDP10 WORD. SECTION TABLE ENTRIES FOR EXEC SECTIONS APPEAR IN BOTH THE USER AND EXECUTIVE PROCESS TABLES. LOCATIONS 400-417 OF THE EXECUTIVE PROCESS TABLE HOLDS SECTION TABLE ENTRIES FOR EXEC SECTIONS 00-17. LOCATIONS 400-417 OF THE USER PROCESS TABLE HOLDS SECTION TABLE ENTRIES FOR EXEC SECTIONS 20-37. THIS AREA IS SIMILAR TO THE PER PROCESS AREA OF THE KI10 PROCESS TABLE EXCEPT THAT THE KL10 CAN PROVIDE ACCESS TO A MUCH LARGER AMOUNT OF CORE (4,000,000 WORDS). LOCATIONS 440-477 OF THE USER PROCESS TABLE HOLDS SECTION TABLE ENTRIES FOR USER SECTIONS 00-37.

8. KI-STYLE PAGE TABLE ENTRIES (SEC = 0) PAGE TABLE ENTRIES:

DATA FOR EVEN VIRTUAL PAGES:

0	ACCESS BIT (A)
1	PUBLIC BIT (P)
2	WRITABLE BIT (W)
3	SOFTWARE BIT (S)
4	CACHE BIT (C)
5-17	PHYSICAL PAGE NUMBER (ADDRESS BITS 14-26)

DATA FOR ODD VIRTUAL PAGES:

18	ACCESS BIT (A)
19	PUBLIC BIT (P)
20	WRITABLE BIT (W)
21	SOFTWARE BIT (S)
22	CACHE BIT (C)
23-35	PHYSICAL PAGE NUMBER (ADDRESS BITS 14-26)

SEE CHAPTER 2,8, PAGING FOR A DESCRIPTION OF KL-STYLE PAGING, [NOT IN BREADBOARD].

9. SECTION TABLE ENTRIES (FULL WORD):

0	ACCESS BIT (A)
1-13	UNUSED
14-27	PHYSICAL HALF PAGE ADDRESS. THIS ADDRESS IS THE ADDRESS OF A PAGE WHICH CONTAINS THE PAGE TABLE WITH FULL WORD ENTRIES FOR THIS SECTION. SEE CHAPTER 2,8, PAGING. [BREADBOARD:THIS ADDRESS IS A HALF PAGE ADDRESS AND THE PAGE TABLE ENTRIES ARE HALF WORD ENTRIES. THE PAGE TABLE CAN BE IN EITHER THE FIRST OR SECOND HALF OF A PAGE FRAME.] THE PAGE TABLE FOR A SECTION IS COMPLETELY DISASSOCIATED FROM THE USER PROCESS TABLE (UPT).
28-35	UNUSED

10. USER PROCESS TABLE (UPT)

0-377 SEC = 0: USER PAGE TABLE WITH HALF WORD ENTRIES, ONE FOR EACH OF THE 512 VIRTUAL PAGES OF A USER PROCESS WITH NO EXTENDED ADDRESSING.

SEC = 1: AVAILABLE TO SOFTWARE.

400-417 SEC = 0: HALF WORD PAGE TABLE ENTRIES WHICH MAPS EXEC PAGES 340 THROUGH 377 ON A PER PROCESS BASIS. THE PER PROCESS EXEC MAPPED REGION IS USED BY THE OPERATING SYSTEM TO MAP PER PROCESS DATA, SUCH AS OPERATING SYSTEM PER-PROCESS VARIABLES AND THE UPT ITSELF.

SEC = 1: SECTION TABLE ENTRIES FOR EXEC SECTIONS 20-37. THUS EXEC SECTIONS 20-37 ARE MAPPED ON A PER PROCESS BASIS. WHEN SEC = 1, THERE ARE NO PER PROCESS PAGE MAP FOR USER SECTION REFERENCE TO EXEC PROCESS DIRECT

PAGES.

420 EXEC AND USER PAGE FAIL WORD STORED HERE ON A PAGE FAULT (KI: USER PAGE FAIL TRAP INSTRUCTION)

421 USER ARITHMETIC OVERFLOW TRAP INSTRUCTION

422 USER PUSHDOWN OVERFLOW TRAP INSTRUCTION

423 USER TRAP-3 TRAP INSTRUCTION

424 MUUO STORED HERE WHEN AN MUUO IS EXECUTED IN USER OR EXEC MODE

425 PC WORD OF MUUO STORED HERE WHEN AN MUUO IS EXECUTED IN USER OR EXEC MODE

426 PC WORD OF PAGE FAIL WORD STORED HERE ON A PAGE FAULT IN USER OR EXEC MODE (KI: EXEC PAGE FAIL WORD STORED HERE)

427 PAGE FAIL NEW PC WORD PICKED UP FROM HERE ON A PAGE FAULT FROM USER OR EXEC MODE (KI: USER PAGE FAIL WORD STORED HERE)

430 KERNEL NEW MUUO PC WORD WHEN NOT EXECUTED IN A TRAP LOCATION

431 KERNEL NEW MUUO PC WORD WHEN EXECUTED IN A TRAP LOCATION

432 SUPERVISOR NEW MUUO PC WORD WHEN NOT EXECUTED IN A TRAP LOCATION

433 SUPERVISOR NEW MUUO PC WORD WHEN EXECUTED IN A TRAP LOCATION

434 CONCEALED NEW MUUO PC WORD WHEN NOT EXECUTED IN A TRAP LOCATION

435 CONCEALED NEW MUUO PC WORD WHEN EXECUTED IN A TRAP LOCATION

436 PUBLIC NEW MUUO PC WORD WHEN NOT EXECUTED IN A TRAP LOCATION

437 PUBLIC NEW MUUO PC WORD WHEN EXECUTED IN A TRAP LOCATION

440-477 SEC = 0: AVAILABLE TO SOFTWARE

SEC = 1: SECTION TABLE ENTRIES FOR USER SECTIONS 0-37 [KI: AVAILABLE TO SOFTWARE]

500 CURRENT AC BLOCK # (CAC), PREVIOUS AC BLOCK # (PAC), PREVIOUS CONTEXT SECTION (PCS) AND CALLED WITH SECTION EXECUTE (CWSX) IS STORED HERE ON AN MUUO EXECUTED IN USER OR EXEC MODE. [KI: AVAILABLE TO SOFTWARE], SEE CHAPTER 2,11, CALLING THE MONITOR (MUUO, PXCT)

501-577 RESERVED FOR FUTURE USE BY HARDWARE [KI: AVAILABLE TO SOFTWARE]

600-777 AVAILABLE TO SOFTWARE

11, EXEC PROCESS TABLE (EPT)

0-37 4 WORD DATA CHANNEL LOGOUT FOR EACH OF THE 8 BUILT-IN DATA CHANNELS, SEE CHAPTER 4.1, MASSBUS CONTROLLER (RH20) AND CHANNELS (MBOX). [KI: AVAILABLE TO SOFTWARE]

0 INITIAL CHANNEL COMMAND

1 INCREMENTED CHANNEL COMMAND

2 STATUS + CHANNEL COMMAND LIST POINTER

3 RESERVED FOR FUTURE USE BY HARDWARE

40 EXEC LUUO STORED HERE

41 EXEC LUUO TRAP INSTRUCTION

42-57 2 WORD PAIRS OF STANDARD PI INTERRUPT INSTRUCTIONS

60-77 4 WORD DTE20 INTERRUPT AND BYTE POINTER LOCATIONS FOR EACH OF THE 4 DTE20 INTERFACES, SEE CHAPTER 4.3, FRONT END INTERFACE (DTE20). [KI: AVAILABLE TO SOFTWARE]

0 IN BYTE POINTER

1 OUT BYTE POINTER

2 INTERRUPT INSTRUCTION

3 UNUSED, RESERVED FOR FUTURE USE BY HARDWARE

100-177 RESERVED FOR FUTURE USE BY HARDWARE [KI: AVAILABLE TO SOFTWARE]

200-377 SEC = 0: EXEC PAGE TABLE WITH HALF WORD ENTRIES, ONE FOR EACH OF THE 256 EXEC VIRTUAL PAGES 400 THROUGH 777

SEC = 1: AVAILABLE TO SOFTWARE

400-417 SEC = 0: AVAILABLE TO SOFTWARE

SEC = 1: SECTION TABLE ENTRIES FOR EXEC SECTIONS 0-17, THESE SECTIONS ARE THE SAME FOR ALL PROCESSES AND MAP THE CODE AND DATA FOR THE COMMON PARTS OF THE OPERATING SYSTEM.

420 RESERVED FOR FUTURE USE BY HARDWARE [KI: EXEC PAGE FAIL TRAP INSTRUCTION]

421 EXEC ARITHMETIC OVERFLOW TRAP INSTRUCTION

422 EXEC PUSHDOWN OVERFLOW TRAP INSTRUCTION

423 EXEC TRAP-3 TRAP INSTRUCTION

424-427 A 4 WORD "HALT STATUS AREA" WHICH IS LOADED EACH TIME THE PROCESSOR IS HALTED, SEE CHAPTER 2.5, MISCELLANEOUS OPCODES:

424 HALT INSTRUCTION OR FULL WORD OF ZEROES

425 PROGRAM COUNTER

426-427 TWO WORDS OF PROCESSOR STATUS

430-777 AVAILABLE TO SOFTWARE

[END OF CH2S9,SPC]

1080,2040,2060 ENGINEERING FUNCTIONAL SPEC - CHAP 2,10

TO: KL10 LIST

TITLE: THE EXTENDED INSTRUCTION SET - REV 2

STATUS: THIS CHAPTER IS RE-WRITTEN TO REFLECT THE RE-DESIGN OF THE STRING INSTRUCTION TO BE THE EXTENDED INSTRUCTION SET. IT INCLUDES RE-DEFINITION OF AC USAGE FOR COMPATABILITY WITH EXTENDED ADDRESSING.

FILE: [EFS]CH2S10.SPC

PDM #: 200-200-007-02

DATE: 5 DEC 75

SUPERSEDED MEMOS: 2P10R0, 22 OCT 74, T. HASTINGS; STRING INSTRUCTION, J. LEONARD, 31 MAY 73; EDIT, J. LEONARD, 22 JUNE 73; BDEC SUBROUTINE TO SIMULATE BINARY TO DECIMAL FUNCTIONS OF STRING, J. LEONARD, JUNE 73; EDIT PATTERN OPERATORS, J. LEONARD, SEPT 5, 74.

ENGINEER: J. LEONARD

APPROVED:

EDITOR: T. HASTINGS

TYPIST: M. PROUTY

REVIEWED:

ABSTRACT

THE EXTEND INSTRUCTION IS A MULTI-FUNCTIONED INSTRUCTION WHICH IS THE MAJOR FEATURE OF THE BUSINESS INSTRUCTION SET (BIS). EXTEND PERFORMS CHARACTER STRING EDITING, DECIMAL TO BINARY CONVERSION, BINARY TO DECIMAL CONVERSION, STRING MOVE, TRANSLATION, AND STRING COMPARE.

REVISION HISTORY

REV DESCRIPTION CHG NO ORIG DATE APPD BY DATE

1.0 INTRODUCTION

THIS CHAPTER ATTEMPTS TO DOCUMENT THE NEW INSTRUCTION EXTEND, (OPCODE 123), WHICH IS THE MAJOR FEATURE OF THE KL10 BUSINESS INSTRUCTION SET.

OPCODE 123 IS THE "EXTEND" INSTRUCTION, WHICH SHOULD BE REGARDED AS AN XCT WHICH INVOKES THE EXTENDED INSTRUCTION SET. ITS AC FIELD ADDRESSES A BLOCK OF 5 AC'S TO BE USED IN THE OPERATION OF THE EIS. THE EFFECTIVE ADDRESS OF THE EXTEND INSTRUCTION IS CALLED E0, AND ADDRESSES AN OPERATOR WORD IN THE PDP-10 INSTRUCTION FORMAT, WHICH MUST BE ONE OF THE SET DEFINED BELOW (OTHERWISE AN MUUO TRAP OCCURS). FOR CERTAIN OPCODES IN (E0), THE OPERATOR WORD IS FOLLOWED BY OPERATION PARAMETERS, REFERED TO IN THE FOLLOWING TEXT AS (E0+1), (E0+2), ETC. SOME OPERATIONS ALSO USE THE EFFECTIVE ADDRESS OF THE OPERATOR WORD, WHICH WILL BE REFERED TO BELOW AS E1.

THE OPERATIONS DEFINED IN THE EXTENDED INSTRUCTION SET ARE LISTED BELOW, WITH THEIR OPCODES--

001	CMPSL	COMPARE STRINGS, SKIP IF LESS
002	CMPSE	COMPARE STRINGS, SKIP IF EQUAL
003	CMPSLE	COMPARE STRINGS, SKIP IF LESS OR EQUAL
005	CMPSGE	COMPARE STRINGS, SKIP IF GREATER OR EQUAL
006	CMPSN	COMPARE STRINGS, SKIP IF NOT EQUAL
007	CMPSG	COMPARE STRINGS, SKIP IF GREATER
004	EDIT	PROCESS STRING ACCORDING TO PATTERN
010	CVTDBO	CONVERT DECIMAL TO BINARY BY OFFSET
011	CVTDBT	CONVERT DECIMAL TO BINARY BY TRANSLATION
012	CVTBDO	CONVERT BINARY TO DECIMAL BY OFFSET
013	CVTBDT	CONVERT BINARY TO DECIMAL BY TRANSLATION
014	MOVSO	MOVE STRING WITH BYTE OFFSET
015	MOVST	MOVE STRING WITH BYTE TRANSLATION
016	MOVSLJ	MOVE STRING UNMODIFIED WITH LEFT JUSTIFICATION
017	MOVSRJ	MOVE STRING UNMODIFIED WITH RIGHT JUSTIFICATION

2.0 DEFINITION OF TERMS

2.1 AC

AC IS THE REGISTER DESIGNATED BY BITS 9-12 OF THE EXTEND INSTRUCTION. THE INTERPRETATION OF AC DEPENDS OF THE PARTICULAR FUNCTION PERFORMED BY THE EXTEND INSTRUCTION. FOR COMPARES, MOVES, AND DECIMAL TO BINARY CONVERSION, THE CONTENTS OF AC ARE INTERPRETED AS FLAGS AND THE SOURCE LENGTH

INDICATION, WHICH IS ALWAYS EXPRESSED AS A NUMBER OF BYTES. IN EDIT, AC CONTAINS FLAGS AND THE ADDRESS OF A MINI-PROGRAM CALLED THE PATTERN. IN BINARY TO DECIMAL CONVERSION, AC CONTAINS THE HIGH-ORDER PART OF THE BINARY INTEGER TO BE CONVERTED.

2,2 AC+1

AC+1 IS THE REGISTER DESIGNATED BY BITS 9-12 OF THE EXTEND INSTRUCTION, PLUS 1 MOD 16. EXCEPT IN BINARY TO DECIMAL CONVERSION, AC+1 CONTAINS A FULL PDP-10 BYTE POINTER ADDRESSING THE SOURCE STRING. THE SOURCE STRING IS PROCESSED AS THOUGH BY ILDB TMP, AC+1. IN BINARY TO DECIMAL CONVERSION, AC+1 CONTAINS THE LOW-ORDER PART OF THE BINARY INTEGER TO BE CONVERTED.

2,3 AC+2

RESERVED FOR FUTURE USE BY DEC HARDWARE.

2,4 AC+3

AC+3 IS THE REGISTER DESIGNATED BY BITS 9-12 OF THE EXTEND INSTRUCTION, PLUS 3 MOD 16. IN COMPARES, MOVES, AND BINARY TO DECIMAL CONVERSION, AC+3 CONTAINS THE LENGTH (IN BYTES) OF THE DESTINATION STRING. IN DECIMAL TO BINARY CONVERSION, AC+3 CONTAINS THE HIGH-ORDER BINARY DATA.

2,5 AC+4

THE NEXT REGISTER AFTER AC+3. IN EDIT, BINARY TO DECIMAL CONVERSION, AND MOVE STRING, A BYTE POINTER TO THE DESTINATION STRING, WHICH IS PROCESSED AS THOUGH BY IDPB TMP, AC+4. IN COMPARE STRING, AC+4 CONTAINS A POINTER TO WHAT WE CALL THE DESTINATION STRING, BUT FOR (HOPEFULLY) OBVIOUS REASONS, WE LOAD BYTES FROM IT RATHER THAN DEPOSITING THEM. IN DECIMAL TO BINARY CONVERSION, AC+4 CONTAINS THE LOW-ORDER BINARY DATA.

2,6 E0

E0 IS THE EFFECTIVE ADDRESS OF THE EXTEND INSTRUCTION. (E0) IS THE WORD CONTAINING THE EXTENDED OPERATION CODE IN BITS 0-8.

2.7 E1

E1 IS THE EFFECTIVE ADDRESS COMPUTED FROM BITS 13-35 OF (E0).

2.8 OFFSET

OFFSET IS BYTE MODIFICATION BY ADDITION OF E1. IN THOSE INSTRUCTIONS WHICH INCLUDE IN THEIR NAMES THE WORD "OFFSET", BYTES FROM THE SOURCE STRING ARE MODIFIED BEFORE USE BY ADDING E1 TO THEM. FOR OFFSET OPERATIONS, E1 IS SIGN EXTENDED TO A FULL WORD BY DUPLICATING BIT 18 IN BITS 0-17.

2.9 TRANSLATED

THIS IS BYTE MODIFICATION BY TRANSLATION THROUGH A TABLE OF HALFWORDS LOCATED AT E1. THIS OCCURS IN EDIT, AND ALL THOSE INSTRUCTIONS WHICH INCLUDE THE WORD "TRANSLATED" IN THEIR NAMES. IN ADDITION TO PROVIDING THE TRANSLATION FUNCTION, THOSE INSTRUCTIONS WHICH USE TRANSLATION CAN CONTROL THE FLAGS IN AC, AND CAN DETECT SPECIAL CHARACTERS IN THE SOURCE STRING.

2.10 S FLAG

IN EDIT AND OTHER TRANSLATE-TYPE INSTRUCTION, BIT 0 OF AC IS CALLED THE S OR SIGNIFICANCE FLAG. IN THESE INSTRUCTION, THE SOURCE STRING IS SCANNED FROM LEFT TO RIGHT, TRANSLATING EACH CHARACTER, BUT AS LONG AS THE S FLAG IS 0, NO REFERENCE TO THE DESTINATION STRING IS MADE. THE S FLAG MAY BE SET BY THE PROGRAMMER WHEN THE INSTRUCTION IS STARTED, OR MAY BE SET BY THE DETECTION OF THE SIGNIFICANCE STARTER BIT IN THE TRANSLATE TABLE, OR IN EDIT, BY EXECUTION OF THE START SIGNIFICANCE OPERATOR. IN DECIMAL TO BINARY CONVERSION, THE S FLAG INDICATES THE SIGNIFICANCE OF THE BINARY NUMBER IN AC+3 AND AC+4. IF THE S FLAG IS 0 WHEN THE INSTRUCTION STARTS, STRING CLEARS AC+3 AND AC+4, AND SETS THE S FLAG. IN BINARY TO DECIMAL CONVERSION, THE S FLAG IS BIT 0 OF AC+3, AND INDICATES WHETHER LEADING ZEROS SHOULD BE STORED IN THE DESTINATION STRING TO FILL IT OUT TO THE LENGTH GIVEN BY RH(AC).

2.11 N FLAG

IN EDIT AND OTHER TRANSLATE-TYPE INSTRUCTIONS, BIT 1 OF AC IS CALLED THE N OR NON-ZERO FLAG. IT MAY BE SET BY THE PROGRAMMER WHEN THE INSTRUCTION IS STARTED, OR MAY BE SET BY DETECTION OF THE SIGNIFICANCE STARTER BIT IN THE TRANSLATE TABLE. IN BINARY TO DECIMAL CONVERSION, THE N FLAG IS BIT 1 OF AC+3, AND IS SET

IF THE BINARY NUMBER IN AC+3 AND AC+4 IS NOT ZERO,

2,12 M FLAG

IN EDIT AND OTHER TRANSLATE-TYPE INSTRUCTIONS, BIT 2 OF AC IS CALLED THE M OR MINUS FLAG. IT MAY BE SET BY THE PROGRAMMER, AND MAY BE SET, CLEARED, OR LEFT AS IS BY THE SIGN CONTROL BITS IN THE TRANSLATE TABLE. IN DECIMAL TO BINARY CONVERSION, THE BINARY RESULT IS NEGATED IF THE M FLAG IS SET AT THE END OF THE CONVERSION. IN BOTH FORMS OF BINARY TO DECIMAL CONVERSION, THE M FLAG IS BIT 2 OF AC+3. IT WILL BE SET AT THE BEGINNING OF THE INSTRUCTION IF THE BINARY NUMBER TO BE CONVERTED IS NEGATIVE.

2,13 FILL CHARACTER

MANY OF THE STRING OPERATIONS OF THE EXTENDED INSTRUCTION SET "FILL" A STRING TO MEET SOME LENGTH CONSTRAINT. FOR THIS PURPOSE THE WORD AT E0+1 IS USED AS THE FILL CHARACTER.

2,14 MESSAGE CHARACTER

EDIT PATTERNS MAY CALL FOR PRE-DEFINED SPECIAL CHARACTERS TO BE INSERTED INTO THE DESTINATION STRING. UP TO 64 SUCH CHARACTERS MAY BE DEFINED FOR ANY EDITING OPERATION, OF WHICH TWO ARE THE FILL AND FLOAT CHARACTERS USED IN THE OPERATION.

2,15 DOUBLE-PRECISION BINARY INTEGER

THE BINARY INTEGERS ACCEPTED AS INPUT BY BINARY TO DECIMAL CONVERSION, AND PRODUCED BY DECIMAL TO BINARY CONVERSION HAVE THE SAME FORMAT AS DOUBLE-WORD INTEGERS PRODUCED AND ACCEPTED BY OTHER INSTRUCTIONS IN THE KL10 (NAMELY MUL, DIV, AND THE DOUBLE-WORD INTEGER ARITHMETIC INSTRUCTIONS DADD, DSUB, DMUL, DDIV)--

INTEGERS ARE REPRESENTED IN TWO'S COMPLEMENT IN TWO ADJACENT WORDS. THE MOST SIGNIFICANT PART IS IN THE WORD WITH LOWEST ADDRESS, AND THAT WORD CARRIES THE SIGN OF THE ENTIRE QUANTITY. THE LEAST SIGNIFICANT PART OCCUPIES BITS 1-35 OF THE WORD WITH HIGHESR ADDRESS. BIT 0 OF THAT WORD IS IGNORED IN ALL OPERANDS, AND IS MADE EQUAL TO THE SIGN OF THE ANSWER IN ALL RESULTS.

3,0 STRING FUNCTIONS

HERE ARE DETAILED DESCRIPTIONS OF THE VARIOUS OPERATIONS DEFINED IN THE EXTENDED INSTRUCTION SET: SIMULATION ROUTINES, DEFINING THESE OPERATIONS IN TERMS OF THE KA/KI INSTRUCTION SET, MAY BE FOUND IN SECTION 5,

3,1 STRING COMPARES

AC BITS 9-35 SPECIFY THE LENGTH OF THE SOURCE STRING, COUNTING IN BYTES, AC+1 CONTAINS A BYTE POINTER TO THE "SOURCE STRING", AC+3 BITS 9-35 SPECIFY THE LENGTH OF THE DESTINATION STRING, AND AC+4 CONTAINS A BYTE POINTER TO THE "DESTINATION STRING", BITS 0-8 OF AC AND AC+3 MUST BE ZERO,

THE SOURCE STRING IS COMPARED AGAINST THE DESTINATION STRING, AND THE INSTRUCTION SKIPS IF THE SPECIFIED SKIP CONDITION IS MET, (E,G, CMPSL SKIPS IF THE SOURCE STRING IS LESS THAN THE DESTINATION STRING), IF THE STRING LENGTHS DIFFER, THE COMPARISON PROCEEDS AS THOUGH THE SHORTER STRING WERE EXTENDED BY FILLING ON THE RIGHT TO THE LENGTH OF THE LONGER STRING, IF THE SOURCE STRING IS THE SHORTER, IT IS EFFECTIVELY EXTENDED WITH BYTES EQUAL TO (E0+1), IF THE DESTINATION STRING IS THE SHORTER, IT IS EXTENDED WITH (E0+2),

3,2 EDIT

AC CONTAINS FLAGS AND THE ADDRESS OF THE PATTERN STRING, AC+1 CONTAINS A BYTE POINTER TO THE SOURCE STRING, AC+3 CONTAINS THE ADDRESS AT WHICH AC+4 IS STORED AT THE TIME SIGNIFICANCE BEGINS, THIS IS CALLED THE MARK POINTER, AC+4 CONTAINS A BYTE POINTER TO THE DESTINATION STRING,

THE PATTERN STRING IS A SEQUENCE OF MINIOPERATIONS WHICH CONTROLS THE CREATION OF THE DESTINATION STRING FROM THE SOURCE STRING, E1 IS THE ADDRESS OF THE TRANSLATION TABLE TO BE USED FOR BYTES SELECTED FROM THE SOURCE STRING, E0+2 CONTAINS THE FLOAT CHARACTER TO BE STORED WHEN SIGNIFICANCE IS STARTED, OR ZERO IF NO FLOAT CHARACTER IS TO BE STORED, E0+1 CONTAINS THE FILL CHARACTER TO BE STORED ON EACH REFERENCE TO THE DESTINATION STRING WHILE SIGNIFICANCE IS OFF, OR ZERO IF THERE IS TO BE NO FILL CHARACTER, MESSAGE CHARACTERS ARE STORED FOLLOWING E0, WITH THE NTH CHARACTER AT E0+N+1,

3,3 CONVERT DECIMAL TO BINARY

CVTDBO AND CVTDBT USE A SOURCE STRING SPECIFIED BY A BYTE POINTER IN AC+1 AND A LENGTH IN AC, PROCESSING IT AS A DECIMAL

DIGIT STRING FROM LEFT TO RIGHT, CONVERTING DIGITS TO A BINARY INTEGER IN AC+3 AND AC+4 UNTIL THE STRING LENGTH IS EXHAUSTED OR A NON-DIGIT IS FOUND IN THE STRING. IN CVTDBO, A NON-DIGIT IS ANY BYTE IN THE SOURCE STRING WHOSE OFFSET VALUE IS NOT IN THE RANGE 0 TO 9. IN CVTDBT, A NON-DIGIT IS ANY BYTE WHOSE TRANSLATION FUNCTION CONTAINS THE ABORT CODE, OR FOR WHICH THE LOW 4 BITS OF THE TRANSLATION FUNCTION EXCEED 9 WHEN SIGNIFICANCE IS ON.

3.4 CONVERT BINARY TO DECIMAL

CVTBDO AND CVTBDT GENERATE A DECIMAL DIGIT STRING AS SPECIFIED BY A BYTE POINTER IN AC+4 AND A LENGTH IN AC+3, REPRESENTING THE MAGNITUDE OF THE BINARY INTEGER IN AC AND AC+1. IN CVTBDO, THE OFFSET IS ADDED TO EACH DIGIT BEFORE STORING A BYTE. IN CVTBDT, DIGITS ARE USED AS AN INDEX INTO THE TRANSLATION TABLE AT E1 TO DETERMINE WHAT BYTE TO STORE FOR EACH. IF THE LENGTH IN AC+3 IS LESS THAN THE NUMBER OF DIGITS REQUIRED TO REPRESENT THE MAGNITUDE OF THE BINARY NUMBER IN AC AND AC+1, THE EXTEND INSTRUCTION TAKES A NON-SKIP EXIT WITHOUT STORING ANYTHING IN THE DESTINATION STRING. IF THE LENGTH IN AC+3 IS GREATER THAN THE NUMBER OF DIGITS REQUIRED TO REPRESENT THE BINARY, AND THE S FLAG IN AC+3 IS 1, BINARY TO DECIMAL CONVERSION STORES FILL CHARACTERS (E0+1) TO RIGHT-ALIGN THE DECIMAL INTEGER IN THE DESTINATION STRING. IF THE S FLAG IS ZERO, OR THE LENGTH IN AC+3 IS EQUAL TO THE NUMBER OF DIGITS REQUIRED, NO FILL CHARACTERS ARE STORED, AND THE DESTINATION STRING BEGINS WITH THE FIRST SIGNIFICANT DIGIT OF THE BINARY INTEGER.

3.5 MOVSO, MOVST

THESE INSTRUCTIONS COPY A BYTE STRING FROM SOURCE TO DESTINATION. MOVSO ADDS E1 TO EACH SOURCE BYTE BEFORE STORING IN THE DESTINATION STRING, AND MOVST TRANSLATES EACH SOURCE BYTE THROUGH A TABLE OF HALFWORDS AT E1. AC+1 CONTAINS THE SOURCE STRING BYTE POINTER, AC BITS 9-35 CONTAIN THE LENGTH OF THE SOURCE STRING, AC+4 CONTAINS THE DESTINATION STRING POINTER, AND AC+3 BITS 9-35 CONTAINS THE DESTINATION STRING LENGTH. IF THE SOURCE STRING IS LONGER THAN THE DESTINATION STRING, THE OPERATION TERMINATES AFTER TRANSFERING THE NUMBER OF BYTES SPECIFIED FOR THE DESTINATION STRING, WITH AC CONTAINING THE DIFFERENCE IN LENGTH. IF THE SOURCE STRING IS SHORTER THAN THE DESTINATION STRING, THE DESTINATION STRING IS FILLED OUT TO ITS FULL LENGTH WITH BYTES EQUAL TO (E0+1).

3.6 MOVSLJ, MOVSRJ

THESE INSTRUCTIONS COPY A SOURCE STRING TO A DESTINATION STRING

WITHOUT MODIFICATION OF BYTES, WITH AN OPTION FOR LEFT OR RIGHT JUSTIFICATION OF THE SOURCE STRING WITHIN THE DESTINATION STRING. THEY ARE EQUIVALENT TO MOVSO WITH AN OFFSET OF ZERO, EXCEPT THAT MOVSRJ STORES FILL CHARACTERS (E0+1) IN THE DESTINATION STRING BEFORE TRANSFERING THE SOURCE STRING, IF THE SOURCE STRING IS SHORTER THAN THE DESTINATION STRING, OR SKIPS OVER THE INITIAL CHARACTERS OF THE SOURCE STRING IF THE SOURCE STRING IS LONGER THAN THE DESTINATION STRING.

MOVSLJ AND MOVSRJ SHOULD BE USED FOR THEIR SPEED ADVANTAGE IN CASES WHERE NO BYTE MODIFICATION IS NEEDED, RATHER THAN MOVSO WITH AN OFFSET OF ZERO.

4.0 EDIT TUTORIAL

EDIT IS INTENDED TO PROVIDE THE FACILITIES NEEDED, PARTICULARLY IN COBOL AND PL/I, TO CREATE FORMATTED CHARACTER STRINGS FOR OUTPUT. THE PRIMARY FEATURES REQUIRED ARE BLANKING OF LEADING ZEROS, AND INSERTION OF A CURRENCY SYMBOL IMMEDIATELY TO THE LEFT OF THE MOST SIGNIFICANT DIGIT. EDIT PROVIDES OTHER USEFUL FEATURES, BUT LET'S START WITH A SIMPLE EDITING EXAMPLE TO SEE HOW IT WORKS. IN ORDER TO UNDERSTAND IT, YOU WILL WANT TO TRACE YOUR WAY THROUGH THE EDIT SIMULATOR IN SECTION 5 AS YOU READ THE DESCRIPTION.

```
;PATTERN CODE AND TRANSLATE TABLE DEFINITIONS

STOP==0          ;TERMINATE EDIT, EXTEND INSTR SKIPS
SELECT==1        ;SELECT NEXT SOURCE BYTE
SIGST==2         ;FORCE SIGNIFICANCE
FLDSEP==3        ;FIELD SEPARATOR
EXCHMD==4        ;EXCH MARK AND DEST POINTERS (AC+3 AND AC+4)

MESSAG==100      ;INSERT MESSAGE CHARACTER
                 ;LOW 6 BITS SELECT ONE OF E0+1 THRU E0+100

SKPM==500        ;SKIP IF M FLAG SET
SKPN==600        ;SKIP IF N FLAG SET
SKPA==700        ;SKIP ALWAYS
                 ;SUCCESSFUL SKIPS ADVANCE OVER 1 TO 64 PATTERN
                 ; BYTES AS SELECTED BY LOW 6 BITS OF OPERATOR

SBIT==400000     ;SET S AND N FLAGS
ABORT==100000    ;ABORT EDIT (NO SKIP)
MCLR==200000     ;CLEAR M FLAG
MSET==300000     ;SET M FLAG

        SETZM   DEST            ;CLEAR SPACE FOR OUTPUT STRING
        SETZM   DEST+1
        MOVEI   AC,PATERN       ;SETUP PATTERN ADDRESS IN AC
        MOVE    AC+1,SRCPNT     ;SOURCE STRING POINTER
        MOVEI   AC+3,MARK       ;WHERE TO STORE SIG PTR
```

```
        MOVE    AC+4,DSTPNT     ;DESTINATION STRING POINTER
        EXTEND  AC,XOP          ;EDIT USING TABLE
        HALT                    ;SOMETHING'S WRONG...
        OUTSTR  DEST            ;SHOW OFF THE RESULT

PATERN: BYTE    (9)SELECT,SELECT,SELECT,SELECT,STOP

SRCPNT: POINT   4,SOURCE        ;ILDB POINTER TO INPUT DATA
                                ; (BCD DIGITS)
DSTPNT: POINT   7,DEST          ;IDPB POINTER TO ASCII OUTPUT

XOP:    EDIT,,TABLE             ;EXTENDED OP EDIT, USING TABLE
        0                       ;INITIALLY, PARAMETERS ALL ZERO
        0
TABLE:  XWD     "0",SBIT!"1"
        XWD     SBIT!"2",SBIT!"3"
        XWD     SBIT!"4",SBIT!"5"
        XWD     SBIT!"6",SBIT!"7"
        XWD     SBIT!"8",SBIT!"9"
REPEAT  3,<XWD  ABORT,ABORT>

MARK:   BLOCK   1               ;SAVED COPY OF DESTINATION PTR
                                ; AT SIGNIFICANCE START TIME
DEST:   BLOCK   2               ;OUTPUT STRING WILL GO HERE
```

THE SIZE OF PATTERN BYTES IS (BY DEFINITION OF EDIT) ALWAYS 9 BITS. SOURCE AND DESTINATION BYTES ARE DEFINED ENTIRELY BY THEIR RESPECTIVE BYTE POINTERS, IN THE USUAL -10 FORM. I HAVE SET UP SRCPNT WITH BYTE SIZE 4, BECAUSE I AM ASSUMING THE SOURCE STRING TO BE PACKED BCD DIGITS. IT COULD AS CONVENIENTLY BE SIXBIT, ASCII, OR EBCDIC (SORRY, RADIX50 WON'T MAKE IT). USING THE BCD ASSUMPTION, CONSIDER THE FOLLOWING SOURCE STRING --

```
SOURCE:  BYTE (4)0,1,0,6
```

EXTEND WILL BEGIN BY DECODING THE WORD AT ITS EFFECTIVE ADDRESS (XOP), AND WILL THEREFORE PERFORM AN EDIT. SINCE AC BITS 4-5 ARE ZERO, THE INITIAL PATTERN BYTE WILL BE TAKEN FROM BITS 0-8 OF THE WORD AT "PATERN"; IE, IT WILL PERFORM "SELECT". THE FIRST BYTE (0) WILL BE LOADED FROM "SOURCE", AND TRANSLATED THROUGH THE LEFT HALF OF LOCATION "TABLE" (LEFT HALF, BECAUSE THE LOW-ORDER BIT OF THE BYTE IS 0, AND LOCATION TABLE BECAUSE THE HIGH ORDER BITS OF THE BYTE ARE 0, WHICH WHEN ADDED TO E1, GIVES THE ADDRESS "TABLE"). THE S FLAG IN AC IS OFF, AND THE S BIT IN THIS TABLE HALFWORD IS NOT SET, SO EDIT ATTEMPTS TO STORE A FILL CHARACTER. E0+1 (THE FILL CHARACTER) IS ZERO, HOWEVER, SO NO CHARACTER IS STORED, AND EDIT PROCEDES TO THE NEXT PATTERN BYTE.

HERE IS ANOTHER SELECT, SO THE SECOND SOURCE DIGIT (1) IS OBTAINED AND TRANSLATED. TRANSLATION YIELDS THE RIGHT HALF OF LOCATION TABLE, WHERE THE S BIT IS SET, SO EDIT COPIES THE

DESTINATION POINTER (AC+4) TO THE MARK POINTER (ADDRESSED BY AC+3), SETS THE S FLAG IN AC, AND ATTEMPTS TO STORE THE FLOAT CHARACTER. THE CONTENTS OF E0+2 IS ZERO, HOWEVER, SO NO FLOAT CHARACTER IS STORED, AND FINALLY THE TRANSLATE FUNCTION (IE, ASCII "1") IS STORED AS THE FIRST CHARACTER IN THE DESTINATION STRING.

THE NEXT PATTERN BYTE IS SELECT, SO EDIT LOOKS AT THE NEXT DIGIT (0) IN THE SOURCE STRING, AND TRANSLATES IT THROUGH TABLE. AT THIS TIME THE S FLAG IS 1, SO THE TRANSLATE FUNCTION (ASCII "0") IS STORED IN THE DESTINATION STRING.

THE FOURTH PATTERN BYTE IS SELECT AGAIN, SO EDIT TRANSLATES THE NEXT SOURCE BYTE (6) THROUGH THE LEFT HALF OF TABLE+3, AND STORES ASCII "6" AS THE THIRD CHARACTER OF THE DESTINATION STRING. THE NEXT PATTERN BYTE IS STOP, SO EXTEND TERMINATES WITH A SKIP, HAVING CREATED A STRING AT DEST EQUIVALENT TO ASCII "106". SIMILARLY, IF THE STRING AT SOURCE HAD BEEN

```
        BYTE (4)11,6,2,3
```

DEST WOULD HAVE BECOME ASCII "9623", AND IF SOURCE HAD BEEN

```
        BYTE (4)0,0,0,4
```

DEST WOULD BE ASCII "4".

SO FAR, I'VE TRIED TO SHOW HOW THE S BIT IN THE TABLE IS USED WITH THE S FLAG IN AC TO SUPPRESS LEADING ZEROS. THE S BIT IS SET IN EACH TABLE ENTRY WHICH CORRESPONDS TO A SOURCE CHARACTER WHICH SHOULD NOT BE SUPPRESSED. HOPEFULLY, IT'S ALSO CLEAR THAT THE INSTRUCTION KNOWS NOTHING ABOUT THE CHARACTER SET BEING USED, SO A TABLE CHANGE COULD BE USED TO SUPPRESS ANY LEADING CHARACTER OR CHARACTERS IN THE SOURCE STRING.

NOW LET'S CHANGE THE EXAMPLE --

```
XOP/    EDIT,,TABLE     ;SAME AS BEFORE
        "<SPACE>"       ;ENABLE ASCII SPACE AS FILL CHAR
        0
```

NOTE: THE SYMBOL <SPACE> IS USED IN THE FOLLOWING DISCUSSION TO REPRESENT A SINGLE SPACE, IN ORDER TO ELIMINATE THE PROBLEM OF COUNTING THEM WHEN THE NUMBER IS IMPORTANT.

HERE WE'VE CHANGED THE "PARAMETER LIST" FOR THE OPERATION TO INCLUDE A FILL CHARACTER. WALKING THROUGH EDIT AGAIN, WE FIND THAT SELECT PICKS UP THE FIRST SOURCE BYTE (0), TRANSLATES IT THROUGH THE LEFT HALF OF TABLE, AND FINDS THAT BOTH THE S FLAG IN AC AND THE S BIT IN THE TABLE ARE ZERO. IT THEREFORE FINDS THE FILL CHARACTER, WHICH IS NON-ZERO, AND STORES IT IN THE DESTINATION STRING. AFTER THIS POINT, THE SEQUENCE IS THE SAME AS IN THE EARLIER EXAMPLE, SO THE RESULTING DESTINATION STRING

IS NOW "<SPACE>106". SOURCE OF 11,6,2,3 WOULD HAVE RESULTED IN "9623", AND SOURCE OF 0,0,0,4 WOULD PRODUCE "<SPACE><SPACE><SPACE>4". THUS THE FILL CHARACTER IS USED TO PRODUCE A CONSTANT FIELD WIDTH IN THE OUTPUT STRING, WHILE SUPPRESSING INSIGNIFICANT LEADING CHARACTERS.

TO DEMONSTRATE USE OF THE FLOAT CHARACTER, LET'S CHANGE THE PARAMETER LIST AGAIN, ADDING A FLOAT CHARACTER --

```
XOP/    EDIT,,TABLE
        "<SPACE>"
        "$"             ;NOW HAVE FLOAT CHAR DOLLAR SIGN
```

RERUNNING THE EXAMPLE, WE SEE THAT THE LEADING ZERO IS AGAIN SUPPRESSED WITH A SPACE, BUT THE SECOND DIGIT OF THE SOURCE STRING (1) TRANSLATES THROUGH THE RIGHT HALF OF TABLE, IN WHICH THE S BIT IS ONE. SINCE THE S FLAG IN AC IS ZERO AT THIS TIME, EDIT RECOGNIZES THIS SITUATION AS SIGNIFICANCE START, SO IT COPIES THE DESTINATION POINTER TO THE MARK POINTER, SETS THE S FLAG IN AC, AND FINDS A NON-ZERO FLOAT CHARACTER. THIS CHARACTER IS STORED IN THE DESTINATION STRING, AND THEN THE TRANSLATED VALUE FROM TABLE IS STORED IN DEST. AFTER THIS POINT, EDITING PROCEDES AS BEFORE, RESULTING IN THE FOLLOWING STRING AT DEST: "<SPACE>$106". THE OTHER SOURCE STRINGS USED AS EXAMPLES WOULD YIELD DESTINATION STRINGS OF "$9623" AND "<SPACE><SPACE><SPACE>$4".

I'M SURE YOU'VE NOTICED THAT I DIDN'T USE A SOURCE STRING OF ALL ZEROS IN MY EARLIER EXAMPLES, AND THAT IF I HAD, I WOULD HAVE GOTTEN NULL RESULTS IN THE DESTINATION STRING. FOR THOSE WHO BELIEVE ZEROS TO BE SOMETIMES SIGNIFICANT, THEREFORE, WE HAVE A SIGNIFICANCE STARTER PATTERN CODE, WHICH TELLS EDIT TO SET THE S FLAG IN AC, INDEPENDENT OF THE SOURCE STRING; IN EFFECT, TO PRETEND IT HAS SIGNIFICANCE. IN THE FIRST EXAMPLE, IF THE SOURCE STRING HAD BEEN 0,0,0,0 NOTHING WOULD HAVE BEEN STORED IN THE DESTINATION STRING. CHANGING THE PATTERN TO

```
PATERN/ SELECT,SELECT,SELECT,SIGST,SELECT,STOP
```

GIVES THE SAME RESULTS FOR THE SOURCE STRINGS USED AS EXAMPLES, BUT FOR 0,0,0,0 NOW GIVES ASCII "0" IN THE DESTINATION STRING. IF FILL CHARACTERS ARE USED, AS IN THE SECOND EXAMPLE, AN ALL-ZERO SOURCE GIVES A DESTINATION STRING CONSISTING ENTIRELY OF FILL CHARACTERS, WHILE INSERTING A SIGST PATTERN OPERATOR JUST BEFORE THE FINAL SELECT GIVES "<SPACE><SPACE><SPACE>0".

NOW LET'S ADD TO THE PARAMETER LIST TO DEMONSTRATE USE OF MESSAGE CHARACTERS --

```
XOP/    EDIT,,TABLE
        "<SPACE>"
        "$"
        ","             ;MAKE COMMA AVAILABLE FOR MESSAGE 2
        "."             ; PERIOD, TOO, AS MESSAGE 3
```

NOW WE CAN CREATE SOME SNAZZIER FORMATS STILL, FOR EXAMPLE,

```
PATERN/ SELECT,MESSAG+2,SELECT,SELECT
        SIGST,SELECT,STOP
```

WITH THE ORIGINAL SOURCE STRING (0,1,0,6) EDIT SEES THE S FLAG OFF WHEN FINDING THE MESSAG CODE IN THE PATTERN, SO IT STORES A FILL CHARACTER, RESULTING IN A DESTINATION STRING OF "<SPACE><SPACE>$106". USED ON THE SECOND SAMPLE STRING (11,6,2,3), HOWEVER, THE MESSAG CODE IS FOUND AFTER THE S FLAG HAS BEEN SET, SO EDIT INSERTS A COMMA, THE DESIGNATED MESSAGE CHARACTER, FOR THIS RESULT: "$9,623". ANOTHER PATTERN

```
PATERN/ SELECT,SELECT,SIGST,MESSAG+3
        SELECT,SELECT,STOP
```

FORCES SIGNIFICANCE BEFORE THE MESSAG CODE, SO A PERIOD WILL ALWAYS BE INSERTED, YIELDING THESE RESULTS ON THE SAMPLE SOURCE STRINGS: "<SPACE>$1.06", "$96.23", AND "<SPACE><SPACE>$.04". MESSAGE CHARACTERS, IN GENERAL, PROVIDE FOR FORMATTING THE OUTPUT BETWEEN SIGNIFICANT CHARACTERS DERIVED FROM THE SOURCE STRING.

YOU'RE PROBABLY GETTING AS BORED WITH THIS EXAMPLE AS I AM, SO LET'S TAKE ON A DIFFERENT KIND OF PROBLEM: CREATE AN OCTAL OUTPUT STRING IN DDT'S "HALFWORD" FORM (IE, LEFT HALF,,RIGHT HALF), WITH LEADING ZEROS SUPPRESSED IN BOTH HALVES. LET'S EVEN GO SO FAR AS TO SUPPRESS THE LEFT HALF AND DOUBLE COMMA ENTIRELY WHEN THE LEFT HALF IS ZERO.

WE'LL NEED A NEW SOURCE POINTER, SINCE WE ARE INTERESTED IN 3-BIT BYTES (OCTAL DIGITS) RATHER THAN BCD DIGITS, AND OF COURSE THE PATTERN WILL BE DIFFERENT, BUT THE SAME TABLE AND DESTINATION POINTER WILL SERVE QUITE ADEQUATELY.

```
SRCPNT/ POINT   3,SOURCE

PATERN/ SELECT,SELECT,SELECT,SELECT
        SELECT,SELECT,MESSAG+2,MESSAG+2
        FLDSEP,SELECT,SELECT,SELECT
        SELECT,SELECT,SIGST,SELECT
        STOP
```

WE WANT NEITHER FILL NOR FLOAT CHARACTERS, SO THE PARAMETER LIST BECOMES --

```
XOP/    EDIT,,TABLE
        0               ;NO FILL
        0               ;NO FLOAT
        ","             ;COMMA AS MESSAGE 2
```

THE FIRST 2 PATTERN WORDS CONTAINS SIX SELECT CODES, CORRESPONDING TO THE SIX OCTAL DIGITS IN THE LEFT HALFWORD OF THE SOURCE. PRESUMABLY, IT'S CLEAR HOW THE OCTAL DIGITS ARE

TRANSLATED THROUGH OUR FORMERLY BCD TABLE, AND FROM THE EARLIER DISCUSSION I ASSUME YOU RECOGNIZE THAT NOTHING WILL BE STORED BY THE FIRST 6 SELECTS IF THE LEFT HALF OF THE SOURCE WORD IS ZERO. REMEMBERING, NOW, THAT MESSAG ATTEMPTS TO STORE A FILL CHARACTER IF THE S FLAG IS OFF, AND THAT THE FILL CHARACTER IS ZERO, YOU CAN SEE THAT NEITHER DIGITS NOR COMMAS WILL BE STORED IF THE LEFT HALF IS ZERO, BUT THAT COMMAS WILL BE STORED IF A SIGNIFICANT (READ NON-ZERO) OCTAL DIGIT IS FOUND IN THE LEFT HALF OF THE SOURCE WORD.

NEXT IN THE PATTERN, WE FIND FLDSEP, WHOSE SOLE FUNCTION IS TO CLEAR THE S, N, AND M FLAGS IN AC. THIS MEANS THAT THE EDITING OPERATION STARTS AFRESH WITH THE RIGHT HALF OF THE SOURCE WORD, SELECTING SOURCE DIGITS, BUT STORING NOTHING UNTIL SIGNIFICANCE IS FOUND IN THE RIGHT HALF. FINALLY, JUST IN CASE THE RIGHT HALF IS ZERO, WE THROW IN A SIGST OPERATOR BEFORE THE 12TH SELECT TO ENSURE THAT SOMETHING GETS OUTPUT.

COBOL HAS A DATA ATTRIBUTE KNOWN AS "BLANK-WHEN-ZERO". IT MEANS THAT AN EDIT OUTPUT FIELD HAVING THAT SPECIFICATION MUST BE TOTALLY BLANKED IF THE DATA IS ZERO, SO THAT THE QUANTITY ".05" WOULD BE REPRESENTED AS SHOWN, BUT ".00" MUST BE BLANKED. IN ORDER TO CREATE ".05" WITH EDIT, WE WOULD FORCE SIGNIFICANCE, USING SIGST, BEFORE THE MESSAG CODE WHICH OUTPUTS THE PERIOD, THEREBY HAVING THE S FLAG ON SO WE GET ".05" RATHER THAN "<SPACE><SPACE>5". HAVING SET IT ARBITRARILY, HOWEVER, WE CAN NO LONGER USE IT TO INDICATE WHETHER THE DATA WAS NON-ZERO. THEREFORE, WE DEPEND ON THE N FLAG, WHICH LIKE THE S FLAG, IS SET BY THE S BIT IN THE TABLE, BUT CANNOT BE TURNED ON BY A PATTERN OPERATOR. THE SKIP OPERATORS IN THE PATTERN ALLOW US TO SPECIFY CERTAIN EDITING OPERATIONS WHICH ARE TO BE OMMITTED WHEN THE SKIP CONDITION IS MET. IN PARTICULAR, SKPN PERMITS EDIT TO OMIT PATTERN STEPS WHICH BLANK THE ENTIRE FIELD, IF THE DATA IS NON-ZERO. ALSO, EXCHMD GIVES US A WAY OF RESETING THE DESTINATION POINTER TO THE VALUE IT HAD WHEN WE STARTED SIGNIFICANCE, SO WE CAN GET BACK TO THE NON-BLANK CHARACTERS. CONSIDER:

```
PATERN/  SELECT,SELECT,SIGST,MESSAG+3
         SELECT,SELECT,SKPN+4,EXCHMD
         MESSAG+0,MESSAG+0,MESSAG+0,MESSAG+0
         STOP

XOP/     EDIT,,TABLE
         "<SPACE>"
         "$"
         ","
         "."
```

WHEN EDIT COMPLETES THE FIRST LINE OF PATTERN, THE DESTINATION STRING WILL BE ONE OF: "$NN.NN", "<SPACE>$N.NN", OR "<SPACE><SPACE>$.NN". IN THE THIRD CASE, IT IS POSSIBLE THAT THE DATA IS ZERO, AND THEREFORE THAT THE N FLAG IS NOT SET. IN ANY OTHER CASE, THE SKPN WILL CAUSE EDIT TO SKIP TO THE STOP

OPERATOR, AND THE FIELD WILL BE LEFT IN THE FORM SHOWN. IF, HOWEVER, THE DATA IS ZERO, THE SKPN WILL NOT SKIP, AND EXCHMD WILL RESET THE DESTINATION POINTER TO THE VALUE IT HAD WHEN SIGNIFICANCE WAS STARTED (IE, TO POINT TO THE BYTE PRECEDING THE "$"). THE FOUR OCCURANCES OF MESSAG+0 WILL STORE THE SELECTED MESSAGE CHARACTER (SPACE) OVER THE DOLLAR SIGN, PERIOD, AND BOTH ZEROS, LEAVING THE FIELD BLANK AS DESIRED.

IT SHOULD BE POINTED OUT THAT THE SKPX PATTERN OPERATORS SKIP OVER N+1 SUCCEEDING PATTERN BYTES, WHERE N IS THE VALUE OF THE SKIP-LENGTH BYTE. (EG, SKPN+0,X PERFORMS THE OPERATOR "X" IF AND ONLY IF THE N FLAG IS ZERO). THUS, A SINGLE SKIP OPERATOR MAY CONTROL THE EXECUTION OF UP TO 64 PATTERN BYTES. IN CASES WHERE MORE THAN 64 MUST BE CONTROLLED, THE CONDITIONAL PART OF THE PATTERN MAY BE BROKEN INTO SECTIONS, THUS:

...SKPX+J, J PATTERN BYTES,SKPA+0,SKPA+K, K+1 PATTERN BYTES...

IF THE SKPX+J SKIPS, IT TRANSFERS OVER THE FIRST GROUP OF J BYTES AND THE SKPA+0, TO LAND ON THE SKPA+K, WHICH TRANSFERS OVER THE SECOND GROUP. IF THE FIRST GROUP DOES NOT CHANGE THE TESTED FLAG, THEN THE FOLLOWING SEQUENCE WORKS:

...SKPX+J, J+1 PATTERN BYTES,SKPX+K, K+1 PATTERN BYTES...

AND OF COURSE, IF THE SKIP IS ONLY TRANSFERING TO A STOP OPERATOR, WE CAN RE-ARRANGE THE PATTERN TO:

...SKPX+0,SKPA+0,STOP, ... ANY NUMBER OF PATTERN BYTES,STOP

THIS LAST SEQUENCE ALSO DEMONSTRATES CONDITIONAL SKIPPING OVER A SKPA TO REVERSE THE SENSE OF A CONDITIONAL SKIP.

IT IS PERHAPS WORTHY OF NOTE, AT THIS POINT, THAT THE SET OF PATTERN OPERATORS CONTAINS NO MECHANISM FOR BRANCHING BACKWARDS (IE, LOOPING). THIS WAS A CONSCIOUS DECISION, BASED ON THE BELIEF THAT THE FACILITIES AVAILABLE FOR LOOP CONTROL ARE NOT REALLY SUFFICIENT, AND IT WOULD BE DIFFICULT TO ADD GOOD ONES. WE DO STILL HAVE RESERVED OPERATORS AVAILABLE FOR FUTURE EXPANSION, AND I WOULD BE PLEASED TO SEE A PROPOSAL WHICH ALLOWS LOOPING. THE MAJOR DIFFICULTY IS IN PRESERVING THE LOOP COUNT IN SUCH A WAY THAT THE INSTRUCTION CAN BE PROPERLY CONTINUED AFTER A PAGE FAULT OR INTERRUPT.

IT IS COMMON PRACTICE, IN THE CARD-ORIENTED WORLD OF BUSINESS DATA PROCESSING, TO INDICATE THE SIGN OF A NUMERIC FIELD IN A CARD BY OVERPUNCHING A DIGIT, USUALLY THE LOW-ORDER DIGIT, WITH A + (12 PUNCH) OR - (11 PUNCH). DUE TO THE WONDERFULNESS OF THE EBCDIC CARD CODE, THIS CONSTRAINS COBOL TO RECOGNIZING THE LETTERS A THROUGH I AS +1 TO +9, J THROUGH R AS -1 TO -9, AND A GRAND ASSORTMENT OF RANDOM CHARACTERS AS + OR -0. SINCE EDIT MAY FIND ITSELF PRESENTED WITH SUCH CHARACTERS, THE TRANSLATE TABLE COMES EQUIPPED WITH CONTROL FUNCTIONS WHICH CAN SET, CLEAR, OR LEAVE ALONE THE M FLAG, SO THAT AN APPROPRIATELY

DEFINED TABLE CAUSES THE M FLAG TO INDICATE THE SIGN OF THE SOURCE DATA. (THE FOURTH CONTROL FUNCTION FROM THE TABLE CAUSES IMMEDIATE TERMINATION OF THE EDIT, PERMITTING THE PROGRAM TO MAKE ANY INTERPRETATION WHATSOEVER OF THE SOURCE BYTE.)

SINCE WE'RE NOW TALKING ABOUT THE FULL CARD CHARACTER SET, WE'LL NEED A DIFFERENT TRANSLATION TABLE. ASSUMING 7-BIT ASCII SOURCE AND DESTINATION, THE TABLE LOOKS LIKE THIS--

```
TABLE:  REPEAT  20,<ABORT,,ABORT>       ;CONTROL CHARS 0-37
        XWD     "0",MSET!"0"                    ;SPACE=0, !=-0
        REPEAT  2,<ABORT,,ABORT>
        XWD     MCLR!"0",ABORT                  ;&=+0
        XWD     ABORT,ABORT
        XWD     ABORT,MCLR!"0"                  ;+=+0
        XWD     ABORT,MSET!"0"                  ;-=-0
        XWD     ABORT,ABORT
        XWD     "0",SBIT!"1"                    ;0,1
        XWD     SBIT!"2",SBIT!"3"               ;2,3
        XWD     SBIT!"4",SBIT!"5"               ;4,5
        XWD     SBIT!"6",SBIT!"7"               ;6,7
        XWD     SBIT!"8",SBIT!"9"               ;8,9
        REPEAT  2,<ABORT,,ABORT>
        XWD     ABORT,MCLR!"0"                  ;?=+0
        XWD     ABORT,SBIT!MCLR!"1"             ;A=+1
        XWD     SBIT!MCLR!"2",SBIT!MCLR!"3"     ;B=+2, C=+3
        XWD     SBIT!MCLR!"4",SBIT!MCLR!"5"
        XWD     SBIT!MCLR!"6",SBIT!MCLR!"7"
        XWD     SBIT!MCLR!"8",SBIT!MCLR!"9"     ;H=+8, I=+9
        XWD     SBIT!MSET!"1",SBIT!MSET!"2"     ;J=-1, K=-2
        XWD     SBIT!MSET!"3",SBIT!MSET!"4"
        XWD     SBIT!MSET!"5",SBIT!MSET!"6"
        XWD     SBIT!MSET!"7",SBIT!MSET!"8"
        XWD     SBIT!MSET!"9",ABORT             ;R=-9
        REPEAT  26,<ABORT,,ABORT>               ;T THRU 177
```

THE IMPORTANT FEATURES OF THIS TRANSLATION TABLE ARE:

1) STRANGE CHARACTERS FOR WHICH WE DO NOT HAVE CORRESPONDING DIGITS ARE MARKED WITH THE ABORT CODE. PRESUMABLY, IF EDIT ENCOUNTERED THEM IN THE SOURCE, WE WOULD ISSUE A DIAGNOSTIC MESSAGE TO THE USER.

2) CHARACTERS REPRESENTING NON-ZERO DIGITS HAVE TABLE ENTRIES IN WHICH THE S BIT IS SET. ENTRIES FOR REPRESENTATIONS OF ZERO HAVE THE S BIT ZERO.

3) CHARACTERS REPRESENTING EXPLICITLY SIGNED DIGITS HAVE TABLE ENTRIES WHICH EXPLICITLY SET OR CLEAR THE M FLAG. UNSIGNED DIGITS DO NOT TOUCH THE M FLAG, SO BY THE END OF THE EDIT, IT WILL EITHER DEFAULT TO ITS INITIAL VALUE, OR TAKE THE VALUE FORCED BY THE RIGHTMOST EXPLICITLY SIGNED DIGIT. IF WE WISHED TO IGNORE SIGNS

ON DIGITS OTHER THAN THE RIGHTMOST, WE WOULD MAKE ALL ENTRIES EXPLICITLY SET OR CLEAR THE M FLAG.

WITH THIS TABLE AND A SOMEWHAT EXTENDED PARAMETER LIST, HERE'S PART OF A PATTERN WHICH RECOGNIZES A NEGATIVE SOURCE AS A CREDIT, AND MARKS THE DESTINATION STRING WITH "CR" IN THAT CASE--

```
...SELECT,SKPM+2,MESSAG+0,MESSAG+0,STOP,MESSAG+4,MESSAG+5,STOP

XOP/    EDIT,,TABLE
        "<SPACE>"
        "$"
        ","
        "."
        "C"
        "R"
```

THE SKPM TESTS THE M FLAG TO CHOOSE WHICH OF TWO MESSAGES TO OUTPUT (EITHER "<SPACE><SPACE>" OR "CR"). THE TEST MIGHT, OF COURSE, BE COMBINED WITH A SKPN TO DO A THIRD OPERATION IF THE DATA WERE ZERO. NOTE THAT EDIT CONSIDERS DIGIT STRINGS TO BE REPRESENTED AS SIGN-MAGNITUDE, SO "-0" EXISTS AND MUST BE ALLOWED FOR.

TO EDIT FROM THIS CARD CODE TO A READABLE FORM WITH, SAY, SIX DIGITS AND TWO DECIMAL PLACES, ONE MIGHT USE A PATTERN LIKE--

```
PATERN/ SELECT,MESSAG+2,SELECT,SELECT
        SIGST,SELECT,MESSAG+3,SELECT
        SELECT,SKPN+0,STOP,SKPM+0
        STOP,EXCHMD,MESSAG+4,STOP

XOP/    EDIT,,TABLE
        "<SPACE>"
        "+"
        ","
        "."
        "-"
```

HERE, WE ASSUME POSITIVE DATA, AND SET UP A FLOATING "+". THEN, IF THE DATA TURNS OUT TO BE NON-ZERO AND NEGATIVE, WE STORE "-" (MESSAG+4) OVER THE FLOATING "+".

5.0 EXTEND SIMULATOR

THE FOLLOWING SUBROUTINE SIMULATES THE FUNCTIONS OF THE EXTEND INSTRUCTION: IT ASSUMES THE EXISTENCE OF A SET OF INVISIBLE AC'S, REFERED TO BY THE NAMES E0, E1, AND T1-T3, AND THAT THE AC NAMED "AC" IS THAT SPECIFIED BY BITS 9-12 OF THE EXTEND INSTRUCTION. E0 IS PRESUMED LOADED WITH THE EFFECTIVE ADDRESS OF THE EXTEND INSTRUCTION.

```
;BIT DEFINITIONS...

S.FLAG==400000   ;SIGNIFICANCE FLAG IN AC LEFT
N.FLAG==200000   ;NON-ZERO FLAG IN AC LEFT
M.FLAG==100000   ;MINUS FLAG IN AC LEFT

S.BIT==400000    ;SIGNIFICANCE STARTER BIT IN TRANSLATE TABLE

EXTEND: MOVEI   E1,@(E0)          ;COMPUTE OFFSET/TRANSLATE BASE
        LDB     T1,[POINT 9,(E0),8]     ;GET EXTENDED OPCODE
        CAIL    T1,20             ;IN DEFINED RANGE?
        JRST    UUO               ;NO, CONVERT TO UUO
        JRST    @.+1(T1)          ;DISPATCH TO HANDLER

        Z       UUO               ;000 UNDEFINED
        Z       CSL               ;001 COMPARE STRING, SKIP LESS
        Z       CSE               ;002  " , SKIP EQUAL
        Z       CSLE              ;003  " , SKIP LESS OR EQUAL
        Z       EDIT              ;004 EDIT
        Z       CSGE              ;005 COMPARE STRING, SKIP GE
        Z       CSN               ;006  " , SKIP NOT EQUAL
        Z       CSG               ;007  " , SKIP GREATER
        Z       DBO               ;010 DECIMAL TO BINARY, OFFSET
        Z       DBT               ;011  " , TRANSLATED
        Z       BDO               ;012 BINARY TO DECIMAL, OFFSET
        Z       BDT               ;013  " , TRANSLATED
        Z       MSO               ;014 MOVE STRING, OFFSET
        Z       MST               ;015  " , TRANSLATED
        Z       MSLJ              ;016  " , LEFT JUSTIFIED
        Z       MSRJ              ;017  " , RIGHT JUSTIFIED
```

[EDIT SIMULATION FOLLOWS -- OTHERS WILL BE PROVIDED AT A LATER DATE]

```
EDIT:   LDB     T3,PATBN          ;PICK UP PATTERN BYTE NO
EDITLP: LDB     T1,PATTBL(T3)     ;PICK UP PATTERN BYTE
        MOVEI   T2,(T1)           ;GET A COPY
        LSH     T2,-6             ;LOOK AT HIGH 3 BITS
        JRST    @.+1(T2)

        Z       EDOPR             ;000 OPERATE
        Z       MESSAG            ;100 SELECT MESSAGE CHARACTER
        Z       PNOP              ;200 RESERVED
        Z       PNOP              ;300 RESERVED
        Z       PNOP              ;400 RESERVED
        Z       SKPM              ;500 SKIP ON M FLAG
        Z       SKPN              ;600 SKIP ON N FLAG
        Z       SKPA              ;700 SKIP ALWAYS

EDOPR:  CAILE   T1,4              ;VALID OPERATE?
        JRST    PNOP              ;NO
        JRST    @.+1(T1)          ;YES, DO IT
```

```
        Z       EDSTOP  ;000 STOP
        Z       EDSEL   ;001 SELECT
        Z       SIGST   ;002 FORCE SIGNIFICANCE (S FLAG)
        Z       FLDSEP  ;003 FIELD SEPARATOR (CLEAR FLAGS)
        Z       EXCHMD  ;004 EXCHANGE MARK AND DEST POINTERS

PATBN:  POINT   2,AC,5          ;PAT BYTE NO IN AC

PATTBL: POINT   9,0(AC),8       ;PATTERN BYTE 0
        POINT   9,0(AC),17      ; BYTE 1
        POINT   9,0(AC),26      ; BYTE 2
        POINT   9,0(AC),35      ; BYTE 3

;HERE TO TERMINATE THE EDIT

EDSTOP: AOS     0(P)            ;CAUSE SKIP RETURN
EDABRT: PUSHJ   P,INCPBN        ;UPDATE PBN PAST THE STOP
        POPJ    P,              ;HERE ON ABORT CODE IN TABLE

;HERE TO GO ON TO NEXT OPERATION IN PATTERN

PNOP:   PUSHJ   P,INCPBN
        JRST    EDITLP

;HERE TO UPDATE PATTERN ADDRESS

INCPBN: ADDI    T3,1            ;INCREMENT IT
        MOVEI   T1,(T3)         ;GET A COPY
        LSH     T1,-2           ;DIVIDE BY 4 FOR WORD ADDR
        ADDI    AC,(T1)         ;UPDATE WORD ADDR IN AC
        ANDI    T3,3            ;KEEP POS'N IN WORD
        DPB     T3,PATBN        ;STUFF IT INTO AC
        POPJ    P,



;HERE WHEN PATTERN CODE IS SELECT

SELECT: ILDB    T1,AC+1         ;GET NEXT SOURCE BYTE
        ROT     T1,-1           ;DIVIDE BY 2, SAVING REMAINDER
        SKIPGE  T1              ;SKIP IF REMAINDER=0
        SKIPA   T1,E(T1)        ;LOW BIT WAS 1, USE RIGHT HALF
        MOVS    T1,E(T1)        ;USE LEFT HALFWORD

;WE NOW HAVE THE TRANSLATE FUNCTION FOR THE CURRENT SOURCE BYTE
; IN THE RIGHT HALF OF T1.  BITS 18-20 SPECIFY CONTROL
; FUNCTIONS,
; AND BITS 21-35 ARE THE TRANSLATED VALUE OF THE SOURCE BYTE.

        TRNE    T1,S,BIT        ;S BIT SET IN TABLE?
        TLO     AC,N,FLAG       ;YES, SET N FLAG

        LDB     T2,[POINT 2,T1,20]
        XCT     CTLTBL(T2)      ;PERFORM REQUESTED CONTROL
```

```
        JUMPL   AC,STDEST       ;IF S FLAG ALREADY 1, STORE
        TRNN    T1,S.BIT        ;IS S BIT SET IN TABLE?
        JRST    STFILL          ;NO, TRY TO STORE FILL CHAR
        TLO     AC,S.FLAG       ;YES, SET S FLAG
        PUSHJ   P,STFLOT        ;MARK THIS, AND TRY FLOAT

STDEST: ANDI    T1,077777       ;KEEP ONLY XLATE FUNCTION VALUE
STUFF:  IDPB    T1,AC+4         ;STORE IT IN DESTINATION STRING
        JRST    PNOP            ;DONE

;TABLE OF CONTROL OPERATIONS FOR TRANSLATE FUNCTIONS

CTLTBL: JFCL                    ;0, DO NOTHING
        JRST    EDABRT          ;1, ABORT THE EDIT
        TLZ     AC,M.FLAG       ;2, CLEAR M FLAG
        TLO     AC,M.FLAG       ;3, SET M FLAG

;HERE WHEN PATTERN CODE SAYS EXCHANGE MARK AND DEST POINTERS

EXCHMD: EXCH    AC+4,0(AC+3)
        JRST    PNOP            ;GO TO NEXT

;HERE WHEN PATTERN WANTS A MESSAGE CHARACTER OUTPUT

MESSAG: JUMPGE  AC,STFILL       ;IF NO SIGNIFICANCE,
                                ; PUT FILL CHAR
        ANDI    T1,77           ;GET MESSAGE SELECTOR
        ADDI    T1,(E0)         ;INDEX INTO PARAMETER LIST
        MOVE    T1,1(T1)        ;PICK UP MESSAGE CHAR
        JRST    STUFF           ;PUT IT INTO DEST STRING


;HERE TO GET AND STORE FILL CHAR, IF ANY

STFILL: SKIPN   T1,1(E0)        ;IS THERE A FILL CHAR?
        JRST    PNOP            ;NO
        JRST    STUFF           ;YES, STORE IT

;HERE IS SUBROUTINE TO STORE FLOAT CHAR

STFLOT: MOVEM   AC+4,0(AC+3)    ;COPY DEST TO MARK POINTER
        SKIPE   T2,2(E0)        ;IF THERE'S A FLOAT CHAR,
        IDPB    T2,AC+4         ;STORE IN DESTINATION STRING
CPOPJ:  POPJ    P,              ;RETURN

;HERE WHEN PATTERN FORCES SIGNIFICANCE

SIGST:  TLON    AC,S.FLAG       ;FORCE AND TEST S FLAG
        PUSHJ   P,STFLOT        ;IT WAS OFF, STORE FLOAT CHAR
        JRST    PNOP

;HERE WHEN PATTERN INDICATES START OF NEW FIELD
```

```
FLDSEP: TLZ     AC,S.FLAG!N.FLAG!M.FLAG ;CLEAR ALL FLAGS
        JRST    PNOP

;HERE FOR SKIPS

SKPM:   TLNN    AC,M.FLAG       ;TEST M FLAG
        JRST    PNOP            ;WAS OFF, IGNORE SKIP LENGTH
        JRST    SKPA            ;TAKE THE SKIP

SKPN:   TLNN    AC,N.FLAG       ;TEST N FLAG
        JRST    PNOP            ;OFF, IGNORE SKIP

SKPA:   ANDI    T1,77
        ADDI    T3,1(T1)        ;ADD THE SKIP DISTANCE
        PUSHJ   P,INCPBN        ;RESOLVE TO NEXT PATTERN BYTE
        JRST    EDITLP


;HERE FOR COMPARES

CSL:    PUSHJ   P,CMPS
        JUMPL   T1,CPOPJ1       ;SKIP IF LESS
        POPJ    P,

CSE:    PUSHJ   P,CMPS
        JUMPE   T1,CPOPJ1       ;SKIP IF EQUAL
        POPJ    P,

CSLE:   PUSHJ   P,CMPS
        JUMPLE  T1,CPOPJ1
        POPJ    P,

CSGE:   PUSHJ   P,CMPS
        JUMPGE  T1,CPOPJ1
        POPJ    P,

CSN:    PUSHJ   P,CMPS
        JUMPN   T1,CPOPJ1
        POPJ    P,

CSG:    PUSHJ   P,CMPS
        JUMPG   T1,CPOPJ1
        POPJ    P,

;HERE FOR ALL STRING COMPARES
; COMMON CODE RETURNS COMPARE CONDITION IN T1

CMPS:   TLNN    AC,777000       ;SRC LENGTH IS AC9-35
        TLNE    AC+3,777000     ;DEST LEN IS AC+3 9-35
        JRST    UUO
        JUMPE   AC,CMP5         ;FILL SRC IF SRC LEN ZERO
        JUMPE   AC+3,CMP3       ;FILL DST IF DST LEN ZERO
```

```
CMP2:   ILDB    T1,AC+1         ;GET SOURCE BYTE
        ILDB    T2,AC+4         ; AND DESTINATION BYTE
        CAME    T1,T2           ;EQUAL?
        JRST    CMPNE           ;NO, STOP HERE
        SOJLE   AC,CMP6         ;IS SRC LEN EXHAUSTED?
        SOJG    AC+3,CMP2       ;NO, HOW ABOUT DEST?

CMP3:   MOVE    T2,2(E0)        ;FILL DEST & COMPARE REMAINING
                                ;SOURCE
CMP4:   ILDB    T1,AC+1         ;GET SRC BYTE
        CAME    T1,T2           ; IS IT EQ DEST FILLER?
        JRST    CMPNE           ;NO, STOP
        SOJG    AC,CMP4         ;YES, LOOP FOR REST OF SRC
        JRST    CMPEQ           ;STRINGS EQUAL...

CMP5:   JUMPE   AC+3,CMPEQ      ;SRC & DEST BOTH ZERO LENGTH?
        JRST    CMP7            ;NO, FILL SRC
CMP6:   SOJLE   AC+3,CMPEQ      ;DEST EXHAUSTED TOO?
CMP7:   MOVE    T1,1(E0)        ;GET SRC FILL CHAR
CMP8:   ILDB    T2,AC+4         ;GET NEXT DEST BYTE
        CAME    T1,T2           ;SRC FILL EQ DEST BYTE?
        JRST    CMPNE           ; NO
        SOJG    AC+3,CMP8       ;YES, LOOP FOR REST OF DEST
                                ;STRING
CMPEQ:  MOVEI   T1,0            ;RETURN EQUALITY
        POPJ    P,

CMPNE:  CAMG    T1,T2           ;SRC GREATER THAN DEST?
        TLOA    T1,400000       ;NO, RETURN LESS
        MOVEI   T1,1            ;YES, RETURN GREATER
        POPJ    P,
```

5.1 TRANSLATE TABLE FORMAT --

ADDR	LEFT HALF	RIGHT HALF
E1-->	TRANSLATE FUNCTION 0	TRANSLATE FUNCTION 1
E1+1-->	TRANSLATE FUNCTION 2	TRANSLATE FUNCTION 3
	.	.
	.	.
	.	.
E1+N-->	TRANSLATE FUNCTION 2*N	TRANSLATE FUNCTION 2*N+1

TRANSLATE FUNCTIONS ARE HALFWORDS, IN WHICH THE HIGH-ORDER 3 BITS CONTROL THE EDITING PROCESS, AND THE LOW-ORDER 15 BITS ARE THE TRANSLATED VALUE OF THE CORRESPONDING SOURCE BYTE:

BIT 0/18	S BIT, SIGNIFIES THAT THIS CHARACTER IS NOT TO BE SUPPRESSED.
BITS 1-2/19-20	IF 0, NO EFFECT.

IF 1, ABORT THE EDIT,
IF 2, CLEAR THE M FLAG,
IF 3, SET THE M FLAG.

5.2 PARAMETER LIST FORMAT

```
E0-->    EXTEND OP, @, XR, Y      ;AC FIELD MUST BE ZERO
E0+1--> FILL CHAR/MESSAGE 0
E0+2--> FLOAT CHAR/MESSAGE 1
E0+3--> MESSAGE 2
E0+4--> MESSAGE 3
            .
            .
            .
E0+100  MESSAGE 77
```

6.0 SIMILARITY TO STRING INSTRUCTION

THE EXTENDED INSTRUCTION SET IS INTENDED TO PROVIDE THE FUNCTIONAL CAPABILITIES OF THE PREVIOUSLY DEFINED "STRING" INSTRUCTION, WHICH IT REPLACES. THE CHANGE WAS MADE TO PROVIDE SMOOTH EXPANSION INTO FUTURE INSTRUCTION SET EXTENSIONS, WHILE RETAINING THE FLAVOR OF THE PDP-10 ARCHITECTURE. FUNCTIONALLY, THIS LIST DIFFERS FROM THE CAPABILITIES OF THE STRING INSTRUCTION IN THAT MOVE AND COMPARE STRING OPERATIONS HAVE SEPARATE LENGTH SPECIFICATIONS FOR THE TWO STRINGS, MOVE OPERATIONS ARE CAPABLE OF RIGHT-JUSTIFYING THE DESTINATION STRING, AND COMPARE OPERATIONS HAVE LOST THE CAPABILITY OF MODIFYING THE SOURCE BYTES BEFORE COMPARISON.

7.0 PROBLEMS

FILLING IS THE PROCESS OF EXTENDING A STRING WITH A CONSTANT CHARACTER TO MEET A PREVIOUSLY SPECIFIED LENGTH. IT OCCURS IN SEVERAL OF THE OPERATIONS OF THE EXTENDED INSTRUCTION SET, BUT THERE IS NO CONSISTENCY WHATEVER TO THE SPEC OF HOW IT OCCURS...

IN STRING MOVE, FILLING ALWAYS OCCURS. IN EDIT, AS CURRENTLY SPEC'D, (E0+1) IS THE FILL CHAR UNLESS (E0+1) IS ZERO, IN WHICH CASE THERE IS NO FILL CHAR. IN BINARY TO DECIMAL CONVERSION, THE EXISTANCE OF A FILL CHARACTER IS CONTROLLED BY THE SIGNIFICANCE FLAG. CAN'T WE COME UP WITH A CONSISTENT MECHANISM?

[END OF 2P10R0.SPC]

1080,2040,2060 ENGINEERING FUNCTIONAL SPEC - CHAP 2,12

TO: KL10 LIST, J. PARSLOW (200 FILE)

TITLE: DEVICE CODE AND OPCODE ASSIGNMENTS - REV 2

STATUS: THIS CHAPTER CONTAINS ALL OF THE OPCODES WHICH HAVE BEEN APPROVED, EVEN THOUGH SOME OF THEM NEED SPECS. BECAUSE SOME OF THE OPCODES REFER TO ANNOUNCED CAPABILITIES, THIS CHAPTER IS COMPANY CONFIDENTIAL.

FILE: [EFS]CH2S12.SPC

PDM #: 200-200-018-02

DATE: 25 APR 75

SUPERSEDED MEMOS: OPCODE ASSIGNMENTS, J. LEONARD, 12 MAR 74

SUPERSEDED SPECS:

ENGINEER: J. LEONARD

APPROVED:

EDITOR: T. HASTINGS

TYPIST: K. PAPPAS

REVIEWED:

ABSTRACT

THIS CHAPTER LISTS THE DEVICE CODE AND OPCODE ASSIGNMENTS FOR ALL PDP-6, KA10, KI10, AND KL10 SYSTEMS. IT DOES NOT DESCRIBE THE DEVICES OR OPCODES. IT DOES INDICATE THE NUMBER OF A CHAPTER IN THE ENGINEERING FUNCTIONAL SPECIFICATION WHERE A DESCRIPTION CAN BE FOUND FOR THE NEW OR CHANGED KL10 OPCODES AND DEVICES.

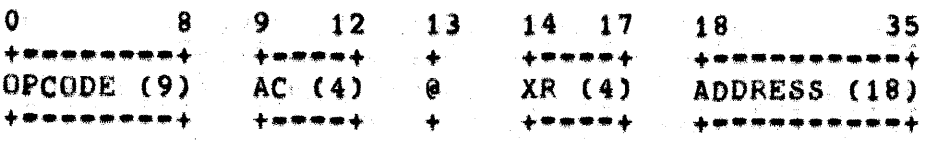
REVISION HISTORY

REV	DESCRIPTION	CHG NO	ORIG	DATE	APPD BY	DATE
1	ADD EXTEND					31 JAN 75
2	REMOVE SXCT, CHANGE PXCT					3 MAR 75

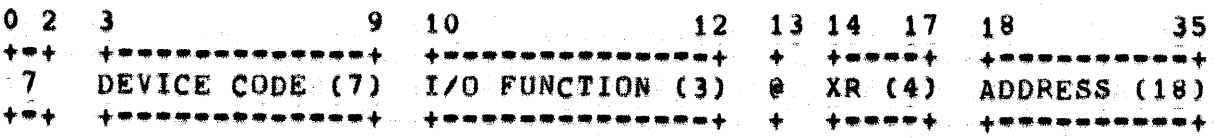
1. SUMMARY

THE PDP-10 ORDER CODE IS DIVIDED INTO SEVERAL CLASSES OF INSTRUCTIONS. HOWEVER THE FORMAT IS THE SAME, EXCEPT FOR THE IO CLASS INSTRUCTIONS AND THE EXTEND INSTRUCTION. OPCODES 000-677 USE THE FIRST 9 BITS OF THE INSTRUCTION TO SPECIFY THE OPCODE. IF THE OPCODE IS EXTEND, BITS 0-8 OF THE CONTENTS OF THE EFFECTIVE ADDRESS SPECIFY ANOTHER OPCODE IN THE EXTENDED SET OF OPCODES. IF THE OPCODE IS XCT, BITS 0-8 OF THE CONTENTS OF THE EFFECTIVE ADDRESS SPECIFY ANOTHER OPCODE IN THE NORMAL SET OF OPCODES. THE I/O INSTRUCTIONS (OPCODES 700-777) ALL HAVE BITS 0-2 = 7. BITS 3-9 SPECIFY THE 7 BIT DEVICE CODE. BITS 10-12 SPECIFY ONE OF 8 GENERAL I/O INSTRUCTIONS.

GENERAL INSTRUCTION FORMAT:



I/O INSTRUCTION FORMAT:



2. OPCODE NUMERICAL LISTING

IN THE FOLLOWING TABLE, ALL OPCODES ARE LISTED IN INCREASING NUMERICAL VALUE IN THE VALUE FIELD.

THE DEFINED SYMBOL OR SYMBOLS ARE GIVEN IN THE SYMBOL FIELD. THESE ARE THE SYMBOLS TO BE USED IN ALL PROGRAMS AND ARE DEFINED IN THE MACRO ASSEMBLER SYMBOL TABLE.

THE NOTE FIELD INDICATES WHETHER THE OPCODE IS DEFINED ON THE PDP-6, KA10, KI10, OR KL10 SYSTEMS USING THE NOTATION 6, A, I AND L RESPECTIVELY AND IN THAT ORDER. A DASH (-) IN THE APPROPRIATE POSITION MEANS THAT THE OPCODE IS NOT IMPLEMENTED ON THAT SYSTEM. A BLANK FIELD IS THE SAME AS 6AIL AND MEANS THE OPCODE IS IMPLEMENTED ON ALL SYSTEMS IN THE SAME WAY. AN ASTERISK (*) AFTER THE NOTE FIELD MEANS THAT THE INSTRUCTION DOES NOT WORK EXACTLY THE SAME ON ALL MACHINES. SEE INDICATED CHAPTER FOR DIFFERENCE. A QUESTION MARK AFTER THE NOTE FIELD MEANS THAT THE INSTRUCTION MAY BE DELETED FROM FUTURE RELEASES OF THE KL10 MICRO-CODE AND MAY NOT BE IMPLEMENTED ON A SUCCESSOR MACHINE.

THE NAME FIELD IS THE OFFICIAL ENGLISH NAME FOR THE OPCODE AND IS THE NAME TO BE USED IN ORDINARY SPEECH AND WRITING. THE CORRESPONDING LETTERS FROM THE SYMBOL ARE CAPITALIZED. THE NAME FIELD ALSO CONTAINS NOTES OF DIFFERENCES.

NOTE: MUUOS (40-77) ARE USED TO CALL THE TOPS10 OPERATING SYSTEM, WHILE JSYS (104) IS USED TO CALL THE VIROS OPERATING SYSTEM. UNUSED OPCODES 100-127 TRAP ON THE KA10. UNUSED OPCODES 247 AND 257 ARE NO-OPERATIONS ON THE KA10. ALL UNUSED OPCODES TRAP ON THE KI10 AND KL10.

VALUE	SYMBOL	NOTE	CHAPTER NAME
000			ILLEGAL
001-037			LUUOS
040	CALL		CALL OPERATING SYSTEM
041	INIT		INITIALIZE USER CHANNEL
042-046			RESERVED FOR CUSTOMER DEFINED CALLS (UUOS) TO THE OPERATING SYSTEM AND CUSTOMER DEFINED OPCODES WITH MODIFIED MICRO-CODE (BY DIGITAL SPECIAL SYSTEMS OR ADVANCED SYSTEMS GROUPS)
047	CALLI		CALL OPERATING SYSTEM IMMEDIATE
050	OPEN		OPEN USER CHANNEL
051	TTCALL		TTY TERMINAL CALL
052-054			RESERVED FOR DEC OPERATING SYSTEMS
055	RENAME		RENAME FILE
056	IN		INPUT BUFFER AND SKIP IF ERROR
057	OUT		OUTPUT BUFFER AND SKIP IF ERROR
060	SETST		SET USER CHANNEL STATUS
061	STATO		SKIP ON STATUS ONES
062	STATUS		READ STATUS
	GETSTS		GET STATUS
063	STATZ		SKIP ON RTATUS ZEROES
064	INBUF		SETUP INPUT BUFFER
065	OUTBUF		SETUP OUTPUT BUFFER
066	INPUT		GET INPUT BUFFER
067	OUTPUT		RELEASE OUTPUT BUFFER
070	CLOSE		CLOSE USER CHANNEL
071	RELEAS		RELEASE USER CHANNEL
072	MTAPE		MAGTAPE FUNCTION
073	UGETF		GET FREE BLOCK NUMBER
074	USETI		SET INPUT BLOCK NUMBER
075	USETO		SET OUTPUT BLOCK NUMBER
076	LOOKUP		LOOKUP FILE ON USER CHANNEL
077	ENTER		ENTER FILE ON USER CHANNEL

100	UJEN	-A--		USER JUMP AND ENABLE
101				ILLEGAL - RESERVED FOR DEC
102				ILLEGAL - RESERVED FOR DEC
103				ILLEGAL - RESERVED FOR DEC
104	JSYS	---L	2,5	JUMP TO SYSTEM SEE VIROS FUNCTIONAL SPECIFICATION AND JSYS MANUAL FOR SYMBOLS FOR EACH CALL AND DESCRIPTION OF ARGUMENTS.
105	ADJSP	---L	2,5	ADJUST STACK POINTER
106				ILLEGAL - RESERVED FOR DEC
107				ILLEGAL - RESERVED FOR DEC
110	DFAD	--IL		DOUBLE PRECISION FLOATING ADD AND ROUND
111	DFSB	--IL		DOUBLE PRECISION FLOATING SUBTRACT AND ROUND
112	DFMP	--IL*	1,3	DOUBLE PRECISION FLOATING MULTIPLY AND ROUND
113	DFDV	--IL*	1,3	DOUBLE PRECISION FLOATING DIVIDE AND ROUND
114	DADD	---L	2,4	DOUBLE PRECISION INTEGER ADD
115	DSUB	---L	2,4	DOUBLE PRECISION INTEGER SUBTRACT
116	DMUL	---L	2,4	DOUBLE PRECISION INTEGER MULTIPLY
117	DDIV	---L	2,4	DOUBLE PRECISION INTEGER DIVIDE
120	DMOVE	--IL	1,3	DOUBLE MOVE
121	DMOVN	--IL	1,3	DOUBLE MOVE NEGATIVE
122	FIX	--IL		FIX
123	EXTEND	---L		EXTENDED OPERATION CODES
124	DMOVEM	--IL*	1,3	DOUBLE MOVE TO MEMORY
125	DMOVNM	--IL*	1,3	DOUBLE MOVE NEGATIVE TO MEMORY
126	FIXR	--IL		FIX AND ROUND
127	FLTR	--IL		FLOAT AND ROUND
130	UFA	-AIL?	1,3	UNNORMALIZED FLOATING ADD
131	DFN	-AIL?	1,3	DOUBLE FLOATING NEGATE
132	FSC	*	1,3	FLOATING SCALE
133	IBP	6AIL*	1,3	INCREMENT BYTE POINTER
	ADJBP	---L	2,5	ADJUST BYTE POINTER
134	ILDB		1,3	INCREMENT POINTER AND LOAD BYTE INTO AC
135	LDB		1,3	LOAD BYTE INTO AC
136	IDPB		1,3	INCREMENT POINTER AND DEPOSIT BYTE INTO MEMORY
137	DPB		1,3	DEPOSIT BYTE INTO MEMORY
140	FAD	*	1,3	FLOATING ADD
141	FADL	6AIL?	1,3	FLOATING ADD LONG

142	FADM	*	1.3	FLOATING ADD TO MEMORY
143	FADB	*	1.3	FLOATING ADD TO BOTH
144	FADR			FLOATING ADD AND ROUND
145	FADRI	-AIL		FLOATING ADD AND ROUND IMMEDIATE
146	FADRM			FLOATING ADD AND ROUND TO MEMORY
147	FADRB			FLOATING ADD AND ROUND TO BOTH
150	FSB	*	1.3	FLOATING SUBTRACT
151	FSBL	6AIL*	1.3	FLOATING SUBTRACT LONG
152	FSBM	*	1.3	FLOATING SUBTRACT TO MEMORY
153	FSBB	*	1.3	FLOATING SUBTRACT TO BOTH
154	FSBR			FLOATING SUBTRACT AND ROUND
155	FSBRI	-AIL		FLOATING SUBTRACT AND ROUND IMMEDIATE
156	FSBRM			FLOATING SUBTRACT AND ROUND TO MEMORY
157	FSBRB			FLOATING SUBTRACT AND ROUND TO BOTH
160	FMP			FLOATING MULTIPLY
161	FMPL	6AIL?	1.3	FLOATING MULTIPLY LONG
162	FMPM			FLOATING MULTIPLY TO MEMORY
163	FMPB			FLOATING MULTIPLY TO BOTH
164	FMPR			FLOATING MULTIPLY AND ROUND
165	FMPRI	-AIL		FLOATING MULTIPLY AND ROUND IMMEDIATE
166	FMPRM			FLOATING MULTIPLY AND ROUND TO MEMORY
167	FMPRB			FLOATING MULTIPLY AND ROUND TO BOTH
170	FDV			FLOATING DIVIDE
171	FDVL	6AIL?	1.3	FLOATING DIVIDE LONG
172	FDVM			FLOATING DIVIDE TO MEMORY
173	FDVB			FLOATING DIVIDE TO BOTH
174	FDVR			FLOATING DIVIDE AND ROUND
175	FDVRI	-AIL		FLOATING DIVIDE AND ROUND IMMEDIATE
176	FDVRM			FLOAING DIVIDE AND ROUND TO MEMORY
177	FDVRB			FLOATING DIVIDE AND ROUND TO BOTH

200	MOVE		MOVE
201	MOVEI		MOVE IMMEDIATE TO AC
202	MOVEM		MOVE TO MEMORY
203	MOVES		MOVE TO SELF
204	MOVS		MOVE SWAPPED
205	MOVSI		MOVE SWAPPED IMMEDIATE TO AC
206	MOVSM		MOVE SWAPPED TO MEMORY
207	MOVSS		MOVE SWAPPED TO SELF
210	MOVN		MOVE NEGATIVE
211	MOVNI		MOVE NEGATIVE IMMEDIATE TO AC
212	MOVNM		MOVE NEGATIVE TO MEMORY
213	MOVNS		MOVE NEGATIVE TO SELF
214	MOVM		MOVE MAGNITUDE
215	MOVMI		MOVE MAGNITUDE IMMEDIATE TO AC
216	MOVMM		MOVE MAGNITUDE TO MEMORY
217	MOVMS		MOVE MAGNITUDE TO SELF
220	IMUL		INTEGER MULTIPLY
221	IMULI		INTEGER MULTIPLY IMMEDIATE
222	IMULM		INTEGER MULTIPLY TO MEMORY
223	IMULB		INTEGER MULTIPLY TO BOTH
224	MUL		MULTIPLY
225	MULI		MULTIPLY IMMEDIATE
226	MULM		MULTIPLY TO MEMORY
227	MULB		MULTIPLY TO BOTH
230	IDIV		INTEGER DIVIDE
231	IDIVI		INTEGER DIVIDE IMMEDIATE
232	IDIVM		INTEGER DIVIDE TO MEMORY
233	IDIVB		INTEGER DIVIDE TO BOTH
234	DIV		DIVIDE
235	DIVI		DIVIDE IMMEDIATE
236	DIVM		DIVIDE TO MEMORY
237	DIVB		DIVIDE TO BOTH
240	ASH		ARITHMETIC SHIFT
241	ROT		ROTATE
242	LSH		LOGICAL SHIFT
243	JFFO	-AIL	JUMP IF FIND FIRST ONE
244	ASHC		ARITHMETIC SHIFT COMBINED
245	ROTC		ROTATE COMBINED
246	LSHC		LOGICAL SHIFT COMBINED
247		6A--	NO OPERATION - RESERVED TO DEC
		--IL	ILLEGAL INSTR TRAP - RESERVED TO DEC

250	EXCH			EXCHANGE AC AND MEMORY
251	BLT			BLOCK TRANSFER
252	AOBJP	*		ADD ONE TO BOTH HALVES OF AC AND JUMP IF POSITIVE [KI DIFFERENT FROM KA]
253	AOBJN	*		ADD ONE TO BOTH HALVES OF AC AND JUMP IF NEGATIVE [KI DIFFERENT FROM KA]
254	JRST	*	1,3	JUMP RESTORE
254040	PORTAL	--IL		PORTAL (JRST 1,)
254100	JRSTF			JUMP AND RESTORE FLAGS (JRST 2,)
254200	HALT			HALT (JRST 4,)
254240	RPCF	---L*	2,2	RESTORE PC AND FLAGS (JRST 5,)
254300	RPCFD	---L*	2,2	RESTORE PC AND FLAGS THEN DISMISS (EXEC-ONLY) (JRST 6,)
254340	EPCF	---L*	2,2	EXCHANGE PC AND FLAGS (EXEC-ONLY) (JRST 7,)
254500	JEN			JUMP AND ENABLE PI CHANNEL (JRST 12,)
254600	SFM	---L*	2,2	SAVE FLAGS IN MEMORY (JRST 14,)
255	JFCL			JUMP ON FLAG AND CLEAR IT
255040	JFOV			JUMP ON FLOATING OVERFLOW (JFCL 1,) [ON PDP-6 THIS WAS PC CHANGE FLAG]
255100	JCRY1			JUMP ON CARRY 1 (JFCL 2,)
255200	JCRY0			JUMP ON CARRY 0 (JFCL 4,)
255300	JCRY			JUMP CARRY (JFCL 6,)
255400	JOV			JUMP ON OVERFLOW (JFCL 10,)
256	XCT			EXECUTE (USER MODE)
	XCT	--I-*		EXECUTIVE EXECUTE (EXEC MODE)
	PXCT	---L*	2,3	KL - PREVIOUS-CONTEXT XCT (EXEC-MODE)
257	NO-OP	6A--		NO-OPERATION
	MAP	--I-		MAP AN ADDRESS (USER MODE)
	MAP	---L*		MAP AN ADDRESS (EXEC MODE ONLY)
260	PUSHJ			PUSH PC DOWN AND JUMP
261	PUSH			PUSH DOWN
262	POP			POP UP
263	POPJ			POP PC UP AND JUMP
264	JSR			JUMP TO SUBROUTINE
265	JSP			JUMP AND SAVE PC
266	JSA			JUMP AND SAVE AC
267	JRA			JUMP AND RESTORE AC

270	ADD	ADD
271	ADDI	ADD IMMEDIATE
272	ADDM	ADD TO MEMORY
273	ADDB	ADD TO BOTH
274	SUB	SUBTRACT
275	SUBI	SUBTRACT IMMEDIATE
276	SUBM	SUBTRACT TO MEMORY
277	SUBB	SUBTRACT TO BOTH
300	CAI	COMPARE AC IMMEDIATE (NO-OP)
301	CAIL	COMPARE AC IMMEDIATE AND SKIP IF AC LESS
302	CAIE	COMPARE AC IMMEDIATE AND SKIP IF AC EQUAL
303	CAILE	COMPARE AC IMMEDIATE AND SKIP IF AC LESS OR EQUAL
304	CAIA	COMPARE AC IMMEDIATE AND SKIP ALWAYS
305	CAIGE	COMPARE AC IMMEDIATE AND SKIP IF AC GREATER OR EQUAL
306	CAIN	COMPARE AC IMMEDIATE AND SKIP IF AC NOT EQUAL
307	CAIG	COMPARE AC IMMEDIATE AND SKIP IF AC GREATER
310	CAM	COMPARE AC WITH MEMORY (NO-OP)
311	CAML	COMPARE AC WITH MEMORY AND SKIP IF AC LESS
312	CAME	COMPARE AC WITH MEMORY AND SKIP IF AC EQUAL
313	CAMLE	COMPARE AC WITH MEMORY AND SKIP IF AC LESS OR EQUAL
314	CAMA	COMPARE AC WITH MEMORY AND SKIP ALWAYS
315	CAMGE	COMPARE AC WITH MEMORY AND SKIP IF AC GREATER OR EQUAL
316	CAMN	COMPARE AC WITH MEMORY AND SKIP IF AC NOT EQUAL
317	CAMG	COMPARE AC WITH MEMORY AND SKIP IF AC GREATER

320	JUMP	JUMP IF AC NEVER (NO-OP)
321	JUMPL	JUMP IF AC LESS
322	JUMPE	JUMP IF AC EQUAL
323	JUMPLE	JUMP IF AC LESS OR EQUAL
324	JUMPA	JUMP ALWAYS
325	JUMPGE	JUMP IF AC GREATER OR EQUAL
326	JUMPN	JUMP IF AC NOT EQUAL
327	JUMPG	JUMP IF AC GREATER
330	SKIP	SKIP NEVER (NO-OP)
331	SKIPL	SKIP IF MEMORY LESS
332	SKIPE	SKIP IF MEMORY EQUAL
333	SKIPLE	SKIP IF MEMORY LESS OR EQUAL
334	SKIPA	SKIP ALWAYS
335	SKIPGE	SKIP IF MEMORY GREATER OR EQUAL
336	SKIPN	SKIP IF MEMORY NOT EQUAL
337	SKIPG	SKIP IF MEMORY GREATER
340	AOJ	ADD ONE TO AC AND JUMP IF NEVER
341	AOJL	ADD ONE TO AC AND JUMP IF LESS
342	AOJE	ADD ONE TO AC AND JUMP IF EQUAL
343	AOJLE	ADD ONE TO AC AND JUMP IF LESS OR EQUAL
344	AOJA	ADD ONE TO AC AND JUMP ALWAYS
345	AOJGE	ADD ONE TO AC AND JUMP IF GREATER OR EQUAL
346	AOJN	ADD ONE TO AC AND JUMP IF NOT EQUAL
347	AOJG	ADD ONE TO AC AND JUMP IF GREATER

L350	AOS	ADD ONE TO MEMORY AND SKIP NEVER
351	AOSL	ADD ONE TO MEMORY AND SKIP IF LESS
352	AOSE	ADD ONE TO MEMORY AND SKIP IF EQUAL
353	AOSLE	ADD ONE TO MEMORY AND SKIP IF LESS OR EQUAL
354	AOSA	ADD ONE TO MEMORY AND SKIP ALWAYS
355	AOSGE	ADD ONE TO MEMORY AND SKIP IF GREATER OR EQUAL
356	AOSN	ADD ONE TO MEMORY AND SKIP IF NOT EQUAL
357	AOSG	ADD ONE TO MEMORY AND SKIP IF GREATER
360	SOJ	SUBTRACT ONE FROM AC AND JUMP NEVER
361	SOJL	SUBTRACT ONE FROM AC AND JUMP IF LESS
362	SOJE	SUBTRACT ONE FROM AC AND JUMP IF EQUAL
363	SOJLE	SUBTRACT ONE FROM AC AND JUMP IF LESS OR EQUAL
364	SOJA	SUBTRACT ONE FROM AC AND JUMP ALWAYS
365	SOJGE	SUBTRACT ONE FROM AC AND JUMP IF GREATER OR EQUAL
366	SOJN	SUBTRACT ONE FROM AC AND JUMP IF NOT EQUAL
367	SOJG	SUBTRACT ONE FROM AC AND JUMP IF GREATER
370	SOS	SUBTRACT ONE FROM MEMORY AND SKIP NEVER
371	SOSL	SUBTRACT ONE FROM MEMORY AND SKIP IF LESS
372	SOSE	SUBTRACT ONE FROM MEMORY AND SKIP IF EQUAL
373	SOSLE	SUBTRACT ONE FROM MEMORY AND SKIP IF LESS OR EQUAL
374	SOSA	SUBTRACT ONE FROM MEMORY AND SKIP ALWAYS
375	SOSGE	SUBTRACT ONE FROM MEMORY AND SKIP IF GREATER OR EQUAL
376	SOSN	SUBTRACT ONE FROM MEMORY AND SKIP IF NOT EQUAL
377	SOSG	SUBTRACT ONE FROM MEMORY AND SKIP IF GREATER

400	SETZ			SET ZEROS TO AC
	CLEAR	6---		CLEAR TO AC (NAME BEING PHASED OUT)
401	SETZI			SET TO ZEROS TO AC IMMEDIATE
	CLEARI	6---		CLEAR TO AC IMMEDIATE (NAME BEING PHASED OUT)
402	SETZM			SET TO ZEROS TO MEMORY
	CLEARM	6---		CLEAR TO MEMORY (NAME BEING PHASED OUT)
403	SETZB			SET TO ZEROS TO BOTH
	CLEARB	6---		CLEAR TO BOTH (NAME BEING PHASED OUT)
404	AND			AND TO AC
405	ANDI			AND TO AC IMMEDIATE
406	ANDM			AND TO MEMORY
407	ANDB			AND TO BOTH
410	ANDCA			AND WITH COMPLEMENT OF AC
411	ANDCAI			AND WITH COMPLEMENT OF AC TO AC IMMEDIATE
412	ANDCAM			AND WITH COMPLEMENT OF AC TO MEMORY
413	ANDCAB			AND WITH COMPLEMENT OF AC TO BOTH
414	SETM			SET MEMORY TO AC
415	SETMI			SET MEMORY TO AC IMMEDIATE
	MOVXA	---L*	2.2	MOVE EXTENDED ADDRESS
416	SETMM			SET MEMORY TO MEMORY
417	SETMB			SET MEMORY TO BOTH
420	ANDCM			AND WITH COMPLEMENT OF MEMORY TO AC
421	ANDCMI			AND WITH COMPLEMENT OF MEMORY IMMEDIATE TO AC
422	ANDCMM			AND WITH COMPLEMENT OF MEMORY TO MEMORY
423	ANDCMB			AND WITH COMPLEMENT OF MEMORY TO BOTH
424	SETA			SET AC
425	SETAI			SET AC TO AC IMMEDIATE
426	SETAM			SET AC TO MEMORY
427	SETAB			SET AC TO BOTH

430	XOR	EXCLUSIVE OR TO AC
431	XORI	EXCLUSIVE OR TO AC IMMEDIATE
432	XORM	EXCLUSIVE OR TO MEMORY
433	XORB	EXCLUSIVE OR TO BOTH
434	IOR	INCLUSIVE OR TO AC
	OR	INCLUSIVE OR TO AC
435	IORI	INCLUSIVE OR TO AC IMMEDIATE
	ORI	INCLUSIVE OR TO AC IMMEDIATE
436	IORM	INCLUSIVE OR TO MEMORY
	ORM	INCLUSIVE OR TO MEMORY
437	IORB	INCLUSIVE OR TO BOTH
	ORB	INCLUSIVE OR TO BOTH
440	ANDCB	AND COMPLEMENTS OF BOTH TO AC
441	ANDCBI	AND COMPLEMENTS OF BOTH TO AC IMMEDIATE
442	ANDCBM	AND COMPLEMENTS OF BOTH TO MEMORY
443	ANDCBB	AND COMPLEMENTS OF BOTH TO BOTH
444	EQV	EQUIVALENCE TO AC
445	EQVI	EQUIVALENCE TO AC IMMEDIATE
446	EQVM	EQUIVALENCE TO MEMORY
447	EQVB	EQUIVALENCE TO BOTH
450	SETCA	SET COMPLEMENT OF AC TO AC
451	SETCAI	SET COMPLEMENT OF AC TO AC IMMEDIATE
452	SETCAM	SET COMPLEMENT OF AC TO MEMORY
453	SETCAB	SET COMPLEMENT OF AC TO BOTH
454	ORCA	INCLUSIVE OR WTH COMPLEMENT OF AC TO AC
455	ORCAI	INCLUSIVE OR WITH COMPLEMENT OF AC TO AC IMMEDIATE
456	ORCAM	INCLUSIVE OR WITH COMPLEMENT OF AC TO MEMORY
457	ORCAB	INCLUSIVE OR WITH COMPLEMENT OF AC TO BOTH

460	SETCM	SET COMPLEMENT OF MEMORY TO AC
461	SETCMI	SET COMPLEMENT OF MEMORY TO AC IMMEDIATE
462	SETCMM	SET COMPLEMENT OF MEMORY TO MEMORY
463	SETCMB	SET COMPLEMENT OF MEMORY TO BOTH
464	ORCM	INCLUSIVE OR WITH COMPLEMENT OF MEMORY TO AC
465	ORCMI	INCLUSIVE OR WITH COMPLEMENT OF MEMORY TO AC IMMEDIATE
466	ORCMM	INCLUSIVE OR WITH COMPLEMENT OF MEMORY TO MEMORY
467	ORCMB	INCLUSIVE OR WITH COMPLEMENT OF MEMORY TO BOTH
470	ORCB	INCLUSIVE OR COMPLEMENTS OF BOTH TO AC
471	ORCBI	INCLUSIVE OR COMPLEMENTS OF BOTH TO AC IMMEDIATE
472	ORCBM	INCLUSIVE OR COMPLEMENTS OF BOTH TO MEMORY
473	ORCBB	INCLUSIVE OR COMPLEMENTS OF BOTH TO BOTH
474	SETO	SET ONES TO AC
475	SETOI	SET ONES TO AC IMMEDIATE
476	SETOM	SET ONES TO MEMORY
477	SETOB	SET ONES TO BOTH

500	HLL	HALF WORD LEFT TO LEFT TO AC
501	HLLI	HALF WORD LEFT TO LEFT IMMEDIATE TO AC
502	HLLM	HALF WORD LEFT TO LEFT TO MEMORY
503	HLLS	HALF WORD LEFT TO LEFT TO SELF
504	HRL	HALF WORD RIGHT TO LEFT TO AC
505	HRLI	HALF WORD RIGHT TO LEFT IMMEDIATE TO AC
506	HRLM	HALF WORD RIGHT TO LEFT TO MEMORY
507	HRLS	HALF WORD RIGHT TO LEFT TO SELF
510	HLLZ	HALF WORD LEFT TO LEFT ZEROS TO AC
511	HLLZI	HALF WORD LEFT TO LEFT ZEROS IMMEDIATE TO AC
512	HLLZM	HALF WORD LEFT TO LEFT ZEROS TO MEMORY
513	HLLZS	HALF WORD LEFT TO LEFT ZEROS TO SELF
514	HRLZ	HALF WORD RIGHT TO LEFT ZEROS TO AC
515	HRLZI	HALF WORD RIGHT TO LEFT ZEROS IMMEDIATE TO AC
516	HRLZM	HALF WORD RIGHT TO LEFT ZEROS TO MEMORY
517	HRLZS	HALF WORD RIGHT TO LEFT ZEROS TO SELF
520	HLLO	HALF WORD LEFT TO LEFT ONES TO AC
521	HLLOI	HALF WORD LEFT TO LEFT ONES IMMEDIATE TO AC
522	HLLOM	HALF WORD LEFT TO LEFT ONES TO MEMORY
523	HLLOS	HALF WORD LEFT TO LEFT ONES TO SELF
524	HRLO	HALF WORD RIGHT TO LEFT ONES TO AC
525	HRLOI	HALF WORD RIGHT TO LEFT ONES IMMEDIATE TO AC
526	HRLOM	HALF WORD RIGHT TO LEFT ONES TO MEMORY
527	HRLOS	HALF WORD RIGHT TO LEFT ONES TO SELF

530	HLLE	HALF WORD LEFT TO LEFT EXTEND SIGN TO AC
531	HLLEI	HALF WORD LEFT TO LEFT EXTEND SIGN IMMEDIATE TO AC
532	HLLEM	HALF WORD LEFT TO LEFT EXTEND SIGN TO MEMORY
533	HLLES	HALF WORD LEFT TO LEFT EXTEND SIGN TO SELF
534	HRLE	HALF WORD RIGHT TO LEFT EXTEND SIGN TO AC
535	HRLEI	HALF WORD RIGHT TO LEFT EXTEND SIGN IMMEDIATE TO AC
536	HRLEM	HALF WORD RIGHT TO LEFT EXTEND SIGN TO MEMORY
537	HRLES	HALF WORD RIGHT TO LEFT EXTEND SIGN TO SELF
540	HRR	HALF WORD RIGHT TO RIGHT TO AC
541	HRRI	HALF WORD RIGHT TO RIGHT IMMEDIATE TO AC
542	HRRM	HALF WORD RIGHT TO RIGHT TO MEMORY
543	HRRS	HALF WORD RIGHT TO RIGHT TO SELF
544	HLR	HALF WORD LEFT TO RIGHT TO AC
545	HLRI	HALF WORD LEFT TO RIGHT IMMEDIATE TO AC
546	HLRM	HALF WORD LEFT TO RIGHT TO MEMORY
547	HLRS	HALF WORD LEFT TO RIGHT TO SELF

550	HRRZ	HALF WORD RIGHT TO RIGHT ZEROS TO AC
551	HRRZI	HALF WORD RIGHT TO RIGHT ZEROS IMMEDIATE TO AC
552	HRRZM	HALF WORD RIGHT TO RIGHT ZEROS TO MEMORY
553	HRRZS	HALF WORD RIGHT TO RIGHT ZEROS TO SELF
554	HLRZ	HALF WORD LEFT TO RIGHT ZEROS TO AC
555	HLRZI	HALF WORD LEFT TO RIGHT ZEROS IMMEDIATE TO AC
556	HLRZM	HALF WORD LEFT TO RIGHT ZEROS TO MEMORY
557	HLRZS	HALF WORD LEFT TO RIGHT ZEROS TO SELF
560	HRRO	HALF WORD RIGHT TO RIGHT ONES TO AC
561	HRROI	HALF WORD RIGHT TO RIGHT ONES IMMEDIATE TO AC
562	HRROM	HALF WORD RIGHT TO RIGHT ONES TO MEMORY
563	HRROS	HALF WORD RIGHT TO RIGHT ONES TO SELF
564	HLRO	HALF WORD LEFT TO RIGHT ONES TO AC
565	HLROI	HALF WORD LEFT TO RIGHT ONES IMMEDIATE TO AC
566	HLROM	HALF WORD LEFT TO RIGHT ONES TO MEMORY
567	HLROS	HALF WORD LEFT TO RIGHT ONES TO SELF
570	HRRE	HALF WORD RIGHT TO RIGHT EXTEND SIGN TO AC
571	HRREI	HALF WORD RIGHT TO RIGHT EXTEND SIGN IMMEDIATE TO AC
572	HRREM	HALF WORD RIGHT TO RIGHT EXTEND SIGN TO MEMORY
573	HRRES	HALF WORD RIGHT TO RIGHT EXTEND SIGN TO SELF
574	HLRE	HALF WORD LEFT TO RIGHT EXTEND SIGN TO AC
575	HLREI	HALF WORD LEFT TO RIGHT EXTEND SIGN IMMEDIATE TO AC
576	HLREM	HALF WORD LEFT TO RIGHT EXTEND SIGN TO MEMORY
577	HLRES	HALF WORD LEFT TO RIGHT EXTEND SIGN TO SELF

600	TRN	TEST AC RIGHT WITH E NO MODIFICATION AND SKIP NEVER
601	TLN	TEST AC LEFT WITH E NO MODIFICATION AND SKIP NEVER
602	TRNE	TEST AC RIGHT WITH E NO MODIFICATION AND SKIP IF ALL MASKED BITS EQUAL 0
603	TLNE	TEST LEFT WITH E NO MODIFICATION AND SKIP IF ALL MASKED BITS EQUAL 0
604	TRNA	TEST AC RIGHT WITH E NO MODIFICATION AND SKIP ALWAYS
605	TLNA	TEST AC LEFT WITH E NO MODIFICATION AND SKIP ALWAYS
606	TRNN	TEST AC RIGHT WITH E NO MODIFICATION AND SKIP IF NOT ALL MASKED BITS EQUAL 0
607	TLNN	TEST AC LEFT WITH E NO MODIFICATION AND SKIP IF NOT ALL MASKED BITS EQUAL 0
610	TDN	TEST AC WITH DIRECT MASK NO MODIFICATION AND SKIP NEVER
611	TSN	TEST AC WITH SWAPPED MASK NO MODIFICATION AND SKIP NEVER
612	TDNE	TEST AC WITH DIRECT MASK NO MODIFICATION AND SKIP IF ALL MASKED BITS EQUAL 0
613	TSNE	TEST AC WITH SWAPPED MASK NO MODIFICATION AND SKIP IF ALL MASKED BITS EQUAL 0
614	TDNA	TEST AC WITH DIRECT MASK NO MODIFICATION AND SKIP ALWAYS
615	TSNA	TEST AC WITH SWAPPED MASK NO MODIFICATION AND SKIP ALWAYS
616	TDNN	TEST AC WITH DIRECT MASK NO MODIFICATION AND SKIP IF NOT ALL MASKED BITS EQUAL 0
617	TSNN	TEST AC WITH SWAPPED MASK NO MODIFICATION AND SKIP IF NOT ALL MASKED BITS EQUAL 0

620	TRZ	TEST AC RIGHT WITH E SET MASKED BITS TO ZEROS AND SKIP NEVER
621	TLZ	TEST AC LEFT WITH E SET MASKED BITS TO ZEROS AND SKIP NEVER
622	TRZE	TEST AC RIGHT WITH E SET MASKED BITS TO ZEROS AND SKIP IF ALL MASKED BITS EQUAL 0
623	TLZE	TEST AC LEFT WITH E SET MASKED BITS TO ZEROS AND SKIP IF ALL MASKED BITS EQUAL 0
624	TRZA	TEST AC RIGHT WITH E SET MASKED BITS TO ZEROS AND SKIP ALWAYS
625	TLZA	TEST AC LEFT WITH E SET MASKED BITS TO ZEROS AND SKIP ALWAYS
626	TRZN	TEST AC RIGHT WITH E SET MASKED BITS TO ZEROS AND SKIP IF NOT ALL MASKED BITS EQUAL 0
627	TLZN	TEST AC LEFT WITH E SET MASKED BITS TO ZEROS AND SKIP IF NOT ALL MASKED BITS EQUAL 0
630	TDZ	TEST AC WITH DIRECT MASK SET MASKED BITS TO ZEROS AND SKIP NEVER
631	TSZ	TEST AC WITH SWAPPED MASK SET MASKED BITS TO ZEROS AND SKIP NEVER
632	TDZE	TEST AC WITH DIRECT MASK SET MASKED BITS TO ZEROS AND SKIP IF ALL MASKED BITS EQUAL 0
633	TSZE	TEST AC WITH SWAPPED MASK SET MASKED BITS TO ZEROS AND SKIP IF ALL MASKED BITS EQUAL 0
634	TDZA	TEST AC WITH DIRECT MASK SET MASKED BITS TO ZEROS AND SKIP ALWAYS
635	TSZA	TEST AC WITH SWAPPED MASK SET MASKED BITS TO ZEROS AND SKIP ALWAYS
636	TDZN	TEST AC WITH DIRECT MASK SET MASKED BITS TO ZEROS AND SKIP IF NOT ALL MASKED BITS EQUAL 0
637	TSZN	TEST AC WITH SWAPPED MASK SET MASKED BITS TO ZEROS AND SKIP IF NOT ALL MASKED BITS EQUAL 0

640	TRC	TEST AC RIGHT WITH E COMPLEMENT MASKED BITS AND SKIP NEVER
641	TLC	TEST AC LEFT WITH E COMPLEMENT MASKED BITS AND SKIP NEVER
642	TRCE	TEST AC RIGHT WITH E COMPLEMENT MASKED BITS AND SKIP IF ALL MASKED BITS EQUAL 0
643	TLCE	TEST AC LEFT WITH E COMPLEMENT MASKED BITS AND SKIP IF ALL MASKED BITS EQUAL 0
644	TRCA	TEST AC RIGHT WITH E COMPLEMENT MASKED BITS AND SKIP ALWAYS
645	TLCA	TEST AC LEFT WITH E COMPLEMENT MASKED BITS AND SKIP ALWAYS
646	TRCN	TEST AC RIGHT WITH E COMPLEMENT MASKED BITS AND SKIP IF NOT ALL MASKED BITS EQUAL 0
647	TLCN	TEST AC LEFT WITH E COMPLEMENT MASKED BITS AND SKIP IF NOT ALL MASKED BITS EQUAL 0
650	TDC	TEST AC WITH DIRECT MASK COMPLEMENT MASKED BITS AND SKIP NEVER
651	TSC	TEST AC WITH SWAPPED MASK COMPLEMENT MASKED BITS AND SKIP NEVER
652	TDCE	TEST AC WITH DIRECT MASK COMPLEMENT MASKED BITS AND SKIP IF ALL MASKED BITS EQUAL 0
653	TSCE	TEST AC WITH SWAPPED MASK COMPLEMENT MASKED BITS AND SKIP IF ALL MASKED BITS EQUAL 0
654	TDCA	TEST AC WITH DIRECT MASK COMPLEMENT MASKED BITS AND SKIP ALWAYS
655	TSCA	TEST AC WITH SWAPPED MASK COMPLEMENT MASKED BITS AND SKIP ALWAYS
656	TDCN	TEST AC WITH DIRECT MASK COMPLEMENT MASKED BITS AND SKIP IF NOT ALL MASKED BITS EQUAL 0
657	TSCN	TEST AC WITH SWAPPED MASK COMPLEMENT MASKED BITS AND SKIP IF NOT ALL MASKED BITS EQUAL 0

660	TRO	TEST AC RIGHT WITH E SET MASKED BITS TO ONES AND SKIP NEVER
661	TLO	TEST AC LEFT WITH E SET MASKED BITS TO ONES AND SKIP NEVER
662	TROE	TEST AC RIGHT WITH E SET MASKED BITS TO ONES AND SKIP IF ALL MASKED BITS EQUAL 0
663	TLOE	TEST AC LEFT WITH E SET MASKED BITS TO ONES AND SKIP IF ALL MASKED BITS EQUAL 0
664	TROA	TEST AC RIGHT WITH E SET MASKED BITS TO ONES AND SKIP ALWAYS
665	TLOA	TEST AC LEFT WITH E SET MASKED BITS TO ONES AND SKIP ALWAYS
666	TRON	TEST AC RIGHT WITH E SET MASKED BITS TO ONES AND SKIP IF NOT ALL MASKED BITS EQUAL 0
667	TLON	TEST LEFT WITH E SET MASKED BITS TO ONES AND SKIP IF NOT ALL MASKED BITS EQUAL 0
670	TDO	TEST AC WITH DIRECT MASK SET MASKED BITS TO ONES AND SKIP NEVER
671	TSO	TEST AC WITH SWAPPED MASK SET MASKED BITS TO ONES AND SKIP NEVER
672	TDOE	TEST AC WITH DIRECT MASK SET MASKED BITS TO ONES AND SKIP IF ALL MASKED BITS EQUAL 0
673	TSOE	TEST AC WITH SWAPPED MASK SET MASKED BITS TO ONES AND SKIP IF ALL MASKED BITS EQUAL 0
674	TDOA	TEST AC WITH DIRECT MASK SET MASKED BITS TO ONES AND SKIP ALWAYS
675	TSOA	TEST AC WITH SWAPPED MASK SET MASKED BITS TO ONES AND SKIP ALWAYS
676	TDON	TEST AC WITH DIRECT MASK SET MASKED BITS TO ONES AND SKIP IF NOT ALL MASKED BITS EQUAL 0
677	TSON	TEST AC WITH SWAPPED MASK SET MASKED BITS TO ONES AND SKIP IF NOT ALL MASKED BITS EQUAL 0

700000	BLKI		BLOCK IN
700000	APRID		READ APR ID
700040	DATAI		DATA IN
700040	RSW	6AI-	READ SWITCHES (DATAI APR,)
700100	BLKO		BLOCK OUT
700100	WRFIL		WRITE CACHE REFILL RAM
700140	DATAO		DATA OUT
700200	CONO		CONDITIONS OUT
700240	CONI		CONDITIONS IN
070030	CONSZ		CONDITIONS IN AND SKIP IF ALL MASKED BITS ZERO
700340	CONSO		CONDITIONS IN AND SKIP IF SOME MASKED BIT ONE
700400	RDERA		READ ERROR ADDRESS REGISTER
700500	SBDIAG		SBUS DIAGNOSTIC FUNCTIONS
701440	SWPIA		SWEEP - INVALIDATE ALL CACHE, NO CORE UPDATE
701500	SWPVA		SWEEP - VALIDATE ALL CORE, LEAVE CACHE VALID
701540	SWPUA		SWEEP - UNLOAD ALL CACHE, INVALIDATING CACHE
701640	SWPIO		SWEEP - INVALIDATE ONE PAGE, NO CORE UPDATE
701700	SWPVO		SWEEP - VALIDATE ONE PAGE, LEAVE CACHE VALID
701740	SWPUO		SWEEP - UNLOAD ONE PAGE, INVALIDATING CACHE
702000	RDPAC		READ PERFORMANCE ANALYSIS COUNTER
702100	WRPAE		WRITE PERFORMANCE ANALYSIS ENABLES

3. ALPHABETICAL LISTING OF OPCODES

SYMBOL	VALUE	NOTE
ADD	270	
ADDB	273	
ADDI	271	
ADDM	272	
ADJBP	133	---L SAME OPCODE AS IBP
ADJSP	105	---L
AND	404	
ANDB	407	
ANDCA	410	
ANDCAB	413	
ANDCAI	411	
ANDCAM	412	
ANDCB	440	
ANDCBB	443	
ANDCBI	441	
ANDCBM	442	
ANDCM	420	
ANDCMB	423	
ANDCMI	421	
ANDCMM	422	
ANDI	405	
ANDM	406	
AOBJN	253	
AOBJP	252	
AOJ	340	
AOJA	344	
AOJE	342	
AOJG	347	
AOJGE	345	
AOJL	341	
AOJLE	343	
AOJN	346	
AOS	350	
AOSA	354	
AOSE	352	
AOSG	357	
AOSGE	355	
AOSL	351	
AOSLE	353	
AOSN	356	
APRID	700000	
ASH	240	
AHSC	244	
BLKI	700000	
BLKO	700100	
BLT	251	
CAI	300	

CAIA	304	
CAIE	302	
CAIG	307	
CAIGE	305	
CAIL	301	
CAILE	303	
CAIN	306	
CALL	040	UUO
CALLI	047	UUO
CAM	310	
CAMA	314	
CAME	312	
CAMG	317	
CAMGE	315	
CAML	311	
CAMLE	313	
CAMN	316	
CLOSE	070	UUO
CMPSE	002	EXTEND
CMPSG	007	EXTEND
CMPSGE	005	EXTEND
CMPSL	001	EXTEND
CMPSLE	003	EXTEND
CMPN	006	EXTEND
CONI	700240	
CONO	700200	
CONSO	700340	
CONSZ	700300	
CVTBDO	012	EXTEND
CVTBDT	013	EXTEND
CVTDBO	010	EXTEND
CVTDBT	011	EXTEND
DATAI	700040	
DATAO	700140	
DFAD	110	**IL
DFDV	113	**IL
DFMP	112	**IL
DFN	131	
DFSB	111	**IL
DIV	234	
DIVB	237	
DIVI	235	
DIVM	236	
DMOVE	120	**IL
DMOVEM	124	**IL
DMOVN	121	**IL
DMOVNM	125	**IL
DPB	137	
EBLT	020	EXTEND
EDIT	004	EXTEND
ENTER	077	UUO
EPCF	254340	

EQV	444	
EQVB	447	
EQVI	445	
EQVM	446	
EXCH	250	
EXTEND	123	---L
FAD	140	
FADB	143	
FADL	141	
FADM	142	
FADR	144	
FADRB	147	
FADRI	145	
FADRM	146	
FDV	170	
FDVB	173	
FDVL	171	
FDVM	172	
FDVR	174	
FDVRB	177	
FDVRI	175	
FDVRM	176	
FIX	122	--IL
FIXR	126	--IL
FLTR	127	--IL
FMP	160	
FMPB	163	
FMPL	161	
FMPM	162	
FMPR	164	
FMPRB	167	
FMPRI	165	
FMPRM	166	
FSB	150	
FSBB	153	
FSBL	151	
FSBM	152	
FSBR	154	
FSBRB	157	
FSBRI	155	
FSBRM	156	
FSC	132	
GETSTS	062	UUO
HALT	25420	
HLL	500	
HLLE	530	
HLLEI	531	
HLLEM	532	
HLLES	533	
HLLI	501	
HLLM	502	

HLLO	520
HLLOI	521
HLLOM	522
HLLOS	523
HLLS	503
HLLZ	510
HLLZI	511
HLLZM	512
HLLZS	513
HLR	544
HLRE	574
HLREI	575
HLREM	576
HLRES	577
HLRI	545
HLRM	546
HLRO	564
HLROI	565
HLROM	566
HLROS	567
HLRS	547
HLRZ	554
HLRZI	555
HLRZM	556
HLRZS	557
HRL	504
HRLE	534
HRLEI	535
HRLEM	536
HRLES	537
HRLI	505
HRLM	506
HRLO	524
HRLOI	525
HRLOM	526
HRLOS	527
HRLS	507
HRLZ	514
HRLZI	515
HRLZM	516
HRLZS	517
HRR	540
HRRE	570
HRREI	571
HRREM	572
HRRES	573
HRRI	541
HRRM	542
HRRO	560
HRROI	561
HRROM	562
HRROS	563
HRRS	543
HRRZ	550

HRRZI	551	
HRRZM	552	
HRRZS	553	
IBP	133	
IDIV	230	
IDIVB	233	
IVIDI	231	
IDIVM	232	
IDPB	136	
ILDB	134	
IMUL	220	
IMULB	223	
IMULI	221	
IMULM	222	
IN	056	UUO
INBUF	064	UUO
INIT	041	UUO
INPUT	066	UUO
IOR	434	
IORB	437	
IORI	435	
IORM	436	
JCRY	255300	
JCRY0	255200	
JCRY1	255100	
JEN	254600	
JFCL	255	
JFFO	243	
JFOV	255040	
JOV	255400	
JRA	267	
JRST	254	
JRSTF	254100	
JSA	266	
JSP	265	
JSR	264	
JSYS	104	---L
JUMP	320	
JUMPA	324	
JUMPE	322	
JUMPG	327	
JUMPGE	325	
JUMPL	321	
JUMPLE	323	
JUMPN	326	
LDB	135	
LOOKUP	076	UUO
LSH	242	
LSHC	246	
MAP	257	--IL

MOVE	200	
MOVEI	201	
MOVEM	202	
MOVES	203	
MOVM	214	
MOVMI	215	
MOVMM	216	
MOVMS	217	
MOVN	210	
MOVNI	211	
MOVNM	212	
MOVNS	213	
MOVS	204	
MOVSI	205	
MOVSLJ	016	EXTEND
MOVSM	206	
MOVSO	014	EXTEND
MOVSB	017	EXTEND
MOVSS	207	
MOVST	015	EXTEND
MOVXA	415	
MTAPE	072	UUO
MUL	224	
MULB	227	
MULI	225	
MULM	226	
OPEN	050	UUO
OR	434	
ORB	437	
ORCA	454	
ORCAB	457	
ORCAI	455	
ORCAM	456	
ORCB	470	
ORCBB	473	
ORCBI	471	
ORCBM	472	
ORCM	464	
ORCB	470	
ORCMI	465	
ORCMM	466	
ORI	435	
ORM	436	
OUT	057	UUO
OUTBUF	065	UUO
OUTPUT	067	UUO
POP	262	
POPJ	263	
PORTAL	254040	
PXCT	256	SAME OPCODE AS XCT AND EXCT
PUSH	261	
PUSHJ	260	

RDERA	700400	
RDPAC	702000	
RELEAS	071	UUO
RENAME	055	UUO
ROT	241	
ROTC	245	
RPCF	254240	
RPCFD	254300	
RSW	700040	
SBDIAG	700500	
SETA	424	
SETAB	427	
SETAI	425	
SETAM	426	
SETCA	450	
SETCAB	453	
SETCAI	451	
SETCAM	452	
SETCM	460	
SETCMB	463	
SETCMI	461	
SETCMM	462	
SETM	414	
SETMB	417	
SETMI	415	
SETMM	416	
SETO	474	
SETOB	477	
SETOI	475	
SETOM	476	
SETSTS	060	UUO
SETZ	400	
SETZB	403	
SETZI	401	
SETZM	402	
SFM	254600	
SKIP	330	
SKIPA	334	
SKIPE	332	
SKIPG	337	
SKIPGE	335	
SKIPL	331	
SKIPLE	333	
SKIPN	336	
SOJ	360	
SOJA	364	
SOJE	362	
SOJG	367	
SOJGE	365	
SOJL	361	
SOJLE	363	
SOJN	366	
SOS	370	

SOSA	374	
SOSE	372	
SOSG	377	
SOSGE	375	
SOSL	371	
SOSLE	373	
SOSN	376	
STATO	061	UUO
STATUS	062	UUO
STATZ	063	UUO
SUB	274	
SUBB	277	
SUBI	275	
SUBM	276	
SWPIA	701440	
SWPIO	701640	
SWPUA	701540	
SWPUO	701740	
SWPVA	701500	
SWPVO	701700	
TDC	650	
TDCA	654	
TDCE	652	
TDCN	656	
TDN	610	
TDNA	614	
TDNE	612	
TDNN	616	
TDO	670	
TDOA	674	
TDOE	672	
TDON	676	
TDZ	630	
TDZA	634	
TDZE	632	
TDZN	636	
TLC	641	
TLCA	645	
TLCE	643	
TLCN	647	
TLN	601	
TLNA	605	
TLNE	603	
TLNN	607	
TLO	661	
TLOA	665	
TLOE	663	
TLON	667	
TLZ	621	
TLZA	625	
TLZE	623	
TLZN	627	
TRC	640	

TRCA	644	
TRCE	642	
TRCN	646	
TRN	600	
TRNA	604	
TRNE	602	
TRNN	606	
TRO	660	
TROA	664	
TROE	662	
TRON	666	
TRZ	620	
TRZA	624	
TRZE	622	
TRZN	626	
TSC	651	
TSCA	655	
TSCE	653	
TSCN	657	
TSN	611	
TSNA	615	
TSNE	613	
TSNN	617	
TSO	671	
TSOA	675	
TSOE	673	
TSON	677	
TSZ	631	
TSZA	635	
TSZE	633	
TSZN	637	
TTCALL	051	UUO
UFA	130	
UGETF	073	UUO
UJEN	100	*A**
USETI	074	UUO
USETO	075	UUO
WRFIL	700100	
WRPAE	702100	
XCT	256	
XOR	430	
XORB	433	
XORI	431	
XORM	432	

4. I/O DEVICE CODE ASSIGNMENTS

EACH I/O INSTRUCTION SPECIFIES A SEVEN BIT DEVICE CODE IN BITS 3-9. FOR EASE OF READING IN AN OCTAL WORD, THE SEVEN BIT DEVICE CODE IS GIVEN AS A 3 DIGIT OCTAL NUMBER AS IT APPEARS IN A 36 BIT WORD. THUS THE LOW ORDER OCTAL DIGIT IS EITHER 0 OR 4, SINCE IT REPRESENTS ONLY ONE BIT. THE CPU COLUMN SPECIFIES WHICH CPUS THE DEVICE IS GOOD FOR. SO FAR THERE ARE 4 CPUS: PDP-6, KA10, KI10, AND KL10. THEIR SYMBOLS ARE 6, A, I, AND L RESPECTIVELY. A DASH MEANS NOT AVAILABLE ON THAT CPU. THE CHAP COLUMN SPECIFIES THE CHAPTER IN THE KL10 FUNCTIONAL SPECIFICATION WHERE THE DEVICE CODE IS EXPLAINED FOR THE KL10. IF A DEVICE IS DIFFERENT FOR DIFFERENT CPUS, IT IS LISTED ON SEPARATE LINES. PDP-6 AND PDP-10 OPTION DESIGNATIONS ARE GIVEN IN BRACKETS, FOLLOWING "6:" OR "10:". SOME DEVICES REQUIRE TWO OR THREE DEVICE CODES WHENEVER THEY ARE PRESENT. SUCH DEVICES ARE INDICATED BY SEPARATING THE DEVICE CODES BY COMMAS ON THE SAME LINE, INSTEAD OF LISTING THEM ON SEPARATE LINES. IN PRINCIPAL KA AND KI DEVICES (EXCEPT BUILT-IN KA OE KI DEVICES) CAN BE PUT ON THE KL10 USING THE I/O BUS ADAPTER, DIA20. THEREFORE THE OPTION DESIGNATIONS FOR THE 1080, 2040, AND 2060 WHICH GO ON THE I/O BUS RETAIN THEIR DECSYSTEM 10 OPTION DESIGNATION, E.G. LP10. DEVICE OPTIONS WHICH ARE PRESENT ONLY ON THE KL10 (1080, 2040, 2060) HAVE A DECSYSTEM 20 OPTION DESIGNATION, E.G. DTE20 OR RH20.

VALUE	SYMBOL	CPU	CHAP	NAME
000	APR	6		ARITHMETIC PROCESSOR
	CPA	6		CENTRAL PROCESSOR
	APR	A		ARITHMETIC PROCESSOR
	APR	I		ARITHMETIC PROCESSOR
	APR	L	2.6	ARITHMETIC PROCESSOR
004	PI	6		PRIORITY INTERRUPT
	PI	A		PRIORITY INTERRUPT
	PI	I		PRIORITY INTERRUPT
	PI	L	2.6	PRIORITY INTERRUPT
010	DRUM	6		DRUM PROCESSOR
	PAG	I		PAGING CONTROL
	PAG	L	2.6	PAGING CONTROL
014	CCI	-AI-		COMPUTER-COMPUTER INTERFACE (PDP-8, -9)[10: DA10]
	CCA	L		CACHE SWEEP
020	CCI2	-AI-		COMPUTER-COMPUTER INTERFACE (PDP-8, -9)[10: DI10]
	TIM	---L	2.7	TIMER CLOCKS (BUILT-IN)
024	ADC	-AI		ANALOG-DIGITAL CONVERTER
	MTR	---L	2.7	ACCOUNTING METERS (BUILT-IN)

030	ADC2	-AI-	ANALOG-DIGITAL CONVERTER
034			UNUSED - RESERVED TO DEC
040			UNUSED - RESERVED TO DEC
044			UNUSED - RESERVED TO DEC
050			UNUSED - RESERVED TO DEC
054			UNUSED - RESERVED TO DEC
060,064	DLB,DLC	-AIL	PDP-11 DATA LINK [10: DL10]
070	CLK	-AIL	REAL TIME CLOCK [10: DK10] NOTE: ON KL, TIM SUPERSEDES DK10
074	CLK2	-AIL	REAL TIME CLOCK [10: DK10] NOTE: ON KL, TIM SUPERSEDES DK10
100	PTP	6AI-	PAPER TAPE PUNCH [6:760,10: BUILT-IN]
110	CDP	-AIL	CARD PUNCH [10: CP10]
114	CDR	6---	CARD READER [6: 461]
120	TTY	6AI-	CONSOLE TELETYPE [6: 626,10: BUILT-IN]
124	LPT	6AIL	LINE PRINTER [6: 646, 10: LP10]
130	DIS	6AIL	DISPLAY [6: 340, 10: 340]
	DIS	-AIL	DISPLAY [10: VP10]
134	DIS2	6AIL	DISPLAY [6: 340, 10: 340]
	DIS2	-AIL	DISPLAY [10: VP10]
140	PLT	-AIL	PLOTTER [10: XY10]
144	PLT2	-AIL	PLOTTER [10: XY10]
150	CR	-AIL	CARD READER [10: CR10]
154	CR2	-AIL	CARD READER [10: CR10]
160		6	PDP-7, 8 INTERFACE TO PDP-6
160,164	DLB2,DLC2	-AIL	PDP-11 DATA LINK [10: DL10]
170	DSK	-AIL	SMALL DISK [10: RC10, RM10B
174	DSK2	-AIL	SMALL DISK [10: RC10]

200	DC	6--	DATA CONTROL
	DTE0	L	10/11 INTERFACE [10: DTE20]
204	DC2	6--	DATA CONTROL
	DTE1	L	10/11 INTERFACE [10: DTE20]
210,214	UTC,UTS	6--	DECTAPE [6: 351]
210	DTE2	L	10/11 INTERFACE [10: DTE20]
214	DTE3	L	10/11 INTERFACE [10: DTE20]
220,224 ,230	MTC,MTC ,MTM	6--	MAGNETIC TAPE
220	PDC	-AIL	PROGRAMMABLE DATA CHANNEL
224	PDC2	-AIL	PROGRAMMABLE DATA CHANNEL
230	LPT3	-AIL	LINE PRINTER
240	DLS	-AIL	DATA LINE SCANNER [10: DC10]
244	DLS2	-AIL	DATA LINE SCANNER [10: DC10]
250	DPC	-AIL	DISK PACK SYSTEM [10: RP10]
254	DPC2	-AIL	DISK PACK SYSTEM [10: RP10]
260	DPC3	-AIL	DISK PACK SYSTEM [10: RP10]
264	DPC4	-AIL	DISK PACK SYSTEM [10: RP10]
270	DF	6--	DISK FILE [6: 270]
	RMC	-AI-	ROTATING MEMORY CONTROLLER (10: RH10)
274	RMC2	-AI-	ROTATING MEMORY CONTROLLER (10: RH10)
300,304	DCSA,DCSB	6---	DATA COMMUNICATION [6: 630]
310			UNUSED - RESERVED TO DEC
314			UNUSED - RESERVED TO DEC
320,324	DTC,DTS	-AIL	DECTAPE [10: TD10]
330,334	DTC2,DTS2	-AIL	DECTAPE [10: TD10]
340,344	TMC,TMS	-AIL	MAGNETIC TAPE [10: TM10]
350,354	TMC2,TMS2	-AIL	MAGNETIC TAPE [10: TM10]
360			UNUSED - RESERVED TO DEC
364			UNUSED - RESERVED TO DEC

370	UNUSED - RESERVED TO DEC
374	UNUSED - RESERVED TO DEC
400	RESERVED FOR CUSTOMER SPECIAL DEVICES
404	RESERVED FOR CUSTOMER SPECIAL DEVICES
410	RESERVED FOR CUSTOMER SPECIAL DEVICES
414	RESERVED FOR CUSTOMER SPECIAL DEVICES
420	RESERVED FOR CUSTOMER SPECIAL DEVICES
424	RESERVED FOR CUSTOMER SPECIAL DEVICES
430	RESERVED FOR CUSTOMER SPECIAL DEVICES
434	RESERVED FOR CUSTOMER SPECIAL DEVICES
440	UNUSED - RESERVED TO DEC
444	UNUSED - RESERVED TO DEC
450	UNUSED - RESERVED TO DEC
454	UNUSED - RESERVED TO DEC
460,464 DSS,DSI -AIL	SINGLE SYNCHRONOUS LINE UNIT [10: DS10]
470,474 DSS2,DSI2 -AIL	SINGLE SYNCHRONOUS LINE UNIT [10: DS10]
500	RESERVED FOR CUSTOMER SPECIAL DEVICES
504	RESERVED FOR CUSTOMER SPECIAL DEVICES
510	RESERVED FOR CUSTOMER SPECIAL DEVICES
514	RESERVED FOR CUSTOMER SPECIAL DEVICES

520				RESERVED FOR CUSTOMER SPECIAL DEVICES
524				RESERVED FOR CUSTOMER SPECIAL DEVICES
530				RESERVED FOR CUSTOMER SPECIAL DEVICES
534				RESERVED FOR CUSTOMER SPECIAL DEVICES
540	MBC0	---L	4.1	MASSBUS CONTROLLER [10: RH20]
544	MBC1	---L	4.1	MASSBUS CONTROLLER [10: RH20]
550	MBC2	---L	4.1	MASSBUS CONTROLLER [10: RH20]
554	MBC3	---L	4.1	MASSBUS CONTROLLER [10: RH20]
560	MBC4	---L	4.1	MASSBUS CONTROLLER [10: RH20]
564	MBC5	---L	4.1	MASSBUS CONTROLLER [10: RH20]
570	MBC6	---L	4.1	MASSBUS CONTROLLER [10: RH20]
574	MBC7	---L	4.1	MASSBUS CONTROLLER [10: RH20]
600				RESERVED FOR CUSTOMER SPECIAL DEVICES
604				RESERVED FOR CUSTOMER SPECIAL DEVICES
610				RESERVED FOR CUSTOMER SPECIAL DEVICES
614				RESERVED FOR CUSTOMER SPECIAL DEVICES
620				RESERVED FOR CUSTOMER SPECIAL DEVICES
624				RESERVED FOR CUSTOMER SPECIAL DEVICES
630				RESERVED FOR CUSTOMER SPECIAL DEVICES
634				RESERVED FOR CUSTOMER SPECIAL DEVICES
640				UNUSED - RESERVED FOR DEC

644	UNUSED - RESERVED FOR DEC
650	UNUSED - RESERVED FOR DEC
654	UNUSED - RESERVED FOR DEC
660	UNUSED - RESERVED FOR DEC
664	UNUSED - RESERVED FOR DEC
670	UNUSED - RESERVED FOR DEC
674	UNUSED - RESERVED FOR DEC
700	RESERVED FOR CUSTOMER SPECIAL DEVICES
704	RESERVED FOR CUSTOMER SPECIAL DEVICES
710	RESERVED FOR CUSTOMER SPECIAL DEVICES
714	RESERVED FOR CUSTOMER SPECIAL DEVICES
720	RESERVED FOR CUSTOMER SPECIAL DEVICES
724	RESERVED FOR CUSTOMER SPECIAL DEVICES
730	RESERVED FOR CUSTOMER SPECIAL DEVICES
734	RESERVED FOR CUSTOMER SPECIAL DEVICES
740	KI/KL UNRESTRICTED* CODES RESERVED FOR CUSTOMERS
744	KI/KL UNRESTRICTED CODES RESERVED FOR CUSTOMERS
750	KI/KL UNRESTRICTED CODES RESERVED FOR CUSTOMERS
754	KI/KL UNRESTRICTED CODES RESERVED FOR CUSTOMERS
760	KI/KL UNRESTRICTED* CODE RESERVED FOR DEC
764	KI/KL UNRESTRICTED* CODE

		RESERVED FOR DEC
770		KI/KL UNRESTRICTED* CODE RESERVED FOR DEC
774		KI/KL UNRESTRICTED* CODE RESERVED FOR DEC

*UNRESTRICTED MEANS I/O INSTRUCTIONS CAN BE DONE IN USER MODE (WITHOUT BEING IN USER IOT MODE)

5. EXTEND OPCODES

THE EXTEND INSTRUCTION DETERMINES A SECONDARY OPCODE FROM THE C(E), THE FOLLOWING IS THE LIST OF SECONDARY OPCODES:

000		ILLEGAL - ILLEGAL INSTRUCTION TRAP
001	CMPSL	COMPARE STRINGS, SKIP IF LESS
002	CMPSE	COMPARE STRINGS, SKIP IF EQUAL
003	CMPSLE	COMPARE STRINGS, SKIP IF LESS OR EQUAL
005	CMPSGE	COMPARE STRINGS, SKIP IF GREATER OR EQUAL
006	CMPSN	COMPARE STRINGS, SKIP IF NOT EQUAL
007	CMPSG	COMPARE STRINGS, SKIP IF GREATER
004	EDIT	PORCESS STRING ACCORDING TO MINI-PROGRAM PATTERN
010	CVTDBO	CONVERT DECIMAL TO BINARY BY OFFSET
011	CVTDBT	CONVERT DECIMAL TO BINARY BY TRANSLATION
012	CVTBDO	CONVERT BINARY TO DECIMAL BY OFFSET
013	CVTBDT	CONVERT BINARY TO DECIMAL BY TRANSLATION
014	MOVSO	MOVE STRING WITH BYTE OFFSET
015	MOVST	MOVE STRING WITH BYTE TRANSLATION
016	MOVSLJ	MOVE STRING UNMODIFIED WITH LEFT JUSTIFICATION
017	MOVSRJ	MOVE STRING UNMODIFIED WITH RIGHT JUSTIFICATION
20	EBLT	EXTEND-BLT
21-777		ILLEGAL - ILLEGAL INSTRUCTION TRAPS

[END OF CH2S12.SPC]

1080,2040,2060 ENGINEERING FUNCTIONAL SPEC - CHAP 2,15

TO: KL10 LIST, J. PARSLOW (200 FILE)

TITLE: MICRO-CODE SPECIAL FUNCTIONS (PROTO & PRODUCTION) REV 1

STATUS: THIS CHAPTER IS A LIST OF THE SPECIAL FUNCTIONS AS IMPLEMENTED ON THE PROTOTYPE AND PRODUCTION MACHINE. THE LIST IS MAINLY INTENDED FOR CHECKOUT PURPOSES, AND THE INFORMATION IS PRESENTED IN SUCH A WAY AS TO AID MAINLY IN DIAGNOSIS.

FILE: [EFS]CH2S15.SPC

PDM #: 200-200-011-01

DATE: 24 APR 75

SUPERSEDED MEMOS: NONE

ENGINEER: R. REID

APPROVED:

EDITOR: T. HASTINGS

TYPIST: M. PROUTY

REVIEWED:

ABSTRACT

EACH MICRO-CODE INSTRUCTION IN THE EBOX IS 75 BITS WIDE. CERTAIN FUNCTIONS DO NOT OCCUR FREQUENTLY ENOUGH TO JUSTIFY OCCUPYING DISTINCT BITS OR FIELDS. CONSEQUENTLY THESE FUNCTIONS THEMSELVES ARE MICRO-CODED IN THREE SEPARATE FIELDS, CALLED SPECIAL FUNCTION FIELDS. THIS SPEC (2P15R0.SPC) IS FOR THE PROTOTYPE AND PRODUCTION MACHINES. SEE 2B15R0.SPC FOR BREADBOARD.

REVISION HISTORY

REV	DESCRIPTION	CHG NO	ORIG	DATE	APPD BY	DATE
0	ORIGINAL			20 JAN 75		
1	PROOF READ			1 APR 75		

0. CONTENTS

1. INTROUCTION

EACH MICRO-CODE WORD IS 75 BITS WIDE. CERTAIN FUNCTIONS DO NOT OCCUR FREQUENTLY ENOUGH TO JUSTIFY OCCUPYING DISTINCT BITS OR FIELDS. CONSEQUENTLY THESE FUNCTIONS THEMSELVES ARE MICRO-CODED IN THREE SEPARATE FIELDS, CALLED SPECIAL FUNCTION FIELDS. EVERY MICRO-CODE WORD CONTAINS THESE THREE FIELDS. EACH MICRO-INSTRUCTION MAY SELECT ONE OF THE FUNCTIONS IN EACH OF THE THREE FIELDS. THE 3 FIELDS ARE:

1. MBOX (MEM) FUNCTIONS (4 BITS)

2. DISPATCH (DISP) AND MISCELLANEOUS (SPEC) (5 BITS)

3. SKIP (SKIP) AND CONDITIONAL (COND) FUNCTIONS (6 BITS)

BECAUSE OF THE MICRO-CODED NATURE OF THESE 3 FIELDS, THE SECOND FIELD CAN CONTAIN EITHER A DISP OR A SPEC FUNCTION, BUT NOT BOTH. SIMILARLY THE THIRD FIELD CAN CONTAIN EITHER A SKIP OR A COND FUNCTION, BUT NOT BOTH.

2. MEM/FUNCTIONS

2.1 GENERAL

A NON-ZERO MEM FIELD IS REQUIRED IN THE MICRO-CODE WHENEVER EITHER AN EBOX REQ IS TO BE SENT TO THE MBOX, OR THE MICRO-CODE IS REQUIRED TO WAIT FOR AN MBOX RESPONSE. BOTH OF THESE REQUIREMENTS ARE DECODED FROM THE CRAM MEM BITS (SEE TABLE BELOW).

2.2 MEM FUNCTIONS DECODED

MEM/FUNCTION		WAIT	NOTE	NOTE	DESCRIPTION
00	-				NOP
01	ARL IND			#	AR LEFT INDEPENDENT
02	MB WAIT	X			MB WAIT
03	SECTION ZERO	X			FORCE SECTION ZERO
04	AREAD		CR	A	AREAD, LOAD AR
05	B WRITE		CR		B WRITE CONDITIONAL
06	FETCH	X	CR	#	FETCH, LOAD ARX
07	REG FUNC	X	UR	#	REGISTER FUNCTION
10	A @		UR		LOAD AR, LOAD ARX
11	B @		UR		LOAD AR, LOAD ARX
12	LOAD AR	X	UR		LOAD AR
13	LOAD ARX	X	UR		LOAD ARX
14	AD FUNC		UR	AD	LOAD FUNC FROM AD
15	BREAD		UR		LOAD AR, LOAD ARX
16	WRITE	X	UR		STORE AR
17	RPW CYCLE	X	UR		READ-PAUSE-WRITE (LOAD AR)

NOTES

WAIT	GENERATES MBOX WAIT (IF MEM CYCLE .AND. -AC REF)
CR	CONDITIONAL REQUEST (SEE INDIVIDUAL DESCRIPTIONS)
UR	UNCONDITIONAL REQUEST
#	CRAM # FIELD IS FURTHER DECODED
AD	AD (00-12) FIELD IS FURTHER DECODED
A	DRAM A FIELD IS FURTHER DECODED

2.3 MEM FUNCTIONS (DETAILED)

00 DEFAULT

01 MEM/ARL IND
DECODES THE CRAM # FIELD AS FOLLOWS (CTL2)

#00	SUBROUTINE CALL
#01	AR (00-08) LOAD
#02	MQ CLR
#03	ARX CLR
#04	ARL CLR
#05	ARR CLR
#06	ARL SEL 4
#07	ARL SEL 2
#08	ARL SEL 1

02 MEM/MB WAIT
WAIT FOR MBOX RESP

03 MEM/SECTION ZERO
LOAD VMA (13-17) WITH 00000

04 MEM/AREAD
THIS FUNCTION IS CONTROLLED BY THE DRAM A FIELD, (THE FUNCTION MEM/AREAD SHOULD BE GIVEN BY THE MICROCODE SIMULTANEOUSLY WITH DISP/AREAD, TO DISPATCH ON THE DRAM A FIELD), THE FIELD IS DECODED AS FOLLOWS

DRAM A

000	
001	FETCH (SEE NOTE)
010	WRITE (UNUSED)
011	PAUSE, WRITE (TEST WRITABILITY)
100	LOAD AR, READ
101	LOAD AR, READ
110	LOAD AR, READ, WRITE
111	LOAD AR, READ, PAUSE, WRITE

NOTE THE CRAM WORD IS SIMULTANEOUSLY GENERATING VMA/PC+1, IF THE DRAM A FIELD IS NOT 001, MCL VMA_AD IS GENERATED TO OVERRIDE THE MICROCODE AND LOAD VMA FROM AD (EFFECTIVE ADDRESS)

NOTE SEE ALSO DISP/AREAD

05 MEM/B WRITE
CONDITIONALLY GENERATE AN MBOX CYC REQ IF DRAM B 01 IS TRUE (MCL5). IT IS INTENDED TO BE USED WITH DISP/DRAM B.

06 MEM/FETCH
CONDITIONALLY GENERATES AN MBOX CYC REQ. (MCL5), THE CONDITION BEING -PI CYCLE OR SKIP SATISFIED OR FETCH EN IN. THE CRAM # FIELD IS DECODED AS SHOWN

#00	UNCONDITIONAL FETCH (NOT GIVEN IN SKIP TESTS)
#01	SKIP TEST
#02	JUMP TEST
#06, 07, 08	SKIP & JUMP CONDITION (DECODE DRAM B FIELD)

00X	UNUSED
01X	38 BIT AD COMPARE
10X	36 BIT COMPARE
11X	36 BIT BIT TEST

NOTES:

CONDITIONAL SKIP	VMA/PC+1, MEM/FETCH, #/20X
CONDITIONAL JUMP	VMA/PC+1, MEM/FETCH, #/50X

07 MEM/REG FUNC
THE CRAM # FIELD IS FURTHER DECODED AS SHOWN

#00	MCL REG FUNC (MCL6)
#01	APR EBOX LOAD REG (APR6)
#02	APR EBOX READ REG (APR6)
#06, 07, 08	(APR6)

1	APR EBOX CCA
2	APR EBOX UBR
3	APR EBOX EBR
4	APR EBOX ERA
5	APR EN REFILL RAM WR
6	(SPARE)
7	APR EBOX SBUS DIAG

10 MEM/A@
USED TO LOAD THE @ WORD OF AN EFFECTIVE ADDRESS CYCLE. THE LOGIC ON MCL4 IS USED TO CALCULATE THE MEMORY CONTEXT IN WHICH THE REFERENCE IS MADE.

11 MEM/B@
USED TO LOAD THE @ WORD OF A BYTE POINTER EFFECTIVE ADDRESS CYCLE. THE LOGIC ON MCL4 IS USED TO CALCULATE THE MEMORY CONTEXT IN WHICH THE REFERENCE IS MADE

12 MEM/LOAD AR
STANDARD READ REFERENCE

13 MEM/LOAD ARX
STANDARD READ REFERENCE

14 MEM/AD FUNC
USED BY BOTH DIAGNOSTICS AND BY THE MICROCODE PAGE REFILL ALGORITHM. THE MBOX CYC REQUIRED IS DETERMINED BY AD BITS AS SHOWN IN THE TABLE BELOW. (THESE MAY ALSO BE READ VIA COND/SEL VMA)

PC AND VMA FLAGS

PC FLAG (SCD)		VMA FLAG (MCL3)
00	PC OVFL	-
01	CRY0	LOAD AR
02	CRY1	LOAD ARX
03	FOV	PAUSE
04	FPD	WRITE
05	USER	VMA USER
06	USER IOT	VMA PUBLIC
07	PUBLIC	VMA PREVIOUS
08	ADR BRK INH	VMA EXTENDED
09	TRAP REQ 2	VMA FETCH
10	TRAP REQ 1	EBOX MAP
11	FXU	-EBOX CACHE
12	DIV CHK.	-EBOX PAGED

15 MEM/BREAD
USED TO READ THE BYTE DATA WORD AFTER THE BYTE POINTER EFFECTIVE ADDRESS CALCULATION IS COMPLETE. THE LOGIC ON MCL4 IS USED TO CALCULATE THE MEMORY CONTEXT IN WHICH THE REFERENCE IS MADE

16 MEM/WRITE
STANDARD WRITE REFERENCE

17 MEM/RPW CYCLE
USED BY THE MICROCODE TO LOAD AR AND DO A WRITE-TEST PRIOR TO THE STANDARD WRITE REFERENCE. INITIATES A READ-PAUSE-WRITE CYCLE IF THE ACCESS IS NON-CACHED.

3. DISP/FUNCTIONS

3.1 GENERAL

THIS FIVE-BIT FIELD IS LOCATED ON THE CRA BOARD FOR HARDWARE TIMING REASONS ASSOCIATED WITH THE DISP FUNCTIONS. OF THE 32 POSSIBLE CODES, 16 ARE DISP (DISPATCH) FUNCTIONS AND 16 ARE SPEC (SPECIAL) FUNCTIONS. FUNCTION 00 IS THE DISP/DIAG ADR TO PERMIT EASILY STARTING OF THE MICRO-CODE AT AN ADDRESS ASSIGNED BY THE CONSOLE. THE NOP (DEFAULT) FUNCTION IS FUNCTION 10 (SPEC/NOP).

3.2 DISP/FUNCTION DECODED

THE DISPATCH FUNCTIONS MODIFY THE CRA ADDRESS AS PRODUCED BY THE CRA BOARD. THE CURRENT CRAM J FIELD IS "OR"ED INTO THE OUTPUT OF THE DISPATCH LOGIC ON CRA1 AND CRA2. A SKIP FUNCTION MAY BE USED TO FURTHER MODIFY THE LEAST SIGNIFICANT ADDRESS BIT.

00	DIAGNOSTIC
01	DRAM J
02	AREAD
03	RETURN
04	PAGE FAIL
05	STATE REG
06	NICOND
07	SHIFT (00-03)
30	MULTIPLY
31	DIVIDE
32	SIGNS
33	DRAM B
34	BYTE
35	NORM
36	EA MOD
37	EA TYPE

3.3 DISP FUNCTIONS (DETAILED)

00 DISP/DIAG (CRA3)
11 BIT DISPATCH ON THE CONTENTS OF THE DIAGNOSTIC ADDRES REGISTER. THIS IS THE DEFAULT DISPATCH (00) AFTER RESET, TO PERMIT STARTING OF THE MICROCODE AT AN ADDRESS LOADED BY THE CONSOLE.

01 DISP/DRAM J (IR1)
DISPATCH ON THE DRAM J FIELD. NOTE THAT THE DRAM J FIELD DOES NOT HAVE BITS IN POSITION 00, 05, 06, AND THAT AN EVEN-ODD PAIR MAY DIFFER ONLY IN POSITION 08, 09, 10 (I.E., THE LAST OCTAL DIGIT).

I.E., DRAM J = 0C CCC 00C XXX
WHERE C IS COMMON ADDRESS BIT
X IS UNIQUE ADDRESS BIT.

02 DISP/AREAD (CRA3)
DISPATCH CONTROLLED BY THE DRAM A FIELD. IF A IS 00 OR 01, THEN THE DISPATCH IS ON THE DRAM J FIELD (IDENTICAL TO DISP/DRAM J). FOR VALUES OF A FROM 02 TO 07, THE DISPATCH IS TO LOCATION 42 TO 47, RESPECTIVELY. THIS FUNCTION IS INTENDED TO BE USED SIMULTANEOUSLY WITH MEM/AREAD. REFER TO MEM/AREAD FOR FURTHER INFORMATION.

03 DISP/RETURN (CRA4)
SUBROUTINE RETURN DISPATCH. THE RETURN ADDRESS MAY OF COURSE BE MODIFIED BY THE J FIELD. THIS FUNCTION POPS THE RETURN ADDRESS OFF THE STACK. NOTE THAT SUBROUTINE CALL IS NOT A DISP FUNCTION, BUT IS DONE VIA SPEC/CALL, OR VIA ARL IND (MEM, SPEC, OR COND FUNCTION).

04 DISP/PAGE FAIL (CLK4)
DISPATCH ON PAGE FAIL CAUSE.

05 DISP/STATE REG (CON3)
16-WAY DISPATCH ON STATE REGISTER. THE STATE REGISTER IS LOADED VIA COND/SR<-#

06 DISP/NICOND (CON2)
16-WAY DISPATCH EVOKED AT THE END OF EVERY MACRO INSTRUCTION TO DETERMINE THE NEXT INSTRUCTION CONDITION. THE NEXT INSTRUCTION MUST PREVIOUSLY HAVE BEEN REQUESTED, AND THE REQUEST MUST HAVE BEEN FOLLOWED BY AT LEAST ONE MICROCODE INSTRUCTION BEFORE THE DISP/NICOND. THIS FUNCTION IS INTENDED TO BE USED WITH A MEM/MB WAIT FUNCTION.

NICOND DISPATCHES AS FOLLOWS (00 IS HIGHEST PRIORITY)

00 PI CYCLE IN PROGRESS
02 STOP (RUN FLAG IS OFF)
04 MTR INTERRUPT REQUEST. (HIGHER PRIORITY THAN NORMAL INTERRUPTS)
06 STANDARD INTERRUPT REQUESTS

12 NORMAL (INSTRUCTION IS IN ARX)
13 TRAP REQUEST

16 AC REF (INSTRUCTION IS IN FAST MEMORY)
17 TRAP REQUEST (AND AC REF)

30 DISP/MULTIPLY (CRA2)
DURING THE MULTIPLY ROUTINE, THE LOOP COUNTER IS COUNTED DOWN IN THE FE REGISTER. DISP/MULTIPLY DISPATCHES AS FOLLOWS

00 - 03 FE POSITIVE, DISPATCH ON MQ34, MQ35
07 FE NEGATIVE

31 DISP/DIVIDE (CRA2)
DISPATCH 8-WAYS ON FE 00, BR 00, AD CRY 00

32 DISP/SIGNS (CRA2)
DISPATCH 8-WAYS ON AR 00, BR 00, AD 00

33 DISP/DRAM B (CRA2)
DISPATCH 8-WAYS ON DRAM B FIELD. OFTEN USED IN CONJUNCTION WITH MEM/BWRITE

34 DISP/BYTE (CRA2)
DISPATCH 8 WAYS ON FIRST PART DONE, BR 12, SCAD 00

35 DISP/NORM (IR3)
DISPATCH EVOKED DURING NORMALIZE ROUTINES TO DISPATCH ONE OF 8-WAYS ON THE MOST SIGNIFICANT ONE BIT IN AD

0	ZERO
1	NEGATIVE
2	AD (01-06)
3	AD 07
4	AD 08
5	AD 09
6	AD 10
7	AD (11-35)

36 DISP/EA MOD (CRA2, SH1)
USED DURING EFFECTIVE ADDRESS CALCULATION, @*2 + (XR)*1

37 DISP/EA TYPE (MCL4)
USED DURING PXCT AND SXCT EFFECTIVE ADDRESS CALCULATIONS

00	NORMAL
01	(PXCT 4 AT AREAD, PXCT 1 AT BREAD) ,AND, CWSX
10	SXCT 0
11	SXCT B

4. SPEC/FUNCTIONS

4.1 GENERAL

THIS FIVE BIT FIELD IS LOCATED ON THE CRA BOARD, WHERE IT IS LOCATED FOR HARDWARE TIMING REASONS ASSOCIATED WITH THE DISP FUNCTION, WITH WHICH THE FIELD IS SHARED, THUS A DISP FUNCTION AND A SPEC FUNCTION CANNOT BE GIVEN IN THE SAME MICRO-CODE WORD.

4.2 SPEC FUNCTION (CTL1)

10	-	20	SECTION HOLD
11	INH CRY 18	21	SBR CALL
12	MQ SHIFT	22	ARL IND
13	SCM ALT	23	MTR CTL
14	CLR FPD	24	FLAG CTL
15	LOAD PC	25	SAVE FLAGS
16	XCRY AR0	26	SP MEM CYCLE
17	GEN CRY 18	27	AD LONG

4.3 SPEC FUNCTIONS (DETAILED)

10 -
DEFAULT NOP, NOTE THAT FUNCTION 00 IS THE DISP/DIAG FUNCTION, USED FOR STARTING THE MICROCODE

11 SPEC/INH CRY 18 (CTL1)
INHIBITS CARRY INTO THE AD LEFT

12 SPEC/MQ SHIFT (CTL2)
USED WITH THE CRAM MQ SEL 1 BIT TO SELECT EITHER OF THE SHIFT INPUTS TO THE MQ REGISTER, THE VARIOUS INPUTS TO THE MQ REGISTER ARE ALSO UNDER CONROL OF DISP/MUL, DISP/DIV, COND/REG CONTROL, AND RESET

13 SPEC/SCM ALT (SCD2)
USED WITH CRAM SCM SEL 2 TO SELECT ONE OF THE FOUR INPUTS TO THE SC REGISTER

14 SPEC/CLR FPD (SCD4)
USED TO CLEAR SCD FPD FLAG

15 SPEC/LOAD PC (CTL1, VMA3)
USED TO LOAD THE PC FROM THE VMA REGISTER

16 SPEC/XCRY AR0 (CTL1)
COMPLEMENTS THE CARRY IN TO THE AD IF THE AR IS NEGATIVE

17 SPEC/GEN CRY 18 (CTL1, IR4)
GENERATES CARRY IN TO THE LEFT HALF OF AD

20 SPEC/SECTION HOLD (MCL4)
FORCES RECIRCULATION OF THE CURRENT VMA SECTION (VMA 13-17)

21 SPEC/SBR CALL (CTL2, CRA4)
PUSHES THE CURRENT MICROCODE ADDRESS ONTO THE SUBROUTINE STACK. NOTE THAT THIS MERELY DEFINES THE RETURN AREA OF THE SUBROUTINE, THE SUBROUTINE NEED NOT BE ENTERED YET, OR EVEN AT ALL. THE CRAM J FIELD IS USED TO MODIFY THIS ADDRESS WHEN RETURNING FROM THE SUBROUTINE.

22 SPEC/ARL IND (CTL2)
DECODES THE CRAM # FIELD AS FOLLOWS

#00	SUBROUTINE CALL
#01	AR(00-08) LOAD
#02	MQ CLR
#03	ARX CLR
#04	ARL CLR
#05	ARR CLR
#06	ARL SEL 4
#07	ARL SEL 2
#08	ARL SEL 1

23 -

24 SPEC/FLAG CTL
DECODES THE CRAM # FIELD AS FOLLOWS

#00	LEAVE USER	CLEARS USER, USER IOT, PUBLIC (SCD5)
#01	SPEC JFCL	OVERRIDES EXEC MODE TO SELECT SCD OV INSTEAD OF SCD PCP (SCD4, SCD5)
#02	PI DISMISS	DISMISS PI CHANNEL (CON5)
#03	HALT	CLEARS RUN FLAG (CON2) IF IO LEGAL
#04	LOAD FLAGS	LOAD PC FLAGS FROM AR (SCD4)
#05	PORTAL	CLR PUBLIC (SCD5) IF PRIVATE PAGE
#07	LOAD FLAGS	LEAVE USER MODE (SCD5)

25 SPEC/SAVE FLAGS (CTL1,
HAS THE FOLLOWING EFFECTS
INHIBITS ADX CRY 36 IF PC+1 INH IS TRUE (CTL1)
SETS PI HOLD AND CLEARS PI CYCLE IF PI CYCLE IS TRUE (CON5)
GENERATES SCD LEAVE USER IF IN PI CYCLE (SCD5)

26 SPEC/SP MEM CYCLE
USED FOR EXTRA CONTROL OF THE MEMORY FUNCTION REQUESTED SIMULTANEOUSLY VIA A MEM/FUNC CODE. DECODES THE CRAM # FIELD AS FOLLOWS

#00	UNCONDITIONAL FETCH	(MCL5)
#01	USER MODE	(MCL2)
#02	EXEC MODE	(MCL3)
#03	FORCE SECTION ZERO	(MCL4)
#04	UPT REFERENCE IF USER	(MCL3)
#05	EPT REFERENCE IF EXEC	(MCL3)
#06	-	
#07	INHIBIT EBOX CACHE	(MCL6)
#08	INHIBIT EBOX MAY BE PAGED	(MCL6)

27 SPEC/AD LONG (CTL1, IR4)
ENABLES CARRIES FROM ADX TO AD. SO DO THE FUNCTIONS DISP/MUL, DISP/DIV, DISP/NORM, AND SPEC/MQ SHIFT.

5. SKIP/FUNCTIONS

5.1 GENERAL

THE SKIP AND COND FUNCTIONS SHARE THE SAME SIX-BIT SPECIAL FIELD. OF THE 64 POSSIBLE FUNCTIONS, 32 ARE CONDITIONAL SKIP FUNCTIONS, CONTROLLING CR ADR 10 (THE LEAST SIGNIFICANT OF THE CRAM ADDRESS LINES.

5.2 SKIP FUNCTION

40	-	CRA2
41	-ODD PARITY	CRA2
42	BR 00	CRA2, EDP4
43	ARX 00	CRA2, EDP2
44	AR 18	CRA2, EDP1
45	AR 00	CRA2, EDP1
46	AC = 0	CRA2, IR1
47	SC 00	CRA2,
50	SC .LT. 36	CRA2,
51	SCAD 00	CRA2,
52	-SCAD=0	CRA2,
53	ADX 00	CRA2, EDP3
54	AD CRY 00	CRA2, IR4
55	AD 00	CRA2, IR3
56	-AD=0	CRA2, IR3
57		
60	FETCH	CON2, CRA2
61	KERNEL MODE	CON2, CRA2
62	USER	CON2, CRA2
63	PUBLIC	CON2, CRA2
64	RD-PSE-WR	CON2, CRA2
65	PI CYCLE	CON2, CRA2
66	EBUS GRANT	CON2, CRA2
67	PI TRANSFER	CON2, CRA2
70	INT REQ	CON2, CRA2
71	-START	CON2, CRA2
72	RUN	CON2, CRA2
73	I/O LEGAL	CON2, CRA2
74	PXCT OR SXCT	CON2, CRA2
75	EBOX PF HANDLE	CON2, CRA2
76	VMA AC REF	CON2, CRA2
77	MTR INT REQ	CON2, CRA2

6. COND/FUNCTIONS

6.1 GENERAL

THE SKIP AND COND FUNCTIONS SHARE THE SAME SIX-BIT SPECIAL FIELD. OF THE 64 POSSIBLE FUNCTIONS, 32 ARE SPECIAL COND FUNCTIONS. THESE DO THE SAME TYPES OF THINGS AS THE SPEC FUNCTIONS, AND ARE ONLY DISTINGUISHED BY COMING FROM A SEPARATE MICRO-CODE FIELD.

6.2 COND FUNCTIONS (CON1, CTL2, SCD3)

00	-	20	DIAG FUNC
01	AR (00-08) LOAD	21	EBOX STATE
02	AR (09-17) LOAD	22	EBUS CTL
03	AR (18-35) LOAD	23	MBOX CTL
04	AR CLR	24	-
05	ARX CLR	25	-
06	ARL IND	26	-
07	REG CTL	27	-
10	FM WRITE	30	VMA #
11	PCF <- #	31	VMA <- #+TRAP
12	FE SHRT	32	VMA <- #+MODE
13	AD FLAGS	33	VMAE <- #+AR(32-35)
14	LOAD IR	34	VMA <- #+2N
15	SPEC INSTR	35	VMA DEC
16	SR <- #	36	VMA INC
17	SEL VMA	37	LOAD VMA HELD

6.3 COND FUNCTIONS (DETAILED)

00 DEFAULT

01 COND/ARLL LOAD (CTL2)
FORCES LOADING OF BITS 00-08 OF THE AR REGISTER

02 COND/ARLR LOAD (CTL2)
FORCES LOADING OF BITS 09-17 OF THE AR REGISTER

03 COND/ARR LOAD (CTL2)
FORCES LOADING OF BITS 18-35 OF THE AR REGISTER

04 COND/AR CLR (CTL2)
FORCES CLEARING OF THE AR REGISTER

05 COND/ARX CLR (CTL2)
FORCES CLEARING OF THE ARX REGISTER

06 COND/ARL IND (CTL2)
DECODES THE CRAM # FIELD AS FOLLOWS

#00	SUBROUTINE CALL
#01	AR(00-08) LOAD
#02	MQ CLR
#03	ARX CLR
#04	ARL CLR
#05	ARR CLR
#06	ARL SEL 4
#07	ARL SEL 2
#08	ARL SEL 1

07 COND/REG CTL (CTL2, CTL1)
DECODES THE CRAM # FIELD AS FOLLOWS

#00	AR (00-08) LOAD
#01	AR (09-17) LOAD
#02	AR (18-35) LOAD
#03	-
#04	-
#05	COND/AR <- EXP TEST
#06	-
#07	MQ SEL #2
#08	MQ SEL #1

10 COND/FM WRITE (CON1, APR5, EDP4)
GENERATES APR FM WRITE TO WRITE AR INTO FAST MEMORY

11 COND/PCF <- # (CON1, SCD5, SCD4)
SETS THE PC FLAGS FROM THE CRAM # FIELD

#00	ARITHMETIC OVERFLOW	OV
#01	FLOATING OVERFLOW	FOV
#02	FIRST PART DONE	FPD
#03	TRAP 2	TRAP REQ 2
#04	TRAP 1	TRAP REQ 1
#05	FLOATING UNDERFLOW	FXU
#06	DIVIDE CHECK	DIV CHK
#07	-	
#08	-	

12 COND/FE SHRT (CON1, SCD2)
CAUSES AN ARITHMETIC SHIFT OF FE RIGHT ONE PLACE

13 COND/AD FLAGS (CON1, SCD5, SCD4)
CONTROLS OVERFLOW AND CARRY FLAGS FROM ADDER OUTPUT

14 COND/LOAD IR (CON1, CON2, IR1)
LOADS THE IR REGISTER, THEN STROBES THE DRAM ON THE NEXT EBOX CYCLE

15 COND/SPEC INSTR (CON1, CON2, CON4, CON5, MCL4)
DECODES THE CRAM # FIELD AS FOLLOWS

#00	PI CYCLE
#01	KERNEL CYCLE
#02	PC+1 INHIBIT
#03	SXCT
#04	PXCT
#05	INTERRUPT DISABLE
#06	ABORT INSTRUCTION (INTERRUPTED)
#07	EBOX HALTED (SIGNAL TO CONSOLE)
#08	-

16 COND/SR <- # (CON1, CON2, CON3, MCL4)
DECODES CRAM # FIELD AS FOLLOWS

#00	SWITCH FROM PXCT 10,4 TO PXCT 2,1
#01	SET SRC CONDITION
#02	SET DST CONDITION
#03,04	HOLD STATUS REGISTER BITS 02, 03
#05,06	CONTROL STATUS REGISTER BITS 00, 01
#07,08	SET STATUS REGISTER BITS 02, 03

17 COND/SEL VMA (CON1, MCL3, VMA4)
SELECTS VMA HELD INSTEAD OF PC AS INPUT TO ADA MIXER ON DATA PATH

20 COND/DIAG FUNC (CON1, CON3, CTL3)
SIMULATES INPUT AND OUTPUT INSTRUCTION TO THE INTERNAL DEVICES. THE INPUT INSTRUCTIONS ARE DONE BY GENERATING THE EQUIVALENT DIAGNOSTIC READ FUNCTION ON CTL3; THE OUTPUT FUNCTION BY GENERATING VARIOUS CONO AND DATAO SIGNALS ON CON3. THE CRAM # FIELD IS DECODED AS FOLLOWS

#00 DELAY THE EBOX CLK
#01
#02 0=OUTPUT, 1=INPUT (IF 0, ENABLE AD ONTO EBUS)

#(02-08) OUTPUT FUNCTIONS

#004 WRPAE (LH) - LOAD LH PERF ANAL ENABLES
#005 WRPAE (RH)
#006 CONO MTR
#007 CONO TIM

#014 CONO APR
#015 CONO PI
#016 CONO PAG
#017 DATAO APR
#020-023 DATAO PAG
(AC BLOCKS, PREV CONTEXT CONTROLLED BY EBUS 00,01)
#024-027 DATAO PAG
(AC BLOCKS, PREV CONTEXT CONTROLLED BY #07,08)
#03X -
#04X -
#05X -
#06X -
#07X -

#(02-08) INPUT FUNCTIONS
THESE FUNCTIONS SIMULATE THE DIAGNOSTIC FUNCTIONS. HERE IS LISTED MERELY THE LOADS WHICH ARE READ.

10X - PI(00-17), MCL(18-23), CTL(24-29), CLK(30-35)
11X - APR(01, 06-17), MTR(18-35)
12X - EDP (00-35)
13X - SCD(02-11), IR(12-17), CON(18-24)
14X - CRA(00-05), CRM(08-11, 14-17, 20-23, 26-29, 32-35)
15X - VMA(13-35)
16X - MBZ(00-08, 14-26, 34-35), MBC(27-33)
17X - CRC(00-04), CCW(05-10), CH(11-14), CCL(16, 18-19), CHX(20-21), CSH(22-29), MBX(30-35)

21 COND/EBOX STATE (CON1, CON4)
DECODE THE CRAM # FIELD AS FOLLOWS

#01,02	SET, HOLD UCODE STATE 01
#03,04	SET, HOLD UCODE STATE 03
#05,06	SET, HOLD UCODE STATE 05
#07,08	SET, HOLD UCODE STATE 07

22 COND/EBUS CTL (CON1, CON3, APR3)
CONTROLS THE PI BOARD AND EXTERNAL I/O TRANSFERS VIA DECODING OF THE CRAM # FIELD

#00	RET	FORCE ECL EBUS RETURN
#01	REQ	REQUEST EBUS CONTROL
#02	REL	RELEASE EBUS AND ALL CONTROLS
#03	DEMAND	ASSERTS EBUS DEMAND
#04	I/O	ASSERTS THE I/O FUNCTION
#05	AC FUNC	SELECTS EBUS FUNCTION FROM AC10,11,12
#06	DEV 0	SELECTS DEVICE 000
#07,08	# FUNC	SELECTS EBUS FUNCTION IF #05 = 0

23 COND/MBOX CTL (CON1, APR5)
DECODES THE CRAM # FIELD AS FOLLOWS

#00	-	
#01	PAGE FAIL	SIMULATES PAGE FAIL (DIAGNOSTICS)
#02	I/O PF ER	SETS I/O PF ERR (DIAGNOSTICS)
#03		
#04	PT DIR WR	WRITE PAGE TABLE DIRECTORY
#05	PR WR	WRITE PAGE TABLE ENTRY
#06		
#07,08	FUNC	SELECT PAGE TABLE FUNCTION

24 SPARE

25 SPARE

26 SPARE

27 SPARE

ALL OF THE FUNCTION 30-34 LOAD THE VMA FROM THE CRAM # FIELD.
DECODED ON (CON1, SCD3, VMA1)

```
30      COND/VMA-#
31      COND/VMA-# + TRAP CYCLE
32      COND/VMA_# + USER*4 + PUBLIC*2 + TRAP*1
33      COND/VMA_# + AR (32-35)
34      COND/VMA_# +2N

35      COND/VMA DEC (CON1)
36      COND/VMA INC (CON1)
        DECREMENT OR INCREMENT THE VMA

37      COND/LOAD VMA HELD (CON1, MCL1, MCL3, VMA4)
        LOADS VMA HELD FROM VMA REGISTER
```

[END OF CH2S15,SPC]

1080,2040,2060 ENGINEERING FUNCTIONAL SPEC - CHAP 2,17

TO: KL10 LIST, J. PARSLOW (200 FILE)

TITLE: ISP DESCRIPTION OF PDP-6, KA, KI, KL PROCESSORS

STATUS: THIS CHAPTER REFLECTS THE ISP DESCRIPTION OF THE 4 PROCESSORS AS OF MAY 1975.

FILE: [EFS]CH2S17.SPC

PDM #: 200-200-038-00

DATE: 11 AUG 75

SUPERSEDED MEMOS: NONE

SUPERSEDED SPECS:

ENGINEER: LLOYD DICKMAN

APPROVED:

EDITOR: T. HASTINGS

TYPIST:

REVIEWED:

DISTRIBUTED:

ABSTRACT

ISP IS A FORMAL LANGUAGE FOR DESCRIBING PROCESSORS. IT WAS DEVELOPED BY GORDEN BELL.

REVISION HISTORY

REV	DESCRIPTION	CHG NO	ORIG	DATE	APPD BY	DATE

```
PDP10 :=

(DECLARE


! GENERAL COMMENTS
!     DEFINITIONS APPLY TO USER MODE (NO USER I/O)
!     FUNCTIONS NOT DEFINED - SET.FLAGS, LONG


! ITEMS TO ADD TO PDP10 INSTRUCTION SIMULATOR (SIM10)
!     INSTRUCTIONS - ADJSP, EXTENDED INSTRUCTION SET
!     USE OF LEFT HALF OF INDEX WORDS
!     USE NON-ZERO AC FIELD IN IBP
!     USE AC FIELD OF JSR
!     USE BITS 0-12 OF INDIRECT ADDRESS
```

```
! PROCESSOR TYPE
! --------------

    MACRO PDP6 := (PROCESSOR EQL 1964)$
    MACRO KA10 := (PROCESSOR EQL 1967)$
    MACRO KI10 := (PROCESSOR EQL 1972)$
    MACRO KL10 := (PROCESSOR EQL 1975)$
    MACRO UNICORN := (PROCESSOR EQL 1976)$


! PC STATE
! --------

    PC.WORD<0:35>;
        OVERFLOW\INTEGER.OVERFLOW<> := PC.WORD<0>;
        CRY.0\CARRY.OUT.OF.BIT.0<> := PC.WORD<1>;
        CRY.1\CARRY.OUT.OF.BIT.1<> := PC.WORD<2>;
        FLT.OVERFLOW\FLOATING.OVERFLOW<> := PC.WORD<3>;
        FPD\FIRST.PART.DONE<> := PC.WORD<4>;
        USER\USER.MODE<> := PC.WORD<5>;
        USER.IO\USER.IO.MODE<> := PC.WORD<6>;
        PUBLIC\PUBLIC.MODE<> := PC.WORD<7>;
        AFI\ADDRESS.FAILURE.INHIBIT<> := PC.WORD<8>;
        TRAP.2\STACK.OVERFLOW<> := PC.WORD<9>;
        TRAP.1\ARITHMETIC.OVERFLOW<> := PC.WORD<10>;
        FLT.UNDERFLOW\FLOATING.UNDERFLOW<> := PC.WORD<11>;
        NO.DIVIDE\DIVIDE.CHECK<> := PC.WORD<12>;
        PC\PROGRAM.COUNTER<18:35> := PC.WORD<18:35>;
    MAP.WORD<18:35>;
        PAGE.FAILURE<> := MAP.WORD<18>;
        P<> := MAP.WORD<19>;
        W<> := MAP.WORD<20>;
        S<> := MAP.WORD<21>;
        NO.MATCH<> := MAP.WORD<22>;
        PHYSICAL.PAGE.ADDRESS<23:35> := MAP.WORD<23:35>;


! MP STATE - PRIMARY VIRTUAL MEMORY OF 2**18 WORDS
! --------

    M\MEMORY[0:#777777]<0:35>;
    AC\ACCUMULATORS[0:#17]<0:35> := M[0:#17]<0:35>;


! BASIC INSTRUCTION FORMAT
! ------------------------

    IR\INSTRUCTION.REGISTER<0:35>;
        OPCODE<0:8> := IR<0:8>;
        A\AC.FIELD<9:12> := IR<9:12>;
        F\FUNCTION<9:12> := IR<9:12>;
```

```
! TEMPORARY DEFINITIONS
! --------------------

    MACRO MIN.2.TO.70 := #600000000000000000000000$      !
    -2**70



! TEMPORARY REGISTERS
! ------------------

    S\SOURCE.OPERAND<0:35>;
    D\DESTINATION.OPERAND<0:35>;
    SA\SOURCE.ADDRESS<18:35>;
    DA\DESTINATION.ADDRESS<18:35>;
    TEMP\TEMPORARY.REGISTER<0:35>;
    TEMP1\TEMPORARY.REGISTER<0:35>;
    TEMPF\TEMPORARY.FLAG.REGISTER<9:12>;
    MASK\MASK.WORD<0:35>;
    SP\STACK.POINTER<0:35>;
    BYTE.PTR\BYTE.POINTER<0:35>;
        POS\POSITION<0:5> := BYTE.PTR<0:5>;
        SIZE<6:11> := BYTE.PTR<6:11>;
    VMA\VIRTUAL.MEMORY.ADDRESS<13:35>;
        I\INDIRECT<13> := VMA<13>;
        X\INDEX<14:17> := VMA<14:17>;
        Y\OFFSET<18:35> := VMA<18:35>;
    E\EFFECTIVE.ADDRESS<18:35>;
    E0\EXTEND.INSTRUCTION.EFFECTIVE.ADDRESS<18:35>;
    E1\EXTENDED.INSTRUCTION.EFFECTIVE.ADDRESS<18:35>;
    LAST\LAST.ADDRESS.IN.EA<18:35>;
    SC\POSITIONS.TO.SHIFT<0:8>;
    LINK\SHIFT.OUT.BIT<>;
    NULL\SHIFT.IN.BIT<>;
    N0\RELATIVE.BYTE.IN.WORD<0:5>;
    N1\BYTES.PER.WORD<0:5>;
    Q\WORDS.TO.CROSS<18:35>;
    R\BYTES.IN.FINAL.WORD<0:5>;
    EXP1\FLOATING.POINT.EXPONENT<0:8>;
    EXP2\FLOATING.POINT.EXPONENT<0:8>;
        MACRO F62 := 35$          ! 62  THESE DEFINITIONS ARE
        USED
        MACRO F63 := 35$          ! 63  TO BYPASS A PROBLEM
        MACRO F97 := 35$          ! 97  IN THE ISP COMPILER WHICH
        MACRO F71 := 35$          ! 71  PROHIBITS MORE THAN 36
        BITS
        MACRO F36 := 35$          ! 36  IN A REGISTER
        MACRO F72 := 35$          ! 72
        MACRO F107 := 35$         ! 107
        MACRO F108 := 35$         ! 108
        MACRO F143 := 35$         ! 143
        MACRO F70 := 35$          ! 70
   MAN1\FLOATING.POINT.MANTISSA<0:F97>;
```

```
    MAN2\FLOATING,POINT,MANTISSA<0:F97>

ERALCED ! END OF REGISTER DECLARATIONS
```

```
! BASIC FUNCTIONS
! ---------------

    EA := (      ! EFFECTIVE ADDRESS CALCULATION
        (IF X NEQ 0 => Y_AC[X]+Y; LAST_X) NEXT
        (DECODE I =>
            VMA<13:17>_0;
            (VMA_M[Y]<13:35> NEXT LAST_VMA NEXT EA)));



    MUUO := (FPD_0; AFI_0; TRAP.1_0; TRAP.2_0);



    NOP := TEMP_TEMP;



    SHIFT.COUNT := SC_E<18>@E<28:35>;
```

```
! BYTE MANIPULATION FUNCTIONS
! ---------------------------

    VALIDATE.BYTE.POINTER := (
        (IF (SIZE GTR POS) AND (SIZE GTR 36) => POS_100-SIZE);
        (IF SIZE GTR 36 => SIZE_36; POS_0) NEXT
        (IF POS GEQ 36 => SIZE_0);
        (IF (POS LSS 36) AND (SIZE LSS 36) AND (POS+SIZE GTR
        36) =>
            SIZE_36-POS)
        );



    ADJUST.BYTE.POINTER := (
        (IF SIZE NEQ 0 =>
            N0_TEMP1+((36-POS)/SIZE);    ! BYTE IN WORD
            N1_36/SIZE NEXT      ! BYTES/WORD
            (IF N1 EQL 0 => OVERFLOW_1; TRAP.1_1; NO.DIVIDE_1);
            (IF N1 NEQ 0 =>
                Q_N0/N1;         ! WORDS TO CROSS
                R_N0-(N0/N1)*N0 NEXT     ! BYTES REMAINING
                POS_36-(R*SIZE);
                (IF PDP6 OR KA10 => BYTE.PTR_BYTE.PTR+Q); ! NOT
                IF INTERRUPTED
                (IF KI10 OR KL10 OR UNICORN =>
                BYTE.PTR<18:35>_TEMP<18:35>+Q)))
       );



    LOAD.BYTE := (
        VALIDATE.BYTE.PTR NEXT
        VMA_BYTE.PTR<13:35> NEXT
        EA; S_0 NEXT
        (IF SIZE NEQ 0 =>
            S_M[VMA<18:35>] "SR0 POS NEXT
            S_S "SL0 (36-SIZE) NEXT
            S_S "SR0 (36-SIZE))
        );



    DEPOSIT.BYTE := (
        VALIDATE.BYTE.PTR NEXT
        VMA_BYTE.PTR<13:35> NEXT
        EA NEXT
        (IF SIZE NEQ 0 =>
            MASK_0 NEXT
            MASK_MASK "SL1 (36-SIZE) NEXT
            MASK_MASK "SR1 (36-POS-SIZE) NEXT
```

```
            D_D ^SL0 (36-SIZE) NEXT
            D_D ^SR0 (36-POS-SIZE) NEXT
            M[VMA<18:35>]_(M[VMA<18:35>] AND MASK) OR D)
        );



    INCREMENT,BYTE,PTR := (
        VALIDATE,BYTE,POINTER;
        TEMP1_1 NEXT
        ADJUST,BYTE,POINTER);
```

```
! FLOATING POINT FUNCTIONS
! ------------------------

    PACK := (
        EXPONENT,CHECK NEXT
        TEMP<0>_MAN1<0>;
        (IF MAN1<0> EQL 0 => TEMP<1:8>_EXP1-128);
        (IF MAN1<0> EQL 1 => TEMP<1:8>_127-EXP1);
        TEMP<9:35>_MAN1<1:27>;
        TEMP1<0>_0;
        TEMP1<1:35>_MAN1<28:F62>);



    UNPACK := (
        (IF TEMP<0> EQL 0 => EXP1_TEMP<1:8>+128);
        (IF TEMP<0> NEQ 0 => EXP1_127-TEMP<1:8>);
        MAN1<0>_TEMP<0>;
        MAN1<1:27>_TEMP<9:35>;
        MAN1<28:F62>_TEMP1<1:35>;
        MAN1<F63:F97>_0);



    SWAP := (
        EXP1_EXP2; EXP2_EXP1;
        MAN1_MAN2; MAN2_MAN1);



    NORMALIZE := (
        (IF PDP6 => (IF MAN1<0:27> EQL 0 => EXP1_0));
        (IF KA10 OR KI10 OR KL10 OR UNICORN =>
            (IF MAN1 EQL 0 => EXP1_0);
            (IF MAN1 NEQ 0 =>
NORMLOOP :=     (IF MAN1<0> EQL MAN1<1> =>
                    FLT,LEFT,SHIFT;
                    EXP1_EXP1-1 NEXT
                    NORMLOOP)))
        );



    ALIGN := (
        (IF EXP1 GTR EXP2 => SWAP) NEXT
        (IF (EXP2-EXP1) GTR 64 => EXP1_0; MAN1_0);
        (IF (EXP2-EXP1) LEQ 64 =>
ALOOP :=    (IF EXP1 NEQ EXP2 =>
                FLT,RIGHT,SHIFT;
                EXP1_EXP1+1 NEXT
```

```
                ALOOP))
        );



    ROUND := (
        (IF MAN1<0> EQL 0 => MAN1<0:F63>_MAN1<0:F63>+1);
        (IF MAN1<0> EQL 1 =>
            (IF PDP6 => MAN1<0:F63>_MAN1<0:F63>+1);
            (IF KA10 OR KI10 OR KL10 OR UNICORN =>
                MAN1<0:F63>_MAN1<0:F63>-1))
        );



    EXPONENT.CHECK := (
        (IF EXP1 GTR 127 =>
            EXP1_EXP1-256;
            OVERFLOW_1;
            FLT.OVERFLOW_1;
            TRAP.1_1);
        (IF EXP1 LSS -128 =>
            EXP1_EXP1+256;
            OVERFLOW_1;
            FLT.OVERFLOW_1;
            FLT.UNDERFLOW_1;
            TRAP.1_1)
        );



    FLT.RIGHT.SHIFT := (MAN1_MAN1 ^SR MAN1<0>);



    FLT.LEFT.SHIFT := (MAN1<1:F97>_MAN1<1:F97> ^SL0 1);
```

```
! EXTENDED FUNCTIONS
! ==================

    TRANSLATE := (
        VMA_E1+S<0:34> NEXT
        (DECODE S<35> =>
            TEMP<0:17>_M[VMA]<0:17>; TEMP<0:17>_M[VMA]<18:35>)
        NEXT
        (IF TEMP<0> EQL 1 => AC[A]<0:1>_'11);    ! SIGNIFICANCE
        (DECODE TEMP<1:2> =>
            NOP;          ! NOP
            RUN;          ! ABORT
            AC[A]<2>_0;   ! CLEAR M
            AC[A]<2>_1);          ! SET M
        S<0:20>_0; S<21:35>_TEMP<3:17>);



    UPDATE.PATTERN ADDRESS := (
        AC[A]<4:5>_AC[A]<4:5>+1;
        (IF AC[A]<4:5> EQL 3 => AC[A]<18:35>_AC[A]<18:35>+1));



    GET.PATTERN BYTE := (
        POS_(3-AC[A]<4:5>)*9;
        SIZE_9;
        BYTE.PTR<12:17>_0;
        BYTE.PTR<18:35>_AC[A]<18:35> NEXT
        LOAD.BYTE NEXT
        PATTERN_S<28:35>);



    SKIP.PATTERN := (
        UPDATE.PATTERN ADDRESS;
        PATTERN<3:8>_PATTERN<3:8>-1 NEXT
        (IF PATTERN<3:8> NEQ #77 => SKIP.PATTERN));



    GET.SRC.BYTE := (
        BYTE.PTR_AC[A+1] NEXT
        INCREMENT.BYTE.PTR NEXT
        AC[A+1]_BYTE.PTR NEXT
        LOAD.BYTE NEXT
        AC[A]<9:35>_AC[A]<9:35>-1);
```

```
    STORE,DST,BYTE := (
        BYTE,PTR_AC[A+4] NEXT
        INCREMENT,BYTE,PTR NEXT
        AC[A+4]_BYTE,PTR NEXT
        DEPOSIT,BYTE NEXT
        AC[A+3]<9:35>_AC[A+3]<9:35>-1);



    LOAD,MESSAGE,CHARACTER := (
        VMA_E1+PATTERN<3:8>+1 NEXT
        S_M[VMA] NEXT
        (IF S NEQ 0 =>
            D_S;
            BYTE,PTR_AC[A+4] NEXT
            INCREMENT,BYTE,PTR NEXT
            AC[A+4]_BYTE,PTR NEXT
            DEPOSIT,BYTE));



    SRC,SETUP := (
        BYTE,PTR_AC[A+1];
        (IF BYTE,PTR<12> EQL 1 => MUUO);
        TEMP1_1 NEXT
        ADJUST,BYTE,POINTER NEXT
        VALIDATE,BYTE,POINTER NEXT
        AC[A+1]_BYTE,PTR);



    DST,SETUP := (
        BYTE,PTR_AC[A+4] NEXT
        (IF BYTE,PTR<12> EQL 1 => MUUO);
        TEMP1_1 NEXT
        ADJUST,BYTE,POINTER NEXT
        VALIDATE,BYTE,POINTER NEXT
        AC[A+4]_BYTE,PTR);
```

```
! INSTRUCTION INTERPRETATION
! --------------------------

    RUN := (
        IR_M[PC];
        PC_PC+1 NEXT
        I,EXECUTION NEXT
        RUN);



! INSTRUCTION SET AND INSTRUCTION EXECUTION PROCESS
! ------------------------------------------------

    I,EXECUTION := (
        VMA_IR<13:35>; LAST_PC NEXT
        EA NEXT
        E_VMA<18:35> NEXT
        I,SET);
```

```
! INSTRUCTION SET
! ---------------

    I,SET := (



    (IF OPCODE EQL #000 => MUUO);          ! MUUO



    (IF (OPCODE GEQ #001) AND (OPCODE LEQ #037) =>          ! LUUO
        (IF PDP6 => MUUO);
        (IF KA10 OR KI10 OR KL10 OR UNICORN =>
            M[40]<0:8>_OPCODE;
            M[40]<9:17>_0;
            M[40]<18:35>_E;
            IR_M[41] NEXT
            I,EXECUTION));



    (IF (OPCODE GEQ #040) AND (OPCODE LEQ #077) => MUUO); !
    MUUO



    (IF OPCODE EQL #100 => MUUO);          ! UJEN



    (IF OPCODE EQL #101 => MUUO);          ! MUUO



    (IF OPCODE EQL #102 => MUUO);          ! MUUO



    (IF OPCODE EQL #103 => MUUO);          ! MUUO



    (IF OPCODE EQL #104 => MUUO);          ! JSYS
```

```
    (IF OPCODE EQL #105 =>          ! ADJSP
        (IF PDP6 OR KA10 OR KI10 => MUUO);
        (IF KL10 OR UNICORN =>
            TEMP<0:17>_AC[A]<0:17>+E;
            TEMP<18:35>_AC[A]<18:35>+E NEXT
            (IF AC[A]<0> EQL TEMP<0> => AC[A]_TEMP);
            (IF AC[A]<0> NEQ TEMP<0> => TRAP.2_1)));



    (IF OPCODE EQL #106 => MUUO);           ! MUUO



    (IF OPCODE EQL #107 => MUUO);           ! MUUO
```

```
! DOUBLE PRECISION FLOATING POINT
! -------------------------------

    (IF OPCODE EQL #110 =>          ! DFAD
        (IF PDP6 OR KA10 => MUUO);
        (IF KI10 OR KL10 OR UNICORN =>
            TEMP_M[E];
            TEMP1_M[E+1] NEXT
            UNPACK NEXT
            EXP1_EXP1+1;
            FLT.RIGHT.SHIFT NEXT
            SWAP;
            TEMP_AC[A];
            TEMP1_AC[A+1] NEXT
            UNPACK NEXT
            EXP1_EXP1+1;
            FLT.RIGHT.SHIFT NEXT
            ALIGN NEXT
            MAN1_MAN1+MAN2 NEXT
            NORMALIZE NEXT
            ROUND NEXT
            PACK NEXT
            AC[A]_TEMP;
            AC[A+1]_TEMP1));



    (IF OPCODE EQL #111 =>          ! DFSB
        (IF PDP6 OR KA10 => MUUO);
        (IF KI10 OR KL10 OR UNICORN =>
            TEMP_M[E];
            TEMP1_M[E+1] NEXT
            UNPACK NEXT
            EXP1_EXP1+1;
            FLT.RIGHT.SHIFT NEXT
            SWAP;
            TEMP_AC[A];
            TEMP1_AC[A+1] NEXT
            UNPACK NEXT
            EXP1_EXP1+1;
            FLT.RIGHT.SHIFT NEXT
            ALIGN NEXT
            MAN1_MAN1-MAN2 NEXT
            NORMALIZE NEXT
            ROUND NEXT
            PACK NEXT
            AC[A]_TEMP;
            AC[A+1]_TEMP1));



    (IF OPCODE EQL #112 =>          ! DFMP
```

```
        (IF PDP6 OR KA10 => MUUO);
        (IF KI10 OR KL10 OR UNICORN =>
            TEMP_M[E];
            TEMP1_M[E+1] NEXT
            UNPACK NEXT
            SWAP;
            TEMP_AC[A];
            TEMP1_AC[A+1] NEXT
            UNPACK NEXT
            EXP1_EXP1+EXP2;
            MAN1_MAN1*MAN2 NEXT
            ROUND NEXT
            (IF KL10 OR UNICORN => NORMALIZE) NEXT
            PACK NEXT
            AC[A]_TEMP;
            AC[A+1]_TEMP1));



    (IF OPCODE EQL #113 =>      ! DFDV
        (IF PDP6 OR KA10 => MUUO);
        (IF KI10 OR KL10 OR UNICORN =>
            TEMP_M[E];
            TEMP1_M[E+1] NEXT
            UNPACK NEXT
            SWAP;
            TEMP_AC[A];
            TEMP1_AC[A+1] NEXT
            UNPACK NEXT
            EXP1_EXP1-EXP2;
            MAN1_MAN1/MAN2 NEXT
            (IF KL10 OR UNICORN => ROUND NEXT NORMALIZE) NEXT
            PACK NEXT
            AC[A]_TEMP;
            AC[A+1]_TEMP1));
```

```
! DOUBLE PRECISION INTEGER
! ------------------------

    (IF OPCODE EQL #114 =>          ! DADD
        (IF PDP6 OR KA10 OR KI10 => MUUO);
        (IF KL10 OR UNICORN =>
            MAN1_AC[A]@AC[A+1]<1:35>+M[E]@M[E+1]<1:35> NEXT
            AC[A]_MAN1<0:35>;
            AC[A+1]<0>_MAN1<0>;
            AC[A+1]<1:35>_MAN1<F36:F71>;
            SET,FLAGS));            ! OVERFLOW,CRY,0,CRY,1,TRAP,1



    (IF OPCODE EQL #115 =>          ! DSUB
        (IF PDP6 OR KA10 OR KI10 => MUUO);
        (IF KL10 OR UNICORN =>
            MAN1_AC[A]@AC[A+1]<1:35>-M[E]@M[E+1]<1:35> NEXT
            AC[A]_MAN1<0:35>;
            AC[A+1]<0>_MAN1<0>;
            AC[A+1]<1:35>_MAN1<F36:F71>;
            SET,FLAGS));            ! OVERFLOW,CRY,0,CRY,1,TRAP,1



    (IF OPCODE EQL #116 =>          ! DMUL
        (IF PDP6 OR KA10 OR KI10 => MUUO);
        (IF KL10 OR UNICORN =>
            MAN1_AC[A]@AC[A+1]<1:35>*M[E]@M[E+1]<1:35> NEXT
            AC[A]_MAN1<0:35>;
            AC[A+1]_MAN1<F36:F71>;
            AC[A+2]_MAN1<F72:F107>;
            AC[A+3]_MAN1<F108:F143>;
            AC[A+1]<0>_MAN1<0>;
            AC[A+2]<0>_MAN1<0>;
            AC[A+3]<0>_MAN1<0>;
            SET,FLAGS));            ! OVERFLOW,CRY,0,CRY,1,TRAP,1



    (IF OPCODE EQL #117 =>          ! DDIV
        (IF PDP6 OR KA10 OR KI10 => MUUO);
        (IF KL10 OR UNICORN =>
            MAN1_AC[A]@AC[A+1]<1:35>/M[E]@M[E+1]<1:35> NEXT
            AC[A]_MAN1<0:35>;
            AC[A+1]<1:35>_MAN1<F36:F71>;
            AC[A+1]<0>_MAN1<0>;
            SET,FLAGS));            !
            NO,DIVIDE,OVERFLOW,CRY,0,CRY,1,TRAP,1
```

```
(IF OPCODE EQL #120 =>          ! DMOVE
    (IF PDP6 OR KA10 => MUUO);
    (IF KI10 => AC[A]_M[E] NEXT AC[A+1]_M[E+1]);
    (IF KL10 OR UNICORN => AC[A]_M[E]; AC[A+1]_M[E+1]));



(IF OPCODE EQL #121 =>          ! DMOVN
    (IF PDP6 OR KA10 => MUUO);
    (IF KI10 =>
        AC[A]_NOT M[E];
        AC[A+1]_-M[E+1] NEXT
        AC[A+1]<0>_0 NEXT
        (IF AC[A+1] EQL 0 => AC[A]_AC[A]+1);
        SET,FLAGS); ! OVERFLOW,CRY,0,CRY,1,TRAP,1
    (IF KL10 OR UNICORN =>
        TEMP_M[E];
        TEMP1_M[E+1] NEXT
        TEMP1<0>_0 NEXT
        (IF TEMP1 EQL 0 => TEMP_TEMP+1) NEXT
        M[A]_TEMP;
        M[A+1]_TEMP1;
        SET,FLAGS));            ! OVERFLOW,CRY,0,CRY,1,TRAP,1



(IF OPCODE EQL #122 =>          ! FIX
    (IF PDP6 OR KA10 => MUUO);
    (IF KI10 OR KL10 OR UNICORN =>
        TEMP_M[E] NEXT
        UNPACK NEXT
        (IF EXP1 GTR 35 => OVERFLOW_1; TRAP,1_1);
        (IF EXP1 LEQ 35 =>
            TEMP<0>_MAN1<0>;
            TEMP<1>_MAN1<0>;
            TEMP<2>_MAN1<0>;
            TEMP<3>_MAN1<0>;
            TEMP<4>_MAN1<0>;
            TEMP<5>_MAN1<0>;
            TEMP<6>_MAN1<0>;
            TEMP<7>_MAN1<0>;
            TEMP<8>_MAN1<0>;
            TEMP<9:35>_MAN1<1:27>;
            SC_EXP1-27;
            (IF SC GTR 0 => TEMP_TEMP ^SL0 SC);
            (IF SC LSS 0 =>
                (DECODE TEMP<0> =>
                    TEMP_TEMP ^SR0 SC; TEMP_TEMP ^SR1 SC));
        AC[A]_TEMP)));
```

```
(IF OPCODE EQL #123 =>          ! EXTEND
    (IF PDP6 OR KA10 OR KI10 => MUUO);
    (IF KL10 OR UNICORN =>
        TEMPF_A;
        E0_E NEXT
        IR_M[E0];
        VMA_IR<13:35> NEXT
        EA NEXT
        E1_VMA<18:35>;
        (IF A EQL 0 => A_TEMPF NEXT EXT.INSTRUCTION.SET);
        (IF A NEQ 0 => MUUO)));



(IF OPCODE EQL #124 =>          ! DMOVEM
    (IF PDP6 OR KA10 => MUUO);
    (IF KI10 => M[E]_AC[A] NEXT M[E+1]_AC[A+1]; FPD_0);
    (IF KL10 OR UNICORN => M[E]_AC[A]; M[E+1]_AC[A+1]));



(IF OPCODE EQL #125 =>          ! DMOVNM
    (IF PDP6 OR KA10 => MUUO);
    (IF KI10 =>
        M[E]_AC[A] NEXT
        M[E+1]_-AC[A+1] NEXT
        M[E+1]<0>_0 NEXT
        (IF M[E+1] EQL 0 => M[E]_M[E]+1);
        SET.FLAGS;  ! OVERFLOW,CRY.0,CRY.1,TRAP.1
        FPD_0);
    (IF KL10 OR UNICORN =>
        TEMP_AC[A];
        TEMP1_AC[A+1] NEXT
        TEMP1<0>_0 NEXT
        (IF TEMP1 EQL 0 => TEMP_TEMP+1) NEXT
        M[E]_TEMP;
        M[E+1]_TEMP1;
        SET.FLAGS));            ! OVERFLOW,CRY.0,CRY.1,TRAP.1
```

```
    (IF OPCODE EQL #126 =>      ! FIXR
        (IF PDP6 OR KA10 => MUUO);
        (IF KI10 OR KL10 OR UNICORN =>
            TEMP_M[E];
            TEMP1_0 NEXT
            UNPACK NEXT
            (IF EXP1 GTR 35 => OVERFLOW_1; TRAP.1_1);
            (IF EXP1 LEQ 35 =>
                TEMP<0>_MAN1<0>;
                TEMP<1>_MAN1<0>;
                TEMP<2>_MAN1<0>;
                TEMP<3>_MAN1<0>;
                TEMP<4>_MAN1<0>;
                TEMP<5>_MAN1<0>;
                TEMP<6>_MAN1<0>;
                TEMP<7>_MAN1<0>;
                TEMP<8>_MAN1<0>;
                TEMP<9:35>_MAN1<1:27>;
                TEMP1_0;
                SC_EXP1-27;
                (IF SC GTR 0 => TEMP_TEMP ^SL0 SC);
                (IF SC LSS 0 =>
FIXR :=             (IF SC NEQ 0 =>
                        (IF TEMP<0> NEQ TEMP<35> => TEMP1_1);
                        TEMP_TEMP ^SR TEMP<0>;
                        SC_SC+1 NEXT
                        FIXR));
            AC[A]_TEMP+TEMP1)));



    (IF OPCODE EQL #127 =>      ! FLTR
        (IF PDP6 OR KA10 => MUUO);
        (IF KI10 OR KL10 OR UNICORN =>
            MAN1<0:35>_M[E];
            MAN1<F36:F97>_0;
            EXP1_27 NEXT
            NORMALIZE;
            (IF MAN1<0> EQL 0 => MAN1<0:28>_MAN1<0:28>+1); !
            ROUND
            (IF MAN1<0> EQL 1 => MAN1<0:28>_MAN1<0:28>-1) NEXT
            PACK NEXT
            AC[A]_TEMP));



    (IF OPCODE EQL #130 =>      ! UFA
        (IF PDP6 OR UNICORN => MUUO);
        (IF KA10 OR KI10 OR KL10 =>
            TEMP_M[E];
            TEMP1_0 NEXT
            UNPACK NEXT
```

```
            EXP1_EXP1+1;
            FLT,RIGHT,SHIFT NEXT
            SWAP;
            TEMP_AC[A];
            TEMP1_0 NEXT
            UNPACK NEXT
            EXP1_EXP1+1;
            FLT,RIGHT,SHIFT NEXT
            ALIGN NEXT
            MAN1_MAN1+MAN2 NEXT
            (IF MAN1<0> EQL MAN1<1> =>
                EXP1_EXP1-1;
                FLT,LEFT,SHIFT) NEXT
            PACK NEXT
            AC[A+1]_TEMP));



    (IF OPCODE EQL #131 =>          ! DFN
        (IF PDP6 OR UNICORN => MUUO);
        (IF KA10 OR KI10 OR KL10 =>
            TEMP_AC[A];
            TEMP1<1:27>_M[E]<9:35>;
            TEMP1<28:35>_0 NEXT
            UNPACK NEXT
            MAN1_-MAN1 NEXT
            PACK NEXT
            AC[A]_TEMP;
            M[E]<9:35>_TEMP1<1:27>));



    (IF OPCODE EQL #132 =>          ! FSC
        (IF PDP6 => MUUO);
        (IF KA10 OR KI10 OR KL10 OR UNICORN =>
            (IF AC[A]<9:35> EQL 0 => AC[A]_0);
            (IF AC[A]<9:35> NEQ 0 =>
                TEMP_AC[A];
                TEMP1_0 NEXT
                UNPACK NEXT
                EXP1_EXP1+E NEXT
                (IF KA10 OR KI10 OR KL10 OR UNICORN =>
                NORMALIZE) NEXT
                PACK NEXT
                AC[A]_TEMP)));
```

```
! BYTE INSTRUCTIONS
! ----------------

    (IF OPCODE EQL #133 =>        ! IBP
        (IF PDP6 => MUUO);
        (IF KA10 OR KI10 OR ((KL10 OR UNICORN) AND (A EQL 0))
        =>
            BYTE,PTR_M[E] NEXT
            INCREMENT,BYTE,POINTER NEXT
            M[E]_BYTE,PTR);
        (IF (KL10 OR UNICORN) AND (A NEQ 0) =>
            BYTE,PTR_M[E];
            TEMP1_AC[A] NEXT
            ADJUST,BYTE,POINTER NEXT
            AC[A]_BYTE,PTR));



    (IF OPCODE EQL #134 =>        ! ILDB
        (IF PDP6 => MUUO);
        (IF KA10 OR KI10 OR KL10 OR UNICORN =>
            BYTE,PTR_M[E] NEXT
            INCREMENT,BYTE,POINTER NEXT
            M[E]_BYTE,PTR NEXT
            LOAD,BYTE NEXT
            AC[A]_S;
            FPD_0));



    (IF OPCODE EQL #135 =>        ! LDB
        (IF PDP6 => MUUO);
        (IF KA10 OR KI10 OR KL10 OR UNICORN =>
            BYTE,PTR_M[E] NEXT
            LOAD,BYTE NEXT
            AC[A]_S;
            FPD_0));



    (IF OPCODE EQL #136 =>        ! IDPB
        (IF PDP6 => MUUO);
        (IF KA10 OR KI10 OR KL10 OR UNICORN =>
            BYTE,PTR_M[E] NEXT
            INCREMENT,BYTE,POINTER NEXT
            M[E]_BYTE,PTR NEXT
            D_AC[A] NEXT
            DEPOSIT,BYTE;
            FPD_0));
```

```
    (IF OPCODE EQL #137 =>          ! DPB
        (IF PDP6 => MUUO);
        (IF KA10 OR KI10 OR KL10 OR UNICORN =>
            BYTE,PTR_M[E] NEXT
            D_AC[A] NEXT
            DEPOSIT,BYTE;
            FPD_0));
```

```
! SINGLE PRECISION FLOATING POINT
! -------------------------------

    (IF (OPCODE GEQ #140) AND (OPCODE LEQ #177) =>
        (DECODE OPCODE<6;8> =>
            TEMP_M[E];
            ((IF PDP6 OR KA10 OR KI10 OR KL10 => TEMP_M[E]);
                (IF UNICORN => MUUO));
            TEMP_M[E];
            TEMP_M[E];
            TEMP_M[E];
            ((IF PDP6 => TEMP_M[E]);
                (IF KA10 OR KI10 OR KL10 OR UNICORN =>
                    TEMP<0:17>_E; TEMP<18:35>_0));
            TEMP_M[E];
            TEMP_M[E]);
        TEMP1_0 NEXT
        UNPACK NEXT
        SWAP NEXT
        TEMP_AC[A];
        TEMP1_0 NEXT
        UNPACK NEXT
        (DECODE OPCODE<4:5> =>
            (ALIGN NEXT ! FAD-
                MAN1_MAN1+MAN2 NEXT
                (IF MAN1<0> NEQ MAN1<8> => FLT,RIGHT,SHIFT)
                NEXT
                PACK);
            (ALIGN NEXT ! FSB-
                MAN1_MAN1-MAN2 NEXT
                (IF MAN1<0> NEQ MAN1<8> => FLT,RIGHT,SHIFT)
                NEXT
                PACK);
            (MAN1_MAN1*MAN2;     ! FMP-
                EXP1_EXP1+EXP2 NEXT
                PACK);
            (MAN1_MAN1/MAN2;     ! FDV-
                EXP1_EXP1-EXP2 NEXT
                PACK)) NEXT
        (DECODE OPCODE<6:8> =>
            (NORMALIZE NEXT      ! -
                PACK NEXT
                AC[A]_TEMP);
            LONG;          ! -L
            (NORMALIZE NEXT      ! -M
                PACK;
                M[E]_TEMP);
            (NORMALIZE NEXT      ! -B
                PACK NEXT
                AC[A]_TEMP;
                M[E]_TEMP);
            (NORMALIZE NEXT      ! -R
                ROUND NEXT
                PACK NEXT
```

```
        AC[A]_TEMP);
    (NORMALIZE NEXT      ! -RI
        ROUND NEXT
        (IF PDP6 => LONG);
        (IF KA10 OR KI10 OR KL10 => PACK NEXT
        AC[A]_TEMP));
    (NORMALIZE NEXT      ! -RM
        ROUND NEXT
        PACK NEXT
        M[E]_TEMP);
    (NORMALIZE; ! -RB
        ROUND NEXT
        PACK NEXT
        AC[A]_TEMP;
        M[E]_TEMP)));
```

```
! MOVE GROUP
! ----------

    (IF (OPCODE GEQ #200) AND (OPCODE LEQ #217) =>
        (DECODE OPCODE<7:8> =>
            S_M[E];         ! -
            S_E;            ! -I
            S_AC[A];        ! -M
            S_M[E]) NEXT            ! -S
        (DECODE OPCODE<5:6> =>
            D_S;            ! MOVE-
            (D<0:17>_S<18:35>; D<18:35>_S<0:17>);           ! MOVS-
            (D_-S NEXT      ! MOVN-
               SET,FLAGS);          ! OVERFLOW,CRY.0,CRY.1,TRAP.1
            (DECODE S<0> => D_S; D_-S) NEXT         ! MOVM-
            SET,FLAGS); ! OVERFLOW,CRY.1,TRAP.1
        (DECODE OPCODE<7:8> =>
            AC[A]_D;        ! -
            AC[A]_D;        ! -I
            M[E]_D;         ! -M
            (M[E]_D; (IF A NEQ 0 => AC[A]_S))));            ! -S
```

```
! MULTIPLY AND DIVIDE
! -------------------

    (IF (OPCODE GEQ #220) AND (OPCODE LEQ #223) =>          ! IMUL
        MAN1<0:17>_0;
        MAN1<18:35>_AC[A] NEXT
        (IF KA10 => (IF AC[A] EQL #400000000000 =>
        MAN1<18:35>_#777777777));
        MAN2<0:17>_M[E];
        MAN2<18:35>_0 NEXT
        (IF (MAN1<18:35> EQL #400000000000)
             AND (MAN2<0:17> EQL #400000000000) =>
           D_MIN,2,TO,70;
            OVERFLOW_1;
            TRAP,1_1);
        (IF (MAN1<18:35> NEQ #400000000000) OR
        (MAN2<0:17> NEQ #400000000000) =>
            MAN1_MAN1*MAN2 NEXT
            TEMP<0>_MAN1<0>;
            TEMP<1:35>_MAN1<F36:F71>;
            (DECODE OPCODE<7:8> =>
                AC[A]_TEMP;         ! -
                AC[A]_TEMP;         ! -I
                M[E]_TEMP;          ! -M
                (AC[A]_TEMP; M[E]_TEMP))));      ! -B



    (IF (OPCODE GEQ #224) AND (OPCODE LEQ #227) =>          ! MUL
        MAN1<0:17>_0;
        MAN1<18:35>_AC[A] NEXT
        (IF KA10 => (IF AC[A] EQL #400000000000 =>
        MAN1<18:35>_#777777777));
        MAN2<0:17>_M[E];
        MAN2<18:35>_0 NEXT
        (IF (MAN1<18:35> EQL #400000000000)
            AND (MAN2<0:17> EQL #400000000000) =>
            D_MIN,2,TO,70;
            OVERFLOW_1;
            TRAP,1_1);
        (IF (MAN1<18:35> NEQ #400000000000) OR
        (MAN2<0:17> NEQ #400000000000) =>
            MAN1_MAN1*MAN2 NEXT
            TEMP<0>_MAN1<0>;
            TEMP<1:35>_MAN1<1:35>;
            (DECODE OPCODE<7:8> =>
                AC[A]_TEMP;         ! -
                AC[A]_TEMP;         ! -I
                M[E]_TEMP;          ! -M
                (AC[A]_TEMP; M[E]_TEMP))));      ! -B
```

```
(IF (OPCODE GEQ #230) AND (OPCODE LEQ #233) =>          ! IDIV
    MAN1<0:17>_0;
    MAN1<18:35>_AC[A] NEXT
    (IF KA10 => (IF AC[A] EQL #400000000000 =>
    MAN1<18:35>_#777777777));
    MAN2<0:17>_M[E];
    MAN2<18:35>_0 NEXT
    (IF (MAN1<18:35> EQL #400000000000)
        AND (MAN2<0:17> EQL #400000000000) =>
        D_MIN,2,TO,70;
        OVERFLOW_1;
        TRAP,1_1);
    (IF (MAN1<18:35> NEQ #400000000000) OR (MAN2<0:17> NEQ
        #400000000000) =>
        MAN1_MAN1/MAN2 NEXT
        TEMP<0>_MAN1<0>;
        TEMP<1:35>_MAN1<F36:F71>;
        (DECODE OPCODE<7:8> =>
            AC[A]_TEMP;         ! -
            AC[A]_TEMP;         ! -I
            M[E]_TEMP;          ! -M
            (AC[A]_TEMP; M[E]_TEMP))));      ! -B


(IF (OPCODE GEQ #234) AND (OPCODE LEQ #237) =>          ! DIV
    MAN1<0:17>_0;
    MAN1<18:35>_AC[A] NEXT
    (IF KA10 => (IF AC[A] EQL #400000000000 =>
    MAN1<18:35>_#777777777));
    MAN2<0:17>_M[E];
    MAN2<18:35>_0 NEXT
    (IF (MAN1<18:35> EQL #400000000000)
        AND (MAN2<0:17> EQL #400000000000) =>
        D_MIN,2,TO,70;
        OVERFLOW_1;
        TRAP,1_1);
    (IF (MAN1<18:35> NEQ #400000000000) OR (MAN2<0:17> NEQ
        #400000000000) =>
        MAN1_MAN1/MAN2 NEXT
        TEMP<0>_MAN1<0>;
        TEMP<1:35>_MAN1<F36:F71>;
        (DECODE OPCODE<7:8> =>
            AC[A]_TEMP;         ! -
            AC[A]_TEMP;         ! -I
            M[E]_TEMP;          ! -M
            (AC[A]_TEMP; M[E]_TEMP))));      ! -B
```

```
! SHIFTS, ROTATES AND COUNTS
! --------------------------

    (IF OPCODE EQL #240 =>          ! ASH
        SHIFT,COUNT NEXT
        (IF SC GTR 0 =>
ASHL :=     (IF SC NEQ 0 =>
                (IF AC[A]<0> NEQ AC[A]<1> => OVERFLOW_1;
                TRAP,1_1);
                AC[A]<1:35>_AC[A]<1:35> ^SL0 1;
                SC_SC-1 NEXT
                ASHL)));
        (IF SC LSS 0 =>
ASHR :=     (IF SC NEQ 0 =>
                AC[A]_AC[A] ^SR AC[A]<0>;
                SC_SC+1 NEXT
                ASHR)));



    (IF OPCODE EQL #241 =>          ! ROT
        SHIFT,COUNT NEXT
        (IF SC GTR 0 => AC[A]_AC[A] ^RL SC);
        (IF SC LSS 0 => AC[A]_AC[A] ^RR SC));



    (IF OPCODE EQL #242 =>          ! LSH
        SHIFT,COUNT NEXT
        (IF SC GTR 0 => AC[A]_AC[A] ^SL0 SC);
        (IF SC LSS 0 => AC[A]_AC[A] ^SR0 SC));



    (IF OPCODE EQL #243 =>          ! JFFO
        (IF PDP6 => MUUO);
        (IF KA10 OR KI10 OR KL10 OR UNICORN =>
            TEMP_AC[A];
            AC[A+1]_0 NEXT
JFFOLOOP :=  (IF TEMP<0> EQL 0 =>
                AC[A+1]_AC[A+1]+1;
                TEMP_TEMP ^SL 0 NEXT
                JFFOLOOP);
            (IF AC[A] NEQ 0 => PC_E)));



    (IF OPCODE EQL #244 =>          ! ASHC
        SHIFT,COUNT NEXT
        (IF SC GTR 0 =>
```

```
ASHCL :=    (IF SC NEQ 0 =>
                (IF TEMP<0> NEQ TEMP<1> => OVERFLOW_1;
                TRAP.1_1);
                AC[A]<1:35>_AC[A]<1:35> ^SL AC[A+1]<1>;
                AC[A+1]<0>_AC[A]<0>;
                AC[A+1]<1:35>_AC[A+1]<1:35> ^SL 0;
                SC_SC-1 NEXT
                ASHCL));
        (IF SC LSS 0 =>
ASHCR :=    (IF SC NEQ 0 =>
                AC[A]<1:35>_AC[A]<1:35> ^SR AC[A]<0>;
                AC[A+1]<0>_AC[A]<0>;
                AC[A+1]<1:35>_AC[A+1]<1:35> ^SR AC[A]<35>;
                SC_SC+1 NEXT
                ASHCR)));



    (IF OPCODE EQL #245 =>      ! ROTC
        SHIFT.COUNT;
        TEMP_AC[A]; TEMP1_AC[A+1] NEXT
        (IF SC GTR 0 =>
            AC[A]_((TEMP@TEMP1) ^RL SC)<71:36>;
            AC[A+1]_((TEMP@TEMP1) ^RL SC)<35:0>);
        (IF SC LSS 0 =>
            AC[A]_((TEMP@TEMP1) ^RR SC)<71:36>;
            AC[A+1]_((TEMP@TEMP1) ^RR SC)<35:0>));



    (IF OPCODE EQL #246 =>      ! LSHC
        SHIFT.COUNT;
        TEMP_AC[A]; TEMP1_AC[A+1] NEXT
        (IF SC GTR 0 =>
            AC[A]_((TEMP@TEMP1) ^SL0 SC)<71:36>;
            AC[A+1]_((TEMP@TEMP1) ^SL0 SC)<35:0>);
        (IF SC LSS 0 =>
            AC[A]_((TEMP@TEMP1) ^SR0 SC)<71:36>;
            AC[A+1]_((TEMP@TEMP1) ^SR0 SC)<35:0>));



    (IF OPCODE EQL #247 => NOP);        ! UNASSIGNED
```

```
! MISC
! ----

    (IF OPCODE EQL #250 => AC[A]_M[E]; M[E]_AC[A]);      ! EXCH



    (IF OPCODE EQL #251 =>       ! BLT
        SA_AC[A]<0:17>; DA_AC[A]<18:35> NEXT
BLOOP:= (M[DA]_M[SA];
        SA_SA+1; DA_DA+1 NEXT
        (IF DA LSS (E+1) => BLTLOOP)) NEXT
        FPD_0); ! AC[A] UNCHANGED IF NO INTERRUPTS



    (IF OPCODE EQL #252 =>       ! AOBJP
        (IF PDP6 OR KA10 => AC[A]_AC[A]+#1000001);
        (IF KI10 OR KL10 OR UNICORN =>
            AC[A]<0:17>_AC[A]<0:17>+1;
            AC[A]<18:35>_AC[A]<18:35>+1) NEXT
        (IF AC[A]<0> EQL 0 => PC_E));



    (IF OPCODE EQL #253 =>       ! AOBJN
        (IF PDP6 OR KA10 => AC[A]_AC[A]+#1000001);
        (IF KI10 OR KL10 OR UNICORN =>
            AC[A]<0:17>_AC[A]<0:17>+1;
            AC[A]<18:35>_AC[A]<18:35>+1) NEXT
        (IF AC[A]<0> EQL 1 => PC_E));



    (IF OPCODE EQL #254 =>       ! JRST
        (IF PDP6 OR KA10 OR KI10 =>
            (IF F<9> EQL 1 => MUUO);
            (IF F<10> EQL 1 => MUUO);
            (IF F<11> EQL 1 => JRSTF);
            (IF F<12> EQL 1 => PORTAL));
        (IF KL10 => (DECODE F =>
            PC_E;
            PORTAL;
            JRSTF;
            MUUO;
            MUUO;
            MUUO;
            MUUO;
            MUUO;
            MUUO;
```

```
            MUUO;
            MUUO;
            MUUO;
            MUUO;
            MUUO;
            MUUO;
            MUUO)));


    PORTAL := (
        (IF PDP6 OR KA10 OR UNICORN => USER_1);
        (IF KI10 OR KL10 => (IF P EQL 0 => PUBLIC_0)));

    JRSTF := (
        PC.WORD<0:4>_M[LAST]<0:4>;
        (IF M[LAST]<5> EQL 1 => USER_1);
        (DECODE M[LAST]<6> =>
            USER.IO_0; (IF USER EQL 0 => USER.IO_1));
        PC.WORD<7:12>_M[LAST]<7:12>);



    (IF OPCODE EQL #255 =>          ! JFCL
        TEMPF_F AND PC.WORD<0:3>;
        PC.WORD<0:3>_(NOT F) AND PC.WORD<0:3>;
        (IF TEMPF NEQ 0 => PC_E));



    (IF OPCODE EQL #256 => IR_M[E] NEXT I.EXECUTION);    ! XCT



    (IF OPCODE EQL #257 =>          ! MAP
        (IF PDP6 OR KA10 OR KL10 OR UNICORN => MUUO);
        (IF KI10 =>
            AC[A]<0:17>_0;
            AC[A]<18>_PAGE.FAILURE;
            AC[A]<19>_P;
            AC[A]<20>_W;
            AC[A]<21>_S;
            AC[A]<22>_NO.MATCH;
            AC[A]<23:35>_PHYSICAL.PAGE.ADDRESS));
```

```
! PROCEDURE LINKAGE
! -----------------

    (IF OPCODE EQL #260 =>       ! PUSHJ
        (IF PDP6 OR KA10 => AC[A]_AC[A]+#1000001;
        (IF KI10 OR KL10 OR UNICORN =>
            AC[A]<0:17>_AC[A]<0:17>+1;
            AC[A]<18:35>_AC[A]<18:35>+1);
        FPD_0; AFI_0; TRAP.1_0; TRAP.2_0 NEXT
        (IF AC[A]<0:17> EQL 0 => TRAP.2_1);
        VMA_AC[A]<18:35> NEXT
        M[VMA]_PC,WORD;
        PC_E);



    (IF OPCODE EQL #261 =>       ! PUSH
        (IF PDP6 OR KA10 => AC[A]_AC[A]+#1000001;
        (IF KI10 OR KL10 OR UNICORN =>
            AC[A]<0:17>_AC[A]<0:17>+1;
            AC[A]<18:35>_AC[A]<18:35>+1) NEXT
        (IF AC[A]<0:17> EQL 0 => TRAP.2_1);
        VMA_AC[A]<18:35> NEXT
        M[VMA]_M[E]);



    (IF OPCODE EQL #262 =>       ! POP
        SP_AC[A] NEXT
        VMA_SP<18:35> NEXT
        TEMP_M[SP] NEXT
        (IF PDP6 OR KA10 => SP_SP-#1000001;
        (IF KI10 OR KL10 OR UNICORN =>
            SP<0:17>_SP<0:17>-1;
            SP<18:35>_SP<18:35>-1);
        (IF PDP6 OR KL10 OR UNICORN => M[E]_TEMP NEXT
        AC[A]_SP);
        (IF KA10 OR KI10 => AC[A]_SP NEXT M[E]_TEMP);
        (IF SP EQL -1 => TRAP.2_1));



    (IF OPCODE EQL #263 =>       ! POPJ
        (IF PDP6 OR KA10 => AC[A]_AC[A]-#1000001;
        (IF KI10 OR KL10 OR UNICORN =>
            AC[A]<0:17>_AC[A]<0:17>-1;
            AC[A]<18:35>_AC[A]<18:35>-1);
        FPD_0; AFI_0; TRAP.1_0; TRAP.2_0 NEXT
        (IF AC[A]<0:17> EQL -1 => TRAP.2_1);
        VMA_AC[A]<18:35>-1 NEXT
        PC_M[VMA]);
```

```
    (IF OPCODE EQL #264 =>          ! JSR
        FPD_0; AFI_0; TRAP.1_0; TRAP.2_0 NEXT
        M[E]_PC,WORD;
        PC_E+1);


    (IF OPCODE EQL #265 =>          ! JSP
        FPD_0; AFI_0; TRAP.1_0; TRAP.2_0 NEXT
        AC[A]_PC,WORD;
        PC_E);


    (IF OPCODE EQL #266 =>          ! JSA
        M[E]_AC[A];
        AC[A]<0:17>_E; AC[A]<18:35>_PC;
        PC_E+1);


    (IF OPCODE EQL #267 =>          ! JRA
        VMA_AC[A]<0:17> NEXT
        AC[A]_M[VMA];
        PC_E);
```

```
! SINGLE PRECISION INTEGER ARITHMETIC
! ----------------------------------

    (IF (OPCODE GEQ #270) AND (OPCODE LEQ #277) =>
        (DECODE OPCODE<7:8> =>
            S_M[E];         ! -
            S_E;            ! -I
            S_M[E];         ! -M
            S_M[E]) NEXT            ! -B
        (DECODE OPCODE<6> => D_AC[A]+S; D_AC[A]-S) NEXT ! ADD,
        SUB
        SET,FLAGS;          ! OVERFLOW,CRY,0,CRY,1,TRAP,1
        (DECODE OPCODE<7:8> =>
            AC[A]_D;        ! -
            AC[A]_D;        ! -I
            M[E]_D;         ! -M
            (AC[A]_D; M[E]_D)));            ! -B
```

```
! ARITHMETIC TESTING
! ------------------

    (IF (OPCODE GEQ #300) AND (OPCODE LEQ #377) =>
        (DECODE OPCODE<3:5> =>
            (TEMP_AC[A]=E NEXT SKIP,TEST);          ! CAI=
            (TEMP_AC[A]=M[E] NEXT SKIP,TEST);       ! CAM=
            (TEMP_AC[A] NEXT JUMP,TEST);            ! JUMP=
            (TEMP_M[E]; ! SKIP=
                (IF A NEQ 0 => AC[A]_M[E]) NEXT SKIP,TEST);
            (AC[A]_AC[A]+1 NEXT ! AOJ=
                SET,FLAGS;          ! OVERFLOW,CRY,0,CRY,1,TRAP,1
                TEMP_AC[A] NEXT JUMP,TEST);
            (M[E]_M[E]+1 NEXT   ! AOS=
                (IF A NEQ 0 => AC[A]_M[E]) NEXT
                SET,FLAGS;          ! OVERFLOW,CRY,0,CRY,1,TRAP,1
                TEMP_M[E] NEXT SKIP,TEST);
            (AC[A]_AC[A]-1 NEXT ! SOJ=
                SET,FLAGS;          ! OVERFLOW,CRY,0,CRY,1,TRAP,1
                TEMP_AC[A] NEXT JUMP,TEST);
            (M[E]_M[E]-1 NEXT   ! SOS=
                (IF A NEQ 0 => AC[A]_M[E]) NEXT
                SET,FLAGS;          ! OVERFLOW,CRY,0,CRY,1,TRAP,1
                TEMP_M[E] NEXT SKIP,TEST)));



    SKIP,TEST := (
        (DECODE OPCODE<6:8> =>
            NOP;            ! -
            (IF TEMP LSS 0 => PC_PC+1); ! -L
            (IF TEMP EQL 0 => PC_PC+1); ! -E
            (IF TEMP LEQ 0 => PC_PC+1); ! -LE
            PC_PC+1;        ! -A
            (IF TEMP GEQ 0 => PC_PC+1); ! -GE
            (IF TEMP NEQ 0 => PC_PC+1); ! -N
            (IF TEMP GTR 0 => PC_PC+1)));           ! -G



    JUMP,TEST := (
        (DECODE OPCODE<6:8> =>
            NOP;            ! -
            (IF TEMP LSS 0 => PC_E);    ! -L
            (IF TEMP EQL 0 => PC_E);    ! -E
            (IF TEMP LEQ 0 => PC_E);    ! -LE
            PC_E;           ! -A
            (IF TEMP GEQ 0 => PC_E);    ! -GE
            (IF TEMP NEQ 0 => PC_E);    ! -N
            (IF TEMP GTR 0 => PC_E)));  ! -G
```

```
! LOGICAL
! -------

    (IF (OPCODE GEQ #400) AND (OPCODE LEQ #477) =>
        (DECODE OPCODE<7;8> =>
            D_M[E];          ! -
            D_E;             ! -I
            D_M[E];          ! -M
            D_M[E]) NEXT             ! -B
        (DECODE OPCODE<3;6> =>
            D_0;             ! SETZ
            D_AC[A] AND D;           ! AND
            D_(NOT AC[A]) AND D;             ! ANDCA
            NOP;             ! SETM
            D_AC[A] AND (NOT D);             ! ANDCM
            D_AC[A];         ! SETA
            D_AC[A] XOR D;           ! XOR
            D_AC[A] OR D;            ! IOR
            D_NOT (AC[A] OR D); ! ANDCB
            D_AC[A] EQV D;           ! EQV
            D_NOT AC[A];             ! SETCA
            D_(NOT AC[A]) OR D; ! ORCA
            D_NOT D;         ! SETCM
            D_AC[A] OR (NOT D); ! ORCM
            D_NOT (AC[A] AND D);             ! ORCB
            D_#777777777777) NEXT            ! SETO
        (DECODE OPCODE<7;8> =>
            AC[A]_D;         ! -
            AC[A]_D;         ! -I
            M[E]_D;          ! -M
            (AC[A]_D; M[E]_D)));             ! -B
```

```
! HALF WORD
! ---------

    (IF (OPCODE GEQ #500) AND (OPCODE LEQ #577) =>
        (DECODE OPCODE<7:8> =>
            (S_M[E]; D_AC[A]);  ! -
            (S_E; D_AC[A]);     ! -I
            (S_AC[A]; D_M[E]);  ! -M
            (S_M[E]; D_M[E])) NEXT      ! -S
        (DECODE OPCODE<3>@OPCODE<6> =>
            D<0:17>_S<0:17>;
            D<0:17>_S<18:35>;
            D<18:35>_S<18:35>;
            D<18:35>_S<0:17>) NEXT
        (DECODE OPCODE<4:5> =>
            (DECODE OPCODE<3> => TEMP_D<18:35>; TEMP_D<0:17>);
            ! -
            TEMP_0;     ! -Z
            TEMP_#777777;       ! -O
            (DECODE S<0> => TEMP_0; TEMP_#7777777)) NEXT ! -E
        (DECODE OPCODE<3> => D<18:35>_TEMP; D<0:17>_TEMP) NEXT
        (DECODE OPCODE<7:8> =>
            AC[A]_D;    ! -
            AC[A]_D;    ! -I
            M[E]_D;     ! -M
            (M[E]_D; (IF A NEQ 0 => AC[A]_D))));        ! -S
```

```
! TEST, MODIFY AND SKIP
! ---------------------

    (IF (OPCODE GEQ #600) AND (OPCODE LEQ #677) =>
        (DECODE OPCODE<5>@OPCODE<8> =>
            TEMP_E;      ! -R
            (TEMP<0:17>_E; TEMP<18:35>_0);      ! -L
            TEMP_M[E];   ! -D
            (TEMP<0:17>_M[E]<18:35>; TEMP<18:35>_M[E]<0:17>))
        NEXT     ! -S
        (DECODE OPCODE<6:7> =>
            NOP;         ! -
            (IF (AC[A] AND TEMP) EQL 0 => PC_PC+1);     ! -E
            PC_PC+1;     ! -A
            (IF (AC[A] AND TEMP) NEQ 0 => PC_PC+1)) NEXT ! -N
        (DECODE OPCODE<3:4> =>
            NOP;         ! -N
            AC[A]_AC[A] AND (NOT TEMP); ! -Z
            AC[A]_AC[A] XOR TEMP;       ! -C
            AC[A]_AC[A] OR TEMP));      ! -O
```

```
! INPUT/OUTPUT AND PROCESSOR DEPENDENT OPERATIONS
! ----------------------------------------------

    (IF (OPCODE GEQ #700) AND (OPCODE LEQ #777) => MUUO)

    );  ! END I.SET
```

```
! EXTENDED INSTRUCTION SET
! ------------------------

    EXT.I.SET := (



    (IF OPCODE EQL #000 => MUUO);        ! UNDEFINED



    (IF OPCODE EQL #004 =>       ! EDIT
        SRC.SETUP NEXT
ELOOP:= GET.PATTERN,BYTE NEXT
        UPDATE.PATTERN,ADDRESS NEXT
        (DECODE PATTERN<0:2> =>
            (IF PATTERN<3:8> EQL #00 => EEND);  ! TERMINATE
                (IF PATTERN<3:8> EQL #01 =>      ! SELECT
                    BYTE.PTR_AC[A+1] NEXT
                    INCREMENT,BYTE.PTR NEXT
                    AC[A+1]_BYTE.PTR NEXT
                    LOAD,BYTE NEXT
                    TRANSLATE NEXT
                    D_S;
                    BYTE.PTR_AC[A+4] NEXT
                    INCREMENT,BYTE.PTR NEXT
                    AC[A+4]_BYTE.PTR NEXT
                    DEPOSIT,BYTE);
                (IF PATTERN<3:8> EQL #02 =>      ! SIGNIFICANCE
                    START
                    (IF AC[A]<0> EQL 0 => AC[A]<0>_1);
                    (IF AC[A]<0> EQL 1 =>
                        AC[A+3]_AC[A+4];
                        PATTERN<3:8>_1 NEXT
                        LOAD,MSG,CHAR));
                (IF PATTERN<3:8> EQL #03 => AC[A]<0:2>_'000); !
                FIELD SEPARATOR
                (IF PATTERN<3:8> EQL #04 =>      ! EXCHANGE
                    MARKER
                    VMA_AC[A+3]<18:35> NEXT
                    AC[A+4]_M[VMA]; M[VMA]_AC[A+4]);
                (IF (PATTERN<3:8> GEQ #05) AND (PATTERN<3:8>
            LEQ #77) => NOP));
            LOAD,MSG,CHAR;       ! MESSAGE CHAR
            NOP;          ! 2XX
            NOP;          ! 3XX
            NOP;          ! 4XX
            (IF AC[A]<2> EQL 1 => SKIP,PATTERN);          ! SKIP
            IF M
            (IF AC[A]<1> EQL 1 => SKIP,PATTERN);          ! SKIP
            IF N
            SKIP,PATTERN) NEXT  ! SKIP PATTERN ALWAYS
        ELOOP NEXT
```

```
EEND := PC_PC+1);



    (IF (OPCODE EQL #001) OR (OPCODE EQL #002)  ! CMP-
        OR (OPCODE EQL #003) OR (OPCODE EQL #005)
        OR (OPCODE EQL #006) OR (OPCODE EQL #007) =>
        SRC.SETUP; DST.SETUP NEXT
CLOOP:= (IF (AC[A]<9:35> EQL 0) AND (AC[A+3]<9:35> EQL 0) =>
            COMPDONE) NEXT
        (IF AC[A]<9:35> EQL 0 =>
            POS_0;
            SIZE_AC[A+1]<6:11>;
            BYTE.PTR<12:17>_0;
            BYTE.PTR<18:35>_E0+1 NEXT
            LOAD.BYTE);
        (IF AC[A]<9:35> NEQ 0 => GET.SRC.BYTE) NEXT
        TEMP_S;
        (IF AC[A+3]<9:35> EQL 0 =>
            POS_0;
            SIZE_AC[A+1]<6:11>;
            BYTE.PTR<12:17>_0;
            BYTE.PTR<18:35>_E0+2 NEXT
            LOAD.BYTE);
        (IF AC[A+3]<9:35> NEQ 0 =>
            BYTE.PTR_AC[A+4] NEXT
            INCREMENT.BYTE.PTR NEXT
            AC[A+4]_BYTE.PTR NEXT
            AC[A+3]<9:35>_AC[A+3]<9:35>-1 NEXT
            LOAD.BYTE) NEXT
        D_S; S_TEMP NEXT
        (DECODE OPCODE<6:8> =>
            NOP;
            (IF S LSS D => CMPSKP);      ! CMPSL
            (IF S NEQ D => CMPEND);      ! CMPSE
            (IF S LEQ D => CMPSKP);      ! CMPSLE
            NOP;
            (IF S GEQ D => CMPSKP);      ! CMPSGE
            (IF S NEQ D => CMPSKP);      ! CMPSN
            (IF S GTR D => CMPSKP)) NEXT        ! CMPSG
        CLOOP;
COMPDONE:=(IF OPCODE NEQ #002 => CMPEND) NEXT
CMPSKP:=PC_PC+1);
CMPEND:=NOP);



    (IF (OPCODE GEQ #010) AND (OPCODE LEQ #011) => NOP); !
        CONVERT DECIMAL TO BINARY
        SRC.SETUP NEXT
        (DECODE AC[A]<0> =>
            MAN1<0:F70>_0;
            (MAN1<0:35>_AC[A+4]<0:35>;
```

```
            MAN1<F36:F70>_AC[A+5]<1:35>) NEXT
DBLOOP := (IF AC[A]<9:35> EQL 0 => DBDONE) NEXT
        GET.SRC.BYTE NEXT
        (IF (S LSS 0) OR (S GTR 9) => DBERR) NEXT
        MAN1<0:F70>_(MAN1<0:F70>*10)+S;
        AC[A]<9:35>_AC[A]<9:35>-1 NEXT
        DBLOOP;
DBDONE := (IF AC[A]<1> NEQ 0 => MAN1<0:F70>_-MAN1<0:F70> NEXT
        AC[A+4]<0:35>_MAN1<0:35>;
        AC[A+5]<0>_MAN1<0>;
        AC[A+5]<1:35>_MAN1<F36:F70>);



    (IF (OPCODE GEQ #012) AND (OPCODE LEQ #013) =>      !
        CONVERT BINARY TO DECIMAL
        MAN1<0:35>_AC[A]<0:35>;
        MAN1<F36:F70>_AC[A+1]<1:35> NEXT
        (IF MAN1<0> EQL 1 => MAN1_-MAN1; AC[A+3]<2>_1);
        (IF MAN1<0:F70> LEQ 999999999999999999999 =>
            TEMP_22; MAN2<0:F70>_100000000000000000000) NEXT
        (IF MAN1<0:F70> LEQ 99999999999999999999 =>
            TEMP_21; MAN2<0:F70>_10000000000000000000) NEXT
        (IF MAN1<0:F70> LEQ 9999999999999999999 =>
            TEMP_20; MAN2<0:F70>_1000000000000000000) NEXT
        .
        .
        .
        (IF MAN1<0:F70> LEQ 99 =>
            TEMP_2; MAN2<0:F70>_10) NEXT
        (IF MAN1<0:F70> LEQ 9 =>
            TEMP_1; MAN2<0:F70>_1) NEXT
       TEMP_AC[A+3]<9:35>-TEMP NEXT
        (IF TEMP GTR 0 =>
            POS_0;
            SIZE_AC[A+4]<6:11>;
            BYTE.PTR<12:17>_0;
            BYTE.PTR<18:35>_E0+1 NEXT
            LOAD.BYTE NEXT
            BYTE.PTR_AC[A+4] NEXT
BDFILL :=   DEPOSIT.BYTE NEXT
            INCREMENT.BYTE.PTR;
            TEMP_TEMP-1 NEXT
            (IF TEMP NEQ 0 => BDFILL));
        (IF TEMP LSS 0 => BDEND) NEXT
        BYTE.PTR_AC[A+4];
BDLOOP :=(IF MAN2<0:F70> NEQ 0 =>
            TEMP1_MAN1<0:F70>/MAN2<0:F70> NEXT
            (IF OPCODE EQL #012 => TEMP1_TEMP1+E1);
            (IF OPCODE EQL #013 =>
                S<0:34>_TEMP1;
                (DECODE (MAN2<0:F70> NEQ 1) OR (AC[A+3]<2> NEQ
                1) =>
```

```
                    S<35>_0; S<35>_1) NEXT
                TRANSLATE NEXT
                D_S) NEXT
            STORE,DST,BYTE;
            MAN2<0:F70>_MAN2<0:F70>-1 NEXT
            BDLOOP) NEXT
        PC_PC+1;
BDEND := NOP);



    (IF (OPCODE GEQ #014) AND (OPCODE LEQ #015) =>        ! MOVS-
        SRC,SETUP; DST,SETUP NEXT
MLOOP:= (IF AC[A+3]<9:35> NEQ 0 =>
            (IF AC[A]<9:35> EQL 0 =>
                POS_0;
                SIZE_AC[A+1]<6:11>;
                BYTE,PTR<12:17>_0;
                BYTE,PTR<18:35>_E0+1 NEXT
                LOAD,BYTE);
            (IF AC[A]<9:35> NEQ 0 => GET,SRC,BYTE) NEXT
            (IF OPCODE EQL #014 => S_S+E);       ! OFFSET
            (IF OPCODE EQL #015 => TRANSLATE) NEXT       !
            TRANSLATE
            STORE,DST,BYTE NEXT
            MLOOP) NEXT
        PC_PC+1));



    (IF (OPCODE GEQ #016) AND (OPCODE LEQ #017) =>        !
        MOVS-J
        SRC,SETUP; DST,SETUP NEXT
        (IF OPCODE EQL #017 =>  ! RIGHT JUSTIFY
            TEMP_AC[A]<9:35>-AC[A+3]<9:35> NEXT
MRJ :=      (IF TEMP LSS 0 =>
                POS_0;
                SIZE_AC[A+1]<6:11>;
                BYTE,PTR<12:17>_0;
                BYTE,PTR<18:35>_E0+1 NEXT
                LOAD,BYTE NEXT
                D_S;
                BYTE,PTR_AC[A+4] NEXT
                INCREMENT,BYTE,PTR NEXT
                AC[A+4]_BYTE,PTR NEXT
                DEPOSIT,BYTE;
                TEMP_TEMP+1 NEXT
                MRJ);
            (IF TEMP GTR 0 =>
                BYTE,PTR_AC[A+1];
                TEMP1_TEMP NEXT
                ADJUST,BYTE,PTR NEXT
```

```
                AC[A+1]_BYTE.PTR)) NEXT
MJLOOP:= (IF AC[A+3]<9:35> NEQ 0 =>
            (IF AC[A]<9:35> EQL 0 =>
                POS_0;
                SIZE_AC[A+1]<6:11>;
                BYTE.PTR<12:17>_0;
                BYTE.PTR<18:35>_E0+1 NEXT
                LOAD.BYTE);
            (IF AC[A]<9:35> NEQ 0 => GET.SRC.BYTE) NEXT
            D_S;
            STORE.DST.BYTE NEXT
            MJLOOP) NEXT
        PC_PC+1));


    (IF (OPCODE GEQ #020) AND (OPCODE LEQ #777) => MUUO) !
    UNDEFINED



    )    ! END EXT.I.SET


    );   ! END PDP10
```

1080,2040,2060 ENGINEERING FUNCTIONAL SPEC - CHAP 3,4

TO: KL10 LIST, J. PARSLOW (200 FILE)

TITLE: EXTERNAL MEMORY ADAPTER (DMA20) - REV 0

STATUS:

FILE: [EFS]CH3S04.SPC

PDM #: 200-200-034-00

DATE: 27 AUG 74

SUPERSEDED MEMOS: SBUS DIAGNOSTIC CYCLE SPEC., P. SULLIVAN, 24 OCT 73

SUPERSEDED SPECS:

ENGINEER: W. BRUCKERT

APPROVED:

EDITOR: M. MILLER

TYPIST: J. MCCARTHY

REVIEWED:

ABSTRACT

THE KL10 HAS TWO KINDS OF MEMORY, INTERNAL AND EXTERNAL. THIS CHAPTER DESCRRIBES THE FUNCTIONAL OPERATION OF THE DMA20 ADAPTER CONNECTED BETWEEN THE MBOX AND THE EXTERNAL CORE MEMORIES (E.G., MF10'S OR MG10'S). ERROR DETECTION AND REPORTING ARE ALSO DETAILED.

REVISION HISTORY

REV	DESCRIPTION	CHG NO	ORIG	DATE	APPD BY	DATE

1. SUMMARY

1. THE KL10 HAS TWO KINDS OF MEMORY, INTJRNAL (E.G., MA20) AND

2. EXTERNAL (MG10).

THIS CHAPTER DESCRIBES THE ADAPTER FOR THE SECOND KIND OF MEMORY.

THE DMA20 IS AN ADAPTER WHICH ENABLES THE KL10 TO ACCESS EXTERNAL CORE MEMORY STORAGE MODULES VIA THE STANDARD KI10 MEMORY BUS. FOUR SUCH KI BUSES CAN BE CONNECTED BETWEEN THE DMA20 AND THE STORAGE MODULES, THEREBY ENABLING FOUR SEPARATE STORAGE MODULES TO BE CYCLING AT ESENTIALLY THE SAME TIME IN A 4-WAY INTERLEAVE MODE. NON-INTERLEAVING AND 2-WAY INTERLEAVING MODES ARE ALSO POSSIBLE.

STORAGE MODULES ARE SET TO THE DESIRED INTERLEAVING MODE BY PHYSICALLY CONFIGURING THE MEMORY ADDRESS SWITCHES ON EACH STORAGE MODULE. THE DMA20 IS SET TO THE DESIRED REQUEST MODE, AND STATUS INFORMATION IS RETRIEVED WITH SBUS DIAGNOSTIC CYCLES.

2. TERMINOLOGY

2.1 KBUS

REFERS TO A MEMORY BUS WHICH MEETS THE KI MEMORY BUS SPECIFICATIONS. IT CONNECTS BETWEEN THE DMA20 AND THE EXTERNAL MEMORY STORAGE MODULES.

2.2 SBUS

INTERNAL STORAGE BUS THAT CONNECTS BETWEEN MBOX AND THE DMA20.

3. GOALS

1. THE DMA20 WILL CHECK PARITY ON ALL DATA TRANSFERS. IT WILL NOT CORRECT BAD PARITY.

2. THE DMA20 WILL BE CAPABLE OF 4-WORD READS AND WRITES.

3. THE DMA20 WILL DETECT NXM'S ON ANY WORD OF A MULTIPLE-WORD REQUEST, EXCEPT THE FIRST WORD. IN THE CASE OF THE FIRST WORD, NXM IS DETECTED BY THE MBOX.

4. THE DMA20 WILL CHECK ADDRESS PARITY ON THE SBUS.

4. NON-GOALS

1. THE DMA20 WILL NOT SUPPORT THE IGN PARITY FEATURE OF THE KBUS.

5. BASIC OPERATION

THE EXTERNAL MEMORY ADAPTER (DMA20) IS THE INTERFACE CONROL ELEMENT BETWEEN THE MBOX AND THE EXTERNAL DESTRUCTIVE-READOUT CORE MEMORY (SEE FIGURE 1). IN ADDITION TO DATA TRANSFERS, THE DMA20 PROVIDES ERROR DETECTION AND REPORTING CAPABILITIES.

THE DMA20 IS EQUIPPED WITH FOUR KBUSES. CONNECTED TO EACH KBUS CAN BE FROM 1 TO 16 STORAGE MODULES, PROVIDING AN OVERALL STORAGE CAPACITY OF 16 X 4 X 64, OR OVER FOUR MILLION, 36-BIT WORDS PLUS PARITY FOR MF10'S, OR 16 X 4 X 128 (OVER 8 MILLION WORDS) FOR MG10'S. COMMUNICATION BETWEEN THE DMA AND THE STORAGE MODULES TAKES PLACE OVER THE KBUSES. COMMUNICATION BETWEEN THE DMA AND THE MBOX TAKES PLACE OVER THE SBUS.

THE DMA HAS A SEPARATE INTERNAL BUFFER STORAGE REGISTER FOR EACH KBUS. THIS ENABLES SINGLE OR MULTIPLE REQUESTS TO BE SENT TO MEMORY OVER THE DIFFERENT KBUSES. DATA BETWEEN THE DMA20 AND THE MBOX IS ALWAYS TRANSFERRED ONE WORD AT A TIME, WITH OR WITHOUT INTERLEAVING OF THE DATA WORDS IN THE DMA BUFFER STORAGE REGISTERS.

TWO TYPES OF MEMORY INTERLEAVING ARE POSSIBLE: 2-WAY AND 4-WAY. WITH 2-WAY INTERLEAVING, MEMORY ADDRESSING ALTERNATES BETWEEN TWO CONSECUTIVE STORAGE MODULES, SUCH AS BETWEEN MODULES 00 AND 01. WITH 4-WAY INTERLEAVING, MEMORY ADDRESSING ROTATES THROUGH FOUR CONSECUTIVE STORAGE MODULES, SUCH AS 00, 01, 02, 03, OR 04, 05, 06, 07.

MODULE INTERLEAVING IS ENABLED BY PHYSICALLY SETTING THE APPROPRIATE INTERRLEAVE SWITCHES BEHIND THE FRONT PANEL DOOR OF THE STORAGE MODULES, AS SHOWN BELOW.

INTERLEAVE SWITCH SETTINGS FOR MF10

	STORAGE MODULE	
MODE	MADR SWITCHES 34	35
NON-INTERLEAVE	NORM	NORM
2-WAY	NORM	INTL
4-WAY	INTL	INTL

FOR THE SYSTEM TO GAIN FULL ADVANTAGE OF THE MEMORY INTERLEAVING, THE DMA20 SHOULD BE SWITCHED (THROUGH SOFTWARE)

TO THE APPROPRIATE "REQUEST MODE", AS SHOWN BELOW, FASTEST OPERATION IS ACHIEVED WITH REQUEST MODE 4 AND THE MADR SWITCHES SET FOR 4-WAY.

DMA20 REQUEST MODE	NON-INTL	R	2-WAY	R	4-WAY	R
MODE 1	A (6)	1	A (5)	2	A (4)	3
MODE 2	NA		A (3)	4	A (2)	5
MODE 4	NA		NA		A (1)	6

A = ALLOWED

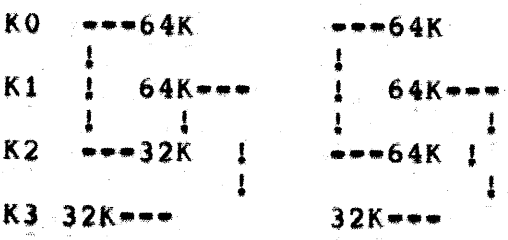
NA = NOT ALLOWED

THE ASSOCIATED NUMBERS IN PARENTHESES ARE LISTED FROM 1 TO 6, WITH 1 BEING THE MOST DESIRABLE AND 6 BEING THE LEAST DESIRABLE CONFIGURATION.

R = RESTRICTIONS (OTHER THAN NORMAL MEMORY CONFIGURATION

1 - 3 NONE

4 WHEN ADDED TOGETHER, KBUSES K0 AND K2 MUST SEE THE SAME ADDRESS RANGE AS KBUSES K1 AND K3, IGNORING BIT 35, FOR EXAMPLE:



THE STORAGE MODULES ON BUSES K0 AND K2 MUST RESPOND TO EVEN ADDRESSES; THAT IS, THE LEAST SIGNIFICANT BIT (LSB) OF MODULE SELECT MUST BE SELECTED FOR A ZERO. THE CONVERSE IS TRUE (ODD ADDRESSES) FOR MODULES ON BUSES K1 AND K3

5 SAME AS ABOVE

6 THE TWO LEAST SIGNIFICANT BITS (OF THE MODULE SELECT SWITCHES) MUST EQUAL THE BUS NUMBER, FOR EXAMPLE:

BIT 18 19	KBUS
0 0	0

0 1	1
1 0	2
1 1	3

INTERLEAVING EFFECTIVELY DECREASES CYCLE TIME, THUS REDUCING PROCESSOR MEMORY WAIT TIME. NORMALLY, THE DMA20 SHOULD BE OPERATED IN REQUEST MODE 4. HOWEVER, DEGRADED INTERLEAVED OPERATION IS ALLOWED WITH REQUEST MODE 2, AND FURTHER MEMORY SYSTEM DEGRADATION IS ALLOWED WITH REQUEST MODE 1. CHANGING FROM MODE 4 TO MODE 2 OR MODE 1 (THROUGH SOFTWARE) RQUIRES CORRESPONDING CHANGES IN THE MADR PHYSICAL SWITCHES.

ONLY STORAGE MODULES OF THE SAME SIZE AND HAVING CONSECUTIVE PORT ADDRESSES WITH AN EVEN PORT ADDRESS FIRST MAY BE INTERLEAVED. FOR EXAMPLE, 2-WAY INTERLEAVING CAN EXIST BETWEEN STORAGE MODULES 0 AND 1 (OR 2 AND 3) BUT NOT BETWEEN 1 AND 2 (OR 3 AND 4); 4-WAY INTERLEAVING CA EXIST BETWEEN STORAGE MODULES 0, 1, 2, 3 BUT NOT BETWEEN 1, 2, 3, 4 (INTERLEAVING CANNOT RUN FROM ONE COLUMN TO ANOTHER). SEVERAL EXAMPLES ARE GIVEN BELOW (SEE FIGURE 2). NOTE THAT, IN 2-WAY, THE STORAGE MODULES DO NOT HAVE TO BE CONNECTED IN NUMERICAL ORDER TO THE KBUSES. THEY MUST BE IN NUMERICAL ORDER FOR 4-WAY.

	KBUS	MODULE NO.	SIZE	OR	SIZE
2-WAY	0	2	32K		64K
	1	3	32K		64K
	2	0	32K		32K
	3	1	32K		32K
4-WAY	0	0	64K		32K
	1	1	64K		32K
	2	2	64K		32K
	3	3	64K		32K

THE OPERATION OF THE DMA20 IS ASYNCHRONOUS WITH RESPECT TO CORE AND SYNCHRONOUS WITH RESPECT TO THE MBOX. MEMORY OPERATION IS INITIATED WHEN THE MBOX REQUESTS A MEMORY CYCLE BY ISSUING A MEMORY ADDRESS AND CYCLE TYPE (READ, WRITE, ETC.) OVER THE SBUS. AFTER DECODING THE INCOMING REQUEST, THE DMA ENABLES THE APPROPRIATE KBUS AND ISSUES A MEMORY CYCLE REQUEST TOGETHER WITH THE MEMORY ADDRESS TO THOSE STORAGE MODULES CONNECTED TO THE KBUS.

IF THE ADDRESSED STORAGE MODULE IS NOT BUSY, IT READS THE

ADDRESS INTO ITS INPUT BUFFER AND RESPONDS WITH AN ACKNOWLEDGE SIGNAL. THE ACKNOWLEDGE SIGNAL IS GATED THROUGH THE DMA TO THE MBOX, INDICATING ACCEPTANCE OF THE MEMORY REQUEST. THE DMA THEN CONTROLS THE REQUESTED CYCLE TYPE, TRANSFERRING DATA BETWEEN THE MBOX AND CORE.

MOST CORE STORAGE MODULES HAVE FOUR PORTS, LABELLED P0, P1, P2, AND P3 TO FACILITATE CONNECTIONS BETWEEN THAT STORAGE MODULE, THE MEMORY INTERFACE (E.G., THE DMA20), AND OTHER PERIPHERALS, SUCH AS AN RH10. ALL PORTS OF EACH STORAGE MODULE SHOULD HAVE THE SAME ADDRESS SWITCH SELECTION. IN THE EVENT OF TWO OR MORE SIMULTANEOUS REQUESTS, THE MEMORY PRIORITY NETWORK ALLOWS ACCESS TO THE PORT HAVING THE HIGHEST PRIORITY. FIRST PRIORITY IS ASSIGNED MEMORY PORT P0; SECOND PRIORITY IS ASSIGNED TO MEMORY PORT P1. THIRD PRIORITY IS SHARED BETWEEN MEMORY PORTS P2 AND P3 SUCH THAT IF PORT P2 WERE SERVICED LAST IN THE PREVIOUS MEMORY CYCLES, PORT P3 WOULD BE GIVEN PRIORITY OVER PORT P2 SHOULD SIMULTANEOUS REQUESTS OCCUR ON THESE PORTS.

IN THE EVENT OF A MALFUNCTION OF ONE PARTICULAR STORAGE MODULE (FIGURE 1), THE MONITOR PROGRAM WILL RESTRUCTURE THE OVERALL CORE MEMORY TO EFFECTIVELY BY-PASS THE FAULTY MODULE PLUS THE OTHER STORAGE MODULES ASSOCIATED WITH IT FOR INTERLEAVING. FOR EXAMPLE, IN 4-WAY INTERLEAVING, IF STORAGE MODULE 03 IS FAULTY, THE MONITOR WILL BY-PASS ALL FOUR CORE STORAGE MODULES 00, 01, 02, AND 03. DIRECT COMMUNICATION BETWEEN THESE FOUR MODULES AND THEIR RESPECTIVE PERIPHERAL(S) WILL ALSO TERMINATE.

DEPENDING ON THE NATURE OF THE MALFUNCTION, THE COMPUTER OPERATOR MAY ELECT TO SHUT DOWN THE SYSTEM AND SET THE MEMORY INTERLEAVE SWITCHES TO DIFFERENT POSITIONS (E.G., NON-INTERLEAVING MODE), THEREBY PERMITTING USAGE OF ALL AVAILABLE MEMORY, EXCEPT FOR THE FAILING BOX, UNTIL THE MALFUNCTION IS CORRECTED. IN THIS CASE, THE DMA MODE MAY ALSO HAVE TO BE CHANGED (E.G., THROUGH SOFTWARE).

6. MEMORY ADDRESSING

THE MANNER IN WHICH AN MF10 STORAGE MODULE IS PROGRAMMED (IN THE KL10 CONFIGURATION) IS ILLUSTRATED IN FIGURE 3. BITS 14 - 19 DETERMINE WHICH ONE OF THE 64 STORAGE MODULES WILL BE SELECTED, FOR A GIVEN INSTRUCTION. BITS 20 - 35 DETERMINE WHICH ONE OF THE 64K WORDS WILL BE ADDRESSED IN THAT PARTICULAR STORAGE MODULE. FOR EXAMPLE, 03 000004 (OCTAL) SELECTS THE FOURTH WORD IN MF10 STORAGE MODULE 03.

WHEN 2-WAY INTERLEAVING IS SELECTED FOR AN MF10, THE LEAST SIGNIFICANT MODULE-ADDRESS BIT (BIT 20 FOR 32K AND BIT 19 FOR 64K) IS SWAPPED WITH THE LEAST SIGNIFICANT WORD-ADDRESS BIT (BIT 35). IN THIS CONFIGURATION, SHOULD THE PROCESSOR GENERATE CONSECUTIVE ADDRESSES STARTING AT ADDRESS 0 OF MF10 NUMBER 0, THE SUBSEQUENT MEMORY CYCLES WOULD ALTERNATE BETWEEN MF10 MEMORIES 0 AND 1, ADDRESSING ONLY THE EVEN WORD LOCATIONS (0,

2, 4, 6, ETC.) UNTIL ALL THE EVEN LOCATIONS WERE ADDRESSED, AND THEN ADDRESSING ALL THE ODD LOCATIONS (1, 3, 5, 7, ETC.) UNTIL ALL THE ODD LOCATIONS WERE ADDRESSED. THIS SEQUENCE WOULD BE CONTINUOUSLY REPEATED UNTIL THE PROCESSOR CEASED GENERATING CONSECUTIVE ADDRESSES.

WHEN 4-WAY INTERLEAVING IS SELECTED FOR AN MF10, THE TWO LEAST SIGNIFICANT MODULE-ADDRESS BITS (BITS 19, 20 FOR 32K AND BITS 18, 19 FOR 64K) ARE SWAPPED WITH THE TWO LEAST SIGNIFICANT WORD-ADDRESS BITS (BITS 34, 35). IN THIS CONFIGURATION, SHOULD THE PROCESSOR GENERATE CONSECUTIVE ADDRESSES STARTING AT ADDRESS 0 OF MF10 NUMBER 0, THE SUBSEQUENT MEMORY CYCLES WOULD ROTATE BETWEEN MF10 MEMORIES 0, 1, 2, 3, STARTING AT ADDRESS 0. MEMORY LOCATIONS WOULD BE ADDRESSED IN THE FOLLOWING SEQUENCE AT EACH OF THE FOUR MF10 MEMORIES:

WORD LOCATION (OCTAL)

0, 4, 10, 14, 20, 24, ETC.

1, 5, 11, 15, 21, 25, ETC.

2, 6, 12, 16, 22, 26, ETC.

3, 7, 13, 17, 23, 27, ETC.

THIS ADDRESSING SEQUENCE (0, 4, 10 . . . 1, 5, 11 . . . 2, 6, 12 . . . 3, 7, 13 . . .) WOULD BE CONTINUOUSLY REPEATED UNTIL THE PROCESSOR CEASED GENERATING CONSECUTIVE ADDRESSES.

FIGURE 4 ILLUSTRATES 4-WAY INTERLEAVING BETWEEN FOUR 64K STORAGE MODULES 00, 01, 02, AND 03, IN THAT ORDER. BEFORE SWAPPING, THE MEMORY LOCATIONS ARE ADDRESSED IN ASCENDING NUMERICAL ORDER, GOING FROM STORAGE MODULE 00 TO STORAGE MODULE 03, UNTIL ALL 256K LOCATIONS ARE ADDRESSED. THUS THE FIRST 64K CONSECUTIVE ADDRESSES START AT 000,000 (OCTAL) AND END AT 177,777 (OCTAL); THE NEXT 64K ADDRESSES START AT 200,000 (OCTAL) AND END AT 377,777 (OCTAL); AND SO FORTH.

WHEN BITS 18 AND 19 ARE SWAPPED WITH BITS 34 AND 35, THE END RESULT IS THAT EVERY FIFTH WORD (0, 4, 10, 14 . . .) IN EACH MODULE (I.E., 0, 1, 2, 3) IS NOW ADDRESSED SEQUENTIALLY, UNTIL 16K WORDS ARE ADDRESSED IN EACH MODULE IN THE FIRST PASS. ADDRESSING THEN LOOPS BACK TO THE BEGINNING OF EACH MODULE, INCREMENTED BY ONE, AND THEN CONTINUES FOR ANOTHER 16K WORDS. THIS PROCESS CONTINUES TWO MORE TIMES UNTIL ALL 64K WORDS IN EACH MODULE ARE ADDRESSED. THIS HAS ESSENTIALLY THE SAME EFFECT AS TRANSFERRING EVERY FIFTH WORD TO OR FROM STORAGE MODULES 00 - 03 SIMULTANEOUSLY OVER THE FOUR KBUSES.

THE NUMBER OF WORDS TO BE TRANSFERRED BETWEEN THE DMA AND THE STORAGE MODULES IN A GIVEN MEMORY CYCLE DEPENDS ON THE NUMBER OF WORD REQUEST LINES (SBUS RQ0 -3) ASSERTED. THESE LINES ALSO

SPECIFY THE ORDER IN WHICH THE WORDS ARE RETRIEVED/WRITTEN. FOR EXAMPLE, A MEMORY REFERENCE CAN BE ISSUED REQUIRING THREE WORDS AT ADDRESSES 00, 01 OR 03. IN THIS CASE, THE DMA ADDRESSING LOGIC WILL ACCESS ADDRESSES 00, 01, SKIP 02, AND IMMEDIATELY ACCESS ADDRESS 03. THUS, TIME IS NOT WASTED REQUESTING AN ACCESS TO AN UNWANTED MEMORY ADDRESS.

7. LOADING RULES

THE SAME GUIDELINES FOR THE KA AND KI MEMORY BUSES APPLY TO EACH DMA20 KBUS.

8. ERROR DETECTION AND REPORTING

THE DMA20 PROVIDES THE CORE MEMORY SYSTEM WITH AN ERROR DETECTION AND REPORTING CAPABILITY. ADDRESS PARITY, READ DATA PARITY, AND WRITE DATA PARITY ARE ALL CHECKED IN THE DMA DURING THE THREE CYCLE OPERATIONS. EACH TYPE OF PARITY ERROR SETS AN APPROPRIATE FLAG IN THE DMA STATUS REGISTER. THE DMA ERROR DETECTION LOGIC ALSO STORES THE ADDRESS OF THE FIRST ERROR IN THE DMA ERROR ADDRESS REGISTER UNTIL ALL ERROR BITS ARE CLEARED. WHEN ANY DATA PARITY OR NXM ERROR IS DETECTED, A CUMULATIVE ERROR SIGNAL (SBUS ERROR) IS SENT TO THE MBOX. WHEN ANY ADDRESS PARITY ERROR IS DETECTED, SBUS ADR PAR ERR IS SENT TO THE MBOX.

IF AN ACKNOWLEDGE (ACK) IS RECEIVED FROM THE FIRST STORAGE MODULE, AND ANY SBSEQUENT REFERENCE IN THE CYCLE DOES NOT RECEIVE AN ACK (I.E., A STORAGE MODULE DID NOT RESPOND), THEN A NON-EXISTENT MEMORY (NXM) FLAG IS SET IN THE DMA STATUS REGISTER IN THE SAME WAY AS A PARITY ERROR. IF THE FIRST WORD DOES NOT RECEIVE AN ACKNOWLEDGE, THE MBOX WILL GENERATE THE NXM INSTEAD.

DURING A DIAGNOSTIC CYCLE, ALL COMMUNICATION (INCLUDING ADDRESSING) BETWEEN THE MBOX AND THE DMA IS PERFORMED OVER THE SBUS DATA LINES. THE OVERALL DIAGNOSTIC CYCLE CONSISTS OF TWO HALVES: TO DMA AND FROM DMA. GENERALLY, DURING THE TO DMA PORTION OF THE CYCLE, DIAGNOSTIC CONDITIONS ARE SET; DURING THE FROM DMA PORTION, THOSE CONDITIONS ARE VERIFIED OR IN SOME CASES ERROR CONDITIONS RETRIEVED.

THE DIAGNOSTIC CYCLE IS INITIATED AND SYNCED TO THE PHASE A CLOCK WITH A COMPLETE CYCLE REQUIRING 500 NS. DURING THE FIRST 250 NS, THE MBOX SENDS OUT A DMA ADDRESS, FUNCTION CODE, AND INFORMATION SPECIFIED BY THE FUNCTION CODE. DURING THE SECOND 250 NS, THE DMA SENDS INFORMATION BACK TO THE MBOX, USING THE DATA LINES AS SPECIFIED BY THE FUNCTION CODE.

TWO FUNCTION CODES ARE DEFINED (FUNCTION 0 AND FUNCTION 1) AND THEIR FORMATS ARE DETAILED IN FIGURES 5 AND 6, RESPECTIVELY. EITHER FUNCTION CODE IS ENABLED IN THE DMA WHEN AN SBUS

DIAGNOSTIC SIGNAL IS SENT TO THE DMA FROM THE MBOX, THE DIAGNOSTIC SIGNAL EFFECTIVELY ENABLES THE DMA DIAGNOSTIC ADDRESS DECODER DURING THE FIRST HALF-CYCLE AND THE TRANSMISSION FUNCTION FOR DATA RETRIEVAL DURING THE SECOND HALF-CYCLE, THOSE BIT POSITIONS WHICH SET OR CLEAR CONDITIONS IN THE FIRST HALF-CYCLE WILL CONTAIN THE NEW DATA WHEN RETRIEVED DURING THE SECOND HALF-CYCLE.

8,1 FUNCTION 0 (TO DMA)

CONTROLLER ADDRESS, THIS FIELD SPECIFIES THE ADDRESS OF THE CONTROLLER TO BE EXERCISED, IN THIS CASE, THE ADDRESS FIELD WILL CONTAIN OCTAL 4, SPECIFYING THE DMA20.

CLEAR BIT, THIS 1-BIT FIELD IS APPLIED TO THE ERROR LOGIC AND, IF A 1, GENERATES THE CLEARING FUNCTION FOR THE FOUR FLOPS IN THE PARITY STATUS REGISTER.

SET INTERLEAVE MODE, THIS 2-BIT FIELD, APPLIED TO THE ERROR LOGIC, SETS THE DMA INTERLEAVE MODE AS DETERMINED BY THE CODE, WITH A CODE OF 00, THE DMA IS SET OFF LINE BY RESETTING BOTH INTERLEAVE FLOPS, THE REMANING THREE CODES SET/RESET THE FLOPS AS REQUIRED FOR THE SELECTED INTERLEAVE MODE, IN REQUEST MODE 1, THE SYSTEM IS SET UP FOR EITHER NON-INTERLEAVING, 2-WAY INTERLEAVING, OR 4-WAY INTERLEAVING, IN REQUEST MODE 2, THE SYSTEM IS SET UP ONLY FOR 2-WAY OR 4-WAY INTERLEAVING, IN REQUEST MODE 4, THE SYSTEM IS SET UP ONLY FOR 4-WAY INTERLEAVING.

LOAD ENABLE, IF A 1, THIS BIT ENABLES GATING OF THE INTERLEAVE MODE FIELD TO THE INTERLEAVE FLOPS, IF THE BIT IS A 0, BITS 6 AND 7 ARE IGNORES BY THE DMA.

FUNCTION CODE, THIS 5-BIT FIELD SPECIFIES THE FUNCTION CODE, IN THIS CASE, OCTAL 00 = FUNCTION CODE 0.

8,2 FUNCTION 0 (FROM DMA)

ERROR REPORTING FLAGS, THIS 6-BIT FIELD RETURNS ERROR INFORMATION FROM THE STATUS REGISTER IN THE ERROR LOGIC, THE OUTPUT OF THE STATUS REGISTER IS ENABLED TO THE SBUS DATA LINES BY AN ERROR TRANSMIT FUNCTION (ERR XMIT) WHICH WAS GENERATED DURING THE FIRST HALF-CYCLE.

REPORT INTERLEAVE MODE, THIS 2-BIT FIELD IDENTIFIES THE CURRENT DMA INTERLEAVE MODE, THE CONDITION OF THE ERROR LOGIC INTERLEAVE FLOPS IS ENABLED TO THE DATA LINESBY ERR XMIT.

ADDRESS INFORMATION, THIS 28-BIT FIELD RETURNS ADDRESS INFORMATION CONCERNING THE FIRST PARITY ERROR, THE OUTPUTS OF THE APPROPRIATE FLOPS (RD RQ, WR RQ, ETC,) AND THE ERROR ADDRESS REGISTER IN THE ERROR LOGIC ARE ENABLED TO THE SBUS

DATA LINES SBY ERR XMIT.

8.3 FUNCTION 1 (TO DMA)

MEMORY CONTROLLER ADDRESS. THIS 5-BIT FIELD SPECIFIES THE ADDRESS OF THE CONTROLLER TO BE EXERCCISED BY THE DIAGNOSTIC CYCLE. IN THIS CASE, THE ADDRESS FIELD WILL CONTAIN OCTAL 4, SPECIFYING THE DMA20.

LOOP-AROUND MODE. WHEN BIT 12 IS A 1, THE DMA IS CONDITIONED FOR THE LOOP-AROUND MODE. THIS MODE CHECKS DATA TRANSFERS BETWEEN THE MBOX AND A SELECTED DMA DATA BUFFER WITHOUT GOING THROUGH CORE. IT IS USED MAINLY DURING THE DIAGNOSTIC CYCLE FOR ISOLATING SYSTEM FAILURES. DURING A LOOP-AROUND OPERATION, A WRITE CYCLE FOLLOWED BY A READ WILL RESULT IN THE WRITE DATA BEING RETURNED ON THE READ CYCLE. THE DATA NEVER GETS SENT TO CORE. THE READ COMMAND CLEARS THE LOOP-AROUND MODE. FOR A NO-FAULT CONDITION, THE RETRIEVED DATA MUST BE IDENTICAL TO THAT DEPOSITED DURING THE PREVIOUS WRITE CYCLE.

FUNCTION CODE. THIS 5-BIT FIELD SPECIFIES THE FUNCTION CODE. IN THIS CASE, OCTAL 01 = FUNCTION CODE 1.

8.4 FUNCTION 1 (FROM DMA)

NUMBER STORAGE MODULE/CONROLLER. THE DMA ALWAYS RETURNS ZEROES IN THIS 8-BIT FIELD.

INDICATE MEMORY TYPE. THIS 4-BIT FIELD INDICATES THE TYPE OF MEMORY CONNECTED. IN THIS CASE, A BIT CONFIGURATION OF OCTAL 02 IS RETURNED, INDICATING A DMA20.

LOOP-AROUND MODE. WHEN SBIT 12 IS A 1, THIS FIELD INDICATES THAT THE DMA WAS CONDITIONED FOR A LOOP-AROUND OPERATION DURING THE PREVIOUS HALF-CYCLE.

8.5 SINGLE-STEP MODE

THE DMA20 WILL OPERATE CORRECTLY WITH REDUCED OR SINGLE-STEP CLOCKS AS CONTROLLED BY THE PROCESSOR.

9. APPENDIX A - LOOSE ENDS

THERE ARE NONE.

FIGURE 1

POSSIBLE CONNECTION OF MF10 STORAGE MODULES TO DMA20

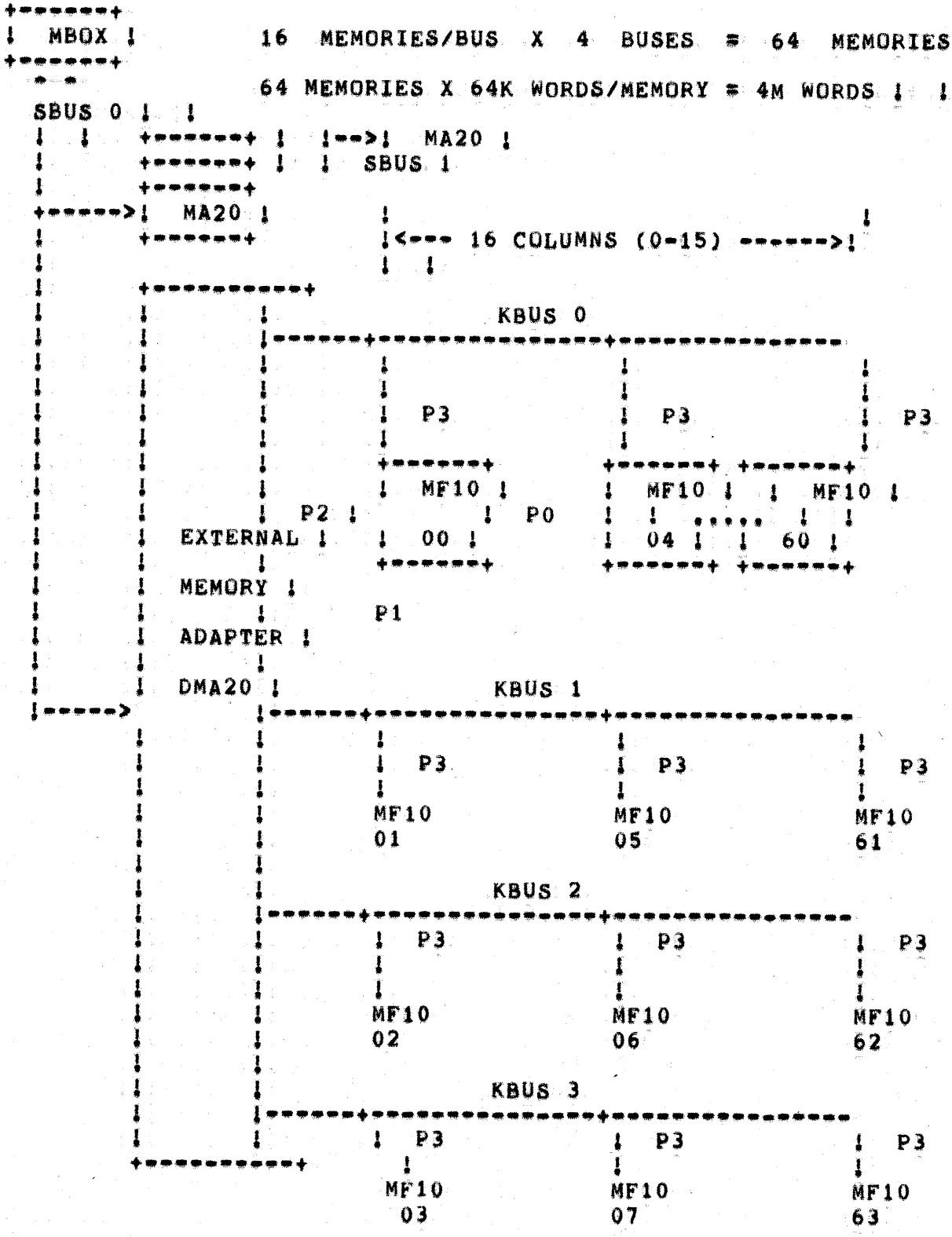
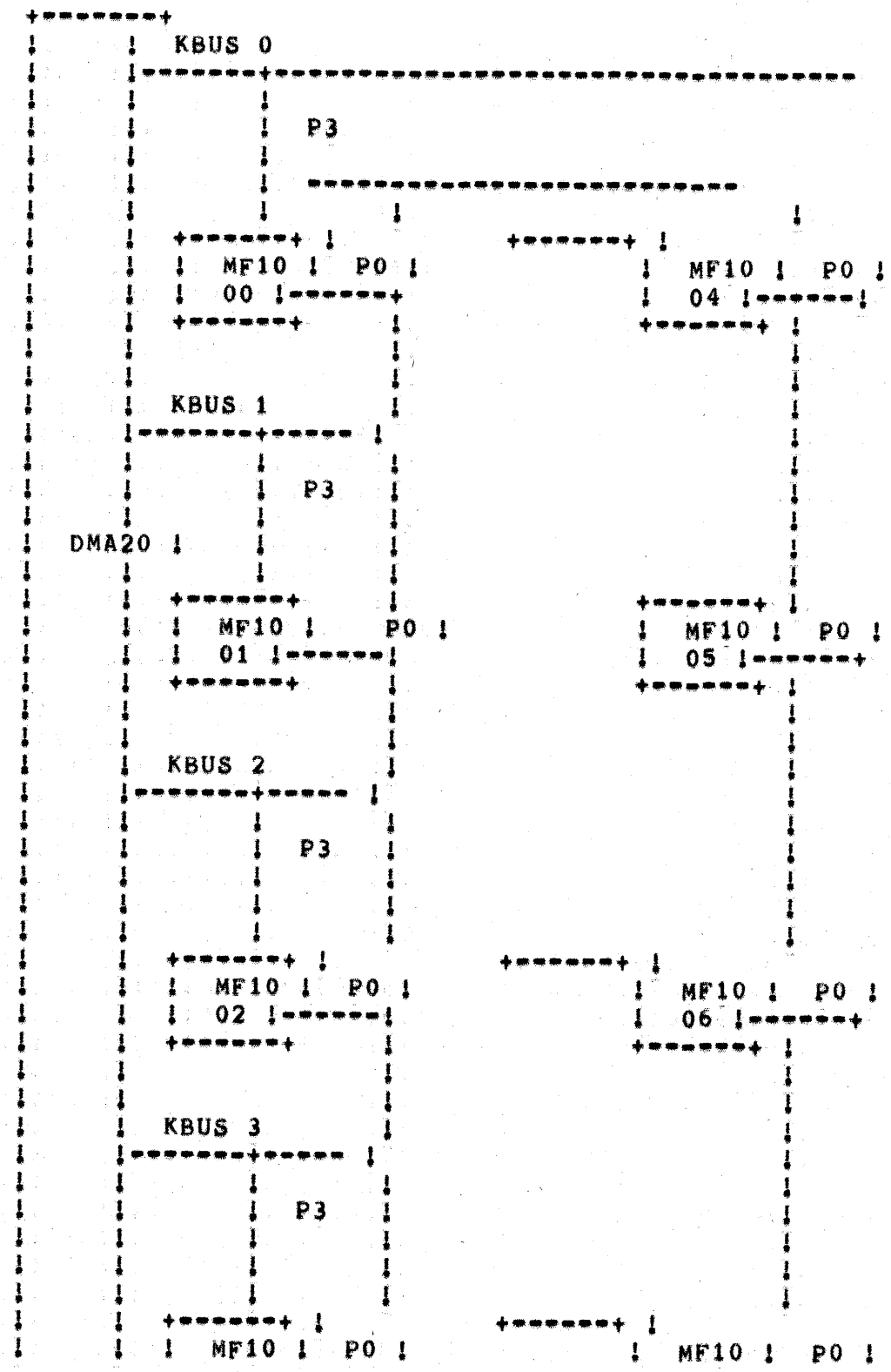


FIGURE 2

CONNECTING PERIPHERAL DIRECTLY TO
STORAGE MODULES




03
07
PERIPHERAL

FIGURE 3

BASIC MEMORY ADDRESSING FOR MF10

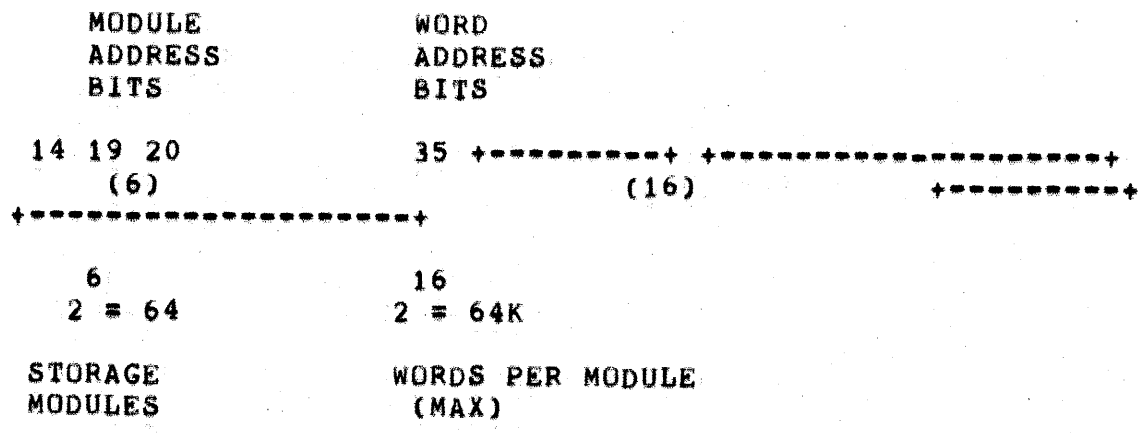


FIGURE 4

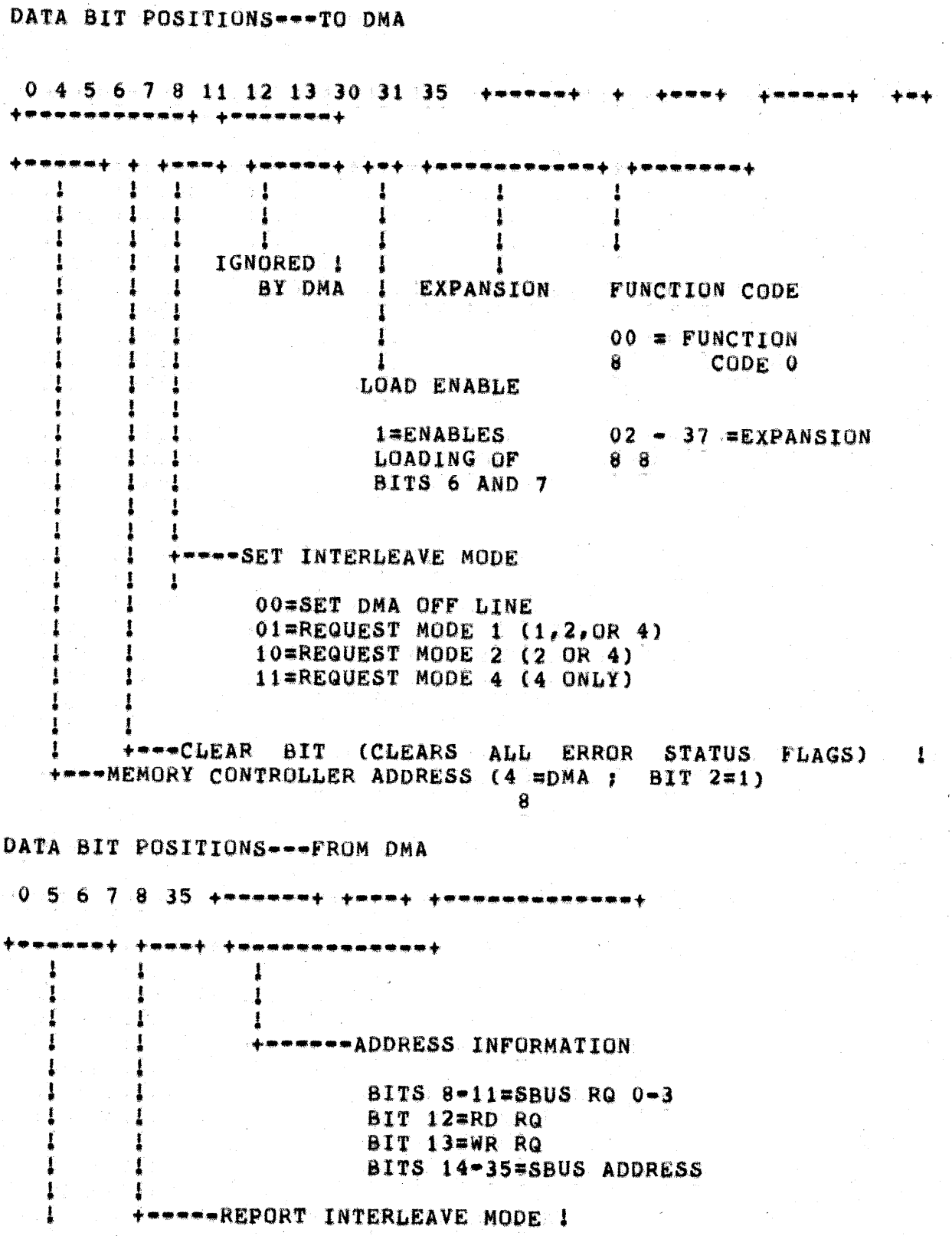
SWAPPING FOR 4-WAY INTERLEAVING
OF MF10 MEMORIES

MEMORY ADDRESS	MEMORY ADDRESS BEFORE SWAPPING				MEMORY ADDRESS AFTER SWAPPING			
	MODULE	WORD ADDRESS	BITS (14-19)	BITS (33-35)	MODULE	WORD ADDRESS	BITS (14-19)	BITS (33-35)
			XXXXXX	XXX			XXXXXX	XXX
0 0 000000 000	0	0	000000	000	0	0	000000	000
8	0	1	000000	001	1	0	000001	000
	0	2	000000	010	2	0	000010	000
	0	3	000000	011	3	0	000011	000
	0	4	000000	100	0	4	000000	100
	0	5	000000	101	1	4	000001	100
	0	6	000000	110	2	4	000010	100
	0	7	000000	111	3	4	000011	100
177777	0	177777						
8		8						
200000	1	0	000001	000	0	1	000000	001
8	1	1			1	1		
	1	2			2	1		
	1	3			3	1		
	1	4			0	5		
	1	5			1	5		
	1	6			2	5		
	1	7			3	5		
377777	1	177777						
8		8						
400000	2	0			0	2		
8	2	1			1	2		
	2	2			2	2		
	2	3			3	2		
	2	4			0	6		
	2	5			1	6		
	2	6			2	6		

	2	7		3 6
577777 8	2	177777 8		
600000 8	3	0		0 3
	3	1		1 3
	3	2		2 3
	3	3		3 3
	3	4		0 7
	3	5		1 7
	3	6		2 7
	3	7		3 7
777777 8	3	177777 8		

FIGURE 5

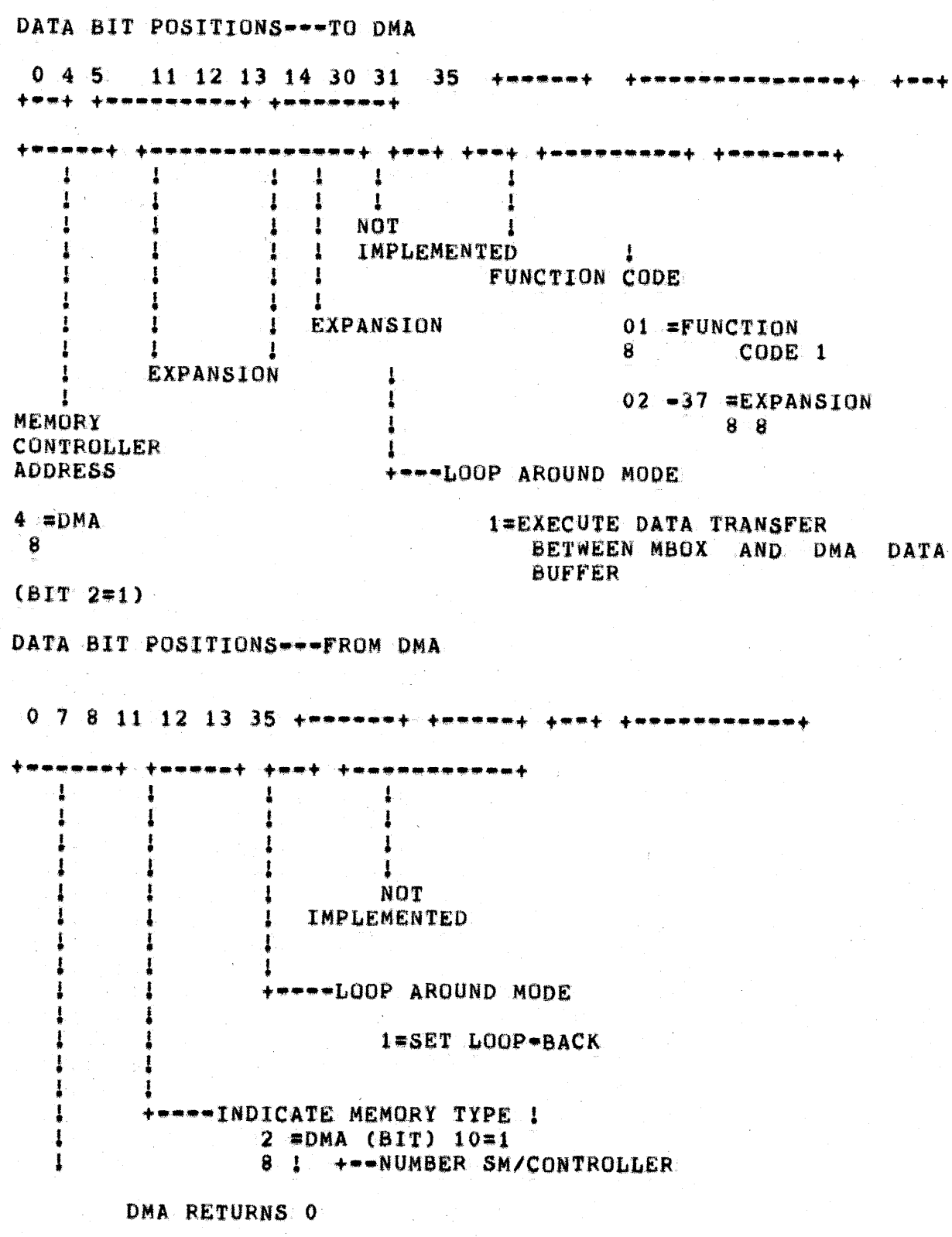
FUNCTION 0 FIELD FORMAT DESCRIPTIONS



```
!              00=OFF LINE
!              01=REQUEST MODE 1
!              10=REQUEST MODE 2
!              11=REQUEST MODE 4 !  +--ERROR REPORTING FLAGS

     BITS 0,1=EXPANSION BIT  2  SET=NXM  FLAG  (DMA)  BIT  3
     SET=READ  PARITY ERROR BIT 4 SET=WRITE PARITY ERROR BIT
     5 SET=ADDRESS PARITY ERROR
```

FIGURE 6

FUNCTION 1 FIELD FORMAT DESCRIPTIONS



[END OF CH3S04.SPC]

1080,2040,2060 ENGINEERING FUNCTIONAL SPEC - CHAP 3,6

TO: KL10 LIST, J. PARSLOW (200 FILE)

TITLE: KL MEMORY BUS (SBUS)

STATUS:

FILE: [EFS]CH3S06.SPC

PDM #: 200-200-032-00

DATE: 9 APR 74

SUPERSEDED MEMOS:

SUPERSEDED SPECS:

ENGINEER: P. GUGLIELMI

APPROVED:

EDITOR: T. HASTINGS

TYPIST:

REVIEWED:

DISTRIBUTED:

ABSTRACT

REVISION HISTORY

REV	DESCRIPTION	CHG NO	ORIG	DATE	APPD BY	DATE

IT IS NOT CURRENTLY POSSIBLE FOR THE MBOX TO CORRECTLY ASSOCIATE SBUS ERRORS DUE TO BAD DATA PARITY ON BOTH READ AND WRITE WITH THE FUNCTIONAL COMPONENT (EBOX OR CHANNEL) WHICH MADE THE REFERENCE. THIS MEANS THAT A TRANSFER ON A CHANNEL COULD BE ABORTED AS A RESULT OF AN ERROR MADE BY AN EBOX PROCESS OR VICE VERSA. THIS SITUATION EXISTS BECAUSE THE DMA20 (CURRENTLY THE ONLY SBUS DEVICE WHICH CHECKS DATA PARITY ON BOTH READS AND WRITES) CAN TAKE UP TO 500 NANO SECONDS BEYOND COMPLETION OF A TRANSFER TO NOTIFY THE MBOX OF AN SBUS ERROR DUE TO BAD DATA PARITY. DURING THIS 500 NANO SECONDS, A MEMORY REFERENCE FOR ANOTHER FUNCTIONAL COMPONENT TO A DIFFERENT BANK OF CORE COULD BE INITIATED. THE MBOX MUST BE ABLE TO ASSOCIATE THE ERROR WITH THE PROPER FUNCTIONAL COMPONENT SO THAT THE HARDWARE CAN TERMINATE THAT PROCESS AND NOT SOME OTHER PROCESS.

TO ACCOMPLISH THIS THE FOLLOWING MODIFICATIONS OF THE SBUS ARE PROPOSED:

1. ADD AN SBUS SIGNAL CALLED "SBUS REQ ODD L" GOING FROM MBOX TO MEMORY.

2. ADD AN SBUS SIGNAL CALLED "SBUS ERROR B L" TO THE SBUS AND RENAME "SBUS ERROR L" TO "SBUS ERROR A L".

THESE NEW SIGNALS WILL BE USED IN THE FOLLOWING MANNER.

THE MBOX WILL MAINTAIN TWO NEW FLIP-FLOPS NAMED PREVIOUS CHANNEL (PREV CHAN) AND REQUEST ODD (REQ ODD). PREVIOUS CHANNEL WILL BE TRUE WHENEVER THE PREVIOUS MEMORY REFERENCE WAS CAUSED BY A CHANNEL. REQUEST ODD WILL TOGGLE EACH TIME A MEMORY REQUEST IS MADE AND WILL BE SENT TO MEMORY ALONG WITH THE CURRENT REQUEST. THIS WILL ALLOW THE CURRENT MEMORY REQUEST TO BE DECLARED EVEN OR ODD SO THAT ALTERNATE MEMORY REQUESTS CAN BE DISTINGUISHED.

THE MEMORY THAT RECOGNIZES THE REQUEST STORES REQ ODD. IF THAT MEMORY DETECTS AN ERROR DURING A TRANSFER, THEN IT NOTIFIES THE PROCESSOR THAT THE ERROR OCCURRED VIA "SBUS ERROR A" AND "SBUS ERROR B". THESE SIGNAL LINES ARE ENCODED AS FOLLOWS.

SBUS	ERROR	MEANING
A	B	
0	0	NO ERROR
0	1	NON ASSOCIATABLE ERROR
1	0	ERROR ON AN EVEN REQUEST
1	1	ERROR ON AN ODD REQUEST

A "NON ASSOCIATABLE ERROR" IS AN SBUS ERROR THAT THE MBOX WILL NOT BE ABLE TO ASSOCIATE WITH ANY PARTICULAR MEMORY TRANSFER

BECAUSE IT TOOK TOO LONG TO DETECT THE ERROR, THE ONLY ERROR THAT FALLS INTO THIS CLASS IS THE 8 MICROSECOND INCOMPLETE CYCLE TIMEOUT ERROR WHICH THE MA20 DETECTS, IF THIS ERROR CONDITION IS DETECTED, THE MBOX WILL SET THE APR SBUS ERROR FLAG IN THE EBOX, THE CHANNELS WILL NOT BE NOTIFIED OF THE OCCURRENCE OF THIS ERROR, THE OCCURRENCE OF THIS ERROR INDICATES A SERIOUS HARDWARE MALFUNCTION.

ON EVEN AND ODD REQUEST SBUS ERRORS, THE MBOX WILL SET THE SBUS APR FLAG IN THE EBOX, LOOK AT THE CURRENT STATE "REQUEST ODD" FLIP-FLOP MBOX AND DECIDE WHETHER THE ERROR IS ASSOCIATED WITH THE CURRENT OR PREVIOUS MEMORY TRANSFER, AND SEND AN ERROR NOTIFICATION TO EITHER THE EBOX OR THE CHANNELS SO THAT THE APPROPRIATE TRANSFER CAN BE RETRIED OR ABORTED.

TO MAKE THIS ALL WORK CERTAIN TIMING CONSTRAINTS MUST BE PLACED ON THE MEMORIES. THEY ARE:

1. ONE WORD CORE READS MUST TAKE $\geq$ 375 NANO SECONDS FROM THE TIME "SBUS START" IS SENT FROM THE PROCESSOR UNTIL THE DATA ARRIVES AT THE PROCESSOR.

2. SBUS ERROR MUST ARRIVE AT THE MBOX $\leq$ 500 NANO SECONDS AFTER THE LAST SBUS ACKN ON WRITES IF THE PROCESSOR IS TO BE ABLE TO ASSOCIATE THE ERROR WITH THE PROCESS THAT CAUSED THE ERROR.

3. SBUS ERROR MUST ARRIVE AT THE MBOX $\leq$ 500 NANO SECONDS AFTER THE LAST WORD OF DATA ON READS IF THE PROCESSOR IS TO BE ABLE TO ASSOCIATE THE ERROR WITH THE PROCESS THAT CAUSED THE ERROR.

4. ANY ERRORS THAT WILL TAKE LONGER THAN 2. OR 3. FALL INTO THE "NON ASSOCIATABLE ERROR" CLASS.

[END OF SBUSDP.SPC]

1080,2040,2060 ENGINEERING FUNCTIONAL SPEC CHAP 3.7

TO: KL10 LIST

TITLE: SBUS DIAGNOSTIC CYCLE

STATUS: THIS CHAPTER IS THE SAME AS THE SUPERCEDED MEMO. IT WILL BE REVIEWED IN MARCH.

FILE: [EFS]CH3S07.SPC

PDM #: 200-200-035-00

DATE: 5 MAR 74

SUPERSEDED MEMOS: SBUS DIAGNOSTIC CYCLE SPEC, P. SULLIVAN 24 OCT

ENGINEER: P. SULLIVAN

APPROVED:

EDITOR: T. HASTINGS

TYPIST: L. NOWELL

REVIEWED:

ABSTRACT

THERE ARE NO SWITCHES ON THE INTERNAL MEMORIES. THE SBUS DIAGNOSTIC CYCLE, IS USED TO SET ADDRESS BOUNDARIES, SET INTERLEAVING, SET MARGINS, AND REPORT ERROR INFORMATION.

REVISION HISTORY

REV	DESCRIPTION	CHG NO	ORIG	DATE	APPD BY	DATE

1. INTRODUCTION

THE SBUS DIAGNOSTIC CYCLE IS USED FOR COMMUNICATION BETWEEN THE MBOX AND THE INTERNAL MEMORY CONTROLLERS. AS THERE ARE NO SWITCHES ASSOCIATED WITH THE INTERNAL MEMORY CONTROLLERS, THINGS SUCH AS ADDRESS BOUNDARIES, INTERLEAVE MODE, MARGINS, ERROR REPORTING, ETC., ARE CONTROLLED USING THE DIAGNOSTIC CYCLE.

THE ORGANIZATION OF THE SBUS FOR EACH CABINET. THERE ARE A MAXIMUM OF TWO MEMORY CONTROLLERS PER CABINET AND EACH CONTROLLER MAY HAVE 2 TO 4 STORAGE MODULES. STORAGE MODULES ARE EITHER 16K X 37 OR 32K X 37 (MA20-E OR MB20-E). IT IS NOT ALLOWED TO MIX MA20 & MB20 STORAGE MODULES ON THE SAME CONTROLLER. EACH MEMORY CONTROLLER HAS A UNIQUE HARD WIRED ADDRESS, SHOWN IN FIGURE 1, FOR DIAGNOSTIC CYCLE.

A DIAGNOSTIC CYCLE IS EXECUTED BY THE MBOX ASSERTING THE SBUS DIAG. SIGNAL AND USEING THE 36 SBUS DATA LINES FOR TRANSFERRING INFORMATION.

THERE WILL BE A PDP-10 MACRO INSTRUCTION WHICH TAKES A 36 BIT INPUT ARGUMENT AND RETURNS A 36 BIT OUTPUT ARGUMENT. THE MICROCODE FOR THIS INSTRUCTION PUTS THE 36 BIT INPUT ARGUMENT IN THE AR AND TELLS THE MBOX TO EXECUTE AN SBUS DIAGNOSTIC CYCLE. WHEN THE MBOX TURNS ON ITS MBOX RESPONSE SIGNAL THE MICROCODE CLOCKS THE CONTENTS OF THE CACHE DATA LINES INTO THE AR AND RETURNS IT AS THE OUTPUT ARGUMENT.

A TIMING DIAGRM FOR A DIAGNOSTIC CYCLE IS SHOWN IN FIGURE 2. DIAGNOSTIC CYCLE ALWAYS STARTS ON THE CLOCK DERIVED FROM PHASE A OF THE SBUS CLOCK AND TAKES 4 PHASE A CLOCK INTERVALS WHICH IS NORMALLY 500NS. THE DIAGNOSTIC CYCLE IS DIVIDED INTO TWO EQUAL PARTS: IN THE FIRST 250NS THE MBOX SENDS OUT A CONTROLLER ADDRESS IN BITS 0 - 4, A FUNCTION FIELD IN BITS 31 - 35, AND INFORMATION IN BITS 5 - 30 SPECIFIED BY THE FUNCTION CODE; IN THE LAST 250NS THE MEMORY CONTROLLER SENDS INFORMATION BACK TO THE MBOX USING BITS 00 - 35 AS DEFINED BY THE FUNCTION CODE. THE SELECTED MEMORY CONTROLLER CLOCKS THE SBUS DATA LINES ON THE 3RD PHASE A CLOCK AND THE MBOX CLOCKS THE SBUS LINES ON THE 5TH PHASE A CLOCK OF A DIAGNOTIC CYCLE AS SHOWN IN FIGURE 2.

TWO FUNCTION CODES, 0 & 1, HAVE BEEN DEFINED AND ARE DESCRIBED IN THE FOLLOWING PAGES. EACH FUNCTION CODE IS DEFINED IN TERMS OF THE INFORMATION SENT BY THE MBOX IN THE FIRST HALF CALLED "TO MEMORY" AND THE INFORMATION SENT BY THE MEMORY CONTROLLER IN THE 1ND HALF CALLED "FROM MEMORY". UNLESS NOTED OTHERWISE THE MA20, MB20 AND DMA20 IMPLEMENT ALL THE FEATURES LISTED UNDER FUNCTION 0 AND 1. ALSO, BITS SET OR CLEARED IN THE FIRST HALF OF A DIAGNOSTIC CYCLE WHEN READ OUT IN THE 2ND HALF OF THE SAME DIAGNOSTIC CYCLE WILL CONTAIN THE NEW DATA. AS THE SUBJECT OF INTERLEAVING IS

SOMEWHAT COMPLEX THERE IS A SEPARATE SECTION DEVOTED TO IT.

2. FUNCTION 0

2.1 TO MEMORY

BITS 0 - 4 MEMORY CONTROLLER ADDRESS

ADDRESSES 0 - 3 ARE ASIGNED TO THE INTERNAL MEMORY CONTROLLERS AS SHOWN IN FIGURE 1. ADDRESS 4 IS ASSIGNED TO THE DMA20 AND ADDRESSES 5 - 37 ARE RESERVED FOR EXPANSION.

BIT 5 CLEAR BITS 0 - 5.

THIS CAUSES THE FLIP FLOPS LISTED IN BITS 0 - 5 OF FUNCTION 0 - FROM MEMORY TO BE CLEARED.

BITS 6, 7 SET INTERLEAVE MODE.

0 = SET CONTROLLER TO OFF LINE STATE: DMA20 ONLY.
1 = NO INTERLEAVE (1 WORD/MEMORY CYCLE) 2 = 2 WAY INTERLEAVE (1 - 2 WORDS/MEMORY CYCLE) 3 = 4 WAY INTERLEAVE (1 - 4 WORDS/MEMORY CYCLE)

INTERLEAVE MODE BASICALLY CONTROLS THE NUMBER OF SIMULTANEOUS MEMORY CYCLES THAT MAY TAKE PLACE AS INDICATED ABOVE.

BITS 8 - 11 SBUS RQ0 - RQ3 ENABLE
8 - SBUS RQ0 ENABLE
9 - SBUS RQ1 ENABLE
10 - SBUS RQ2 ENABLE
11 - SBUS RQ3 EANBLE
IT IS THE PROGRAMMERS RESPONSIBILITY TO LOAD BITS 8 - 11 CONSISTENT WITH BITS 6 AND 7.

BITS 8 - 11 ARE USED IN CONJUCTION WITH THE INTERLEAVE MODE AS FOLLOWS:

IF A CONTROLLER IS SET IN INTERLEAVE 1 MODE THEN RQ0 THRU RQ3 SHOULD ALL BE ENABLED IN THE CONTROLLER; IF A CONTROLLER IS SET TO INTERLEAVE 2 OR 3 MODE THEN RQ0 & RQ2 SHOULD BE ENABLED IN THE EVEN CONTROLLER AND RQ1 & RQ3 ENABLED IN THE ODD CONTROLLER.

ONE POSSIBLE WAY OF MAKING AN INTERNAL CONTROLLER APPEAR TO BE OFF LINE IT TO NOT ENABLE ANY OF THE SBUS RQ - RQ3 ENABLES.

BITS 8 - 11 ARE IGNORED BY THE DMA20.

BIT 12 — THIS BIT MUST BE A "ONE" TO ENABLE LOADING OF BITS 6 - 11, OTHERWISE BITS 6 - 11 ARE IGNORED.

BITS 13 - 30 — RECERVED FOR FUTRE USE BY HAREWARE.

BITS 31 - 35 — FUNCTION CODE 00 FOR FUNCTION 0, 8 FUNCTION 02 - 37 ARE RESERVED FOR FUTURE USE HARDWARE, 8

MA20 AND MB20 CONTROLLERS ONLY DECODE BIT 35 OF THIS FIELD.

2.2 FROM MEMORY

BITS 0 - 5 — 0 - 1 FUTURE
2 INCOMPLETE CYCLE
3 READ PAR ERR
4 WRITE PAR ERR
5 ADR PAR ERR

THESE ARE ERROR REPORTING FLAGS IN THE MEMORY CONTROLLERS, WHICH ARE CLEARED WHEN BIT 5 IS ON A "ONE" IN THE 1ST HALF OF FUNCTION 0. THE DMA20 DOES NOT IMPLEMENT THE INCOMPLETE CYCLE BIT AND THE MA20 AND MB20 CONTROLLERS DO NOT IMPLEMENT THE READ OR WRITE PAR ERROR BITS.

BITS 6 - 7 THESE BITS REPORT THE INTERLEAVE MODE OF THE CONTROLLER WITH THE 0 COMBINATION USED TO INDICATE AN OFF LINE OR NON EXISTENT MEMORY CONTROLER.

BITS 8 - 35 THESE BITS CONTAIN THE ADDRESS INFORMATION OF THE MOST RECENT CYCLE OR OF THE 1ST PARITY ERROR. BITS 8 - 11 CONTAIN THE SBUS RQ0 - RQ3, BITS 12 AND 13 THE RD AND WR RQ AND BITS 14 - 35 THE SBUS ADR BITS. THIS FEATURE IS NOT IMPLEMENTED IN THE MA20 OR MB20.

3. FUNCTION 1

3.1 TO MEMORY

BITS 0 - 4 — MEMORY CONTROLLER ADDRESS (SEE FUNCTION 0)

BITS 5 - 11 — EXPANDSION

BIT 12 LOOP AROUND MODE. LOOP AROUND MODE IS A DIAGNOSTIC FEATURE USED FOR CHECKING DATA TRANSFERS BETWEEN THE MBOX AND THE MEMORY DATA REGISTER IN EACH MEMORY STORAGE MODULE WITHOUT GOING THRU THE CORES. A LOOP AROUND CYCLE CONSISTS OF THE FOLLOWING STEPS:

1. SET LOOP AROUND MODE USING FUNCTION 1 DIAGNOSTIC CYCLE.
2. DO AN ORDINARY WRITE CYCLE TO ANY WORD IN THE SUBJECT CONTROLLER.
3. DO AN ORDINARY READ CYCKE TO THE SAME WORD IN THE SAME CONTROLLER IN (2).

LOOP AROUND MODE IS AUTOMATICALLY CLEARED BY THE HARDWARE AFTER STIP (3) ABOVE.

BIT 13 FUTURE

BITS 14 - 21 MEMORY CONTROLLER ADDRESS BOUNDARIES

THESE BITS ARE USED TO SET UP PART OF THE ADDRESS BOUNDARY FOR EACH MEMORY CONTROLLER. BITS 14 - 17 CORRESPOND TO SBUS ADR 14 - 17 AND ARE SET UP THE SAME WAY AS ADDRESS SWITCHES ON CURRENT MEMORIES. A MATCH BETWEEN BITS 14 - 17 AND BITS 14 - 17 OF THE SBUS ADDRESS LINES IS ONE OF THE CONDITIONS NECESSARY TO START A MEMORY CYCLE. NOT IMPLEMENTED IN DMA20.

BITS 18 - 21 CORRESPOND TO SBUS ADR 18 - 21 AND DETERMINE THE LOWER ADDRESS BOUNDARY FOR THESE BITS. MA20 CONTROLLER USE BITS 18 - 21 SINCE THE STORAGE MODULES ARE 16K WORDS WHEREAS MB20 CONTROLLERS IGNORE BIT 21 SINCE THE STORAGE MODULE SIZE IS 32K WORDS. BITS 18 - 2L ARE SET UP TO ADDRESS THE LOWEST ADDRESS STORAGE MODULE WITHIN A MEMORY CONTROLLER CONSTRAINED BY THE HARD WIRED BITS SHOWN IN FIGURES 3 AND 4. A SECOND CONDITION FOR STARTING A MEMORY CYCLE IS THAT SBUS ADDRESS BITS 18 - 21 ARE = THE LOWER BOUNDARY REGISTER.

NOT IMPLEMENTED IN DMA20.

BITS 22 - 25 MEMORY CONTROLLER ADDRESS BOUNDARY

THESE BITS IN CONJUNCTION WITH 14 - 21 DEFINE THE ADDRESS BOUNDARIES FOR EACH MEMORY CONTROLLER. BITS 22 - 25

CORRESPOND TO SBUS ADR 18 - 21 AND DETERMINE THE UPPER ADDRESS BOUNDARY FOR THESE BITS. AS ABOVE, THE MA20 USES BITS 22 - 25 AND THE MB20 IGNORES BIT 25. BITS 22 - 25 ARE SET TO ADDRESS THE HIGHEST ADDRESS STORAGE MODULE WITHIN A MEMORY CONTROLLER CONSTRAINED BY THE HARD WIRED BITS SHOWN IN FIGURES 3 AND 4. A THIRD CONDITION FOR STARTING A MEMORY CYCLE IS THAT SBUS ADR BITS 18 - 21 ARE = THE UPPER BOUNDARY REGISTER.

NOT IMPLEMENTED IN DMA20.

THEREFORE IN ORDER TO START A MEMORY CYCLE IN AN INTERNAL MEMORY, IT IS NECESSARY TO MEET THE THREE CONDITIONS SPECIFIED ABOVE AND IN ADDITION SBUS RQN MUST MATCH SBUS RQN ENABLE SET UP IN BITS 8 - 11 OF FUNCTION 0-TO MEMORY. ANOTHER WAY OF MAKING A MEMORY CONTROLLER APPEAR TO BE OFF LINE IS TO SET THE UPPER BOUNDARY REGISTER TO LOWER VALUE THAN THE LOWER BOUNDARY REGISTER OR VICE VERSA.

BIT 26 THIS BIT MUST BE A "ONE" TO ENABLE LOADING OF BITS 14 - 25, OTHEREWISE BITS 14 - 25 ARE IGNORED. NOT IMPLEMENTED IN DMA20.

BITS 27 - 30 TURN ON THE MARGIN AS SPECIFIED BELOW.

27	28	29	30	
0	0	0	0	NO OP
0	0	0	1	CLEAR ALL MARGIN
0	0	1	X	CURRENT MARGIN
0	1	0	X	STROBE MARGIN
1	0	0	X	THRESHOLD MARGIN

THE "X" IN BIT 30 POSITION ABOVE MEANS THAT BIT 30 SELECTS LOW MARGIN WHEN "ZERO" AND HIGH MARGIN WHEN A "ONE". ONLY ONE MARGIN AT A TIME MAY BE TURNED ON. NOT IMPLEMENTED IN DMA20.

BITS 31 - 35 FUNCTION CODE = 01 FOR FUNCTION 1.
8

FUNCTION 02 - 37 ARE RESERVED FOR FUTURE 8 EXPANSION. MA20 AND MB20 CONTROLLERS ONLY DECODE BIT 35 OF THIS FIELD.

3,2 FROM MEMORY

BITS 0 - 7 THESE BITS INDICATE THE NUMBER OF STORAGE MODULES PER CONTROLLER. EACH BIT ON A "ONE" INDICATES THE PRESENCE OF A STORAGE MODULE. IF ALL BITS ARE 0 MEANS DON'T KNOW SIZE. DMA20 SENDS BACK 0.

MA20 & MB20 INDICATES STORAGE MODULES AS FOLLOWS,

0	1	2	3	4	5	6	7	
0	0	0	0	0	0	0	1	SM0
0	0	0	0	0	0	1	0	SM1
0	0	0	0	0	1	0	0	SM2
0	0	0	0	1	0	0	0	SM3

BITS 8 - 11 MEMORY TYPE
- 0 = CUSTOMER
- 1 = MA20
- 2 = DMA20
- 3 = MB20
- 4 - 17 EXPANSION (8)

BIT 12 LOOP AROUND SET

NOTE: THE FOLLOWING BITS (13 - 35) ARE NOT IMPLEMENTED IN DMA20,

BIT 13 RESERVED FOR FUTURE

BIT 14 - 21 ADDRESS BOUNDARY SET UP IN 1ST HALF OF FINCTION 1

BITS 22 - 25 ADDRESS BOUNDARY SET UP IN 1ST HALF OF FUNCTION 1

BITS 26 - 29 EXPANSION

BIT 30 MARGINS SELECTED

BIT 31 EXPANSION

BITS 32 - 35 INDICATES WHICH WORD RQS ARE ENABLED
- 32 - SBUS RQ0 ENABLE
- 33 - SBUS RQ1 ENABLE
- 34 - SBUS RQ2 ENABLE
- 35 - SBUS RQ3 ENABLE

4, INTERLEAVING

INTERLEAVING FOR CURRENT EXTERNAL MEMORIES ON THE KL10 IS HANDLED EXACTLY THE SAME WAY IT IS FOR KA/KI MEMORIES, THAT IS, THRU THE 2 INTERLEAVE SWITCHES ON THE PORT ADDRESS SWITCH PANEL, WHICH INTERCHANGE MA BITS 34 AND 35 WITH THE TWO LOW ORDER UNIT SELECT BITS,

DMA20 IN INTERLEAVE 1 MODE: ADDRESS & INTERLEAVE SWITCHES MAY BE SET IN ANY CONFIGURATION,

INTERLEAVE 2 MODE: ALL MEMORIES MUST BE SET FOR 2-WAY OR 4-WAY INTERLEAVING; ALL MEMORIES ON KBUS0 AND KBUS2 MUST BE SET TO RESPOND TO EVEN, ADDRESSES; ALL MEMORIES ON KBUS1

AND KBUS3 MUST RESPOND TO ODD ADDRESSES.

INTERLEAVE 3 MODE: ALL MEMORIES MUST BE SET FOR 4-WAY INTERLEAVING; ALL MEMORIES ON KBUSN MUST RESPOND TO ADR 34, 35 = N.

SELECTION OF ONE OF THE THREE INTERLEAVING MODES FOR INTERNAL MEMORY IS DONE THRU THE DIAGNOSTIC CYCLE AS PREVIOUSLY DESCRIBED. THE STORAGE MODULES, ASSOCIATED WITH EACH MEMORY CONTROLLER, ARE ASSIGNED TO A FIXED PORTION OF THE CONTROLLER ADDRESS SPACE WHICH IS DEPENDENT ON THE INTERLEAVE MODE, AS SHOWN IN FIGURE 3 FOR MA20 AND FIGURE 4 FOR MB20. A MEMORY CONTROLLER CAN OCCUPY THOSE REGIONS OF THE ADDRESS SPACE SASSIGNED BY BITS 14 - 25 OF FUNCTION 1.

THE STORAGE MODULES OCCUPY CONTIGUOUS 16K BLOCKS WITHIN A CONTROLLER'S ADDRESS SPACE.

IN INTERLEAVE 2 MODE, STORAGE MODULES 0 - 3 ARE DECODED BY ADDRESS BITS 19 & 20 WHILE THE DISTINCTION BETWEEN EVEN AND ODD CONTROLLER IS MADE BY SBUS RQ0 OR RQ2 AND SBUS RQ1 OR RQ3 RESPECTIVELY. THEREFORE, IN INTERLEAVE 2 MODE IT IS NECESSARY TO HAVE STORAGE MODULES IN EACH CONTROLLER THAT DECODE BITS 19 & 20 THE SAME. IN THIS MODE, A CONTROLLER MAY CYCLE ONLY ONE STORAGE MODULE/MEMORY CYCLE HOWEVER 2 CONTROLLERS MAY CYCLE IN PARALLEL WITH A RESULTANT OF TWO WORDS/ MEMORY CYCLE.

IN INTERLEAVE 3 MODE, THE STORAGE MODULE DECODING IS DONE BY SBUS ADR BIT 19 & SBUS RQ N AS SHOWN. THIS MODE REQUIRES THAT EACH CONTROLLER HAVE THE SAME NUMBER OF STORAGE MODULES WHICH MAY BE; SM0 AND SM1, OR SM2 AND SM3, OR SM0 - SM3. IN THIS MODE, A CONTROLLER MAY CYCLE 1 OR 2 STORAGE MODULES IN PARALLEL AND TWO CONTROLLERS MAY CYCLE IN PARALLEL WITH A RESULTANT OF 4 WORDS/MEMORY CYCLE.

A METHOD FOR MAKING A STORAGE MODULE APPEAR TO BE OFF LINE IS THE FOLLOWING: SET THE LOWER ADDRESS BOUNDARY REGISTER TO MATCH THE ADDRESS OF THE NEXT STORAGE MDULE AND SET THE UPPER ADDRESS BOUNDARY REGISTER TO MATCH THE ADDRESS OF THE HIGHEST ADDRESS STORAGE MODULE. FOR EXAMPLE, REFER TO FIGURE 3 AND ASSUME THAT IT IS DESIRED TO MAKE SM1 APPEAR TO BE OFF LINE: THEN ASSUMING WE ARE WORKING IN THE LOWEST REGION OF THE ADDRESS SPACE, THE LOWER ADDRESS BOUNDARY REGISTER WOULD BE SET TO 0010 AND THE UPPER ADDRESS BOUNDARY REGISTER TO 0100. USING THIS SCHEME ANY NUMBER OF CONTIGUOUS STORAGE MODULES MAY BE DESELECTED IN INTERLEAVE 1 MODE; IN INTERLEAVE 2 MODE IT IS POSSIBLE TO DESELECT ANY NUMBER OF PAIRED CONTIGUOUS STORAGE MODULES IN A PAIR OF INTERLEAVED MEMORY CONTROLLERS (FOR EXAMPLE, SM2 IN MA20 CONTROLLERS 0 & 1); IN INTERLEAVE 3 MODE IT IS POSSIBLE ONLY TO DESELECT ONE OR BOTH OF THE GROUPS OF 4 WAY INTERLEAVED STORAGE MODULE IN A PAIR OF INTERLEAVED MEMORY CONTROLLERS (FOR EXAMPLE, SM0 AND SM1 IN CONTROLLER 0 AND 1

OR SM2 AND SM3 IN CONTROLLERS 0 AND 1 OR SM0 - SM3 IN CONTROLLERS 0 AND 1). IT IS ALSO POSSIBLE TO DESELECT A STORAGE MODULE BY INTRODUCING A HOLE(S) INTO THE PAGE TABLES.

[END OF CH3S07.SPC]

1080,2040,2060 ENGINEERING FUNCTIONAL SPEC - CHAP 4.1

TO: KL10 LIST

TITLE: MASSBUS CONTROLLER (RH20) AND CHANNELS (MBOX) - REV 2

STATUS: THIS CHAPTER HAS THE LATEST CHANGES IN RH20 CONI STATUS NAMES

FILE: [EFS]CH4S01.SPC

PDM #: 200-200-015-02

DATE: 23 JUN 75

SUPERSEDED SPEC: MASSBUS CONTROLLER HARDWARE SPECIFICATION (RH20), V. KU, 8 JUNE 73

ENGINEER: V. KU, L. CARPENITO

APPROVED:

EDITOR: T. HASTINGS

TYPIST: L. NOWELL

REVIEWED:

ABSTRACT

THE MASSBUS CONTROLLER (RH20) IS AN INTERFACE BETWEEN THE KL10 CBUS AND UP TO 8 HIGH SPEED MAGTAPE MASTER DRIVES, DRUMS, AND/OR DISKS. EACH DEVICE MUST MEET THE "MASSBUS INTERFACE STANDARD". THIS CHAPTER ALSO DESCRIBES THE BUILT-IN CHANNELS, WHICH ARE LOCATED IN THE MBOX. EACH CONTROLLER HAS ITS OWN CHANNEL.

REVISION HISTORY

REV	DESCRIPTION	CHG NO	ORIG	DATE	APPD BY	DATE

1. GENERAL

THE MASSBUS CONTROLLER (RH20) IS AN INTERFACE BETWEEN THE KL10 EBUS, CBUS AND UP TO 8 DEVICES PROVIDED THAT EACH DEVICE (DRUM, DISK, MAGTAPE, ETC.) MEET THE "MASSBUS INTERFACE STANDARD", THE CONTROLLER WILL HAVE SUFFICIENT HARDWARE FEATURES TO INTERFACE TO SINGLE-PORTED DRIVES AS WELL AS DUAL-PORTED DRIVES AND WILL ALLOW TIMESHARED SWAPPING AND PAGING SOFTWARE TO OPERATE THE SYSTEM EFFICIENTLY WITH MINIMUM LATENCY. UP TO EIGHT CONTROLLERS CAN BE CONNECTED TO A KL10 SYSTEM AND ONE OR MORE CONTROLLERS CAN OPERATE (READING, WRITING, ETC.) SIMULTANEOUSLY. ONLY ONE DEVICE PER CONTROLLER CAN TRANSFER DATA AT ANY ONE TIME BUT NON-DATA TRANSFER COMMANDS (SEEK, REWIND, ETC.) CAN OVERLAP AND CAN BE ISSUED TO ANY DRIVE AT ANY TIME AND CAN INTERRUPT ON COMPLETION EVEN DURING A TRANSFER. FURTHERMORE THE SOFTWARE CAN KEEP ONE COMMAND AHEAD SO THAT ADJACENT SECTORS CAN BE ACCESSED WITH NO LATENCY EVEN THOUGH SWITCHING TRACKS OR DIRECTION OF TRANSFER.

EACH MASSBUS CONTROLLER IS CONNECTED TO THE MEMORY SYSTEM WITH ITS OWN BUILT-IN CHANNEL. THE CHANNEL PROVIDES SCATTER-GATHER CAPABILITY IN MEMORY AND READ BACKWARDS (FOR MAGNETIC TAPE). THE CHANNELS PASS PARITY TO AND FROM THE 18 BIT + PARITY MASSBUS AND THE 36 BIT + PARITY MEMORY. PARITY IS CHECKED ON BOTH SIDES OF THE CHANNEL.

FEATURES WILL BE INCLUDED TO ALLOW PROGRAM DIAGNOSIS OF THE CONTROLLER WITH OR WITHOUT ANY DEVICE ATTACHED TO THE CONTROLLER. NEITHER INDICATORS NOR OFF-LINE TEST MODE WILL BE PROVIDED.

FROM THE PROGRAMMER'S POINT OF VIEW THERE ARE TWO SETS OF PROGRAMMABLE REGISTERS PROVIDED BY THE RH20, INTERNAL AND EXTERNAL. INTERNAL REGISTERS ARE CONTROLLER REGISTERS AND CONTAIN INFORMATION ASSOCIATED WITH THE CURRENT OR NEXT TRANSFER OR WITH INFORMATION FOR ALL OF THE DRIVES. THE EXTERNAL REGISTERS ARE DRIVE REGISTERS AND CONTAIN INFORMATION ASSOCIATED WITH A PARTICULAR DRIVE. THE PROGRAM MUST SPECIFY WHICH DRIVE IT IS INTERESTED IN BEFORE REFERRING TO AN EXTERNAL REGISTER. THUS THE RH20 PROVIDES A WINDOW TO THE EXTERNAL REGISTERS IMPLEMENTED IN THE DRIVE.

FIG. 1 SHOWS THE KL10 MASS STORAGE SYSTEM BLOCK DIAGRAM.

TWO RH20S CAN BE ATTACHED TO THE SAME MASSBUS, ALTHOUGH ONLY ONE CAN BE ENABLED AT A TIME. THE SOFTWARE CAN DYNAMICALLY DISABLE ONE AND ENABLE THE OTHER TO PROVIDE FOR A FAILSOFT RECOVERY FROM A FAILING CONTROLLER IN A SINGLE OR MULTI-PROCESSING SYSTEM.

2. REFERENCE

1. "MASSBUS INTERFACE STANDARD" AS REVISED

2. "DISK, DRUM, AND MAGTAPE I/O ON THE KL10" (PAUL GUGLIELMI, NOVEMBER 2, 1972) AS REVISED

3. "EBUS SPECIFICATION" (VIC KU, APRIL 17, 1973) AS REVISED

4. "CBUS FUNCTIONAL SPECIFICATION" (VIC KU, APRIL 25, 1973) AS REVISED

5. "ADVANTAGES OF A SECONDARY INSTRUCTION REGISTER IN THE KL10 MASSBUS CONTROLLER" (TOM HASTINGS, NOVEMBER 16, 1972)

3, CONTROLLER AND DEVICE SPECIFICATIONS

PORTIONS OF THE FOLLOWING PARAGRAPHS ARE EXTRACTED FROM MEMOS LISTED IN 2,0,

3,1 DEVICE CODE

EACH CONTROLLER IS ASSIGNED ONE UNIQUE DEVICE CODE, A TOTAL OF EIGHT DEVICE CODES HAVE BEEN ASSIGNED SINCE UP TO EIGHT CONTROLLERS CAN EXIST IN A KL10 SYSTEM, THE DEVICE CODE ASSIGNMENTS ARE:

CONTROLLER	DEVICE CODE	SYMBOL	CONTROLLER'S PHYSICAL NUMBER	EPT LOCATION
1ST	540	MBC0	0	0
2ND	544	MBC1	1	4
3RD	550	MBC2	2	10
4TH	554	MBC3	3	14
5TH	560	MBC4	4	20
6TH	564	MBC5	5	24
7TH	570	MBC6	6	30
8TH	574	MBC7	7	34

3,2 PHYSICAL CONTROLLER NUMBER

EACH CONTROLLER IS ASSIGNED A PHYSICAL NUMBER ACCORDING TO THE PHYSICAL SLOTS INTO WHICH THE CONTROLLER MODULES RESIDE, BOTH THE DEVICE CODE AND THE PHYSICAL NUMBER OF THE CONTROLLER WILL BE HARD WIRED ON THE KL10 BACK PANEL,

THE PHYSICAL NUMBER OF A CONTROLLER IS USED BY THE EBOX TO IDENTIFY THE SOURCE OF PRIORITY INTERRUPTS,

3,3 COMMAND REGISTER

EACH CONTROLLER WILL HAVE TWO COMMAND REGISTERS (PRIMARY AND SECONDARY), A COMMAND IN THE PRIMARY COMMAND REGISTER SHALL BE EXECUTED IMMEDIATELY PROVIDED NO TRANSFER ERROR CONDITION (DEFINED IN 6,3) IS DETECTED IN THE CONTROLLER, THE SECONDARY COMMAND REGISTER SERVES AS A COMMAND LOOKAHEAD,

THE COMMAND IN THE SECONDARY COMMAND REGISTER WILL BE EXECUTED AS SOON AS THE COMMAND IN THE PRIMARY COMMAND REGISTER IS

TERMINATED AND NO TRANSFER ERROR IS DETECTED IN THE CONTROLLER. CAUTION: ONLY READ/ WRITE COMMANDS SHALL BE LOADED INTO THE COMMAND REGISTERS. THE PROGRAMMER SHALL LOAD NON READ/WRITE (SEEK REWIND ETC.) COMMAND DIRECTLY INTO THE DRIVE'S "CONTROL REGISTER". THE SECONDARY COMMAND REGISTER ALLOWS THE SOFTWARE TO SPECIFY THE NEXT COMMAND TO BE DONE BEFORE THE CONTROLLER HAS FINISHED THE CURRENT ONE. THIS MEANS THAT THE CONTROLLER CAN START THE NEXT COMMAND IMMEDIATELY AFTER THE CURRENT COMMAND IS DONE INSTEAD OF WAITING FOR A SOFTWARE INTERRUPT ROUTINE TO SUPPLY THE NEXT COMMAND. SUCH A WAIT WOULD CAUSE THE NEXT SECTOR TO BE MISSED. WITHOUT THIS, THE SOFTWARE CAN ONLY TRANSFER EVERY OTHER BLOCK OR PAGE FOR DIFFERENT USERS.

3.4 INTERRUPT

THE CONTROLLER WILL INTERRUPT THE PROCESSOR IMMEDIATELY WHEN:

1. A R/W COMMAND IS TERMINATED (WITH OR WITHOUT TRANSFER ERROR)

2. AN "ATTENTION" SIGNAL (CAUSED BY SEEK COMPLETE, ETC.) IS DETECTED ON THE MASSBUS BY THE CONTROLLER PROVIDED THAT "ATTENTION INTERRUPT ENABLE (SEE 6.4) IS TRUE.

3. A "REGISTER ACCESS ERROR" IS DETECTED IN THE CONTROLLER WHEN A DRIVE REGISTER IS LOADED OR READ (SEE 6.1.0)

3.5 EBUS & CBUS

THE CONTROLLER SHALL INTERFACE TO ONE EBUS AND ONE CBUS. ALL DATA TRANSFER TAKES PLACE OVER THE CBUS AND THE COMMANDS ARE LOADED VIA THE EBUS. REGISTER READ/WRITE AND STATUS READ/WRITE ARE PERFORMED VIA THE EBUS.

3.6 DEVICE

THE CONTROLLER SHALL BE DESIGNED TO MEET THE "MASSBUS INTERFACE STANDARD" AND SHALL CONTROL DEVICES (FIXED-HEAD DISK, DRUM, MOVING- HEAD DISK, MAGTAPE DRIVE, ETC.) THAT MEET THE "MASSBUS INTERFACE STANDARD". EACH DEVICE SHALL IMPLEMENT A SUBSET OF THE STANDARD. DETAILED SPECIFICATIONS OF EACH DEVICE (RS04, RP04, TU16, ETC.) WILL BE MADE AVAILABLE BY THE DEVICE DESIGNER.

4. REGISTERS ACCESSIBLE BY SOFTWARE

4.1 REGISTER SUMMARY

THERE ARE TWO GROUPS OF REGISTERS ACCESSIBLE BY THE PROGRAMMER:

1. REGISTERS IN A DRIVE (EXTERNAL) AND

2. REGISTERS IN A CONTROLLER (INTERNAL)

TABLE 1 LISTED ALL THE REGISTERS IN THE CONTROLLER AND TABLE 2 LISTED ALL THE REGISTERS IN THE DRIVE (NOT ALL DRIVES WILL NECESSARILY IMPLEMENT ALL REGISTERS). IN THESE TABLES, "READ" MEANS TO TRANSFER A REGISTER TO THE EBOX; "WRITE" MEANS TO LOAD A REGISTER FROM THE EBOX.

THE PRIMARY (SECONDARY) COMMAND REGISTER IS MADE UP OF TWO INTERNAL REGISTERS:

1. PRIMARY (SECONDARY) BLOCK ADDRESS REGISTER AND

2. PRIMARY (SECONDARY) TRANSFER CONTROL REGISTER

TABLE 1 INTERNAL REGISTER SUMMARY

REG.# (OCTAL)	READ/ WRITE	REGISTER NAME	BRIEF DESCRIPTION
70	R/W	SECONDARY BLOCK ADDRESS	CONTAINS THE STARTING BLOCK ADDRESS (DISKS) OR NUMBER OF FRAMES (MAGTAPE) OF THE BACKUP COMMAND.
71	R/W	SECONDARY TRANSFER CONTROL	CONTAINS THE BLOCK COUNT AND THE FUNCTION CODE OF THE BACKUP COMMAND.
72	READ ONLY	PRIMARY BLOCK ADDRESS	CONTAINS THE STARTING BLOCK ADDRESS (DISKS) OR NUMBER OF FRAMES (MAGTAPE) OF THE PRIMARY COMMAND.
73	READ ONLY	PRIMARY TRANSFER CONTROL	CONTAINS THE INITIAL BLOCK COUNT AND FUNCTION CODE OF THE PRIMARY COMMAND.
74	R/W	INTERRUPT VECTOR INDEX REGISTER	CONTAINS THE INTERRUPT VECTOR USED DURING PRIORITY INTERRUPT OPERATION.
75	READ ONLY	READ REGISTER	CONTAINS DATA RECEIVED FROM THE MASSBUS DATA BUS. USED TO VERIFY DATA DURING A DIAGNOSTIC WRITE OPERATION
76	WRITE ONLY	WRITE REGISTER	CONTAINS DATA TO BE SENT OUT TO THE MASSBUS DATA BUS DURING A DIAGNOSTIC READ OPERATION.
77	WRITE ONLY	DIAGNOSTIC CONTROL	CONTAINS DIAGNOSTIC BITS TO SIMULATE MASSBUS

REGISTER OPERATIONS.

TABLE 2 EXTERNAL REGISTER SUMMARY

REG.#	READ/ WRITE	REGISTER NAME	BRIEF DESCRIPTION
00	R/W	CONTROL	CONTAINS FUNCTION CODE AND GO BIT.
01	READ ONLY	STATUS	CONTAINS ALL NON-ERROR STATUS INFORMATION PLUS THE ERROR SUMMARY BIT.
02	R/W	ERROR 1	INDIVIDUAL ERROR INDICATIONS
03	R/W	MAINTENANCE	CONTAIN DIAGNOSTIC MAINTAINANCE FUNCTIONS.
04	R/W	ATTENTION SUMMARY	SPECIAL PSEUDO-REGISTER WHICH SHOWS THE "ATTENTION ACTIVE" STATUS OF ALL DRIVES, ONE BIT PER DRIVE. (WRITE A 1 TO RESET BITS IN DRIVE)
05	R/W	DESIRED SECTOR/ TRACK ADDRESS	CONTAINS ADDRESS OF A TRACK AND SECTOR TO BE READ OR WRITTEN.
06	READ ONLY	DRIVE TYPE	BITS TO INDICATE THE CHARACTERISTICS OF THE DRIVE.
07	READ ONLY	LOOK-AHEAD	CONTAINS THE CURRENT ROTATIONAL POSITION OF A DISK.
10	R/W	ERROR 2	CONTAINS ADDITIONAL ERROR BITS.
11	R/W	OFFSET	CONTROL OF OFFSETTING OF MOVABLE HEADS.
12	R/W	DESIRED CYLINDER ADDRESS	CONTAINS ADDRESS OF A CYLINDER TO WHICH A SEEK IS TO BE DONE ON MOVING HEAD DISK.
13	READ ONLY	CURRENT CYLINDER ADDRESS	CONTAINS THE CYLINDER ADDRESS CORRESPONDING TO THE CURRENT ARM POSITION.
14	READ ONLY	SERIAL NUMBER	CONTAINS THE LOWEST FOUR DECIMAL DIGITS OF THE SERIAL OF THE DRIVE.

15	R/W	ERROR 3	CONTAINS MORE ERROR BITS.
16	READ ONLY	ECC POSTION	CONTAINS THE POSITION OF A BURST ERROR.
17	READ ONLY	ECC PATTERN	CONTAINS THE PATTERN OF A DETECTED BURST ERROR.

4.2 DETAIL DESCRIPTION OF THE REGISTERS

FIGURES 2, 3, AND 4 SHOW THE BIT MAPS OF THE REGISTERS.

4.2.1 WRITING AN INTERNAL REGISTER

TO WRITE A WRITABLE INTERNAL REGISTER, THE PROGRAMMER USES A DATAO TO:

1. SPECIFY THE DESIRED REGISTER NUMBER (BIT 0-5 = 7X). THE REGISTER NUMBER IS STORED IN THE CONTROLLER (PREPARATION REGISTER).

2. SET THE "LOAD REGISTER" BIT (LR) (BIT 6) TO 1.

3. SPECIFY THE CONTENT OF THE REGISTER (BIT 18 THRU 35) AFTER LOADING IS DONE.

4.2.2 READING AN INTERNAL (CONTROLLER) REGISTER

TO READ A READABLE INTERNAL REGISTER, THE PROGRAMMER USES A DATAO TO;

1. SPECIFY THE DESIRED REGISTER NUMBER (BIT 0-5 = 7X). THE REGISTER NUMBER IS STORED IN THE CONTROLLER (PREPARATION REGISTER).

2. SET THE "LOAD REGISTER" BIT (BIT 6) TO 0. SUBSEQUENT DATAI INSTRUCTIONS WILL TRANSFER THE CONTENTS OF THE DESIRED REGISTER (ON DATA BITS 18 THROUGH 35) SPECIFIED BY THE CONTENTS OF THE "PREPARATION REGISTER" TO THE EBOX. BITS 0 THROUGH 5 WILL CONTAIN THE REGISTER NUMBER FOR VERIFICATION. REPEATED DATAI INSTRUCTIONS WILL READ THE SAME REGISTER PROVIDED THAT NO DATAO WITH LR = 0 OR 1 IS DONE IN BETWEEN. IN OTHER WORDS, DATAI

WILL ALWAYS READ THE REGISTER SPECIFIED BY THE LAST DATAO.

4.2.3 WRITING AN EXTERNAL (DRIVE) REGISTER

THE PROGRAMMER USES A DATAO TO:

1. SPECIFY THE DESIRED REGISTER NUMBER (BIT 0 THROUGH 5, WITH BIT 0 = 0). THE REGISTER NUMBER IS STORED IN THE CONTROLLER (PREPARATION REGISTER).

2. SET THE "LOAD REGISTER " BIT (BIT 6) TO 1.

3. SPECIFY THE "DISABLE REGISTER ACCESS ERROR STOP" BIT (BIT 9, DRAES).

4. SPECIFY THE "CONTROL BUS EVEN PARITY" BIT (BIT 18).

5. SPECIFY THE DRIVE NUMBER (BITS 15 - 17).

6. SPECIFY THE CONTENTS OF THE REGISTER (BITS 20 - 35) AFTER LOADING IS DONE.

THE CONTROLLER, ON DETECTING THE DATAO, WILL LOAD THE SPECIFIED DRIVE REGISTER VIA THE MASSBUS. IF THE "CONTROL BUS EVEN PARITY" BIT IS SET, THE CONTROLLER WILL GENERATE EVEN PARITY, OTHERWISE, ODD PARITY IS GENERATED (EVEN PARITY IS PROVIDED FOR DIAGNOSTIC PURPOSES AS USUALLY THIS BIT WILL BE 0). IF THE DRIVE DOES NOT RESPOND PROPERLY (SEE 6.10) "REGISTER ACCESS ERROR" (CONI BIT 24) WILL BE SET. "REGISTER ACCESS ERROR" WILL GENERATE AN INTERRUPT AND WILL INHIBIT THE PROGRAM FROM WRITING ANY MORE REGISTERS (INTERNAL OR EXTERNAL) UNLESS "DISABLE REGISTER ACCESS ERROR STOP" WAS SPECIFIED BEFORE THE ERROR IS MADE. IN THIS CASE NO INTERRUPT WILL BE GENERATED BY "REGISTER ACCESS ERROR" AND ANY REGISTER CAN BE ACCESSED FREELY. A DATAI WILL ALWAYS READ BACK THE REGISTER JUST LOADED AND CAN BE USED TO READ BACK THE REGISTER NUMBER AND THE DRIVE NUMBER FROM WHICH THE "REGISTER ACCESS ERROR" WAS MADE. THIS INFORMATION WILL BE USEFUL IN THE ERROR REPORTING SYSTEM.

4.2.4 READING AN EXTERNAL (DRIVE) REGISTER

TO READ A READABLE DRIVE REGISTER, THE PROGRAMMER USES A DATAO TO:

1. SPECIFY THE DESIRED REGISTER NUMBER (BIT 0 - 5, WITH BIT 0 = 0). THE REGISTER NUMBER IS STORED IN THE "PREPARATION REGISTER".

2. SET THE "LOAD REGISTER" BIT (BIT 6) TO 0

3. SPECIFY THE "DISABLE REGISTER ACCESS ERROR STOP" BIT (BIT 9, DRAES)

4. SPECIFY BIT 18 "CONTROL BUS EVEN PARITY"

5. SPECIFY THE DRIVE NUMBER (BITS 15 - 17)

SUBSEQUENT DATAI WILL TRANSFER TO THE EBOX, VIA THE EBUS,

1. THE CONTENTS OF THE DESIRED REGISTER (BIT 20 - 35)

2. THE PARITY BIT SENT IN BY THE DRIVE (BIT 19)

3. "CONTROL BUS EVEN PARITY" (BIT 18) - BIT SPECIFIED BY THE PREVIOUS DATAO

4. "DRIVE SELECT REGISTER" (BIT 15 - 17) - DRIVE NUMBER SPECIFIED BY THE PREVIOUS DATAO

5. BITS 11-14 ARE NOT USED AND RETURN ZERO'S

6. "TRA" (BIT 10) - THIS BIT WILL BE SET IF THE DRIVE RESPONDS WITH "TRANSFER" (THIS IS THE NORMAL DRIVE RESPONSE)

7. "DISABLE REGISTER ACCESS ERROR STOP" (BIT 9) - SPECIFIED BY THE PREVIOUS DATAO

8. "CONTROL BUS PARITY ERROR" (BIT 8) - THIS BIT IS SET IF THE CONTROLLER DETECTS A PARITY ERROR ON THE 16 MASSBUS CONTROL LINES. THE CONTROLLER CHECKS FOR ODD PARITY ERROR WHEN READING AN EXTERNAL REGISTER OTHER THAN THE ATTN SUMMARY REGISTER

9. BIT 7 IS NOT USED

10. BIT 6 (LR) RETURNS ZERO.

11. "REGISTER SELECT" (BIT 0 - 5) SPECIFIED BY THE PREVIOUS DATAO

THE "REGISTER ACCESS ERROR" BIT (CONI BIT 24) WILL BE SET IF "CONTROL BUS PARITY ERROR" IS SET OR IF "TRA" IS NOT RECEIVED (IF A DRIVE IS NOT PRESENT A 2US TIMEOUT SETS RAE). INTERRUPT WILL BE GENERATED AND LOADING OF ANY MORE REGISTERS WILL BE INHIBITED UNLESS "DISABLE REGISTER ACCESS ERROR STOP" WAS SPECIFIED IN THE "PREPARATION REGISTER". THUS A PROGRAM CAN LOAD A NUMBER OF REGISTERS AND CHECK FOR A REGISTER ACCESS ERROR ONCE AT THE END (BEFORE THE OPERATION STARTS). IN THIS CASE NO INTERRUPT WILL BE GENERATED. REPEATED DATAI INSTRUCTIONS WILL ACCESS THE SAME DRIVE REGISTER.

"TRA", "CONTROL BUSPARITY ERROR" SIGNALS ARE GENERATED FOR EACH DATAI INSTRUCTION AND ARE NOT REMEMBERED IN THE CONTROLLER EXCEPT BY SETTING THE REGISTER ACCESS ERROR BIT.

5. DETAILED DESCRIPTION OF THE INTERNAL REGISTERS

5.1 PRIMARY AND SECONDARY BLOCK ADDRESS REGISTER (REGISTER NUMBERS 72 AND 70) (PBAR AND SPAR). THESE REGISTERS ARE CONTAINED IN 4X4 MEMORY CHIPS ON THE M(%% MOFULE> THE BITS ARE CALLED BILE BIT XX(1).

THESE REGISTERS CONTAIN THE STARTING BLOCK ADDRESS (16 BITS) OF A DATA TRANSFER. IN THE CASE OF A DRUM OR DISK, THE BLOCK ADDRESS SPECIFIES THE STARTING SURFACE AND SECTION ADDRESS. IN THE CASE OF A MAGTAPE DRIVE, THE BLOCK ADDRESS REGISTER IS USED TO SPECIFY THE NUMBER OF FRAMES PER RECORD TO BE WRITTEN OR READ.

THE SECONDARY BLOCK ADDRESS REGISTER (SBA) MUST BE LOADED BEFORE THE SECONDARY TRANSFER CONTROL REGISTER (STCR = 71) SINCE THE CONTROLLER WILL EXECUTE THE COMMAND AS SOON AS THE SECONDARY TRANSFER CONTROL REGISTER IS LOADED. IF THE NEXT SECTOR OR RECORD OF THE SAME DRIVE IS TO BE READ/WRITE, THE SECONDARY BLOCK ADDRESS REGISTER NEED NOT BE LOADED. THIS WORKS BECAUSE MASSBUS DEVICES LEAVE THEIR BLOCK ADDRESS REGISTER INCREMENTED AND THE RH20 DOES NOT CHANGE THEM, IF SBAR WAS NOT LOADED WHEN STCR WAS LOADED.

5.2 PRIMARY AND SECONDARY TRANSFER CONTROL REGISTER (REGISTER NUMBERS 73 AND 71) (PTCR, STCR). THESE REGISTERS ARE CONTAINED IN 4X4 MEMORY CHIPS ON THE M8555 MODULE. THE BITS ARE CALLED FILE BIT XX(1).

THESE REGISTERS CONTAIN THE LENGTH (IN 2'S COMPLEMENT) OF THE DATA TRANSFER (NUMBER OF SECTORS OR RECORD) AND THE FUNCTION CODE TO SPECIFY A READ/WRITE OPERATION (NOTE TO PROGRAMMER: DO NOT LOAD ANY NON READ/WRITE FUNCTION CODE INTO THE SECONDARY TRANSFER CONTROL REGISTERS). IN THE CASE OF MAGTAPE, ONLY ONE RECORD AT A TIME SHALL BE SPECIFIED.

THE "RESET COMMAND LIST POINTER" BIT (FILE BIT 7), IF SET, WILL BE SENT AS A CBUS RESET TO THE CHANNEL (VIA THE CBUS) AT THE BEGINNING OF A COMMAND EXECUTION TO RESET THE COMMAND LIST POINTER IN THE CHANNEL ASSOCIATED WITH THE CONTROLLER TO A PREDETERMINED VALUE (MEMORY LOCATION ZERO OF THE "EXEC PROCESS TABLE" + 4 TIMES THE CONTROLLER'S PHYSICAL NUMBER + 0).

THE "DISABLE TRANSFER ERROR STOP" BIT (FILE BIT 19), IF SET, IS USED TO OVERRIDE THE RH20 MASSBUS LOOPBACK "DATA PARITY ERROR" (CONI BIT 18) BITS AND "DRIVE EXCEPTION" (CONI BIT 19). THIS IS USEFUL WHEN THE PROGRAM WISHES TO READ IN ALL DATA ALTHOUGH THERE IS A BAD SPOT ON THE DISK, FOR EXAMPLE,THE ERROR CONDITIONS ARE STORED AND CAN BE TESTED WITH A CONI INSTRUCTION. NOTE: ONCE XFER ERR IS SET THE RH20 WILL SET DONE AND AABORT THE XFER.

THE "STORE" BIT (BIT 10), IF SET, WILL BE SENT TO THE CHANNEL TO STORE ALL STATUS INFORMATIONS ASSOCIATED WITH THE CURRENT

TRANSFER INTO MEMORY SO THAT THE PROGRAM CAN DETERMINE THE EXACT STATUS OF THE CURRENT TRANSFER.

5.3 INTERRUPT VECTOR INDEX REGISTER (REGISTER NUMBER 74)

BITS 27 THROUGH 35 ARE SPECIFIED BY THE PROGRAMMER AS A RELATIVE ADDRESS WITHIN THE EPT. ACTUALLY THIS REGISTER IS NOT NEEDED, BUT WAS LEFT IN WHEN THE EBUS WAS CHANGED TO ALLOW AN ADDRESS RELATIVE TO THE EPT. ALL SUBSEQUENT DEVICES HAVE PREWIRED EPT RELATIVE ADDRESSES. [BREADBOARD: EXEC VIRTUAL ADDRESS]. THE CONTROLLER WILL SEND THESE BITS, AS THE CONTROLLER'S INTERRUPT VECTOR, TO THE EBOX (SEE "EBUS SPECIFICATION" FOR DETAILED INTERRUPT OPERATION).

5.4 READ REGISTER (REGISTER NUMBER 75)

THIS REGISTER IS USED MAINLY BY THE DIAGNOSTIC PROGRAMMER. DURING A DIAGNOSTIC WRITE OPERATION, DATA SENT OUT BY THE CONTROLLER (ON THE MASSBUS DATA BUS) CAN BE MADE TO LOOP BACK TO THE READ REGISTER AND CAN BE READ IN FOR DATA VERIFICATION.

5.5 WRITE REGISTER (REGISTER NUMBER 76)

THE REGISTER SHALL BE LOADED ONLY BY THE DIAGNOSTIC PROGRAMMER. CONTENTS OF THE WRITE REGISTER ARE USED TO SIMULATE DATA INPUTS TO THE CONTROLLER (ON THE MASSBUS DATA BUS) DURING A DIAGNOSTIC READ OPERATION. BITS 18-35 ARE DATA BITS AND BIT 17 SHALL BE THE COMPUTED PARITY BIT (BY THE PROGRAMMER) FOR BITS 18-35.

5.6 DIAGNOSTIC CONTROL REGISTER (REGISTER NUMBER 77)

MOST MASSBUS OPERATIONS CAN BE SIMULATED BY THE BITS IN THIS REGISTER.

5.6.1 CLOCK (BIT 35)

THIS BIT IS USED TO SIMULATE THE MASSBUS SYN CLOCK SIGNAL. THE TRAILING EDGE OF THIS BIT WILL STROBE IN OR SEND OUT DATA FROM/TO THE MASSBUS DATA BUS.

THE LEADING AND TRAILING EDGES OF THIS SIGNAL ARE GENERATED BY SIMPLY SETTING AND THEN RESETTING BIT 35.

5.6.2 ATTENTION (BIT 34)

THIS BIT IS USED TO SIMULATE THE MASSBUS ATTENTION SIGNAL. INTERRUPT WILL BE GENERATED WHEN THIS BIT IS SET PROVIDE THAT THE ATTENTION INTERRUPT ENABLE BIT (CONO BIT 30) HAS BEEN SET.

5.6.3 BIT 33

NOT USED.

5.6.4 EBL (BIT 32)

THIS BIT WILL SIMULATE THE MASSBUS END OF BLOCK SIGNAL. THE BLOCK COUNTER WILL BE INCREMENTED BY THE LEADING EDGE OF THIS BIT.

5.6.5 EXEPTION (BIT 31)

THIS BIT WILL SIMULATE THE MASSBUS EXCEPTION SIGNAL. THE SETTING OF THIS SIGNAL ALONG WITH EBL (BIT 32) SET WILL TERMINATE THE CURRENT COMMAND PROVIDED THAT "DISABLE TRANSFER ERROR STOP" (5.2) IS NOT SET.

5.6.6 READ/WRITE (BIT 30)

THIS BIT, WHEN SET, WILL ENABLE OUTPUT DATA (ON THE MASSBUS DATA BUS) TO BE LOOPED BACK TO BECOME INPUT DATA TO ENABLE TESTING OF THE DATA PATH AND THE DRIVERS/RECEIVERS. THIS BIT SHALL BE SET TO "1" ONLY FOR A READ SIMULATION AND SHALL BE CLEARED FOR A WRITE SIMULATION.

5.6.7 EVEN PARITY CHECK (BIT 29)

THIS BIT, WHEN SET, WILL ENABLE THE CONTROLLER TO CHECK FOR EVEN PARITY, INSTEAD OF ODD PARITY, ON THE MASSBUS CONTROL BUS. THE PARITY NETWORK CAN BE CHECKED BY THIS MEANS.

5.6.8 BLOCK ADDRESS REGISTER TEST (BIT 28)

THIS BIT, WHEN SET, WILL INHIBIT THE CONTROLLER FROM SENDING OUT THE CONTENT OF THE SECONDARY TRANSFER CONTROL REGISTER TO A DRIVE. INSTEAD, THE SECONDARY BLOCK ADDRESS REGISTER IS SENT OUT TWICE AND CAN BE CHECKED BY THE DIAGNOSTIC PROGRAM.

5.6.9 CONTROL BUS TEST (BIT 27)

THIS BIT, WHEN SET, WILL LOOPBACK THE MASSBUS CONTROL BUS TO ENABLE TESTING OF THE DRIVES/RECEIVERS AND RELATED LOGIC. NOTE: THE DIAGNOSTIC SOFTWARE MUST SELECT A DRIVE NUMBER WHICH IS NOT PRESENT OR IS NOT POWERED UP. THIS MAY NOT BE POSSIBLE IN A CONFIGURATION WITH 8 DUAL PORT DRIVES WITHOUT SOFTWARE INTERVENTION IN BOTH SYSTEMS.

5.6.10 TRANSFER (BIT 26)

THIS BIT, WHEN SET, WILL SIMULATE THE MASSBUS TRANSFER SIGNAL TO BE GENERATED WHENEVER A DRIVE REGISTER IS BEING ACCESSED.

5.7 DESCRIPTION OF THE DRIVE REGISTER - DUE TO THE SENSITIVE NATURE OF THE "MASSBUS INTERFACE STANDARD"

INTERESTED PEOPLE ARE RECOMMENDED TO OBTAIN COPIES OF THE SPECIFICATION FROM MR. JOHN LEVY (11/45 ENGINEERING). THE DRIVE REGISTERS, READ/WRITE OPERATION, DUAL PORT DRIVE PROTOCOL, ETC. ARE CLEARLY SPECIFIED AND DESCRIBED IN THE

SPECIFICATION,

6, STATUS REGISTER (CONO AND CONI)

BIT MAPS OF THE CONO AND CONI ARE SHOWN IN FIG, 5, EVERY CONI BIT WILL BE CLEARED BY AT LEAST THE FOLLOWING TWO CONDITIONS EXCEPT WHERE SPECIFIED,

1, THE RESET SIGNAL FROM THE EBUS; OR

2, CONO BIT 25 (CLEAR MASSBUS CONTROLLER),

6,1 BITS 33, 34, AND 35-PRIORITY INTERRUPT CHANNEL

INTERRUPT TO THE SPECIFIED PRIORITY INTERRUPT CHANNEL IS ENABLED IF NOT ALL THE BITS ARE ZERO,

CONDITIONS THAT CAUSE INTERRUPT ARE:

COMMAND DONE (BIT 32), MASSBUS ATTENTION (BIT 28) IF MBUS ATTN ENB (1) IS SET AND REGISTER ACCESS ERROR (BIT 24), IF DISABLE REGISTER ACCESS ERROR STOP IS NOT SET (PRE 0 REG BIT 09),

6,2 BIT 32 - COMMAND DONE

THIS BIT IS SET WHEN THE COMMAND IN THE PRIMARY COMMAND REGISTER IS TERMINATED WITH OR WITHOUT TRANSFER ERROR, AN INTERRUPT WILL BE GENERATED, COMMAND DONE IS CLEARED BY CONO BIT 32 = 1 (CLEAR COMMAND DONE),

6,3 BIT 31 - PRIMARY COMMAND REGISTER FULL

THIS BIT IS SET WHEN A COMMAND IN THE SECONDARY COMMAND REGISTER IS TRANSFERRED INTO THE PRIMARY COMMAND REGISTER WHEN THE PRIMARY COMMAND REGISTER IS NOT FULL AND ALL TRANSFER ERROR (IF ANY) CONDITIONS RESULTED FROM THE LAST TRANSFER HAVE BEEN CLEARED, THE COMMAND IN THE PRIMARY COMMAND REGISTER WILL BE EXECUTED IMMEDIATELY, THIS BIT IS RESET:

1, WHEN THE COMMAND IS TERMINATED WITH OR WITHOUT TRANSFER ERROR,

2, BY CONO BIT 31 = 1 (STOP TRANSFER),

TRANSFER ERROR CONDITIONS THAT WILL TERMINATE A TRANSFER COMMAND AND WILL INHIBIT THE EXECUTION OF THE NEXT TRANSFER COMMAND ARE;

1, DATA PARITY ERROR (CONI BIT 18) OR DRIVE EXCEPTION (CONI BIT 19) IS SET AND THE "DISABLE TRANFER ERROR STOP" BIT IN THE PRIMARY TRANSFER COMMAND REGISTER IS NOT SET,

2. LONG WORD COUNT ERROR (BIT 20).

3. SHORT WORD COUNT ERROR (BIT 21).

4. CHANNEL ERROR (BIT 22).

5. DRIVE RESPONSE ERROR (CONI BIT 23).

6. DATA OVERRUN (CONI BIT 26)

COMMAND DONE (CONI BIT 32) WILL BE SET WHEN THE COMMAND IN THE PRIMARY COMMAND REGISTER IS TERMINATED WITH OR WITHOUT TRANSFER ERROR.

CONO BIT 31 (STOP TRANSFER) WILL TERMINATE THE CURRENT TRANSFER BUT SHOULD BE USED BY THE SOFTWARE ONLY WHEN THE TRANSFER IS HUNG. ALL STATUS BITS WILL NOT BE CLEARED AND CAN BE EXAMINED BY CONI INSTRUCTIONS.

6.4 BIT 30 - MASSBUS ATTENTION ENABLE

THIS BIT, WHEN SET, WILL ENABLE THE MASSBUS "ATTENTION" SIGNAL TO GENERATE AN INTERRUPT.

6.5 BIT 29 - SECONDARY COMMAND REGISTER FULL

THIS BIT IS SET AS SOON AS THE SECONDARY TRANSFER CONTROL REGISTER (REG. 71 IS LOADED. THIS BIT IS RESET:

1. BY SETTING CONO BIT 29 = 1 (DELETE SECONDARY COMMAND REGISTER FULL).

2. AFTER TRANSFERRING THE CONTENTS OF THE SECONDARY COMMAND REGISTER INTO THE PRIMARY COMMAND REGISTER. THIS TAKES PLACE WHEN THE PRIMARY COMMAND REGISTER IS NOT FULL AND NO TRANSFER ERROR CONDITION EXISTS IN THE CONTROLLER. ERROR CONDITIONS ARE DEFINED IN 6.3. CONO BIT 29 = 1 (DELETE SECONDARY COMMAND REGISTER FULL) WILL DELETE THE CONTENTS OF THE COMMAND IN THE SECONDARY COMMAND REGISTER IF THAT COMMAND HAS NOT BEEN TRANSFERRED INTO THE PRIMARY COMMAND REGISTER. WHENEVER THE PROGRAMMER FOUND IT NECESSARY TO DELETE THE SECONDARY COMMAND THE PROGRAMMER SHOULD DO THE DELETION ONLY WHEN THERE IS AT LEAST ONE MORE SECTOR TO BE TRANSFERRED IN THE CURRENT DATA TRANSFER COMMAND.

6.6 BIT 28 - MASSBUS ATTENTION

THIS BIT IS SET BY ANY DRIVE ON THE MASSBUS WHENEVER ATTENTION CONDITION (SEEK COMPLETE ETC.) OCCURS IN THE DRIVE. THE BIT WILL GENERATE AN INTERRUPT IF NOT ALL PIA BITS ARE ZERO AND IF "MASSBUS ENABLE" IS SET. "MASSBUS ENABLE" IS SET BY CONO BIT 30 = 1. MASSBUS ATTENTION WILL BE SET EVEN WHEN MASSBUS ATTENTION INTERRUPT ENABLE IS NOT SET; IT JUST WON'T INTERRUPT

UNTIL MASSBUS ATTENTION ENABLE BECOMES SET, THE MASSBUS ATTENTION BIT WILL BE RESET ONLY AFTER ALL THE ATTENTION CONDITIONS IN ALL DRIVES ARE CLEARED. THE PROGRAM MUST SELECT THE APPROPRIATE DRIVES AND WRITE THE APPROPRIATE EXTERNAL (DRIVE) REGISTERS.

CONO BIT 28 = 1 (RESET COMMAND LIST POINTER) WILL CAUSE THE CHANNEL TO RESET THE COMMAND LIST POINTER, ASSOCIATED WITH CONTROLLER, TO POINT TO EPT + 4 TIMES THE CONTROLLER'S PHYSICAL NUMBER + 0. CONO BIT 28 SHOULD BE USED WHEN THERE IS NO TRNSFER IN PROGRESS, TO RESYNCHRONIZE THE MBOX AND THE PROGRAM.

6.7 BIT 27 - MASSBUS ENABLE

THIS BIT MUST BE SET (BY CONO BIT 27 = 1) TO ENABLE THE CONTROLLER TO INTERFACE TO ANY MASSBUS DRIVE.

IN SYSTEMS WHERE MORE THAN ONE CONTROLLERS ARE CONNECTED TO THE SAME MASSBUS, THE SOFTWARE SHALL MAKE SURE THAT ONLY ONE CONTROLLER AT ANY TIME IS ENABLED (BIT 27 =1). THIS FEATURE IS MAINLY INTENDED FOR FAILSOFT RECONFIGURATION IN SINGLE AND MULTI-PROCESSING SYSTEMS. THE DUAL PORT OPTION SHOULD BE USED WHERE CONCURRENT TRANSFERS ARE DESIRED.

THE OTHER CONTROLLER MUST BE DISCONNECTED FROM THE MASSBUS WITH CONO BIT 27 = 0, ELSE BUT ERRORS WILL OCCUR BECAUSE OF IMPROPER BUS TERMINATION.

6.8 BIT 26 - DATA OVERRUN

THIS BIT IS SET WHEN:

1. A MASSBUS DRIVE IS INPUTING DATA FASTER THAN THE CONTROLLER CAN TRANSMIT TO THE CHANNEL; OR

2. A MASSBUS DRIVE IS DEMANDING DATA FASTER THAN THE CONTROLLER IS RECEIVING FROM THE CHANNEL.

CONO BIT 26 = 1 (TRANSFER ERROR CLEAR) WILL CLEAR:

1. DATA OVERRUN (CONI BIT 26)
2. DRIVE RESPONSE ERROR (CONI BIT 23)
3. CHANNEL ERROR (CONI BIT 22)
4. SHORT WORD COUNT (CONI BIT 21)
5. LONG WORD COUNT (CONI BIT 20)
6. DRIVE EXCEPTION (CONI BIT 19)
7. DATA PARITY ERROR (CONI BIT 18)

6.9 BIT 25 - CHANNEL NOT READY

THIS STATUS BIT IS CONTROLLED SOLELY BY THE CHANNEL AND CAN NOT BE SET OR RESET BY THE PROGRAM. THE CHANNEL SETS THIS BIT EITHER WHEN THE SYSTEM IS INITIALIZED OR AFTER THE CHANNEL HAS COMPLETED THE PREVIOUS TRANSFER AND HAS ENCOUNTERED A HALT

COMMAND WORD OR HALT BIT IN A COMMAND WORD AND IS READY TO START A NEW TRANSFER. THE CHANNEL RESETS THIS BIT WHEN IT IS EITHER ACTIVELY DOING DATA TRANSFER OR IS IN THE PROCESS OF FINISHING UP THE CURRENT DATA TRANSFER OPERATION (HOUSE-CLEANING) AND IS NOT READY FOR THE NEXT DATA TRANSFER OPERATION.

CONO BIT 25 (CLEAR MASSBUS CONTROLLER) = 1 WILL INITIALIZE THE CONTROLLER AND THE MASSBUS DRIVES TO THEIR POWERUP STATES.

IF THIS BIT IS USED TO CLEAR THE CONTROLLER WHEN THERE IS A DATA TRANSFER COMMAND IN PROGRESS THE PROGRAM MUST ALSO DO A CONO INSTRUCTION WITH BIT 28 = 1 (RESET COMMAND LIST POINTER) TO CLEAR THE MBOX OTHERWISE THE MBOX CAN BE IN A HUNG STATE.

6,10 CONI BIT 24 - REGISTER ACCESS ERROR

THIS BIT IS SET WHEN A DRIVE REGISTER IS ACCESSED AND ONE OR MORE OF THE FOLLOWING ERROR CONDITIONS OCCUR:

1. CONTROL BUS TIME OUT - THIS ERROR CONDITION OCCURS WHEN A NONEXISTANT DRIVE IS SELECTED. IF THE ATTENTION SUMMARY REGISTER IS SELECTED, THE CONTROLLER WILL EXPECT A CONTROL BUS TIME OUT AND, THEREFORE, WILL NOT SET THE ERROR BIT.

2. CONTROL BUS PARITY ERROR - THIS ERROR CONDITION OCCURS WHEN A DRIVE REGISTER IS READ AND THE CONTROLLER DETECTS A PARITY ERROR ON THE MASSBUS CONTROL BUS. IF THE ATTENTION SUMMARY REGISTER IS READ, THE CONTROLLER WILL INHIBIT PARITY CHECKING AND THIS ERROR WILL NOT BE SET.

THE CONTROLLER WILL NOT ALLOW FURTHER WRITING OF ANY REGISTER IF "REGISTER ACCESS ERROR" IS SET AND THE INSTRUCTION THAT CAUSED THE ERROR DID NOT SPECIFY "DISABLE REGISTER ACCESS ERROR STOP".

CONO BIT 24 = 1 WILL CLEAR REGISTER ACCESS ERROR AND THE ASSOCIATED INTERRUPT REQUEST.

6,11 BIT 23 - DRIVE RESPONSE ERROR

THIS CONI BIT IS SET WHEN A DRIVE DOES NOT RESPONSE WITH "TRANSFER" WHEN THE CONTROLLER IS LOADING A READ/WRITE COMMAND INTO A DRIVE.

DRIVE RESPONSE ERROR WILL TERMINATE THE CURRENT TRANSFER AND COMMAND DONE WILL BE SET. THIS BIT IS CLEARED BY CONO BIT 26. CONO BIT 23 IS NOT USED.

6,12 BIT 22 - CHANNEL ERROR

THIS CONI STATUS BIT IS SET WHEN THE CHANNEL, DURING A DATA

TRANSFER OPERATION, DETECTS AN ERROR CONDITION THAT WILL AFFECT THE VALIDITY OF THE DATA TRANSFER. THE CONTROLLER UPON SENSING THIS BIT, WILL TERMINATE THE CURRENT DATA TRANSFER AND COMMAND DONE WILL BE SET. THIS BIT IS CLEARED BY CONO BIT 26. CONO BIT 22 IS NOT USED.

6.13 BIT 21 - SHORT WORD COUNT

THIS BIT WILL BE SET IF THE CHANNEL HAS TRANSFERRED ALL THE DATA SPECIFIED BY THE CHANNEL COMMAND WORD BUT THE CONTROLLER IS STILL NOT DONE WITH THE TRANSFER. THE CONTROLLER WILL TERMINATE THE CURRENT DATA TRANSFER AND WILL SET COMMAND DONE AND TRANSFER ERROR. THIS BIT IS CLEARED BY CONO BIT 26. CONO BIT 21 IS NOT USED.

6.14 BIT 20 - LONG WORD COUNT

THIS BIT WILL BE SET IF THE CONTROLLER HAS TRANSFERRED THE SPECIFIED NUMBER OF SECTOR (SPECIFIED IN THE BLOCK COUNT FIELD OF THE PRIMARY COMMAND REGISTER) BUT THE CHANNEL HAS NOT REACHED THE FINAL WORD COUNT SPECIFIED IN THE CHANNEL COMMAND WORD. THE CONTROLLER WILL TERMINATE THE CURRENT DATA TRANSFER AND WILL SET COMMAND DONE AND TRANSFER ERROR. THIS BIT IS CLEARED BY CONO BIT 26. CONO BIT 20 IS NOT USED.

6.15 BIT 19 - DRIVE EXCEPTION

THIS BIT IS SENT TO THE CONTROLLER BY THE DRIVE SPECIFIED IN THE CURRENT READ/WRITE COMMAND WHEN A TRANSFER ERROR CONDITION (DATA PARITY, ETC.) IS DETECTED IN THE DRIVE. THE CONTROLLER WILL STORE THE BIT AND WILL TERMINATE THE CURRENT TRANSFER. COMMAND DONE AND TRANSFER ERROR WILL BE SET. THIS BIT IS CLEARED BY CONO BIT 26. CONO BIT 19 IS NOT USED.

DRIVE EXCEPTION WILL NOT CAUSE THE READ/WRITE COMMAND TO BE TERMINATED IF "INHIBIT TRANSFER ERROR STOP" IS SPECIFIED IN THE PRIMARY COMMAND REGISTER. CONO BIT 19 IS NOT USED.

6.16 BIT 18 - DATA PARITY ERROR

THIS BIT IS SET WHEN THE CONTROLLER DETECTS A PARITY ERROR ON THE MASSBUS DATA BUS DURING A READ AND WRITE COMMAND. IN THE CASE OF A WRITE COMMAND, THE DRIVE INVOLVED MAY ALSO DETECT A PARITY ERROR. THE CONTROLLER WILL TERMINATE THE READ/WRITE COMMAND AND WILL SET COMMAND DONE AND TRANSFER ERROR. CONO BIT 26 WILL CLEAR THIS ERROR. CONO BIT 18 IS NOT USED.

DATA PARITY ERROR WILL NOT TERMINATE THE READ/WRITE COMMAND IF "INHIBIT TRANSFER ERROR STOP" IS SPECIFIED IN THE PRIMARY COMMAND REGISTER. CONO BIT 18 IS NOT USED.

6.17 BITS 0 - 17 - NOT USED, 7-9 SPARE

7. NOTES ON CONTROLLER OPERATION

THE FOLLOWING STEPS CAN BE USED TO PROGRAM THE CONTROLLER TO PERFORM A READ/WRITE OPERATION.

7.1 DRIVE TYPE

THE PROGRAM READS THE "DRIVE TYPE" REGISTER OF EACH MASSBUS DRIVE TO DETERMINE HOW MANY AND WHAT TYPE OF DRIVES ARE ON EACH CONTROLLER.

7.2 SEEK TO THE DESIRED CYLINDER

(APPLIES ONLY TO MOVING-HEAD DISK DRIVES) BY READING THE "CURRENT CYLINDER" REGISTER OF EACH MOVING-HEAD DRIVE, THE PROGRAM CAN DETERMINE WHICH DRIVES REQUIRE SEEK OPERATION. THE PROGRAM CAUSES A DRIVE TO SEEK TO DESIRED CYLINDER BY LOADING THE DRIVES "DESIRED CYLINDER" REGISTER WITH THE DESIRED CYLINDER NUMBER AND THEN THE "CONTROL REGISTER" WITH A SEEK COMMAND. THE DRIVE WILL ASSERT "ATTENTION" WHICH IN TURN WILL CAUSE THE CONTROLLER TO INTERRUPT THE PROGRAM AFTER THE SEEK OPERATION IS DONE.

7.3 SECTOR OPTIMIZATION (APPLIES TO MOVING-HEAD AND FIXED HEAD DISK DRIVES)

SECTOR OPTIMIZATION CAN BE DONE BY READING THE DRIVES "LOOK AHEAD REGISTER" WHICH GIVE THE ROTATIONAL POSITION OF A DISK.

7.4 CHANNEL COMMAND WORD LOADING

THE PROGRAMMER LOADS THE CHANNEL COMMAND WORD INSTRUCTIONS INTO MEMORY (SEE 9.0)

7.5 COMMAND LOADING

AFTER THE ABOVE ARE DONE THE PROGRAM CAN LOAD THE SECONDARY COMMAND REGISTER (SECONDARY BLOCK ADDRESS REGISTER FIRST AND THEN THE SECONDARY TRANSFER CONTROL REGISTER) WITH A R/W COMMAND.

7.6 NORMAL READ/WRITE

AFTER THE PROGRAM LOADED THE SECONDARY BLOCK ADDRESS REGISTER (OPTIONAL) AND THE SECONDARY TRANSFER CONTROL REGISTER CONTROL REGISTER (COMPULSORY), THE SECONDARY COMMAND REGISTER IS TRANSFERRED INTO THE PRIMARY COMMAND REGISTER IF THE PRIMARY COMMAND REGISTER IS NOT FULL AND NO TRANSFER ERROR CONDITION EXISTS IN THE CONTROLLER. THE CONTROLLER WILL AUTOMATICALLY ADDRESS AND LOAD TWO DRIVE REGISTER: THE "DESIRED BLOCK ADDRESS REGISTER" (THE "FRAME COUNTER REGISTER" IN THE CASE OF MAGTAPE DRIVE) AND THE "CONTROL REGISTER" (ONLY THE "CONTROL REGISTER" IS LOADED IF "PRIMARY BLOCK ADDRESS REGISTER" IS NOT FULL). THE CONTROLLER'S DATA CONTROL PORTION IS THEN SET UP TO

RECEIVE OR TRANSMIT DATA BY DECODING THE FUNCTION BITS IN THE PRIMARY TRANSFER COMMAND REGISTER. CAUTION: ONLY READ/WRITE COMMAND SHALL BE LOADED INTO THE SECONDARY TRANSFER CONTROL REGISTER. THE PROGRAM SHALL LOAD ALL NON READ/WRITE COMMANDS INTO THE CONTROL REGISTER OF A DRIVE DIRECTLY. THE BLOCK COUNTER IS INCREMENTED AFTER EACH BLOCK OF DATA HAS BEEN TRANSFERRED TO THE MBOX. "PRIMARY COMMAND REGISTER FULL" IS RESET WHEN THE BLOCK COUNTER IS INCREMENTED TO ZERO AND "COMMAND DONE" WILL BE SET.

7.7 CBUS OPERATION

REFER TO CHANNEL BUS (CBUS) - MBOX/RH20 INTERFACE, CHAPTER 5.1.

7.8 LOADING ANOTHER READ/WRITE COMMAND

WHEN THE CONTROLLER AND THE MBOX IS ACTIVELY DOING THE READ/WRITE COMMAND ON THE PRIMARY COMMAND REGISTER, THE PROGRAMMER CAN DETERMINE AND SET UP THE NEXT READ/WRITE COMMAND TO BE PERFORMED BY THE CONTROLLERS BY LOADING THE SECONDARY COMMAND REGISTER.

8. NOTES ON MAGTAPE DRIVE PROGRAMMING:

8.1

ONLY ONE RECORD PER COMMAND SHALL BE SPECIFIED TO BE READ/WRITTEN

8.2

IN THE CASE OF READ REVERSE OPERATION, THE MBOX (CHANNEL CONTROL LOGIC) WILL DECREMENT (INSTEAD OF INCREMENT) THE MEMORY BUFFER LOCATIONS.

8.3

READ REVERSE ON MAGTAPE DRIVES WILL NOT GET MEANINGFUL DATA IF THE NUMBER OF FRAME ON THE RECORD IS NOT A MULTIPLE OF WORDS.

THEREFORE, IF A PROGRAMMER WISHES TO WRITE PART OF A PDP10 CORE WORD ON TAPE AND TO BE ABLE TO READ THE RECORD BACK LATER WITH A READ REVERSE COMMAND, HE MUST WRITE THE ENTIRE WORD ON TAPE.

8.4 REGISTERS 0-40

THESE REGISTERS ARE THE SO-CALLED EXTERNAL REGISTERS BECAUSE THEY ARE IMPLEMENTED IN EACH DRIVE AND DEPEND ON THE TYPE OF DRIVE. SEE TABLE 2 IN SECTION 4.1 FOR THE NAMES OF THE EXTERNAL REGISTERS. SEE THE MASSBUS SPECIFICATION AND THE FUNCTIONAL SPECIFICATIONS FOR EACH OF THE INDIVIDUAL DRIVES RP04, RS04, AND TM02/TU16.

CONVENTIONS USED IN THE FOLLOWING TABLE. BOTH THE READ (CONI)

AND WRITE (CONO) BIT NAMES ARE GIVEN. USUALLY THE NAMES ARE THE SAME SINCE A CONO BIT USUALLY EITHER:

1. CLEARS THE BIT WHICH IS READ ON A CONI

2. WRITES THE BIT A 0 OR 1 WHICH IS READ ON A CONI

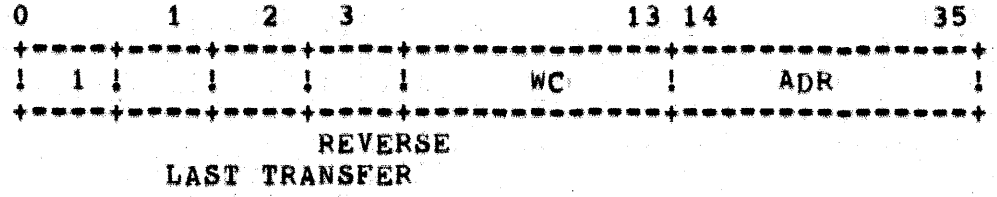
THE BIT NAME FIELD INDICATES WHETHER THE BIT IS CLEARED OR JUST WRITTEN ON A CONO. THE WRITE BIT IS GIVEN A DIFFERENT SYMBOL AND NAME IF IT PERFORMS AN UNRELATED FUNCTION. IF A BIT ALWAYS READS ON A CONI AS 0, ITS SYMBOL IS "0". IF A BIT IS IGNORED OR WRITES ON A CONO, ITS SYMBOL IS "IGN".

9. CHANNEL COMMAND WORD

9.1 A CHANNEL COMMAND WORD (CCW) WILL USE 22 BIT PHYSICAL MEMORY ADDRESSES TO SPECIFY JUMP, HALT ADDRESSES AND THE STARTING ADDRESSES OF DATA TRANSFERS.

9.2 CHANNEL COMMAND WORD FORMATS

9.2.1 DATA TRANSFER



9.2.1.1 WC=POSITIVE COUNT EQUAL TO THE NUMBER OF WORDS TO BE TRANSFERRED.

9.2.1.2 ADR=PHYSICAL MEMORY ADDRESS OF THE FIRST LOCATION (IF BIT 2=0) OR LAST LOCATION (IF BIT 2=1) IN THE DATA BUFFER.

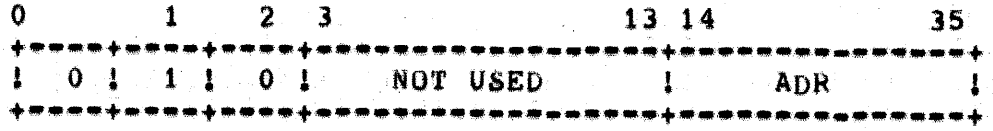
1. IF ADR=0 THEN THE CCW DOES NOT MOVE ANY DATA TO OR FROM MEMORY. ON READS, DATA COMING FROM THE RH20 IS THROWN AWAY. ON WRITES, THE CHANNEL SUPPLIES FULL WORDS OF BLOCK FILL WORDS (FETCH FROM LOCATION EPT + 60 TO 63) TO THE RH20 TO FILL THE REMAINING OF THE CURRENT BLOCK. ON BOTH READS AND WRITES, WC IS DOWN COUNTED EACH TIME A WORD IS REQUESTED BY THE RH20. WHEN WC GOES TO ZERO THE NEXT CCW IS FETCHED (IF BIT 1=0); THE CHANNEL WILL HALT, WHEN WC=0, IF BIT 1=1. IN THIS CASE, THE CHANNEL COMMAND LIST POINTER WILL POINT TO THE CURRENT CCW LOCATION +1.

2. IF ADR NON 0 THEN THE WC IS DOWN COUNTED EACH TIME A WORD IS MOVED TO/FROM MEMORY. THE ADR IS UP COUNTED (IF BIT 2=0) OR DOWN COUNTED (IF BIT 2=1) EACH TIME A WORD IS MOVED TO/FROM MEMORY.

THIS ALLOWS READ REVERSE OPERATIONS ON MAGTAPE SYSTEMS. THE PROGRAMMER MUST SET BIT 2 TO ZERO FOR ALL WRITE OPERATIONS.

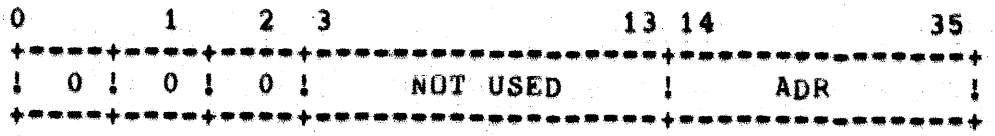
WHEN WC GOES TO ZERO THE NEXT CCW IS FETCHED (IF BIT 1=0); THE CHANNEL WILL HALT, WHEN WC=0, IF BIT 1=1. THE CHANNEL COMMAND LIST POINTER WILL POINT TO THE CURRENT CCW+1.

9.2.2 JUMP. USED TO JUMP TO A NEW CHANNEL COMMAND LIST.



ADR=PHYSICAL MEMORY ADDRESS FROM WHICH THE NEXT CCW WILL BE PICKED UP. NO DATA IS TRANSFERRED BY THIS COMMAND WORD.

9.2.3 HALT. THIS CCW CAUSES THE CHANNEL TO HALT. THE CHANNEL COMMAND LIST POINTER WILL POINT TO "ADR".



9.2.4 WHEN THE CHANNEL RECEIVES A "START" SIGNAL FROM THE RH20, THE CHANNEL WILL USE THE CHANNEL COMMAND LIST POINTER TO FETCH A NEW CCW IF THE RH20 DOES NOT SPECIFY "RESET". IF "RESET" IS SPECIFIED, THE CHANNEL WILL GO TO EPT+4(RH20'S PHYSICAL NUMBER)+0 FOR THE NEW CCW. THE NEW CCW SHALL BE A JUMP CCW IN THIS CASE SINCE THE CHANNEL COMMAND LIST POINTER MAY BE HOLDING A RANDOM ADDRESS AND A MEANINGFULL ADR NEEDED TO BE LOADED.

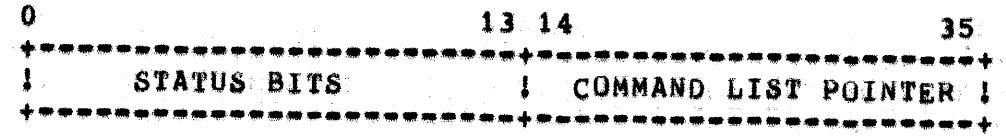
9.2.5 IN ORDER TO INCREASE THE CHANCE OF DOING NEXT SECTOR READ/WRITE, THE LAST CCW OF A DATA TRANSFER SHALL HAVE BIT 1=1 (LAST TRANSFER). THIS SAVES THE CHANNEL FROM DOING ONE MEMORY REFERENCE.

10. CHANNEL STATUS WORD

THE CHANNEL, UPON DETECTING ANY ERROR OR AFTER RECEIVING A "STORE REQUEST FROM THE RH20, WILL STORE TWO STATUS WORDS IN:

EPT+4 (RH20'S PHYSICAL NUMBER)+1=STATUS WORD 1, AND
EPT+4 (RH20'S PHYSICAL NUMBER)+2=STATUS WORD 2.

1. STATUS WORD 1



BIT 0 THIS BIT IS ALWAYS A ONE.

BIT 1 (MEMORY PARITY ERROR): THIS BIT IS SET IF MEMORY PARITY ERROR OCCURS WHENEVER THE CHANNEL REFERENCES MEMORY.

BIT 2 (NOT SBUS ERROR): THIS BIT IS RESET IF MEMORY PROBLEMS OTHER THAN PARITY AND NONEXISTENT MEMORY OCCURS.

BIT 3 (NOT WORD COUNT=0): THIS BIT IS RESET IF THE WORD COUNT IS ZERO WHEN THE STATUS WORDS ARE STORED INTO MEMORY.

BIT 4 (NON EXISTENT MEMORY): THIS BIT IS SET IF THE CHANNEL REFERENCES A NON EXISTENT MEMORY LOCATION.

BIT 5,6,7 0'S

BIT 8 (SPARE)

BIT 9 (LAST TRANSFER ERROR): SME MEMORY ERROR CONDITIONS MAY NOT BE DETECTED FOR UP TO 25US (NON EXISTENT MEMORY, ETC.). WHENEVER AN ERROR IS DETECTED AFTER THE RH20 HAS TERMINATED A DATA TRANSFER COMMAND, THE CHANNEL WILL SET BIT 9 AND WILL STORE THE STATUS WORDS, THE NEW DATA TRANSFER COMMAND (IF ANY) WILL BE ABORTED. THE PROGRAMMER SHALL EXAMINE THIS BIT AFTER EACH TRANSFER TO MAKE SURE THAT THE TRANSFER HAS NO ERROR.

BIT 10 (RH20 ERROR): THIS BIT IS SET IF THE RH20 SENDS A "START" SIGNAL WHEN THE CHANNEL'S "READY" BIT IS TRUE.

BIT 11 (LONG WC ERROR): THIS BIT IS SET IF THE RH20 HAS COMPLETED THE DATA TRANSFER COMMAND AND THE CHANNEL HAS NOT REACHED THE FINAL WORD COUNT SPECIFIED IN THE CCW.

BIT 12 (SHORT WORD COUNT ERROR): THIS BIT IS SET IF THE CHANNEL HAS TRANSFERRED ALL THE DATA SPECIFIED BY THE CCW AND THE RH20 IS STILL NOT DONE WITH THE TRANSFER.

BIT 13 (OVERRUN ERROR): EACH RH20 HAS 16 DATA BUFFERS IN THE CHANNEL. THIS BIT IS SET WHEN

1. THE RH20 IS INPUTTING DATA AND THE DATA BUFFER IS FULL, OR

2. THE RH20 IS DEMANDING DATA AND THE CHANNEL DATA BUFFER IS EMPTY.

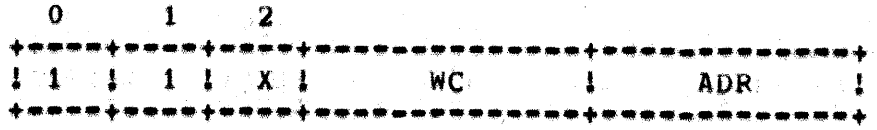
BIT 14-35 (CHANNEL COMMAND LIST POINTER): POINTS AT CURRENT CCW+1.

2. STATUS WORD 2 - CONTAINS THE LAST CCW WITH UP-TO-DATE WORD COUNT AND ADDRESS.

11. PROGRAMMING NOTES

PROGRAMMING NOTES ON READING/WRITING NEXT SECTORS ON A RS04 OR RP04 DRIVE - THE RS04 AND RP04 DRIVES HAVE FINITE GAPS BETWEEN SECTORS. THE FOLLOWING PROGRAMMING NOTES SHOULD BE OBSERVED WHEN READ/WRITE NEXT SECTOR IS DESIRED IN ORDER TO INCREASE THE CHANCE OF WINNING:

1. THE LAST CCW OF ANY DATA TRANSFER SHOULD BE:

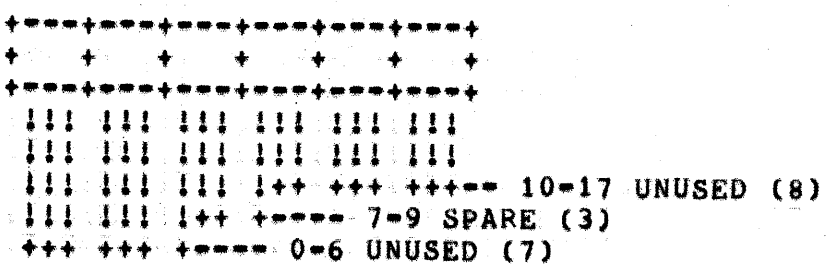


X=1 IF READ REVERSE
THIS ALLOWS THE CHANNEL TO TERMINATE THE CURRENT TRANSFER
WITHOUT HAVING TO FETCH A HALT CCW.

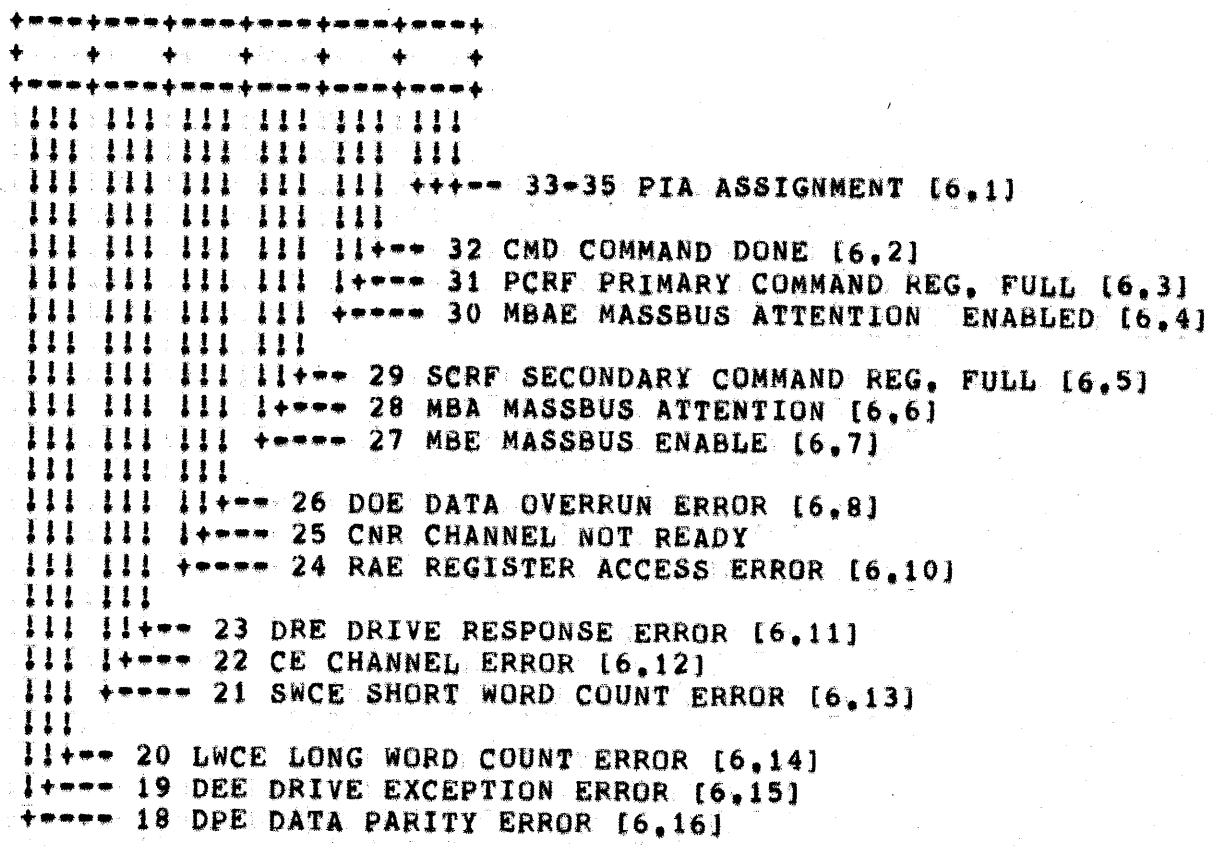
2. THE FIRST CCW OF THE NEXT TRANSFER SHOULD BE A DATA CCW AND
SHOULD IMMEDIATELY FOLLOW (IN MEMORY LOCATION) THE LAST DATA
CCW OF THE LAST TRANSFER. THIS ALLOWS THE CHANNEL TO GET
READY FOR THE NEW TRANSFER WITHOUT HAVING TO FETCH A JUMP
CCW. UP TO TWO CONSECUTIVE JUMP CCWS ARE ALLOWED AFTER THE
FIRST DATA CCW OF A TRANSFER IN ORDER TO MINIMIZE DATA OVERRUN.

12. REGISTER LAYOUTS

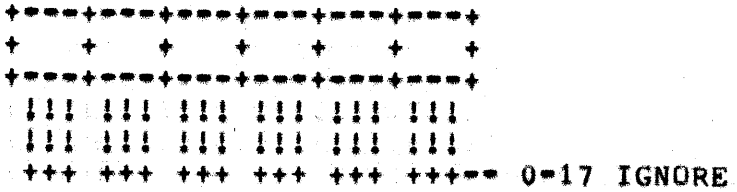
CONI DTEX, E (LH)



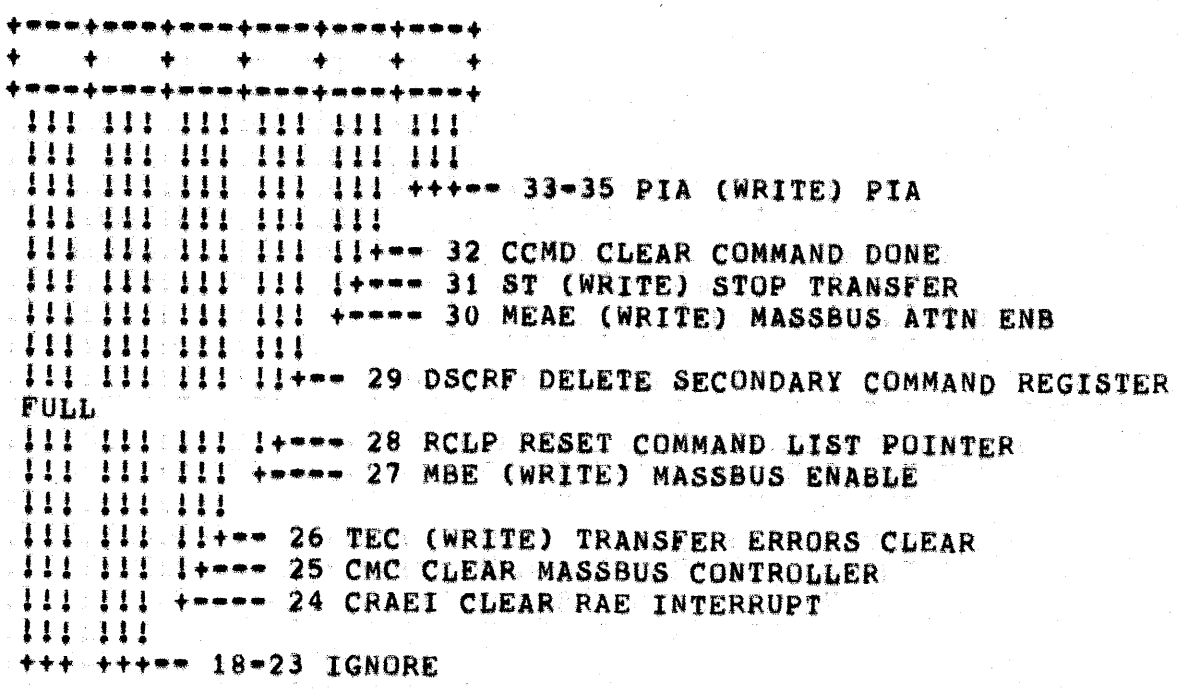
CONI DTEX, E (RH)



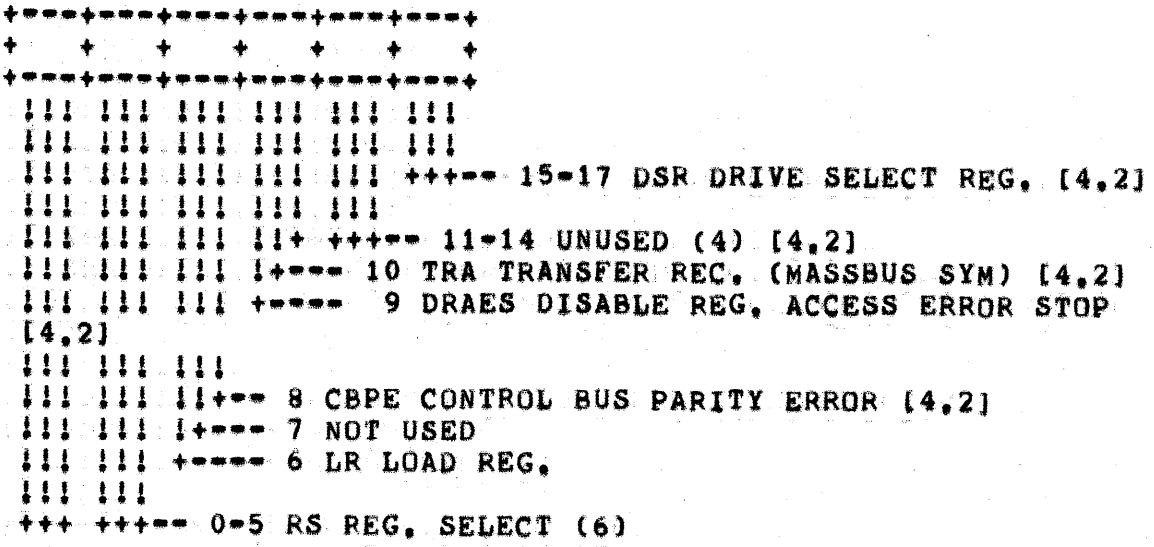
CONO DTEX, E (LH)



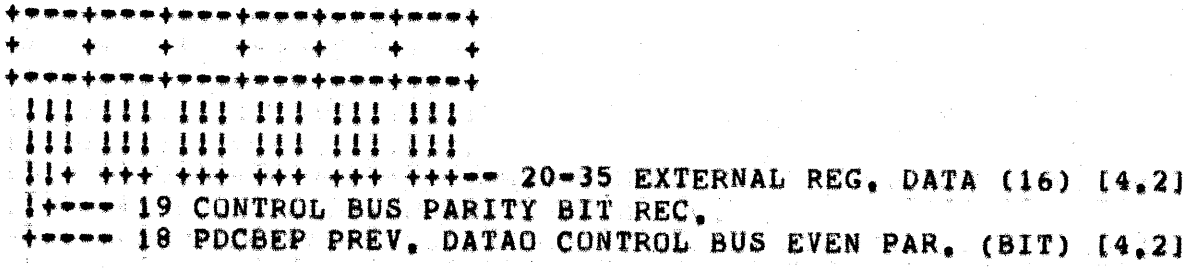
CONO DTEX, E (RH)



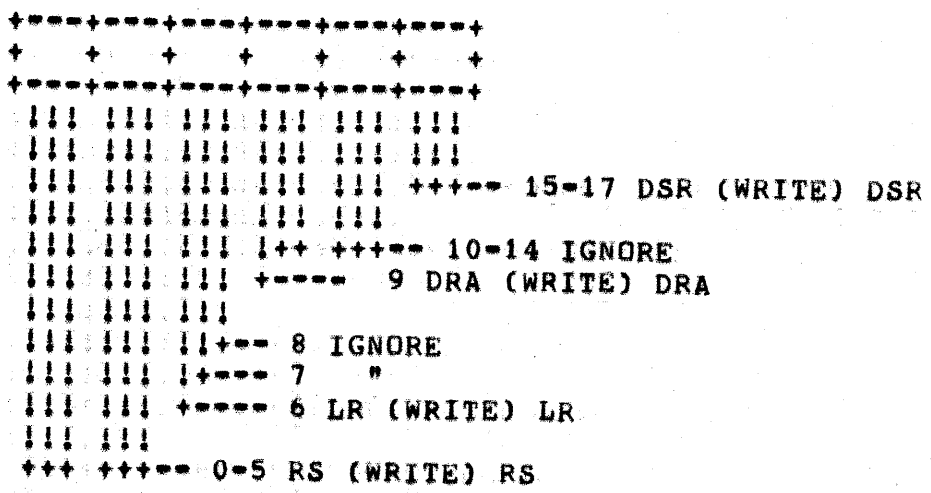
DATAI EXTERNAL (DRIVE) REGISTERS (00-37) (LH)



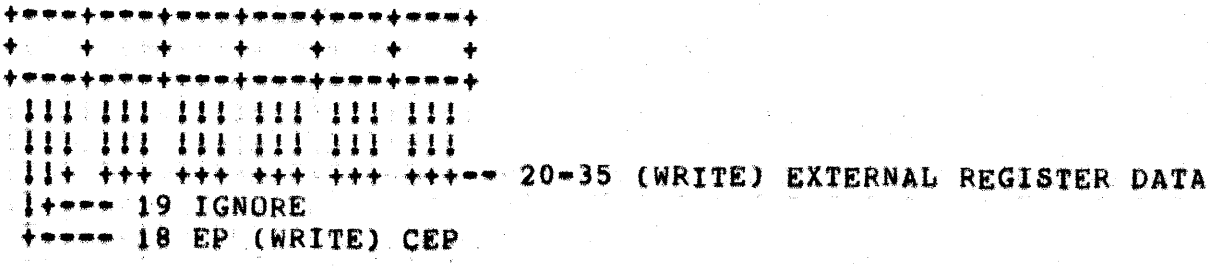
DATAI EXTERNAL (DRIVE) REGISTERS (00-37) (RH)



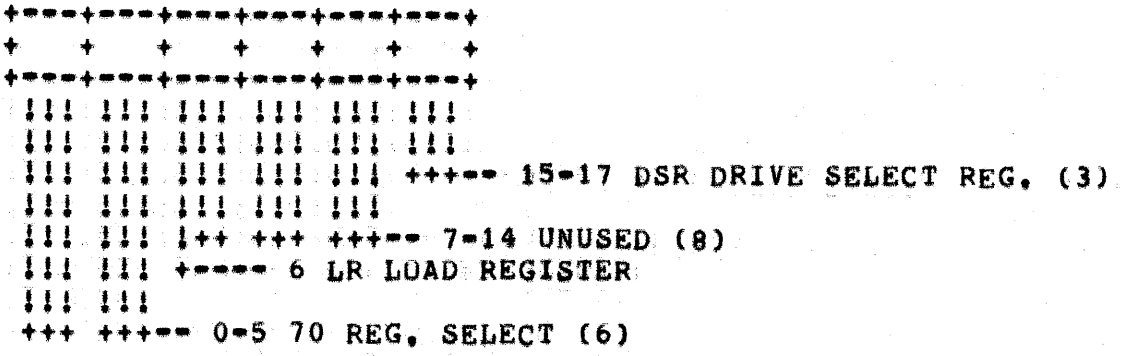
DATAO EXTERNAL (DRIVE) REGISTERS (00-37) (LH)



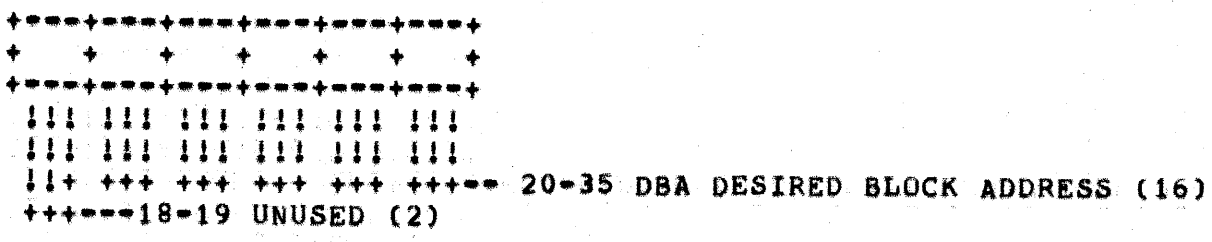
DATAO EXTERNAL (DRIVE) REGISTERS (00-37) (RH)



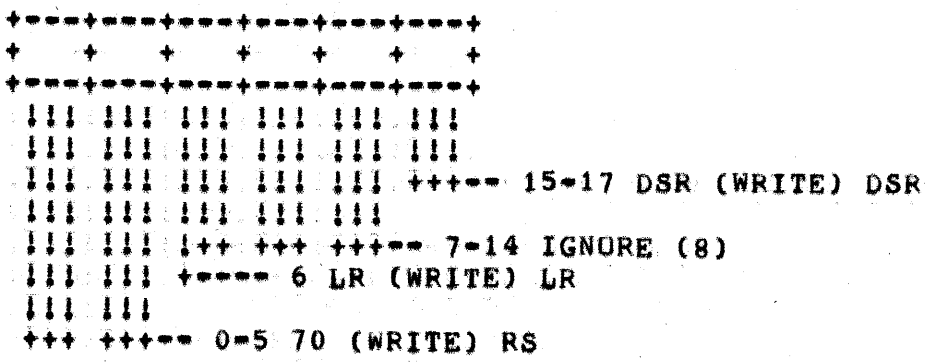
DATAI SECONDARY BLOCK ADDRESS REGISTER (SBA = 70) (LH)



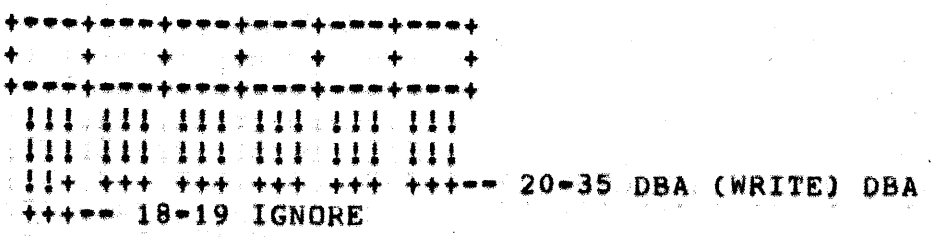
DATAI SECONDARY BLOCK ADDRESS REGISTER (SBA = 74) (RH)



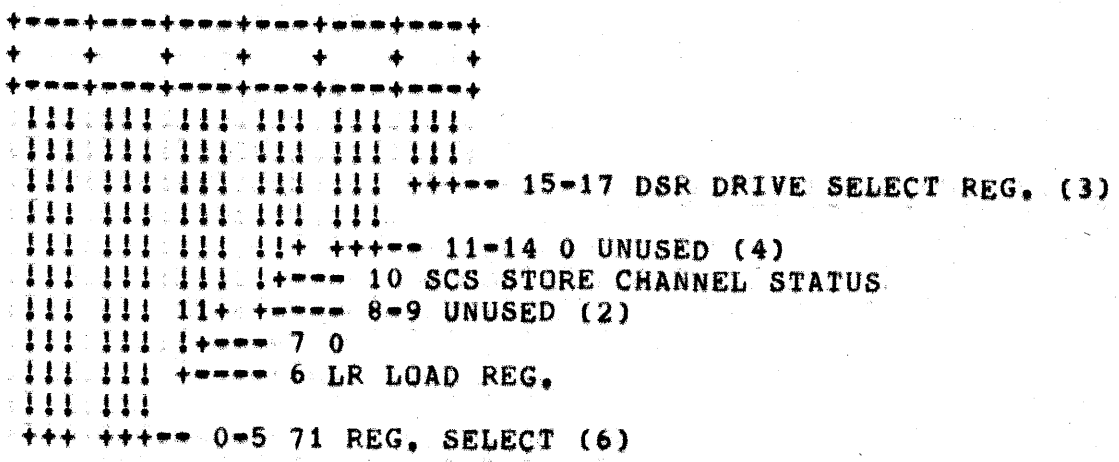
DATAO SECONDARY BLOCK ADDRESS REGISTER (SBA = 70) (LH)



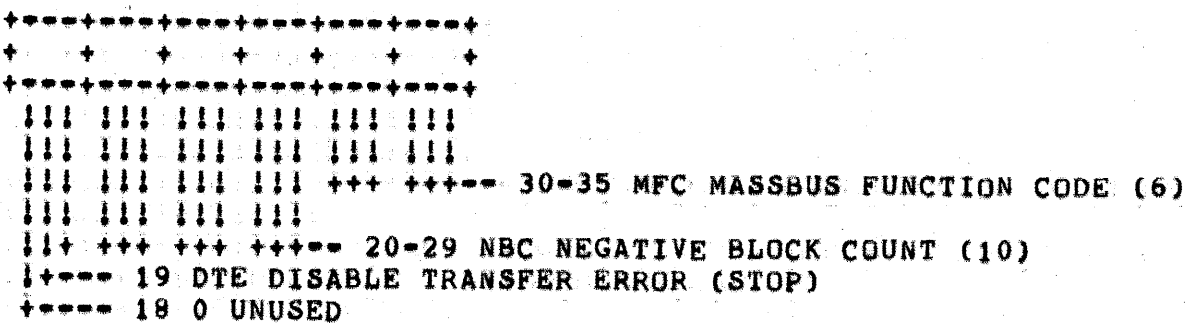
DATAO SECONDARY BLOCK ADDRESS REGISTER (SBA = 70) (RH)



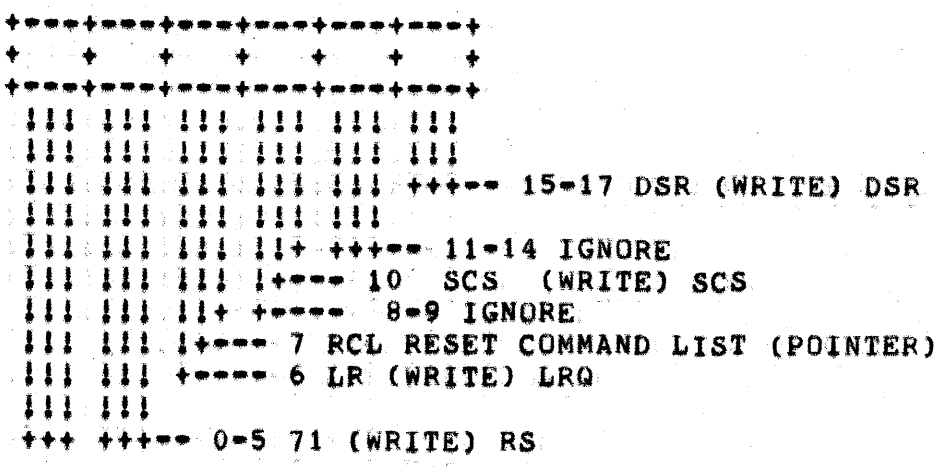
DATAO SECONDARY TRANSFER CONTROL REGISTER (SCT = 71) (LH)



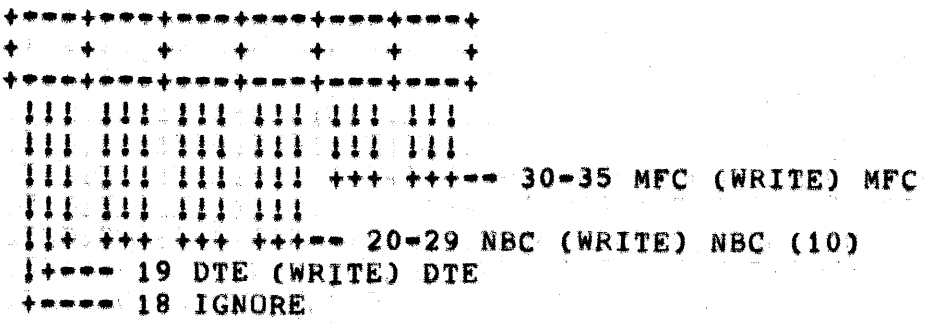
DATAO SECONDARY TRANSFER CONTROL REGISTER (STC = 71) (RH)



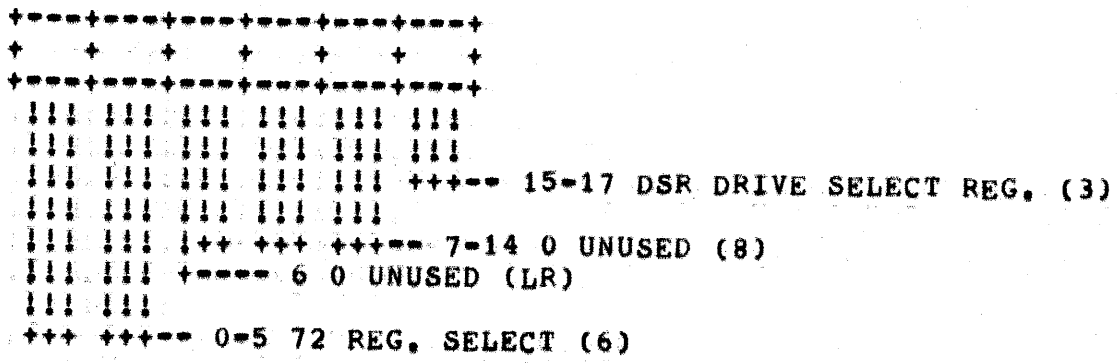
DATAI SECONDARY TRANSFER CONTROL REGISTER (STC = 71) (LH)



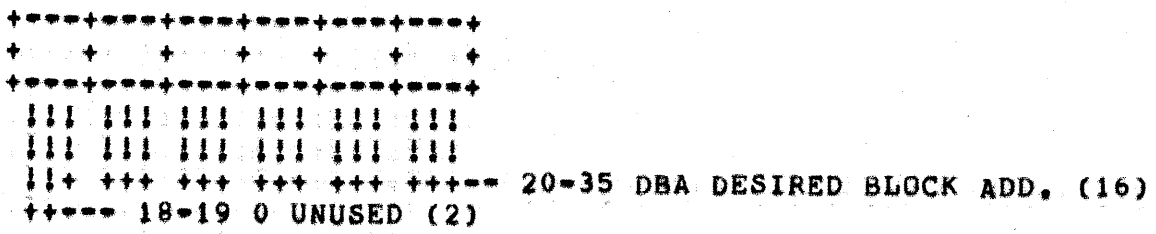
DATAI SECONDARY TRANSFER CONTROL REGISTER (STC = 71) (RH)



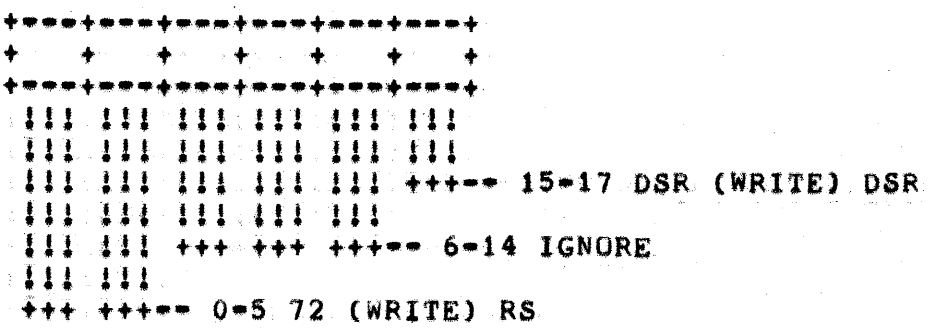
DATAO PRIMARY BLOCK ADDRESS REGISTER (PBA = 72) (LH)



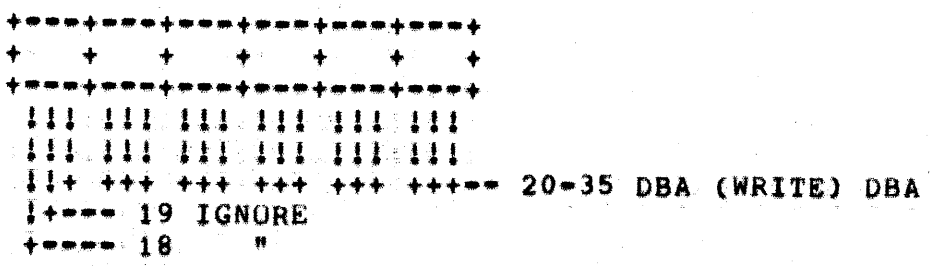
DATAO PRIMARY BLOCK ADDRESS REGISTER (PBA = 72) (RH)



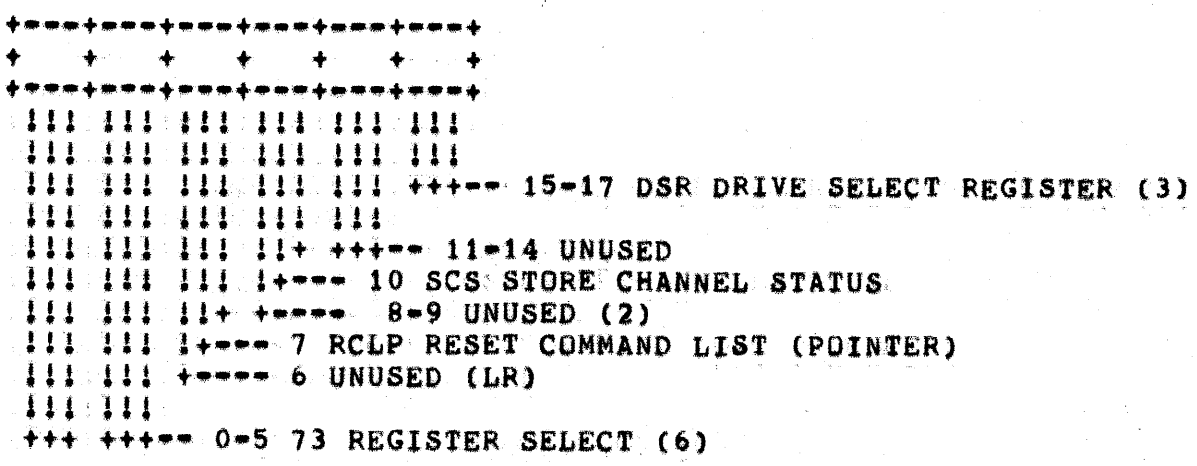
DATAI PRIMARY BLOCK ADDRESS REGISTER (PBA = 72) (LH)



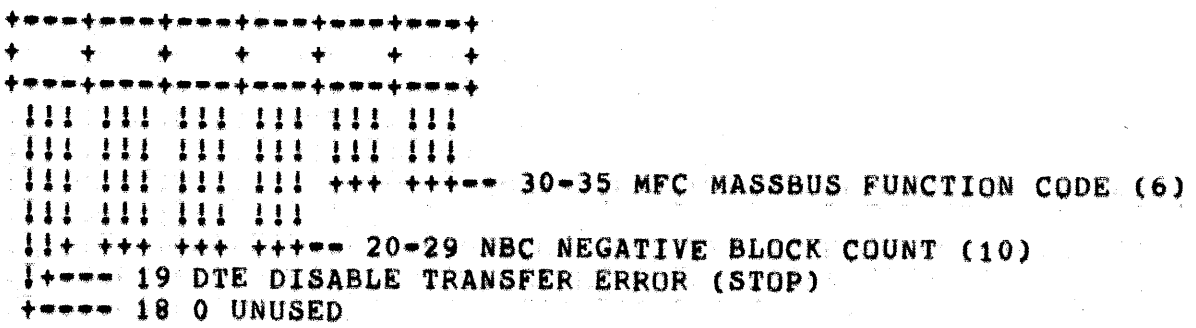
DATAI PRIMARY BLOCK ADDRESS REGISTER (PBA = 72) (RH)



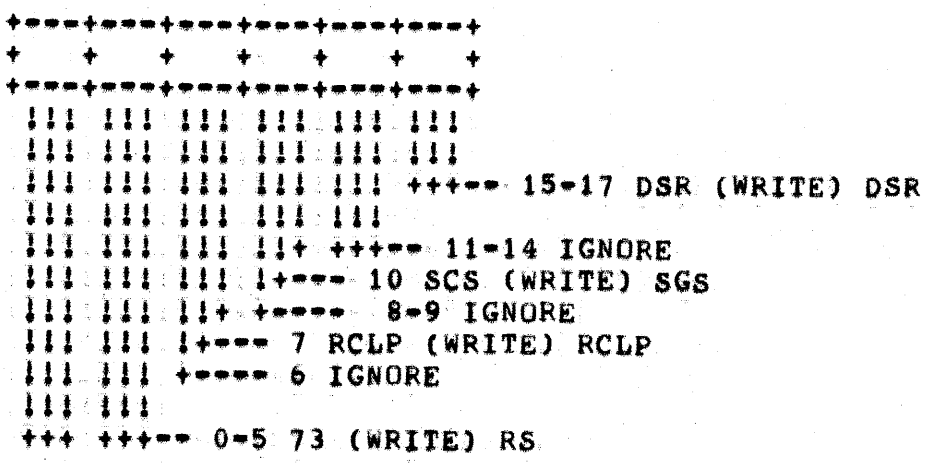
DATAO PRIMARY TRANSFER CONTROL REGISTER (PTC = 73) (LH)



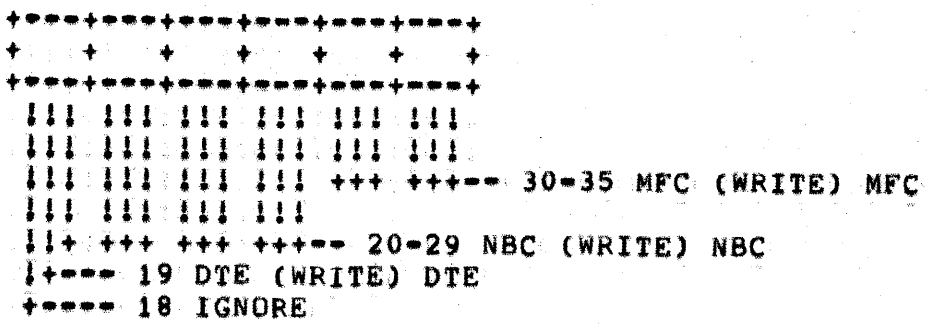
DATAO PRIMARY TRANSFER CONTROL REGISTER (PTC = 73) (RH)



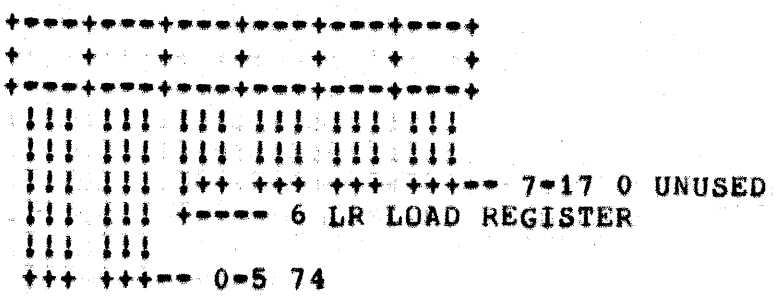
DATAI PRIMARY TRANSFER CONTROL REGISTER (PTC = 73) (LH)



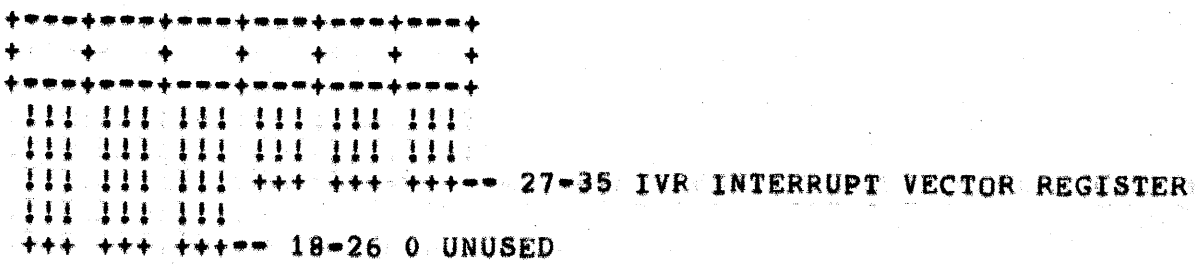
DATAI PRIMARY TRANSFER CONTROL REGISTER (PTC = 73) (RH)



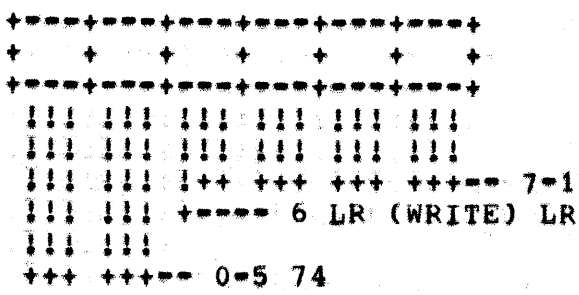
DATAO INTERRUPT VECTOR INDEX REGISTER (IVI = 74) (LH)



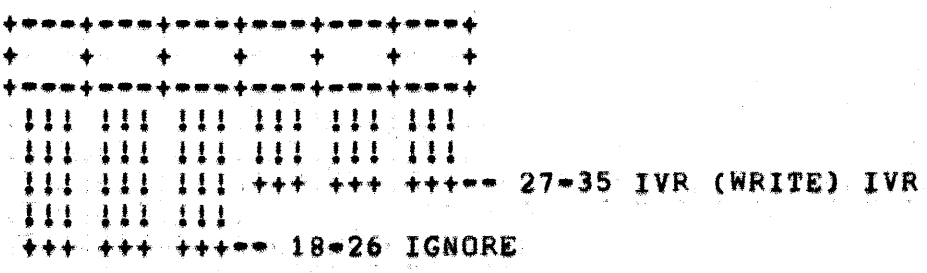
DATAO INTERRUPT VECTOR INDEX REGISTER (IVI = 74) (RH)



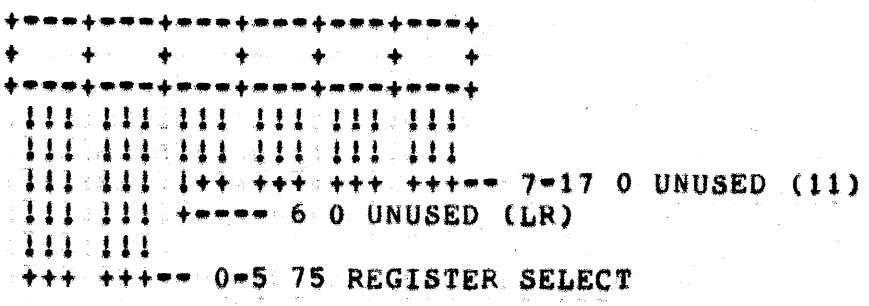
DATAI INTERRUPT VECTOR INDEX REGISTER (IVI = 74) (LH)



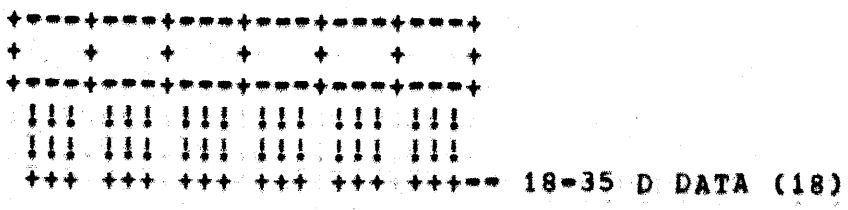
DATAI INTERRUPT VECTOR INDEX REGISTER (IVI = 74) (RH)



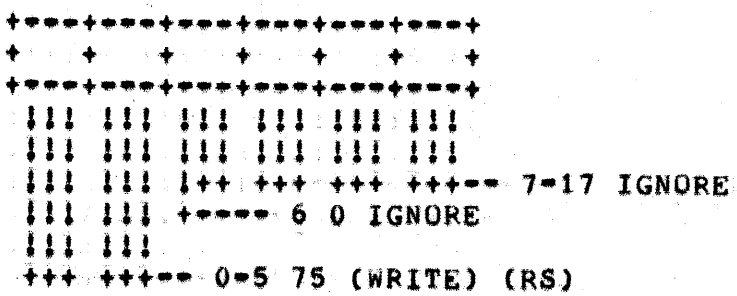
DATAI READ REGISTER (RR = 75) (LH)



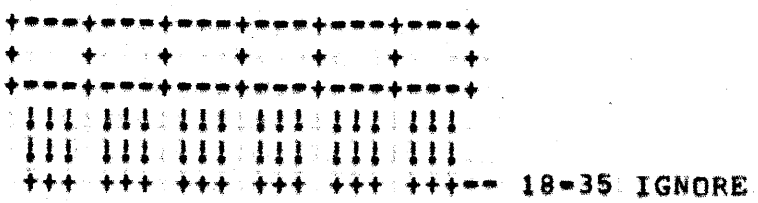
DATAI READ REGISTER (RR = 75) (RH)



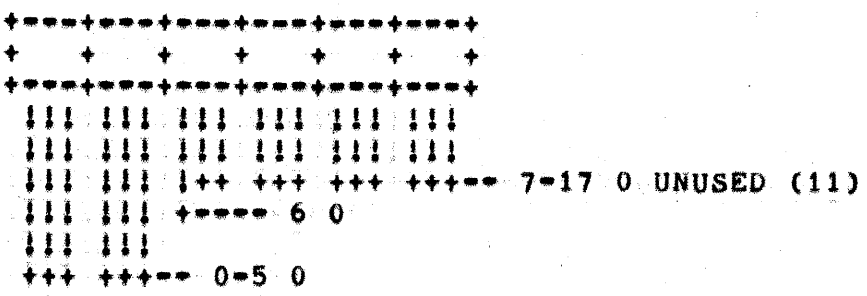
DATAO READ REGISTER (RR = 75) (LH)



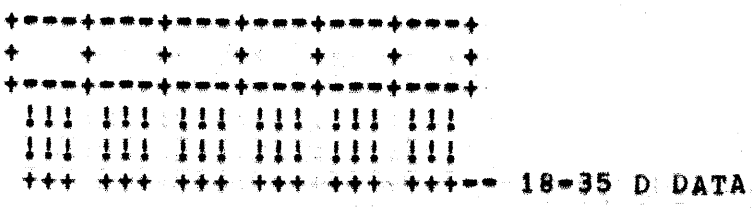
DATAO READ REGISTER (RR = 75) (RH)



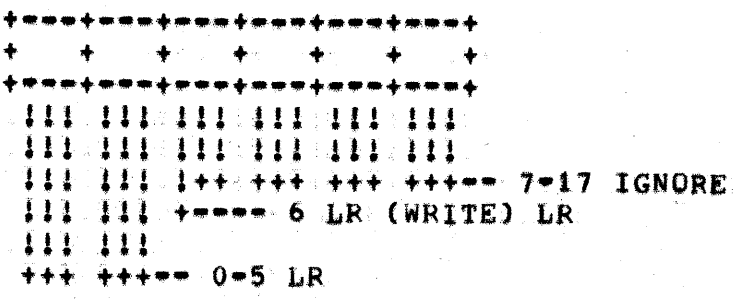
DATAI WRITE REGISTER (WR = 76) (LH)



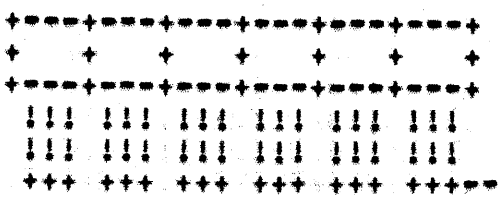
DATAI WRITE REGISTER (WR = 76) (RH)



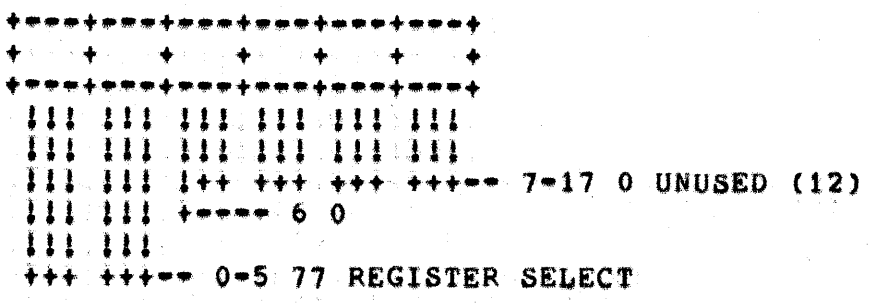
DATAO WRITE REGISTER (WR = 76) (LH)



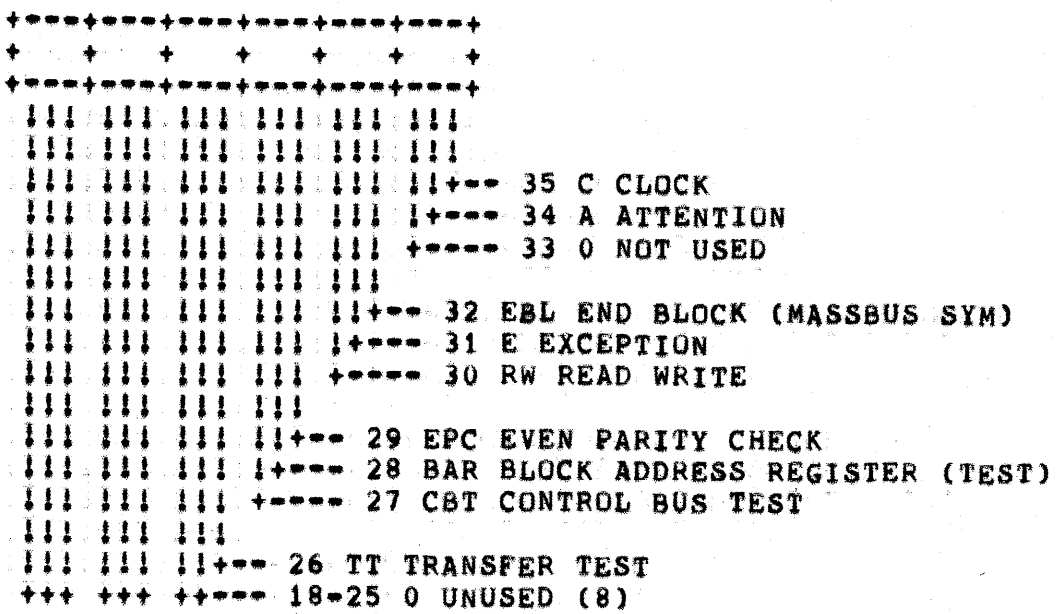
DATAO WRITE REGISTER (WR = 76) (RH)



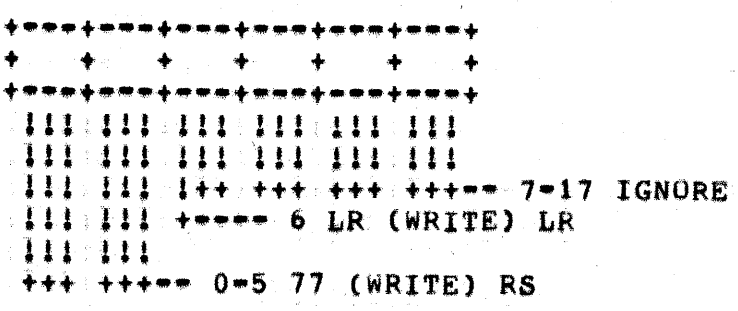
DATAO DIAGNOSTIC CONTROL REGISTER (DCR = 77) (LH)



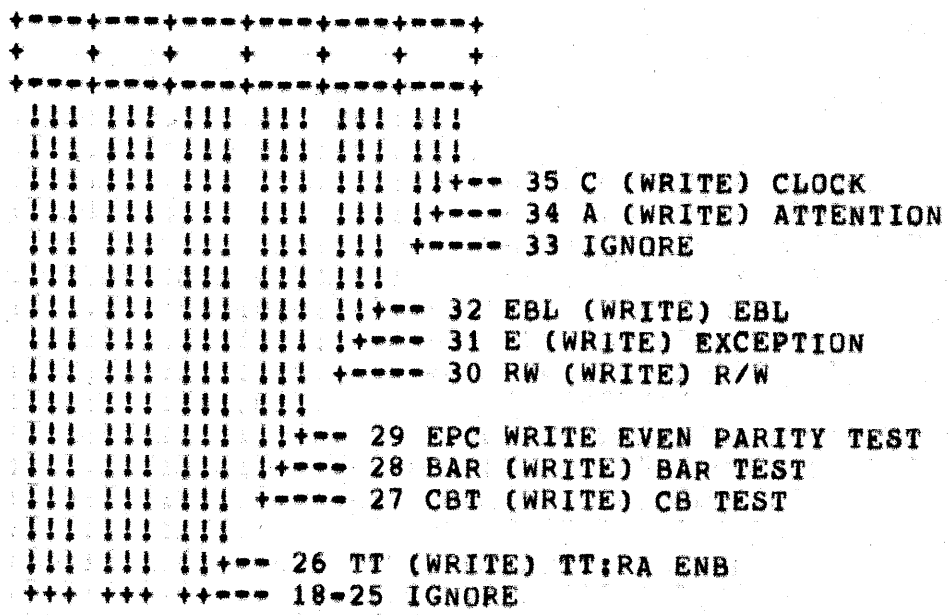
DATAO DIAGNOSTIC CONTROL REGISTER (DCR = 77) (RH)



DATAI DIAGNOSTIC CONTROL REGISTER (DCR = 77) (LH)



DATAI DIAGNOSTIC CONTROL REGISTER (DCR = 77) (RH)



[END CH4S01,SPC]

1080,2040,2060 ENGINEERING FUNCTIONAL SPEC - CHAP 4,3

TO: KL10 LIST, J. PARSLOW (200 FILE), D. NIGRO (NOTEBOOK)

TITLE: FRONT END INTERFACE (DTE20) - REV 7

STATUS: THIS CHAPTER INCLUDES THE RESULT OF THE FUNC SPEC REVIEW. IT INCLUDES JUD'S APRIL 2 IOP PROPOSAL. THIS CHAPTER DESCRIBES THE EBUS PARITY, UNIBUS PARITY, AND PI 0 FUNCTIONS WHICH ARE NOT IN THE BREADBOARD. LASTLY IT CONTAINS THE -11 LAND APPROVED VECTOR ADDRESSES.

FILE: [EFS]CH4S03.SPC

PDM #: 200-200-012-07

DATE: 23 APR 75

SUPERSEDED MEMOS: INITIAL SPEC FOR 10/11 INTERFACE, T. EGGERS, 10 OCT 72; DTE20 DEPOSIT/EXAMINE, J. LEONARD, 8 NOV 73

SUPERSEDED SPECS: 10/11 INTERFACE SPECIFICATION, W. BRUCKERT, 13 APRIL 73; DTE20.MEM, W. BRUCKERT, 26 MARCH 74

ENGINEER: W. BRUCKERT, T. EGGERS

APPROVED:

EDITOR: T. HASTINGS

TYPIST: J. MCCARTHY

REVIEWED: 28 MAY 74, 16 MAY 74

ABSTRACT

THE FRONT END INTERFACE OR DTE20 PROVIDES AN INTERFACE BETWEEN A KL10 CPU AND A PDP-11, CALLED A FRONT END. UP TO 4 DTE20S MAY BE CONNECTED TO A SINGLE KL10 CPU. UP TO 4 DTE20S MAY BE CONNECTED TO A SINGLE FRONT END. THE DTE20 ALLOWS THE PDP-11 TO PERFORM CONSOLE FUNCTIONS, SUCH AS EXAMINE AND DEPOSIT. THE DTE20 PROVIDES A VARIABLE-SPEED, MEMORY-TO-MEMORY BYTE TRANSFER CAPABILITY. THE PDP-11 DRIVES UNIT RECORD EQUIPMENT AND COMMUNICATION GEAR ATTACHED TO THE PDP-11 UNIBUS. FINALLY THE DTE20 ALLOWS THE PDP-11 TO DIAGNOSE THE KL10 CPU AND MANY OTHER COMPONENTS.

REVISION HISTORY

REV	DESCRIPTION	CHG NO	ORIG	DATE	APPD BY	DATE
0	INPUT TO REVIEW			28 MAR 74		
1	CHANGES FROM REVIEW + UNIBUS			10 MAY 74		
2	CHANGES FROM 16 MAY REVIEW			20 MAY 74		
3	LATEST UPDATE			3 SEP 74		
4	-11 LAND VECTOR ADDRESSES			2 DEC 74		
5	FINAL -11 LAND VECTOR ADDRESSES			20 JAN 75		
6	UPDATE EXAMINE/DEPOSIT			17 MAR 75		
7	ADDED CAUTION USING BIS			23 APR 75		(17 DEC)

0. CONTENTS

1. INTRODUCTION

EACH CPU IN A KL10 SYSTEM CAN HAVE FROM 1 TO 4 PDP-11S ATTACHED EACH SERVING AS A SO-CALLED FRONT END PROCESSOR. EACH PDP-11 IS CONNECTED TO THE KL10 BY A SEPARATE INTERFACE, CALLED THE DTE20. THE PURPOSES OF THE FRONT ENDS ARE VARIED. SOME PDP-11S CAN BE MULTI-PURPOSE WHILE OTHERS ARE DEDICATED. THE PURPOSES OF FRONT ENDS CAN BE:

1. HANDLING UNIT RECORD EQUIPMENT SUCH AS A CARD READER AND A LINE PRINTER.

2. HANDLING ASYNCHRONOUS COMMUNICATION GEAR.

3. HANDLING SYNCHRONOUS COMMUNICATION GEAR.

4. PROVIDING A LONG TERM POWER LINE FREQUENCY CLOCK.

5. REPLACING THE LIGHTS AND SWITCHES OF A TRADITIONAL COMPUTER CONSOLE WITH AN INTERACTIVE CONSOLE COMMAND LANGUAGE.

6. DIAGNOSING THE KL10 CPU AND OTHER FUNCTIONAL COMPONENTS.

7. RUNNING A DEDICATED REAL-TIME DATA ACQUISITION SYSTEM OR CONTROLLING AN ON-LINE EXPERIMENT. (NOT SUPPORTED IN SOFTWARE INITIALLY.)

8. BOOTSTRAPPING THE SYSTEM.

THE DTE20 INTERFACE HAS BEEN DESIGNED WITH THE ABOVE PURPOSES IN MIND. FROM 1 TO 4 DTE20S ARE BUILT INTO A KL10 CPU AND INTERFACE WITH IT USING THE EBUS. THE DTE20 GENERATES PARITY ON DEPOSIT DATA AND DETECTS PARITY ERRORS ON BYTE TRANSFER DATA AND EXAMINE DATA OVER THE EBUS. THE DTE20 CONNECTSTO THE PDP-11 AS A STANDARD UNIBUS PERIPHERAL USING INTERRUPT AND DEVICE ADDRESSES. UP TO 4 DTE20S CAN BE CONNECTED TO A PDP-11. IN A 4 KL10 CPU SYSTEM, THERE CAN BE 4 PDP-11S, WHERE EACH CAN TALK TO ALL OF THE KL10S, WITH AN EFFECTIVE CROSS-BAR SWITCH MADE UP OF 16 DTE20S. AT THE OTHER EXTREME, IT IS ALSO POSSIBLE TO HAVE 16 PDP-11S, WHERE EACH PDP-11 CAN ONLY TALK TO 1 KL10 CPU EACH USING 1 DTE20. COMBINATIONS IN BETWEEN ARE ALSO POSSIBLE. IT HAS NOT YET BEEN DETERMINED WHICH OF THESE POSSIBLE CONFIGURATIONS WILL BE SUPPORTED BY SOFTWARE. THE DTE20 USES BOTH THE NPR (DIRECT MEMORY ACCESS) AND VECTOR INTERRUPT FEATURES OF THE PDP-11. THE DTE20 ALSO CONTAINS THE LOGIC TO DETECT PDP-11 CORE MEMORY PARITY ERRORS DURING NPR TRANSFERS, IF THE MEMORY BEING ACCESSED CONTAINS THE PARITY OPTION (MF11UP). SEE MF11UP SPECIFICATION FOR ERROR REGISTER INFORMATION. FINALLY THE DTE20 FOLLOWS THE RECENTLY APPROVED UNIBUS PARITY SCHEME. SEE SECTION 13 FOR DTE20 DESCRIPTION. SEE APPENDIX FOR APPROVED UNIBUS PARITY SPECIFICATION.

THE DTE20 PROVIDES THE FOLLOWING CAPABILITIES:

1. CONSOLE FUNCTIONS OF EXAMINE AND DEPOSIT, RESTRICTED AND UNRESTRICTED.

2. DOORBELL FUNCTION WHERE THE -11 CAN INTERRUPT THE -10 AND VICE VERSA.

3. HIGH SPEED SIMULTANEOUS TWO-WAY TRANSFER OF VARIABLE BYTE DATA BETWEEN -11 AND -10 MEMORY.

4. DIAGNOSTIC BUS FOR THE -11 TO DIAGNOSE THE -10.

5. KL10 INITIATED BOOTSTRAP STARTUP OF THE PDP-11.

6. MECHANISM (DIAGNOSTIC BUS) TO LOAD MICRO-CODE INTO THE CRAM, EXECUTE PDP-10 INSTRUCTIONS, AND START AND STOP THE -10.

2. TERMINOLOGY

2.1 -11 OWNED COMMUNICATION REGION

AN AREA OF -10 MEMORY DEFINED BY THE DEPOSIT RELOCATION AND PROTECTION WORD IN THE EPT (EPTDPW), WRITTEN BY THE -11 USING PROTECTED DEPOSITS, AND READ BY THE -10. IT IS USED FOR COORDINATING STATUSES, PREPARING FOR BYTE TRANSFER OPERATIONS, AND PASSING LIMITED AMOUNTS OF DATA. EACH -11 IN THE SYSTEM HAS A SEPARATE -11 OWNED COMMUNICATION REGION IN -10 MEMORY WHICH IT ALONE CAN MODIFY.

2.2 -10 OWNED COMMUNICATION REGION

AN AREA OF -10 MEMORY DEFINED PURELY IN SOFTWARE, SEPARATE FROM THE -11 OWNED COMMUNICATION REGION, WRITTEN BY THE -10 AND READ BY THE -11 USING PROTECTED EXAMINES. IT IS USED FOR COORDINATING STATUSES, PREPARING BYTE TRANSFER OPERATIONS, AND PASSING LIMITED AMOUNTS OF DATA.

2.3 PRIVILEGED FRONT END

A PRIVILEGED FRONT END IS A PDP-11 ATTACHED TO A KL10 VIA A DTE20 WHICH CAN USE THE DIAGNOSTIC BUS AND CAN DO UNPROTECTED DEPOSITS. A MANUAL SWITCH (PRIVILEGED) ON EACH DTE20 IS USED TO INDICATE PRIVILEGED FRONT END OR RESTRICTED FRONT END FOR THE ATTACHED -11. A PRIVILEGED FRONT END CAN CRASH THE -10. [BREADBOARD AND 1080 ALWAYS PRIVILEGED]

2.4 RESTRICTED FRONT END

A RESTRICTED FRONT END IS A PDP-11 ATTACHED TO A KL10 VIA A DTE20 WHICH CANNOT CRASH THE KL10 IF THE -10 HARDWARE AND SOFTWARE ARE WORKING CORRECTLY. A RESTRICTED FRONT END IS PREVENTED FROM USING THE DIAGNOSTIC BUS. A RESTRICTED FRONT END CAN ONLY ACCESS -10 MEMORY AFTER THE -10 HAS DONE A CONO TO ALLOW USE OF DTE PI 0. WHEN SO ENABLED, THE RESTRICTED FRONT END CAN ONLY EXAMINE IN A -10 OWNED COMMUNICATION REGION AND CAN ONLY DEPOSIT IN ITS OWN -11 OWNED COMMUNICATION REGION.

2.5 NORMAL TERMINATION

A FLIP-FLOP AND INTERRUPT OCCURRING AT THE END OF A BYTE DATA TRANSFER. THE TERM DONE IS NOT USED BECAUSE UNLIKE ALL OTHER DONE FLAGS THE NORMAL TERMINATION FLAG IS NOT SET IF AN ERROR OCCURS. INSTEAD THE ERROR TERMINATION FLAG IS SET.

2.6 PROTECTED EXAMINES AND DEPOSITS

AN EXAMINE OR DEPOSIT WHICH IS RELOCATED AND RANGE CHECKED BY THE -10. THE RELOCATION AND PROTECTION FOR EXAMINE IS SEPARATE FROM THAT OF DEPOSIT. A PRIVILEGED FRONT END CAN OVERRIDE THE EXAMINE AND DEPOSIT PROTECTION CHECKS. A RESTRICTED FRONT END

CAN NOT OVERRIDE THE EXAMINE OR DEPOSIT PROTECTION CHECKS. [BREADBOARD: NOT PRESENT]

2.7 EXAMINE REGION

A REGION IN -10 MEMORY ACCESSED BY THE -11 USING PROTECTED EXAMINES.

2.8 DEPOSIT REGION

A REGION IN -10 MEMORY ACCESSED BY THE -11 USING PROTECTED DEPOSITS.

2.9 RELATIVE ADDRESS

AN ADDRESS SPECIFIED BY -11 SOFTWARE ON A PROTECTED EXAMINE OR DEPOSIT ADDRESS. THE ADDRESS IS RELATIVE TO THE EXAMINE OR DEPOSIT REGION AND SO RUNS FROM 0 TO THE MAX RELATIVE ADDRESS SPECIFIED BY THE -10 IN THE APPROPRIATE EPT WORD.

2.10 MBZ

ABBREVIATION FOR MUST BE ZERO. SOFTWARE MUST MAKE SURE A BIT LABELED WITH MBZ IS 0 WHEN WRITING A REGISTER. BITS SO LABELED ARE RESERVED TO DEC FOR FUTURE HARDWARE, ECOS, MICRO-CODE CHANGES. AT PRESENT THE BITS ARE IGNORED BY THE HARDWARE.

3. GOALS

3.1 SUPPORT THE FRONT END FUNCTIONS MENTIONED IN THE INTRODUCTION, SECTION 1.

3.2 RESTRICTED FRONT END CANNOT CRASH OR SERIOUSLY DEGRADE A PROPERLY PROGRAMMED KL10. FURTHERMORE A RESTRICTED FRONT END CANNOT VIOLATE THE SECURITY OF THE SYSTEM. THUS A RESTRICTED FRONT END HAS NO MORE PRIVILEGES OR CAPABILITIES THAN A USER PROGRAM. [BREADBOARD: PROTECTION NOT PRESENT]

3.3 ALLOW 18 BIT ADDRESSING IN PDP-11 MEMORY ON BYTE TRANSFERS FOR FUTURE GROWTH. HOWEVER THE TWO EXTENSION BITS ARE THE SAME FOR BOTH DIRECTIONS. [BREADBOARD: ONLY 16 BIT]

3.4 SUPPORT UNIBUS PARITY.

3.5 PROVIDE SUFFICIENT ERROR CHECKING AND DIAGNOSTIC FACILITIES TO HELP ISOLATE FAILURES TO THE BOARD LEVEL.

3.6 DESIGN HARDWARE SO THAT IT WILL BE COMPATIBLE WITH 11/05 AND 11/40 CPUS AND THEIR SUCCESSORS.

3.7 MINIMIZE CONFLICT OF DEVICE INTERRUPT AND REGISTER ADDRESSES BETWEEN REGULARLY SUPPORTED KL10 SYSTEM PDP-11 PERIPHERALS, LIKELY PDP-11 PERIPHERALS PURCHASED FROM DEC, CUSTOMER BUILT DEVICES, AND FUTURE DEC BUILT PDP-11 PERIPHERALS.

4. NON-GOALS

4.1 GATHER READ IS NOT PROVIDED. A HANDSHAKE WITH EXAMINE/DEPOSIT RITUAL MUST BE USED TO GATHER READ.

4.2 BYTE TRANSFERS CAN ONLY GO TO EXEC VIRTUAL ADDRESS SPACE IN SECTION 0.

4.3 CAN ONLY EXAMINE AND DEPOSIT IN THE CURRENT AC BLOCK WHILE SYSTEM RUNS. [BREADBOARD ALLOWS ALL AC BLOCKS BUT ONLY PHYSICAL ADDRESSES.]

4.4 BYTE POINTER IN EPT WILL DO NOTHING USEFUL WITH INDEX AND INDIRECT FIELD. SOFTWARE MUST MAKE SURE 0.

4.5 ALLOW BYTE TRANSFERS INTO EXEC VIRTUAL EXTENDED SECTIONS USING DOUBLE WORD BYTE POINTERS.

4.6 THERE IS NO POSITIVE ERROR STATUS BIT FOR NXM IN -11 MEMORY. NXM CAN BE IDENTIFIED ONLY BY THE ABSENCE OF ALL OTHER SPECIFIC ERROR BITS WHEN THE GENERAL ERROR BIT IS SET.

5. CONVENTIONS USED IN SPEC

BECAUSE -11 "OUTPUT" IS -10 "INPUT", THE TERMS INPUT AND OUTPUT ARE NOT USED. INSTEAD DIRECTION IS SPECIFIED IN TERMS OF ITS DESTINATION, I.E., TO-10 OR TO-11. PDP-11 BIT NUMBERS LESS THAN 10 HAVE A LEADING ZERO, WHILE -10 BIT NUMBERS DO NOT. ALL -11 REGISTERS ARE LABELED ACCORDING TO PDP-11 CONVENTIONS, WHERE THE LEAST SIGNIFICANT BIT IS 00. THE ONLY EXCEPTION TO THIS RULE IS THE DS LINES IN DIAG1 IN WHICH THE MOST SIGNIFICANT BIT IS DS0. ALL -10 REGISTERS AND LOCATIONS ARE LABELED ACCORDING TO PDP-10 CONVENTIONS, WHERE THE MOST SIGNIFICANT BIT IS 0. BIT ASSIGNMENTS FOR -11 AND -10 WORDS ARE ALWAYS LISTED FROM LEFT TO RIGHT. ALL REGISTERS, AS WELL AS THE BITS IN EACH REGISTER, HAVE BEEN GIVEN MNEMONICS. THUS CONSISTENCY BETWEEN PRINTS, MANUALS, DIAGNOSTICS, SYSTEM SOFTWARE AND DECALS ON HARDWARE IS POSSIBLE.

UNLESS OTHERWISE SPECIFIED ALL BITS WHICH THE SOFTWARE CAN READ FROM REGISTERS IN THE -10 AND -11 REFLECT THE CURRENT STATE OF THE HARDWARE (EVEN THOUGH A RAM IS SOMETIMES USED TO HOLD A COPY OF THE REGISTERS). EXCEPTIONS ARE INDICATED BY SAYING: READ RETURNS THE VALUE STORED BY THE SOFTWARE ON THE LAST REGISTER WRITE.

THE NEXT FEW SECTIONS DESCRIBE HOW TO PROGRAM THE VARIOUS FUNCTIONS IN THE MOST STRAIGHTFORWARD WAY. THE REMAINING SECTIONS DESCRIBE THE REGISTER LAYOUTS IN DETAIL.

6. SUMMARY OF REGISTERS (N = DTE20 INTERFACE # 0-3)

6.1 PDP-10 ACCESSABLE REGISTERS (DTE0 = 200, DTE1 = 204, DTE2 = 210, DTE3 = 214)

LOCATION		FUNCTION
CONI	DTEN,	DTE20 STATUS, INTERRUPT FLAGS, ERRORS, ETC.
CONO	DTEN,	DTE20 STATUS AND CONTROL
DATAO	DTEN,	BYTE COUNT FOR TO-10 BYTE TRANSFERS
DATAI	DTEN,	RETURN ZEROS (RESERVED FOR DEC FOR FUTURE HARDWARE)

6.2 PDP-10 MEMORY LOCATIONS IN THE EPT

EIGHT LOCATIONS ARE ASSIGNED TO EACH DTE20 (N = 0,1,2,3).

LOCATION	SYMBOL	NAME
140 + 8*N	EPTEBP	TO ELEVEN BYTE POINTER
141 + 8*N	EPTTBP	TO TEN BYTE POINTER
142 + 8*N	EPTDII	DTE20 INTERRUPT INSTRUCTION
143 + 8*N	UNUSED	
144 + 8*N	EXAMINE	PROTECT WORD
145 + 8*N	EXAMINE	RELOC WORD
146 + 8*N	DEPOSIT	PROTECT WORD
147 + 8*N	DEPOSIT	RELOC WORD

6,3 PDP-11 LOCATIONS (BASE = 774400 + 40*N, WHERE N = 0-3)

NOTE: ADDRESSES 0-26 ARE STORED IN RAM AND ARE NOT AFFECTED BY UNIBUS INIT. ADDRESSES 30-36 ARE IMPLEMENTED AS REGISTERS AND ARE INITIALIZED BY UNIBUS INIT.

ADDRESS	SYMBOL	USE	NAME
BASE + 0	DLYCNT	TO-10 TO-11	DELAY COUNT
BASE + 2	DEXWD3	DEP/EX	DEPOSIT/EXAMINE DATA WORD 3 [BREADBOARD: DEXWD1]
BASE + 4	DEXWD2	DEP/EX	DEPOSIT/EXAMINE DATA WORD 2
BASE + 6	DEXWD1	DEP/EX	DEPOSIT/EXAMINE DATA WORD 1 [BREADBOARD: DEXWD3]
BASE + 10	TENAD1	DEP/EX	10 ADDRESS WORD 1 FOR DEPOSIT/EXAMINE
BASE + 12	TENAD2	DEP/EX	10 ADDRESS WORD 2 FOR DEPOSIT/EXAMINE
BASE + 14	TO10BC	TO-10	TO-10 BYTE COUNT
BASE + 16	TO11BC	TO-11	TO-11 BYTE COUNT
BASE + 20	TO10AD	TO-10	TO-10 PDP-11 MEMORY ADDRESS
BASE + 22	TO11AD	TO-11	TO-11 PDP-11 MEMORY ADDRESS
BASE + 24	TO10DT	TO-10	TO-10 PDP-11 DATA WORD
BASE + 26	TO11DT	TO-11	TO-11 PDP-11 DATA WORD
BASE + 30	DIAG1	DIAG	DIAGNOSTIC WORD 1
BASE + 32	DIAG2	DIAG	DIAGNOSTIC WORD 2
BASE + 34	STATUS	ALL	STATUS WORD
BASE + 36	DIAG3	TO-10 DIAG	DIAGNOSTIC WORD 3

6,4 PDP-11 CORE LOCATIONS

DTE20 NUMBER	PDP-11 DEVICE (BASE) REGISTERS	PDP-11 VECTOR INTERRUPT	PDP-10 DEVICE CODE	PDP-10 SYMBOL
0	774400	774	200	DTE0
1	774440	770	204	DTE1
2	774500	764	210	DTE2
3	774540	760	214	DTE3

6,5 EBUS COMMUNICATION

SEE CHAPTER 5,2, EBUS - EBOX TO RH20, DTE20, DIA20 INTERFACE

IOP	FUNCTION	2 VECTOR INTERRUPT [BREADBOARD: NOT USED]
IOP	FUNCTION	4 DATAO
IOP	FUNCTION	5 DATAI
IOP	FUNCTION	6 BYTE TRANSFER

THE DTE20 ALLOWS THE SOFTWARE TO SET THE FOLLOWING FIELDS OF THE 36 BIT IOP FUNCTION WORD,

1. SPACE FIELD [0-2] SPECIFYING ADDRESS SPACE
2. UNUSED BITS [11-12]
3. ADDRESS FIELD [13-35]

THE PI BOARD SUPPLIES THE PHYSICAL CONTROLLER FIELD [7-10], THE DTE20 ASSERTS QUALIFIER [6] FOR ALL EXAMINES AND DEPOSITES BY A RESTRICTED FRONT END WHETHER PROTECTED OR NOT, THE DTE20 ASSERTS QUALIFIER [6] FOR ALL PROTECTED EXAMINES AND DEPOSITS BY A PRIVILEGED FRONT END AND DOES NOT ASSERT IT IF THE PRIVILEGED FRONT END MAKES AN UNPROTECTED EXAMINE OR DEPOSIT,

7. CONSOLE FUNCTIONS

ONE OF THE FUNCTIONS OF THE -11 IS TO PROVIDE THE TRADITIONAL COMPUTER CONSOLE FUNCTIONS. THE -11 IS PROGRAMMED TO ACCEPT CONSOLE COMMANDS TO DISPLAY LOCATIONS IN -10 MEMORY, START THE KL10, STOP THE KL10, CHANGE LOCATIONS IN KL10 MEMORY, ETC. THIS IS DONE ON ONE OR MORE OF THE TERMINALS CONNECTED TO THE -11. SEE CHAPTER 1.6, CONSOLE FUNCTIONS, FOR A DESCRIPTION OF THE CONSOLE COMMANDS.

7.1 CONSOLE SWITCHES

THE DTE20 PROVIDES EXAMINE AND DEPOSIT FUNCTIONS SO THAT THE PDP-11 CAN FETCH OR CHANGE ANY LOCATION IN THE COMMUNICATION AREA OF THE KL10 PHYSICAL MEMORY WHILE THE SYSTEM IS RUNNING OR EXECUTING A HALT. THE EBOX CLOCK MUST BE RUNNING [BREADBOARD: A PI MUST BE ASSIGNED AND CANNOT BE HALTED]. THE EXAMINE AND DEPOSIT FUNCTIONS (AND BYTE TRANSFERS) ARE HANDLED AS A PI REQUEST AND HAS PRIORITY HIGHER THAN ANY PROGRAMMED PI LEVEL. IT IS REFERRED TO AS PI 0, EVEN THOUGH EXAMINE AND DEPOSIT WORK WHEN THE PI SYSTEM IS TURNED OFF AND THE CPU IS HALTED. [BREADBOARD: EXAMINE AND DEPOSIT WORK AT SAME PROGRAMMED PI CHANNEL AS FOR BYTE TRANSFERS.]

THERE ARE TWO TYPES OF EXAMINE/DEPOSIT FUNCTION, PROTECTED AND UNPROTECTED. PROTECTED EXAMINES AND DEPOSITS ARE NORMAL FOR PRIVILEGED AND RESTRICTED FRONT ENDS. EXAMINES ARE RELOCATED AND PROTECTED SEPARATELY FROM DEPOSITS. IN THIS WAY THE EXAMINE REGION CAN BE MADE TO INCORPORATE ALL OF THE REGIONS WHICH THE -10S AND THE -11S CAN DEPOSIT INTO, WHILE THE DEPOSIT REGION CAN BE SEPARATE FOR EACH -11. UNPROTECTED EXAMINES AND DEPOSITS ARE ABNORMAL, REQUIRE SPECIAL PROGRAMMING, AND MAY ADDRESS ANY AREA IN -10 MEMORY IN ANY OF THE FOLLOWING ADDRESS SPACES: (1) EXEC PROCESS TABLE, (2) EXEC VIRTUAL ADDRESS, (3) PHYSICAL. IN ORDER FOR A RESTRICTED FRONT END TO DO EXAMINES OR DEPOSITS OF ANY KIND, THE -10 MUST ENABLE PI 0. PRIVILEGED FRONT ENDS CAN ALWAYS USE PI 0 FOR ANY KIND OF EXAMINE OR DEPOSIT.

7.2 CONSOLE LIGHTS

OTHER CONSOLE FUNCTIONS, SUCH AS DISPLAYING THE CONTENTS OF CERTAIN CPU REGISTERS OR MEMORY LOCATIONS, MUST BE DONE WITH THE COOPERATION OF THE OPERATING SYSTEM. THE OPERATING SYSTEM MUST PERIODICALLY STORE THE QUANTITIES TO BE DISPLAYED IN THE COMMUNICATIONS AREA. INFORMATION SO DISPLAYED MAY INCLUDE THE STATE OF THE PI SYSTEM, THE CURRENT JOB NUMBER BEING RUN, THE NUMBER OF ACTIVE JOBS, THE PROGRAM COUNTER ON THE LAST CLOCK INTERRUPT, ETC. IT IS POSSIBLE TO SIMULATE ALL OF THE KI CONSOLE LIGHTS USED WHILE THE SYSTEM IS IN NORMAL OPERATION. MAJOR KL CPU STATE INFORMATION IS CONTINUOUSLY AVAILABLE ON THE 7 DS LINES IN DIAG1 WHILE THE SYSTEM RUNS. IF THE -10 CRASHES, THE PDP-11 CAN USE THE DIAGNOSTIC BUS TO DETERMINE ADDITIONAL

HARDWARE STATUS INFORMATION. THE PDP-11 MUST NOT USE THE DIAGNOSTIC BUS FOR DATA TRANSFERS DURING NORMAL SYSTEM OPERATION BECAUSE IT WILL INTERFERE WITH TRAFFIC ON THE EBUS.

7.3 COMMUNICATION REGION RELOCATION AND PROTECTION [BREADBOARD: NOT PRESENT]

COMMUNICATION REGIONS ARE THE NORMAL MEANS FOR COMMUNICATION CONTROL INFORMATION BETWEEN -10S AND -11S. FOR RESTRICTED FRONT ENDS, THE -10 MUST ENABLE PI 0 BEFORE EXAMINES OR DEPOSITS (PROTECTED ONLY ALLOWED) CAN BE DONE. FOR RELIABILITY, EACH -10 AND EACH -11 HAS A COMMUNICATION REGION WHICH IT ALONE CAN WRITE. EACH -10 AND EACH -11 CAN BE SET UP TO ALLOW READING ALL OF THE COMMUNICATION REGIONS IN THE SYSTEM. AGAIN FOR RELIABILITY, THE -11S ALWAYS USE PROTECTED EXAMINES AND DEPOSITS FOR DEALING WITH COMMUNICATION REGIONS, BECAUSE THE KL10 RELOCATES ALL PROTECTED EXAMINES AND DEPOSITS THE -11 SOFTWARE DEALS WITH RELATIVE ADDRESSES WHICH ARE RELATIVE WITH RESPECT TO THE EXAMINE OR DEPOSIT REGIONS NO MATTER WHICH OF THE 4 FRONT ENDS IT IS. FIRST THE KL10 (MICRO-CODE) TAKES THE RELATIVE EXAMINE OR DEPOSIT ADDRESS AND COMPARES IT WITH THE PROTECTION WORD IN THE EPT FOR THAT DTE20, IF THE RELATIVE ADDRESS EXCEEDS THE PROTECTION THEN AN EXAMINE RETURNS 0 AND A DEPOSIT DOES NOTHING. IF THE RELATIVE ADDRESS IS LESS THAN OR EQUAL TO THE PROTECTION WORD, THE RELATIVE ADDRESS IS ADDED TO THE 23 BIT PHYSICAL ADDRESS STORED IN THE RELOCATION WORD FOR THAT DTE20. THE RELOCATED ADDRESS IS THEN USED TO REFERENCE THE DATA. ALL PAGING VIOLATIONS WILL CAUSE AN IO PAGE FAIL INTERRUPT IN THE APR AND ARE CONSIDERED SERIOUS SYSTEM ERRORS.

THE EXAMINE PROTECTION WORDS ARE STORED IN EPT LOCATIONS:

144 + N*8	EPTEPW	EXAMINE PROTECTION
145 + N*8		EXAMINE RELOCATION

THE DEPOSIT PROTECTION WORDS ARE STORED IN EPT LOCATIONS:

146 + N*8	EPTDPW	DEPOSIT PROTECTION
147 + N*8		DEPOSIT RELOCATION

A PROTECTION WORD SPECIFIES THE NUMBER OF WORDS ACESSIBLE. E.G., AN EXAMINE PROTECTION WORD OF 100 SPECIFIED THAT THE ONLY LEGAL ADDRESSES AN -11 MAY SPECIFY FOR PROTECTED EXAMINES ARE THOSE IN THE RANGE 0-77. A RELOCATION WORD SPECIFIES THE PHYSICAL ADDRESS TO WHICH PROTECTED ADDRESS 0 WILL BE RELOCATED. THUS, IF THE DEPOSIT PROTECTION WORD IS 20 AND THE DEPOSIT RELOCATION WORD IS 24610, THE -11 MAY STORE INTO PHYSICAL LOCATIONS 24610 THROUGH 24627 BY USE OF ADDRESES 0-17.

7.4 PROGRAMMING - COMMON TO EXAMINE AND DEPOSIT

IN ORDER TO SPECIFY A 36 BIT PDP-10 DATA WORD, THREE -11 WORDS ARE USED. THEY ARE DEPOSIT/EXAMINE DATA WORD 1 (DEXWD1), DEPOSIT/EXAMINE DATA WORD 2 (DEXWD2), AND DEPOSIT/EXAMINE DATA WORD 3 (DEXWD3).

IN ORDER TO SPECIFY A 23 BIT PDP-10 ADDRESS, TWO -11 WORDS ARE USED. THEY ARE TEN ADDRESS WORD 1 (TENAD1) AND TEN ADDRESS WORD 2 (TENAD2). THE HIGH ORDER PART OF TENAD1 IS USED FOR CONTROL. TENAD1[DEP] SPECIFIES WHETHER AN EXAMINE OR DEPOSIT IS TO BE DONE. FOR A PRIVILEGED FRONT END TENAD1[PRTOFF] CAN BE SET BY THE SOFTWARE IN ORDER TO PERFORM AN UNPROTECTED EXAMINE OR DEPOSIT. ON UNPROTECTED OPERATIONS, THE SPACE FIELD SPECIFIES THE TYPE OF ADDRESS: EXEC VIRTUAL, EXEC PROCESS TABLE (EPT), OR PHYSICAL. [BREADBOARD: ONLY PHYSICAL IS PROVIDED. HOWEVER THE BREADBOARD CAN EXAMINE ANY AC BLOCK.] AN EXEC VIRTUAL, OR PHYSICAL ADDRESS, LESS THAN 20 REFERENCES THE CURRENT AC BLOCK OF THE MACHINE. (REMEMBER THE KL10 DIFFERS FROM THE KI IN THAT THERE IS NO DISTINCTION BETWEEN EXEC ACS AND USER ACS. THE FIRST 7 OF THE 8 AC BLOCKS CAN BE USED FOR EITHER EXEC OR USER MODE.)

THE EXAMINE OR DEPOSIT FUNCTION IS STARTED WHEN THE -11 PROGRAM WRITES TENAD2. NO PROGRAM INTERRUPTS ARE GENERATED ON THE -10 OR THE -11 SIDE TO SIGNAL COMPLETION OF THE EXAMINE OR DEPOSIT. THEREFORE THE -11 PROGRAM MUST CHECK FOR COMPLETION BY LOOKING AT THE STATUS [DEXDON] BIT. THE DTE20 CLEARS DEXDON WHEN THE -11 WRITES TENAD2, SO THE SOFTWARE NEVER NEEDS TO. DATA IN TENAD1, TENAD2, DEXWD1, DEXWD2, DEXWD3 REMAIN INTACT AFTER AN OPERATION. THEREFORE THE -11 MAY PERFORM REPEATED PROTECTED EXAMINES OR DEPOSITS, MERELY BY WRITING THE TENAD2 WORD EACH TIME. AN EXAMINE FOLLOWED BY A DEPOSIT (CHANGING ONLY TENAD1 AND TENAD2) WILL RESULT IN MOVING DATA FROM ONE KL10 CORE LOCATION TO ANOTHER. FOR UNPROTECTED OPERATIONS, THE -11 MUST RELOAD THE PROTECT OFF BIT (TENAD1[PRTOFF] BETWEEN EACH OPERATION.

8. DOORBELL FUNCTION

THE DOORBELL FUNCTION ALLOWS EACH -10 TO INTERRUPT EACH -11 CONNECTED BY A DTE20 AND VICE VERSA. THE -11S CANNOT INTERRUPT EACH OTHER.

THE DOORBELL CONSISTS OF A PROGRAMMABLE INTERRUPT AND A STATUS BIT. IN ORDER FOR THE PDP-11 TO INTERRUPT THE PDP-10, THE -11 SETS THE REQUEST 10 INTERRUPT FLOP (INT10S = BIT 08) IN THE -11 STATUS WORD, STATUS. WHEN THIS BIT IS SET, THE DTE20 GENERATES AN INTERRUPT IN THE KL10 WITH A STATUS BIT SET IN THE CONI WORD (TO10DB = BIT 26) INDICATING THAT THE PDP-11 CPU HAS PROGRAMMED AN INTERRUPT OF THE KL10. THIS PROCEDURE WORKS IN A REVERSED BUT IDENTICAL MANNER FOR THE KL10 INTERRUPTING THE PDP-11. THE -10 SETS THE 10 REQUESTING 11 INTERRUPT BY DOING A CONO DTEX, DOOR. THE -11 DISCOVERS THE CAUSE FOR THE INTERRUPT BY LOOKING AT BIT TO11DB = BIT 11 IN STATUS. COMMUNICATION IS DONE VIA A WORD (OR WORDS) IN THE COMMUNICATION REGION IN KL10 MEMORY. A WORD OR WORDS IS CHOSEN AND DEPOSIT AND EXAMINE FEATURES ARE USED BY THE -11 TO GAIN ACCESS TO THESE WORDS. THIS MECHANISM IS USED BY BOTH PROCESSORS TO INDICATE TO THE OTHER PROCESSOR THAT IT IS POWERING DOWN. FOR INSTANCE, IF THE KL10 NOTICES THAT ITS POWER IS DISAPPEARING, IT WILL SET A BIT IN A WORD WHICH IS ASSIGNED FOR POWER FAILURE NOTIFICATION. THE KL10 THEN INTERRUPTS THE PDP-11. THE PDP-11, AS PART OF ITS STANDARD INTERRUPT ROUTINE, WILL ALWAYS CHECK FOR THE KL10 POWERFAIL BIT IN THE COMMUNICATION REGION. IN THIS WAY, THE PDP-11 IS NOTIFIED THAT THE KL10 POWER IS DISAPPEARING. IN A SIMILAR WAY THE -11 COULD INTERRUPT THE -10 ON EVERY TICK OF THE POWER LINE CLOCK (50 OR 60 HZ).

THE DOORBELLS CONSIST OF ONE FLOP FOR EACH DIRECTION. TO AVOID LOSING INTERRUPTS ON THE -11 SIDE, THE -11 PROGRAM SHOULD CLEAR THE DOORBELL, CHECK TO SEE IF IT CLEARED (IN CASE THE -10 WAS TRYING TO SET IT AT THE SAME TIME), TRY AGAIN IF IT DIDN'T CLEAR, AND THEN POLL FOR THE CAUSE OF THE INTERRUPT.

9. BYTE TRANSFER FUNCTION - COMMON TO-10 AND TO-11

DURING THE BYTE TRANSFER FUNCTION THE DTE20 TRANSFERS FIELDS OF INFORMATION BETWEEN THE PDP-11 AND THE EBOX. ON THE KL10 SIDE, THE FIELDS ARE OF VARIABLE LENGTH AND ARE ACCESSED THROUGH A PDP-10 BYTE POINTER. ON THE PDP-11 SIDE THE FIELDS ARE EITHER 8 BITS WIDE AND ARE STORED IN CONSECUTIVE BYTES OR ARE 16 BITS WIDE AND ARE STORED IN CONSECUTIVE WORDS. IF THE FIELD INTO WHICH THE INFORMATION IS BEING STORED IS NARROWER THAN THE FIELD FROM WHICH IT WAS READ, AS MANY OF THE RIGHTMOST BITS AS WILL FIT ARE STORED. IF THE FIELD INTO WHICH THE INFORMATION IS BEING STORED IS WIDER THAN THE FIELD FROM WHICH IT WAS READ, THE INFORMATION IS RIGHT ALIGNED AND PADDED WITH ZEROS ON THE LEFT.

IN ORDER TO PERFORM A TRANSFER THE FOLLOWING ACTIONS MUST BE DONE:

1. THE -11 SHOULD SPECIFY THE TRANSFER RATE AND UNIBUS ADDRESS BITS 17-16 (CAN BE DONE ONCE AT SYSTEM STARTUP). IF IT IS NOT SPECIFIED, AN UNDETERMINED TRANSFER RATE WILL OCCUR TO ONE OF THE FOUR 32K MEMORY REGIONS.

2. THE -11 MUST SPECIFY WHETHER BYTE OR WORD MODE IS TO BE USED IN THE -11.

3. THE SENDER MUST SPECIFY THE ADDRESS OF THE SOURCE STRING.

4. THE RECEIVER MUST SPECIFY THE ADDRESS OF THE DESTINATION STRING.

5. THE RECEIVER MUST SPECIFY THE BYTE COUNT OF THE STRING TO BE RECEIVED.

6. THE RECEIVER MUST SPECIFY WHETHER IT ALONE (SCATTER WRITE) OR BOTH CPUS ARE TO RECEIVE NORMAL TERMINATION INTERRUPTS (I BIT = 1).

BYTES MAY BE STORED IN THE -11 AS EITHER ONE BYTE PER PDP11 16-BIT WORD (1 TO 16 BITS OF DATA) OR ONE BYTE PER 8-BIT PDP-11 BYTE (1 TO 8 BITS OF DATA). BYTE ADDRESSES ARE SPECIFIED IN THE -10 USING REGULAR KL10 BYTE POINTERS IN THE EPT. BYTE POINTERS ARE INTERPRETED IN EXEC VIRTUAL ADDRESS SPACE, SECTION 0 ONLY. WARNING: THE INDEX FIELD OF THE BYTE POINTERS SHOULD BE ZERO. OTHERWISE THE EBOX WILL INDEX USING THE CURRENT CONTENTS OF THE EXEC OR USER INDEX REGISTER AT THE TIME OF THE TRANSFER. INDIRECTION SHOULD NOT BE USED, SINCE THE INDIRECT WORD WILL NOT BE INCREMENTED AS WITH ALL BYTE POINTER OPERATIONS. WARNING: BYTE SIZES GREATER THAN 16 BITS FOR TO-11 TRANSFERS CAN CAUSE FALSE PARITY ERRORS. SEE SECTION 13.

9.1 SUGGESTED PROTOCOLS

THERE ARE 3 FORMS (MAYBE MORE) OF PROTOCOL POSSIBLE FOR HANDLING BYTE TRANSFERS DEPENDING ON HOW THE LENGTH OF THE BYTE STRING IS DETERMINED. SCATTER WRITE IN THE RECEIVER IS PROVIDED BY THE HARDWARE, BUT GATHER READ IN THE SENDER IS NOT.

FOR SAFETY REASONS THE RECEIVER IS IN CONTROL OF HOW MANY CHARACTERS ARE TO BE TRANSFERRED BEFORE THE RECEIVER GETS A NORMAL TERMINATION INTERRUPT. THE RECEIVER ALSO DETERMINES WHETHER THE SENDER WILL RECEIVE A NORMAL TERMINATION INTERRUPT OR NOT FOR THE CURRENT PART OF THE STRING (I BIT).

9.1.1 PRE-AGREED STRING LENGTH METHOD - THE FIRST METHOD IS FOR THE SENDER TO TELL THE RECEIVER HOW LONG THE STRING IS USING EXAMINE/DEPOSIT. THE RECEIVER THEN SETS UP THE BYTE COUNT AND SPECIFIES THAT BOTH CPUS ARE TO BE INTERRUPTED AT TERMINATION (I BIT = 1).

9.1.2 STRING LENGTH IN HEADER METHOD - THE SECOND METHOD IS FOR THE SENDER TO PASS THE LENGTH OF THE STRING AS THE FIRST FEW BYTES OF THE STRING (A HEADER). THE RECEIVER SETS A BYTE COUNT WHICH IS A GUARANTEED MINIMUM, AND REQUESTS TO INTERRUPT THE RECEIVER ONLY (I BIT = 0). THE RECEIVER FIELDS THE INTERRUPT, READS THE BYTE COUNT FROM THE STRING, SETS UP A NEW BYTE COUNT AND POSSIBLY A NEW DESTINATION ADDRESS AND CONTINUES THE TRANSFER. BOTH CPUS GET THE NORMAL TERMINATION INTERRUPT (I BIT = 1) WHEN THE STRING IS COMPLETE. NOTICE THAT THIS SAME TECHNIQUE CAN BE USED IF THE RECEIVER WISHES TO BREAK A LONG STRING INTO SEVERAL SHORT BUFFERS TO ACHIEVE A SCATTER WRITE IN ITS MEMORY.

9.1.3 VARIABLE LENGTH (GOOD ONLY TO-11) METHOD - IN THE THIRD METHOD THE RECEIVER (-11) STILL SETS UP A BYTE COUNT, HOWEVER THE TRANSFER TERMINATES EARLY WHEN A NULL CHARACTER IS ENCOUNTERED (Z BIT = 1). THIS MODE IS USEFUL FOR TRANSFERRING ASCIZ STRINGS WHICH HAVE WIDE USE IN PDP-10 SOFTWARE. NOTICE THAT THE -11 CAN BREAK UP RECEIPT OF A LONG ASCIZ STRING USING THE I BIT DESCRIBED IN 9.1.2.

10. TO-10 BYTE TRANSFERS

SEE SECTION 9 FOR DESCRIPTION WHICH IS COMMON FOR BOTH DIRECTIONS. THE FOLLOWING DESCRIPTION ASSUMES THAT THE -10 HAS DISCOVERED THE LENGTH OF THE STRING WHICH IT IS TO RECEIVE USING ONE OF THE TWO METHODS DESCRIBED IN SECTION 9.1.

1. -11 WRITES THE DELAY COUNT WORD (DLYCNT) WITH THE NEGATIVE NUMBER OF 500 NANO SECOND INTERVALS TO DELAY BETWEEN BYTE TRANSFERS ACROSS THE DTE20 IN BITS 13-00. BITS 15-14 SPECIFY UNIBUS ADDRESS BITS 17-16 FOR BOTH TO-10 AND TO-11 BYTE TRANSFERS. (DLYCNT [15-00] MAY BE WRITTEN AT SYSTEM STARTUP SINCE IT IS NEVER RESET BY THE HARDWARE.)

2. -11 WRITES ADDRESS OF SOURCE STRING (TO10AD [15-00]) [BREADBOARD: TO10AD[15-01]]

3. -11 SETS -11 BYTE OR WORD MODE BIT DIAG3[TO10BM] [BREADBOARD: BIT 00 IN TO10AD]

4. -10 ALLOCATES CORE TO RECEIVE STRING.

5. -10 SETS UP BYTE POINTER IN EPT TO RECEIVE DATA (EPTTBP).

6. -10 SETS NEGATIVE BYTE COUNT AND INDICATES THAT BOTH CPUS ARE TO RECEIVE THE NORMAL TERMINATION INTERRUPT (DATAO DTEX, -BYTE COUNT WITH I (BIT 23) SET). THIS DATAO STARTS THE TRANSFER.

7. THE DTE20 INTERFACE AND EBOX WORK AS FOLLOWS DURING THE TRANSFER:

```
DO
10CHAR: BEGIN
        COPY DLYCNT [13-00] TO COUNTER [13-00];

DELAY:          BEGIN (DELAY LOOP)
                ADD COUNTER [13-00]+1 TO COUNTER [13-00];
                DELAY 500 NANOSECONDS;
                IF EXAMINE/DEPOSIT REQUEST, THEN

                        BEGIN (EXAMINE/DEPOSIT TAKES PRIORITY)
                        DO EXAMINE/DEPOSIT
                        END (EXAMINE/DEPOSIT)

                IF COUNTER [13]=0, THEN EXIT LOOP DELAY;
                END (OF DELAY LOOP)

                COPY BC [11-00] TO COUNTER [11-00];
                IF TO10BC=0, THEN

                        BEGIN (NORMAL TERMINATION)
                        IF TO10BC[I] = 1, THEN
                           INTERRUPT BOTH CPU'S
                        ELSE (TO10BC[I] = 0)
                           INTERRUPT JUST THE -10;
                        EXIT LOOP 10CHAR (NORMAL TERMINATION)
                        END (NORMAL TERMINATION)

                IF TO10BC IS NON-ZERO, THEN
                INCREMENT THE BYTE COUNT (TO10BC <- TO10BC+1);
                PERFORM AN NPR FETCH FROM -11 MEMORY FOR A BYTE
                   OR WORD DEPENDING ON MODE;
                INCREMENT THE -11 ADDRESS (TO10AD);
                DTE20 GENERATES AN IOP INTERRUPT ON THE EBUS;
                WHEN THE INTERRUPT REQUEST IS HONORED, THE
                   DTE20 EXECUTES AN IOP API FUNCTION 06;
                THE KL10 MICRO-CODE PERFORMS AN IDPB USING THE
                   TO10 BYTE POINTER (EPTTBP);
                IF EXAMINE/DEPOSIT REQUEST, THEN

                        BEGIN
                        DO EXAMINE/DEPOSIT
                        END

                IF TO11 READY (CHAR, TO BE TRANSFERRED), THEN

                        BEGIN
                        DO 11CHAR
                        END

        END (GO BACK TO 10CHAR)
```

NOTE: THAT IF AN EXAMINE OR DEPOSIT OCCURS DURING A BYTE TRANSFER, IT IS DONE IMMEDIATELY AND THE DELAY IS RESTARTED.

THE SOFTWARE COULD LOCK OUT DATA TRANSFERS BY EXAMINING OR DEPOSITING AT A RATE FASTER THAN THE BYTE TRANSFER DELAY.

11. TO-11 BYTE TRANSFERS

SEE SECTION 9 FOR A DESCRIPTION WHICH IS COMMON FOR BOTH DIRECTIONS. THE FOLLOWING DESCRIPTION ASSUMES THAT THE -11 HAS DISCOVERED THE LENGTH OF THE STRING WHICH IT IS TO RECEIVE USING ONE OF THE THREE METHODS DESCRIBED IN SECTION 9.1.

1. -11 WRITES THE DELAY COUNT WORD (DLYCNT) WITH THE NEGATIVE NUMBER OF 500 NANO SECOND INTERVALS TO DELAY BETWEEN BYTE TRANSFERS ACROSS THE DTE20 IN BITS 13-00. BITS 15-14 SPECIFY UNIBUS ADDRESS BITS 17-16 FOR BOTH TO-10 AND TO-11 BYTE TRANSFERS. (DLYCNT [15-00] MAY BE WRITTEN AT SYSTEM STARTUP TIME SINCE IT IS NEVER RESET BY THE HARDWARE.)

2. -10 WRITES ADDRESS OF SOURCE STRING (BYTE POINTER IN EPTEBP).

3. -11 ALLOCATES CORE TO RECEIVE STRING.

4. -11 SETS UP ADDRESS TO RECEIVE DATA (TO11AD[15-00]) [BREADBOARD: TO11AD[15-01]]

5. -11 WRITES THE TO-11 BYTE COUNT WORD (TO11BC) WHICH SETS THE FOLLOWING:

    1. NEGATIVE BYTE COUNT (TO11BC[11-00])

    2. INDICATES THAT BOTH CPUS ARE TO RECEIVE THE NORMAL TERMINATION INTERRUPT (TO11BC[INT10] = 1)

    3. WHETHER TERMINATION IS ALSO TO OCCUR ON TRANSFER OF A NULL CHARACTER (TO11BC [ZSTOP]). -11 BYTE OR WORD MODE (TO11BC[TO11BM]) [BREADBOARD: BIT TO11AD[00]]

    4. WRITING TO11BC STARTS THE TRANSFER (PROVIDING TO11AD HAS ALSO BEEN WRITTEN SINCE THE LAST NORMAL OR ERROR TERMINATION).

6. THE DTE20 INTERFACE AND EBOX WORK AS FOLLOWS DURING THE TRANSFER:

```
DO
11CHAR: BEGIN (LOOP FOR EACH BYTE TRANSFERRED)
        COPY DLYCNT [13-00] TO COUNTER [13-00];

DELAY:          BEGIN (DELAY LOOP)
                ADD COUNTER [13-00]+1 TO COUNTER [13-00];
                DELAY 500 NANOSECONDS;
                IF EXAMINE/DEPOSIT REQUEST, THEN

                        BEGIN (EXAMINE/DEPOSIT)
                        DO EXAMINE/DEPOSIT
                        END (EXAMINE/DEPOSIT)

                IF COUNTER [13=0], THEN EXIT LOOP DELAY;
                END (OF DELAY LOOP)

                COPY BC [11-00] TO COUNTER [11-00];
                IF TO11BC=0, THEN

                        BEGIN (NORMAL TERMINATION)
                        IF TO11BC[I]=1, THEN
                        INTERRUPT BOTH CPU'S
                        ELSE (TO11BC[I]=0)
                        INTERRUPT JUST THE -11 CPU;
                        EXIT LOOP 11CHAR
                        END (NORMAL TERMINATION)

                IF TO11BC IS NON-ZERO, THEN
                INCREMENT THE BYTE COUNT (TO11BC <- TO11BC+1);
                DTE20 PUTS AN IOP INTERRUPT REQUEST ON EBUS;
                WHEN INTERRUPT REQUEST IS HONORED, THE DTE20
                   EXECUTES AN IOP API FUNCTION 06;
                THE KL10 MICRO-CODE PERFORMS AN ILDB USING THE
                   TO-11 BYTE POINTER (EPTEBP);
                IF TO11BC[ZSTOP]=1, THEN

                        BEGIN (STOP ON NULL OPTION CHECK)
                        IF BYTE JUST TRANSFERRED IS A 0, THEN

                                BEGIN (NORMAL TERMINATION)
                                IF TO11BC[I]=1, THEN
                                   INTERRUPT BOTH CPU'S
                                ELSE INTERRUPT JUST THE -11;
                                EXIT LOOP 11CHAR;
                                END (NORMAL TERMINATION)
                        END (STOP ON NULL OPTION CHECK)

                THE DTE20 PERFORMS AN NPR TRANSFER IN THE -11
                   WHICH STORES THE BYTE OF DATA (0-16 BITS
                   WORTH) IN AN -11 BYTE OR WORD;

                THE DTE20 INCREMENTS THE TO-11 ADDRESS BY
                   1 OR 2 (TO11AD);
```

END (OF LOOP TO TRANSFER A BYTE, GO BACK TO 11CHAR)

12. GENERAL PROGRAMMING INFORMATION

THERE ARE FOUR ERRORS WHICH CAN CAUSE A TRANSFER ERROR CONDITION THAT IS DETECTED BY THE HARDWARE. THEY ARE UNIBUS PARITY ERRORS, UNIBUS TIMEOUT ERRORS, PDP-11 MEMORY PARITY ERRORS, OR EBUS PARITY ERRORS. ALL OF THESE ERRORS SET A GENERAL ERROR BIT STATUS(TO10ER) OR STATUS(TO11ER). IN ADDITION ALL OF THE ERRORS EXCEPT UNIBUS TIMEOUT ERRORS ALSO HAVE ANOTHER ERROR BIT. A BUS TIMEOUT IS MOST LIKELY CAUSED BY A NONEXISTENT MEMORY REFERENCE ALTHOUGH IT CAN BE CAUSED BY A VARIETY OF HARDWARE PROBLEMS.

THE EBUS LOGIC ATTEMPTS TO RUN THE EBUS AT FULL SPEED. IT CAN, HOWEVER, HOLD UP THE BUS DURING A DEPOSIT IF A READ AND WRITE IS PERFORMED TO A DTE20 RAM LOCATION (FIRST 12 LOCATIONS OF -11 INTERFACE ADDRESSES). THIS HOLD-UP OCCURS WHEN THE READ OR WRITE TO THE RAM LOCATION OCCURS AFTER THE INTERRUPT HAS BEEN HONORED BUT BEFORE THE ADDRESS HAS BEEN SENT TO THE -10. THE MAXIMUM TIME THE EBUS WILL BE HELD UP IN ADDITION TO THE NORMAL PROCESSING TIME IS APPROXIMATELY 2 MICRO-SECONDS. ALL READS AND WRITES TO THE RAM LOCATIONS SLOW DOWN THE INTERFACE AS THEY STEAL CYCLES FROM THE TRANSFER LOGIC, ALTHOUGH THEY DO NOT SLOW DOWN THE EBUS OPERATION EXCEPT IN THE CASE NOTED.

BYTE OPERATIONS ARE ONLY PERMISSIBLE ON THE -11 STATUS WORD. IF RAM LOCATIONS ARE ADDRESSED AS BYTES, A BUS TIME OUT BY THE -11 PROCESSOR WILL OCCUR. IF THE DIAGNOSTIC WORDS ARE ADDRESSED AS BYTES, NO ERROR WILL OCCUR. THIS APPLIES ONLY TO DATOB UNIBUS CYCLES. DATI BYTE OPERATIONS ARE PERMISSIBLE FOR ALL LOCATIONS AS THE RECEIVER OF THE DATA IS RESPONSIBLE FOR THE BYTE OPERATIONS. THIS DOES PROHIBIT BYTE INSTRUCTIONS WHICH USE BOTH DATI AND DATOB UNIBUS CYCLES SUCH AS INCB.

TO-10 AND TO-11 BYTE POINTERS ARE EVALUATED IN EXEC VIRTUAL ADDRESS SPACE. (SECTION 0 IS ALWAYS ASSUMED). IF PAGING IS TURNED ON, ALL ADDRESSES ARE TRANSLATED ACCORDING TO THE EXEC PAGE MAP AND THE PROTECTION IS CHECKED ASSUMING A REFERENCE FROM KERNEL MODE.

A EBUS RESET (CONO PI, OR POWER UP) EXECUTED BY THE -10 WILL NOT CLEAR ALL OF A PRIVILEGED DTE20, BUT WILL CLEAR PART OF IT. A RESET WILL ALWAYS CLEAR THE KL10 STATUS FLOPS, THE ASSIGNED PI CHANNEL, AND PI LEVEL 0 ENABLE. IF A DTE20 IS RESTRICTED, THE EBUS RESET WILL DO A COMPLETE RESET TO THE DTE20 TO INSURE THAT THE SYSTEM INITIALIZES PROPERLY. A PRIVILEGED FRONT END LOSES ONLY THE ABILITY TO PERFORM STATUS INTERRUPTS AFTER A EBUS RESET HAS BEEN ISSUED. THIS MEANS THAT A UNIBUS INIT (RESET, POWER UP, PRESSING -11 CONSOLE START SWITCH) MUST BE GENERATED ON EACH -11 SYSTEM CONNECTED TO A DTE20 THAT IS A PRIVILEGED FRONT END AFTER THE KL10 IS POWERED UP.

TO-10 AND TO-11 BYTE STRING TRANSFERS WILL GO TO COMPLETION EVEN THOUGH THE CPU IS HALTED. IF A -10 ISSUES A RESET (CONO

PI, OR POWER UP) ALL RESTRICTED DTE20S ON ITS EBUS ARE COMPLETELY INITIALIZED.

SINCE PI 0 INTERRUPTS ARE ALWAYS HONORED (EXCEPT IF DISABLED ON RESTRICTED FRONT END), BYTE TRANSFERS, DEPOSITS AND EXAMINESWORK AS LONG AS THE MICRO-CODE IS LOADED AND RUNNING EVEN IF IN A HALT LOOP.

CAUTION:

BIT SETS (BIS) AND OTHER SIMILAR INSTRUCTIONS SHOULD BE USED WITH GREAT CARE WHEN APPLIED TO DTE REGISTERS. THE -11 CPU PERFORMS A DATI, "ORS" IN THE BIT AND THEN PERFORMS A DATO. FOR BITS WHICH READ AND WRITE DIFFERENT FUNCTIONS THIS CAN CAUSE UNEXPECTED AND DISASTROUS RESULTS. FOR EXAMPLE, IN THE STATUS REGISTER, BIT 2 READS AS DEXDON (DEPOSIT OR EXAMINE DONE) AND WRITES AS EBUSPS (EBUS PARITY ERROR SET). A BIT SET INSTRUCTION WITH DEXDON ASSERTED WILL CAUSE AN APPARENT EBUS PARITY ERROR.

13. ERROR DETECTION

THE DTE20 WILL GENERATE/CHECK PARITY ON DEPOSIT/EXAMINE DATA (36 BITS) AND BYTE STRING DATA (16 BITS). IT WILL NOT CHECK OR GENERATE PARITY FOR CONI DTEN, CONO DTEN, OR DATAO DTEN. THE SOFTWARE SHOULD CHECK FOR ERRORS BY EXAMINING THE TERMINATION WORDS. THE PARITY SCHEME ALSO IMPOSES ONE RESTRICTION ON THE BYTE POINTER USED FOR TO-11 TRANSFERS. A BYTE SIZE OF LARGER THAN 16 BITS CANNOT BE USED UNLESS THE BITS TO THE LEFT OF THE RIGHTMOST 16 BITS CONTAIN EVEN PARITY. IF A PARITY ERROR OCCURS, THE ERROR TERMINATION BIT STATUS [TO11ER] (IF A TO-11 TRANSFER) WILL BE SET AND THE EBUS PARITY ERROR FLAG STATUS [BPARER] WILL BE SET (FOR TO11 AND EXAMINE OPERATIONS). IF AN EXAMINE OPERATION WAS IN PROGRESS WHEN A TO -11 TRANSFER OPERATION HAD AN ERROR TERMINATION (STATUS [TO11ER]) DUE TO AN EBUS PARITY ERROR, IT IS NOT POSSIBLE FOR THE SOFTWARE TO DETERMINE IF THE EXAMINE OPERATION HAD A PARITY ERROR. IF THIS HAPPENS, THE SOFTWARE SHOULD RETRY THE EXAMINE OPERATION. WHEN A PARITY ERROR OCCURS, THE BAD DATA IS STORED IN THE RAM AND CAN BE RETRIEVED FOR ERROR REPORTING. THE DTE20 SOMETIMES SWAPS THE LEFT AND RIGHT BYTES FOR BYTE MODE PRIOR TO WRITING THE BYTES INTO THE RAM. THEREFORE THE TERMINATION TO-11 ADDRESS WORD (TO11AD) SHOULD BE EXAMINED TO DETERMINE IF THE LEFT AND RIGHT HALVES WERE SWAPPED. IF THE TERMINATION ADDRESS IS EVEN THE BYTES WERE SWAPPED (THIS APPLIES ONLY TO TRANSFERS IN BYTE MODE).

13.1 UNIBUS PARITY

THE DTE20 GENERATES AND CHECKS PARITY USING THE NEW UNIBUS PARITY SCHEME. THIS SCHEME ALLOWS A MIXTURE OF PARITY AND NON-PARITY DEVICES TO BE ATTACHED TO THE UNIBUS. THE BASIC STRATEGY IS FOR THE SENDER (CPU, PERIPHERAL, OR MEMORY) TO SEND A PARITY BIT (PB) AND A BIT SAYING WHETHER IT IS GENERATING PARITY OR NOT (PA). THE RECEIVER CAN THEN CHECK FOR ERRORS OR NOT DEPENDING ON THE SENDER:

PA	PB	
0	0	SENDER NOT GENERATING PARITY
1	1	SENDER GENERATING PARITY AND PARITY BIT = 1
1	0	SENDER GENERATING PARITY AND PARITY BIT = 0
0	1	CORE MEMORY PARITY ERROR, NO UNIBUS PARITY CHECKING IN EFFECT

THE DTE20 CHECKS UNIBUS PARITY AND GENERATES UNIBUS PARITY ACCORDING TO THIS SCHEME.

THE DTE20 GENERATES UNIBUS PARITY FOR ALL PDP-11 DATI FROM THE FIRST 12 DTE20 REGISTERS, BUT NOT THE LAST 4 REGISTERS (DIAG1, DIAG2, DIAG3, AND STATUS). THE LAST 4 ARE NOT DONE BECAUSE THEY ARE REGISTERS NOT RAM LOCATIONS AND SO WOULD REQUIRE A BUFFER REGISTER FOR EACH. THE DTE20 CHECKS UNIBUS PARITY FOR ALL PDP-11 DATO TO ALL 16 OF THE DTE20 REGISTERS.

THE DTE20 UNIBUS ADDRESS AND DATA RECEIVERS ARE ON TWO DIFFERENT BOARDS, M8553 AND M8552 RESPECTIVELY. TWO ERROR FLAGS ARE PROVIDED, SO THAT FAULT ISOLATION TO THE BOARD CAN BE ACHIEVED WHEN THE ERROR IS THE DTE20 UNIBUS RECEIVERS. THE -11 DATO UNIBUS PARITY ERROR BIT, DIAG3[4], IS SET WHEN THE DTE20 DETECTS EVEN PARITY ON ADDRESS AND DATA COMBINED ON BOARD M8553. THE DATO UNIBUS RECEIVE ERROR BIT, DIAG3[2], IS SET WHENEVER THE PARITY OF JUST THE DATA COMPUTED ON ONE BOARD IS DIFFERENT FROM THE PARITY COMPUTED FOR JUST THE DATA COMPUTED ON THE OTHER BOARD. THEREFORE THE TWO ERROR BITS CAN BE INTERPRETED AS FOLLOWS, ASSUMING SINGLE BIT ERRORS:

DATO UNIBUS PARITY ERR DIAG3[DUPE]	DATO UNIBUS RECEIVE ERR DIAG3[DURE]	MEANING FOR UNIBUS DATO'S
0	0	NO ERROR HAS OCCURRED OR NO PARITY INFORMATION WAS SENT.
0	1	THE PARITY COMPUTED ON ADDRESS AND DATA (ON BOARD M8552) WAS CORRECT (ODD). THE DATA PARITY COMPUTED ON THE M8552 BOARD DID NOT AGREE WITH THAT COMPUTED ON THE M8553 BOARD. THEREFORE A RECEIVER OR PARITY TREE FAILURE OCCURRED ON THE M8552 BOARD.
1	0	THE PARITY ON THE ADDRESS AND DATA (ON BOARD M8552) WAS INCORRECT (EVEN). THE DATA RECEIVED ON BOTH BOARDS CONTAINED THE SAME PARITY. THEREFORE INCORRECT ADDRESS OR DATA WAS SENT, OR AN ADDRESS RECEIVER OR PARITY TREE IN THE M8552 BOARD FAILED IN THE DTE20.
1	1	THE PARITY OF THE ADDRESS AND DATA RECEIVED ON THE M8552 BOARD WAS INCORRECT (EVEN). THE PARITY OF THE DATA RECEIVED BY EACH BOARD WAS DIFFERENT. THEREFORE THE M8553 HAS A BAD RECEIVER.

WHENEVER A PARITY ERROR OCCURS DURING A BYTE TRANSFER OPERATION IN A PARTICULAR DIRECTION, THE DTE20 WILL STOP PROCESSING DATA IN THAT DIRECTION AS SOON AS POSSIBLE. IF A DEPOSIT OR EXAMINE IS IN PROGRESS IT WILL COMPLETE. IN ALL OTHER CASES THE DTE20 WILL STOP IMMEDIATELY. IF THE PROGRAM DECIDES THAT THE PARITY ERROR EFFECTS WERE HARMLESS, THE OPERATIONS THAT WERE IN PROGRESS WHEN THE PARITY ERROR CAN BE CONTINUED BY CLEARING THE ERROR FLAGS (EITHER DATO UNIBUS RECEIVE ERROR (DIAG3[DURE]) OR DATO UNIBUS PARITY ERROR (DIAG3[DUPE]). IF THE OPERATIONS THAT WERE IN PROGRESS CANNOT BE CONTINUED, A RESET TO THE DTE20 SHOULD BE PERFORMED.

THE BAD DATA AND ADDRESS ARE READ OUT FROM A SHIFT REGISTER IN DIAG3[13-08], 5 BITS AT A TIME. EACH TIME THE LOCATION IS READ, A SHIFT COMMAND MUST BE ISSUED PRIOR TO READING THE NEXT 5 BITS OF DATA BY WRITING DIAG3[SCD]. THE DIAG3 REGISTER ONLY SHIFTS WHEN A PARITY ERROR FLAG IS SET AND LOADS IF THE PARITY ERROR FLAGS ARE CLEARED. THEREFORE, IT WILL CAPTURE THE FIRST PARITY ERROR DATA ONLY ON SUCCESSIVE PARITY ERRORS.

IF A PARITY ERROR OCCURS DURING TO10 NPR CYCLES, AN INTERRUPT OCCURS WITH NPR UNIBUS PARITY ERROR FLAG DIAG3[NUPE] IS SET, AS WELL AS TO10 ERROR TERMINATION, STATUS[TO10ER]. THE DATA RECEIVED IS LEFT IN TO10 PDP-11 DATA WORD, TO10DT, AND THE TO10 PDP-11 MEMORY ADDRESS, TO10AD, CONTAINS THE PDP-11 ADDRESS READ FROM. THIS ERROR CANNOT BE CONTINUED FROM AND BOTH TO10BC AND TO10AD MUST BE LOADED TO START A TRANSFER.

IF A PARITY ERROR OCCURS ON A DATO TO CLEAR THE PARITY ERROR FLAGS (DIAG3) THE FLAGS WILL GET CLEARED AND AN INTERRUPT WILL NOT BE GENERATED. IF A PARITY ERROR OCCURS ON A DATO TO THE LOCATION CONTAINING THE CLEAR PARITY ERROR FLAG (DIAG3), BUT THE CLEAR ERROR BIT IS NOT TRUE, THE ERROR FLAGS WILL GET SET.

14. DIAGNOSING THE KL10

14.1 DIAGNOSTIC BUS OPERATION

ALL KL10 DIAGNOSTIC FUNCTIONS AND CONSOLE FUNCTIONS (EXCEPT DEPOSIT AND EXAMINE) ARE PERFORMED OVER THE DIAGNOSTIC PORTION OF THE EBUS. THIS SPECIFICATION EXPLAINS THE OPERATION OF THE DIAGNOSTIC BUS. FOR A SPECIFICATION OF THE OPERATIONS AVAILABLE SEE THE KL10 SPECIFICATION.

14.2 DIAGNOSTIC BUS DESCRIPTION

THE DIAGNOSTIC BUS CONTAINS TEN SIGNAL LINES AS FOLLOWS:

DS00 - 06 - DIAGNOSTIC SELECT LINES - THE -11 SENDS ENCODED DIAGNOSTIC FUNCTIONS TO THE KL10 ON THESE LINES. THESE LINES CAN BE READ BY THE -11 AT

ANY TIME EVEN WHILE THE REST OF THE EBUS IS ACTIVE FOR OTHER DEVICES.

DIAG STROBE - DIAGNOSTIC STROBE - THIS LINE IS ASSERTED TO INDICATE THAT THE DIAGNOSTIC SELECT LINES ARE STABLE, AND THAT THE INDICATED FUNCTION SHOULD BE PERFORMED.

DFUNC - DIAGNOSTIC FUNCTION WHEN TRUE, CAUSES THE KL10 TO DISABLE THE BASIC CPU STATUS FROM THE DS LINES, SWITCH THE TRANSLATOR (ONLY FOR THE DS LINES) TO CONVERT TTL TO ECL, AND PUT THE EBUS TRANSLATOR UNDER CONTROL OF DS BITS 00 AND 01.

THE DIAGNOSTIC BUS IS CONTROLLED AS FOLLOWS:

14,2,1 DIAGNOSTIC CPU STATUS READ - ALL BITS IN DIAG WORD1 MUST BE LOADED WITH ZEROS. THE CPU STATUS MAY THEN BE READ FROM THE DS LINES AFTER 1 MICRO-SECOND HAS BEEN ALLOWED FOR THE LINES TO SETTLE.

14,2,2 DIAGNOSTIC FUNCTIONS ONLY (I,E., NO 36-BIT TRANSFERS) - THE DESIRED FUNCTION CODE BITS SHOULD BE SET ALONG WITH DIAG COMMAND START (DIAG1[DCOMST]) AND DIAG FUNCTION (DIAG1[DFUNC]). THIS WILL RESULT IN THE FUNCTION BEING SENT TO THE KL10. WHEN DIAG COMMAND START IS A ZERO, THE FUNCTION HAS BEEN SENT. ALL FUNCTION BITS MUST BE LOADED WITH THE DESIRED VALUE EACH TIME A NEW COMMAND IS SENT. THE DIAG SEND BIT HAS NO EFFECT UPON THIS OPERATION. DIAG KL10 MUST NOT BE SET OR A 36-BIT DATA TRANSFER WILL TAKE PLACE. THIS OPERATION SHOULD NOT TAKE MORE THAN 2,0 MICROSECONDS.

14,2,3 DIAGNOSTIC FUNCTIONS WITH 36-BIT DATA TRANSFER -

SENDING DATA TO THE KL10 - NO OTHER OPERATIONS (I,E. BYTE STRING TRANSFERS) MAY BE IN PROGRESS WHILE DOING 36-BIT DIAGNOSTIC DATA TRANSFERS. THE DATA SHOULD BE LOADED INTO DEXWD1-3 (SAME BIT ASSIGNMENTS AS WITH A DEPOSIT OR EXAMINE). DIAG KL10 SHOULD THEN BE SET. A DEPOSIT OPERATION SHOULD BE STARTED. WHEN THE TRANSFER IS COMPLETE (DEXDON SET), THE DIAGNOSTIC FUNCTION SHOULD BE LOADED AS DESCRIBED ABOVE THE DIAG KL10, DIAG SEND, DIAG COMMAND START AND DIAG FUNCTION SET. THE OPERATION IS COMPLETE WHEN DIAG COMMAND START IS ON A ZERO.

RECEIVING DATA FROM THE KL10 - THE DIAGNOSTIC FUNCTION SHOULD BE LOADED WITH DIAG KL10 SET, DFUNC SET, DIAG SEND CLEAR, AND DIAG COMMAND START SET. WHEN DIAG COMMAND START IS CLEAR, THE FUNCTION IS COMPLETE AND THE DATA IS IN DEX WD1-3. NO OTHER OPERATIONS (I,E., BYTE STRING TRANSFERS) MAY BE IN PROGRESS DURING THIS OPERATION.

ALL THE KL10 DIAGNOSTIC FUNCTIONS ARE DISABLED WHEN THE PRIVILEGED/RESTRICTED MODE SWITCH IS SET TO RESTRICTED MODE. THIS BIT CAN BE TESTED BY READING STATUS [RM]. WHEN THE SWITCH

IS SET TO RESTRICTED MODE, STATUS [RM] IS SET (I.E., THE DEVICE IS RESTRICTED AND CANNOT SEND DIAGNOSTIC FUNCTIONS).

15. DIAGNOSING THE DTE20

THE INTERFACE CONTAINS MANY FEATURES WHICH ENABLE DIAGNOSING OF THE INTERFACE. IT IS DESIGNED TO BE DIAGNOSED USING THREE BASIC METHODS:

1. WITHOUT USING OR DISTURBING THE EBUS
2. WITH LOOP-BACK ON THE EBUS BUT WITHOUT THE KL10 OR WITHOUT THE KL10 RUNNING
3. WITH THE KL10 RUNNING

THE INTERFACE IS CHECKED OUT MAINLY IN A SINGLE STEP MANNER. FULL SPEED OPERATION MAY ONLY BE CHECKED WITH A RUNNING KL10. DIAG1 CONTAINS THE DIAGNOSE 10/11 INTERFACE BIT (DIAG1[D1011]. WHEN DIAG1[D1011] IS SET, THE FOLLOWING OCCURS:

THE INTERFACE CLOCK IS DISABLED AND SINGLE STEP OPERATION COMMENCES. INTERRUPTS ARE INHIBITED FROM BEING SENT TO THE KL10. THE INTERFACE OPERATES IN THE NORMAL MANNER EXCEPT THAT EBUS OPERATIONS NEVER COMPLETE BECAUSE NO INTERRUPTS ARE ISSUED TO THE KL10. THEREFORE, A BIT HAS BEEN PROVIDED WHICH ALLOWS SETTING THE EBUS COMPLETE, ALLOWING THE OPERATION TO CONTINUE.

THE INTERFACE CONTROL IS RUN BY AN UP-COUNTER AND THREE DECODERS. THE DECODERS ARE SELECTED BY THE MAJOR STATE FLOPS. THE UP-COUNTER IS LOADABLE BY THE RIGHTMOST 4 BITS OF DIAG WORD 2. THIS ENABLES ANY MINOR LOGIC STATE TO BE EXECUTED. THE MAJOR STATES ARE NOT LOADABLE, HOWEVER THEY NATURALLY CYCLE UNTIL A CONDITION OCCURS WHICH INDICATES THAT THE OPERATION IS READY TO TAKE PLACE. THESE MAJOR STATE BITS ARE READABLE.

16. RELOADING THE PDP-11

THE -10 CAN RELOAD THE -11 BY "PUSHING" THE -11'S RELOAD BUTTON CONO[SR11B] AND "RELEASING" IT CONO[CR11B], 150 MILLISECONDS LATER. THIS WILL RESULT IN THE ROM SEQUENCE BEING PERFORMED (SEE SOFTWARE FRONT END SPECIFICATION). THIS RESTART TAKES A CONSIDERABLE AMOUNT OF TIME. THE -10 CAN RECOGNIZE THE START OF THE ROM EXECUTION BY CHECKING THE POWER FAIL BIT (CONI[DEAD11]). CONI[DEAD11] GOES TRUE AT THE START OF THE RELOAD SEQUENCE. WHEN CONI[DEAD11] IS FALSE AGAIN THE HARDWARE PORTION OF THE RELOAD SEQUENCE IS COMPLETE AND CONTROL HAS BEEN SHIFTED TO SOFTWARE IN THE PDP-11. NO CONSTANTS OR STATUS BITS (OTHER THAN RAM LOCATIONS) CAN BE SET PRIOR TO POWER FAIL GOING FALSE AGAIN BECAUSE THE PDP-11 GENERATES AN INIT WHICH WILL CLEAR THEM. THE ROM WILL LOAD A PROGRAM WHICH WILL DETERMINE WHETHER THE -10 DESIRED A WARM RESTART (FROM UNIBUS HUNG, ETC.) OR A COLD RESTART, INCLUDING COMPLETE REBOOT. THERE IS A SECOND SWITCH ON THE DTE20 WHICH ENABLES OR DISABLES THE -10

FROM RELOADING THE -11. IT IS LABELED "ENABLE 10 LOAD 11" AND "DISABLE 10 LOAD 11".

PROCEDURE TO DUMP AND LOAD THE PDP-11:

THE FOLLOWING IS THE PROCEDURE WHICH THE KL10 EXECUTES IN ORDER TO DUMP AND/OR BOOTSTRAP THE PDP-11 THROUGH THE DTE20:

1. CLEAR THE DTE20 AND INITIATE A BM873 BUTTON #4 BOOTSTRAP OPERATION - CONO [SR11B!CL11PT!CLTO11!CLTO10!PILDEN]

2. WAIT TO SEE PDP11 POWER FAIL (AC LOE = TRUE) - CONI [DEAD11] = 1

3. WAIT TO PAP11 POWER RECOVER (AC LOW = FALSE) - CONI [DEAD11] = 0

4. WAIT AT LEAST ANOTHER 150 MILLISECONDS AND THEN CLEAR THE RELOAD -11 BUTTON - CONO [CR11B]

5. SET BYTE COUNTER TO A SPECIAL CODE (1365 OCTAL) - DATAO [1365]

6. RING PDP11'S DOORBELL - CONO[TO11DB]

7. WAIT UNTIL "-10 RINGING -11'S DOORBELL" IS TURNED OFF BY THE -11 (I.E., UNTIL CONI[TO11DB] BECOMES ZERO.

8. ENABLE THE DTE20 TO USE PI 0 INTERRUPTS (I.E., SET CONO[PILDEN!PI0ENB].

9. SET UP THE TO-10 BYTE POINTER (IN THE EPT) FOR THE FIRST 3.5K.

10. SET UP THE BYTE COUNTER FOR THE FIRST 3.5K, INDICATING "INTERRUPT -10 ONLY" - DATAO [1000]

11. WAIT FOR "TO-10 DONE" OR "TO-10 ERRROR" - CONI [TO10DN!TO10ER]

12. NOTE WHETHER THERE WAS AN ERROR (CONI [TO10ER]) AND THEN TURN OFF TO10DN AND TO10ER - CONO [CLTO10]. IF ERROR, GO TO STEP 17.

13. IF END OF 28K, GO TO STEP 17.

14. SET UP TO-10 BYTE POINTER (IN THE ERT) FOR THE NEXT 3.5K.

15. SET UP THE BYTE COUNTER FOR THE NEXT 3.5K INDICATING "INTERRUPT -10 ONLY" (DATAO [1000], UNLESSS THIS IS THE LAST 3.5K (OF 28K), IN WHICH CASE INDICATE "INTERRUPT BOTH PROCESORS" (DATAO [TO10IB!1000]).

16. GO TO STEP 11.

17. SET UP TO-11 BYTE POINTER (IN THE EPT) FOR "PDP11 BOOTSTRAP". NOTE THAT THE FIRST WORD OF THIS "PDP11 BOOTSTRAP" MUST BE THE BIT PATTERN 000240 (A PDP11 NOP INSTRUCTION).

18. RING THE PDP11'S DOORBELL - CONO [TO11DB]

19. WAIT FOR EITHER TO11DB TO GO OFF (CONI[TO1DB] = 0), OR TO10DB TO COME ON (CONI[TO10DB] = 1.

20. IF NO ERROR WAS NOTED IN STEP 12, TO11DB SHOULD GO OFF (TO10DB COMING ON INDICATES A MASSIVE SCREWUP). IF AN ERROR WAS NOTED IN STEP 12, TO11DB GOING OFF INDICATES THAT THE ERROR WAS "NON-FATAL" (NON-EX-MEM OR -11 MEMORY PARITY) AND THE -11 IS PROCEEDING. TO10DB COMING ON INDICATES THAT THE ERROR WAS "FATAL" AND THE -11 IS HALTING (AT LOCATION 173714). IN THIS LATTER CASE THE -10 MUST RESTART FROM STEP 1.

21. IF TO11DB WENT OFF, WAIT FOR "TO-11 DONE" OR "TO-11 ERROR" - CONI [TO11DN!TO11ER]

22. NOTE WHETHER THERE WAS AN ERROR -(CONI [TO11ER]).

23. TURN OFF TO11DN AND TO11ER AND RING THE PDP11'S DOORBELL - CONO [TO11DB!CLTO11]

24. WAIT FOR EITHER TO11DB TO GO OFF (CONI[TO11DB] = 0), OR TO10DB TO COME ON (CONI[TO10DB] = 1).

25. TO11DB GOING OFF INDICATES THAT THE PDP11 FOUND NO ERRORS AND IS TRANSFERRING CONTROL TO THE CODE WHICH WAS JUST RECEIVED FROM THE -10. IN THIS CASE THE -10 SHOULD START FOLLOWING THE PROTOCOL OF THIS CODE.

26. TO10DB COMING ON INDICATES THAT THE PDP11 HAS FOUND AN ERROR (OR THAT THE FIRST WORD TRANSMITTED WASN'T THE BIT PATTERN 000240), AND THE PDP11 IS HALTING (AT LOCATION 173766). IN THIS CASE THE -10 MUST RESTART FROM STEP 1.

17. KL10 ACCESSABLE REGISTERS

17.1 CONI/CONO REGISTER

-10 BIT	SYMBOL	NAME AND DESCRIPTION
0-19		UNUSED. READ: 0S; WRITE: IGNORED, MBZ, RESERVED TO DEC.
20		RESTRICTED MODE
	RM	READ: A 1 INDICATES DTE IS SWITCHED TO RESTRICTED MODE, A 0, PRIVILEGED MODE.
		WRITE: MBZ
21	DEAD11	11 POWER FAILURE
		READ: A 1 INDICATES THAT THE -11 POWER IS NOT CORRECT (THE UNIBUS SIGNAL "AC LOW" IS ASSERTED) AND THAT NO BYTE TRANSFERS CAN TAKE PLACE.
		WRITE: MBZ
22	TO11DB	REQUEST -11 DOORBELL INTERRUPT
		READ: IF 1, THE -10 HAS REQUESTED A -11 DOORBELL INTERRUPT AND THE -11 HAS NOT YET CLEARED IT (STATUS[INT11C]).
		WRITE: A 1 CAUSES A DOORBELL INTERRUPT IN THE -11, SETTING -10 REQUEST 11 INTERRUPT ON THE -11 SIDE OF INTERFACE, STATUS [TO11DB]. -10 CANNOT CLEAR. -11 CLEARS WITH STATUS[INT11C]
23		CLEAR RELOAD -11 BUTTON
		READ: 0
	CR11B	WRITE: CLEAR RELOAD -11 BUTTON
24		SET RELOAD -11 BUTTON
		READ: 0
	SR11B	WRITE: SET RELOAD -11 BUTTON
25		UNUSED. READ: 0; WRITE: MBZ
26		-11 REQUESTING -10 INTERRUPT

TO10DB READ: THE -11 HAS REQUESTED A DOORBELL INTERRUPT SENT TO THE -10.

CL11PT WRITE: A 1 CLEARS THE -11 REQUESTING -10 INTERRUPT FLAG ON THE -11 SIDE, STATUS [TO10DB], AND ON THE -10 SIDE CONI [TO10DB]

27 TO11ER TO-11 ERROR TERMINATION

READ: INDICATES THAT AN ERROR OCCURRED DURING A TO-11 TRANSFER, CLEARED BY STATUS [CLTO11]

WRITE: MBZ

28 UNUSED, READ: 0; WRITE: MBZ

29 TO-11 NORMAL COMPLETION

TO11DN READ: A TO-11 TRANSFER WAS COMPLETED AND AN ERROR DIDN'T OCCUR AND THE "I" BIT, TO11BC [INT10] HAD SEEN SET BY THE -11. THE TRANSFER IS COMPLETED IF (1) THE BYTE COUNT (TO11BC) BECAME EQUAL TO 0 OR (2) THE -11 HAD SET THE "Z" BIT, TO11BC [ZSTOP], AND A NULL CHARACTER WAS ENCOUNTERED.

CLTO11 WRITE: A 1 CLEARS THE TO-11 NORMAL TERMINATION FLAG AND THE TO-11 ERROR TERMINATION FLAG, THE -10 PI REQUEST LEVEL IS CLEARED SO THAT THE INTERRUPT MAY BE DISMISSED.

30 TO-10 NORMAL TERMINATION

TO10DN READ: THE BYTE COUNTER FOR THE TO-10 TRANSFER (DATAO DTEN,) BECAME EQUAL TO 0 AND AN ERROR DID NOT OCCUR.

CLTO10 WRITE: A 1 CLEARS THE TO-10 NORMAL TERMINATION FLAG, STATUS [TO10DN], AND TO-10 ERROR TERMINATION FLAG, STATUS [TO10ER], THE -10 PI REQUEST LEVEL IS CLEARED. [BREADBOARD: IGNORED]

31 TO-10 ERROR TERMINATION

TO10ER READ: AN ERROR (-11 MEMORY PARITY OR UNIBUS TIME OUT ERROR, BUT NOT EBUS PARITY ERROR) OCCURRED DURING THE TO-10 TRANSFER.

	PILDEN	WRITE: PI LOAD ENABLE. A 1 MEANS LOAD THE PI INTERRUPT CHANNEL NUMBER FROM BITS 33-35 AND PI LEVEL 0 ENABLED FROM BIT 32. [BREADBOARD: CLTO10]
32	PI0ENB	PI LEVEL 0 ENABLED [BREADBOARD: IGNORED] READ: A 1 MEANS THE DTE20 IS ENABLED TO PERFORM EXAMINES, DEPOSITS, AND BYTE TRANSFERS AT PI LEVEL 0 BY THE EBOX. THE DTE20 IS AUTOMATICALLY ENABLED IF THE -11 IS A PRIVILEGED -11 EVEN THOUGH THIS BIT IS 0. WRITE: FOR A RESTRICTED PDP-11 AND CONO[PILDEN] = 1, A 0 DISABLES PI LEVEL 0. A 1 ENABLES PI LEVEL 0.
33-35	PIA	PI CHANNEL NUMBER READ: THE CURRENT PI CHANNEL ASSIGNMENT FOR DOORBELL INTERRUPTS, BYTE TRANSFER NORMAL AND ERROR TERMINATIONS. WRITE: LOADS PI CHANNEL ASSIGNMENT IF CONO[PILDEN] = 1.

17,2 DATAI DTEN, E

DATAI DTEN IS NOT PROVIDED. ALL 0S ARE RETURNED ON A READ, INCLUDING EBUS PARITY.

17,3 DATAO DTEX, E

-10 BIT	SYMBOL	NAME AND FUNCTION
0-22		UNUSED. READ: 0S; WRITE: MBZ
23	TO10IB	TO-10 "I" BIT IF 1, SET TO10IB. IF 0, CLEAR TO10IB. IF 1, THE -10 HAS SET THE "I" BIT FOR A TO-10 BYTE TRANSFER. BOTH THE -10 AND THE -11 WILL BE INTERRUPTED ON NORMAL TERMINATION. IF 0, THE -10 HAS NOT SET THE "I" BIT FOR A TO-10 BYTE TRANSFER. ONLY THE -10 WILL BE INTERRUPTED ON NORMAL TERMINATION. THE -10 MAY THEN RESET THE TO-10 BYTE POINTER, EPTTBP, BEFORE RELOADING THE TO-10 BYTE COUNT (DATAO DTEN,) AND PERFORMING A SCATTER READ.
24-35		NEGATIVE BYTE COUNT TWOS COMPLEMENT OF THE NUMBER OF CHARACTERS LEFT TO TRANSFER UNTIL A TO-10 NORMAL TERMINATION OCCURS. A -1 WILL TRANSFER 1 CHARACTER TO THE -10 BEFORE A NORMAL TERMINATION. A 0 WILL TRANSFER 0 BYTES BEFORE A NORMAL TERMINATION.

18. PDP-11 REGISTERS

THE 16 PDP-11 REGISTERS ARE LISTED HERE IN ALPHABETICAL ORDER.

THE CONTENTS OF 36 BIT KL10 WORDS APPEAR IN THE 3 -11 WORDS DEXWD1, DEXWD2, AND DEXWD3 ON AN EXAMINE. THE CONTENTS OF THESE WORDS ARE WRITTEN IN A KL10 WORD ON A DEPOSIT. THE DTE20 DOES NOT RESET THESE WORDS AFTER AN EXAMINE OR A DEPOSIT SO THEY MAY BE USED FOR SUBSEQUENT OPERATIONS. NOTE THAT THE THREE WORDS ARE ASSIGNED DESCENDING RATHER THAN ASCENDING ADDRESSES IN -11 ADDRESS SPACE SO THAT SUCCESSIVE BYTES CAN BE READ (BUT NOT WRITTEN) USING INCREMENT BYTE INSTRUCTIONS, I.E. BITS 28-35, 20-27, 12-19, 4-11, 0-3. [BREADBOARD: DEXWD1 = 164002, DEXWD3 = 164006]

DEXWD1: DEPOSIT OR EXAMINE WORD 1 (174006 + 40*N)

-11 BIT	NAME
15-04	UNUSED
	READ: 0, WRITE: MBZ
03-00	KL10 DATA WORD BITS 0-3

DEXWD2: DEPOSIT OR EXAMINE WORD 2 (174004 + 40*N)

-11 BIT	NAME
15-00	KL10 DATA WORD BITS 4-19

DEXWD3: DEPOSIT OR EXAMINE WORD 3 (174002 + 40*N)

-11 BIT	NAME
15-00	KL10 DATA WORD BITS 20-35

DIAG1: DIAGNOSTIC WORD 1 (174030 + 40*N)

THESE BITS ARE FOR DIAGNOSTIC USE ONLY.

-11 BIT	SYMBOL	NAME
15-09	DSC	DIAGNOSTIC SELECTION CODES READ: BITS 15-09 ARE DS00-DS06
15-12	DS00-DS03	UNUSED: 0
11	DS04	1 = KL CLOCK ERROR STOP THE KL10 INTERNAL CLOCK (32MHZ) HAS FROZEN DUE TO A HARDWARE MALFUNCTION. ONE OF THE FOLLOWING: CONTROL RAM, DISPATCH RAM, OR FAST MEMORY PARITY ERROR OR FIELD SERVICE TEST CONDITION.
10	DS05	1 = RUN (1) THE MICRO-CODE EXAMINES THIS FLAG BETWEEN PDP-10 INSTRUCTIONS. THE MICRO-CODE WILL ENTER THE HALT LOOP, IF THIS FLAG IS OFF. THIS FLAG IS UNDER CONTROL OF THE -11 USING TWO DIAGNOSTIC FUNCTIONS. THE -10 CANNOT AFFECT RUN.
09	DS06	1 = HALT (1) THIS SIGNAL IS SET TO A 1 WHEN THE XXX MICRO-CODE ENTERS THE HALT LOOP AND IT CLEARS THE SIGNAL WHEN IT LEAVES THE LOOP. WRITE: SPECIFIES THE DIAGNOSTIC SELECTION CODE NOTE: WRITE FUNCTIONS WITH DS00 AND DS01 = 0 MAY BE DONE PREDICTABLY WHILE THE SYSTEM IS RUNNING WITHOUT BEING IN DIAGNOSTIC MODE. THUS THE -11 CAN SAMPLE -10 STATUS WITHOUT FEAR OF CLOBBERING DATA ON THE EBUS.
08	DEX	DEPOSIT OR EXAMINE MAJOR STATE READ: A 1 MEANS INTERFACE MAJOR STATE IS DEPOSIT OR EXAMINE WRITE: MBZ
07		TO-10 TRANSFER MAJOR STATE

	TO10	READ: A 1 MEANS INTERFACE MAJOR STATE IS TO-10 TRANSFER, 0 MEANS NOT IN TO-10 TRANSFER STATE.
	DFUNC	WRITE: A 1 CAUSES THE KL10 CPU TO STOP SENDING BASIC STATUS ON THE DS LINES SO THAT A LOOP BACK TEST CAN BE PERFORMED ON THE DS LINES. IF ANY OF THE DS LINES ARE SET (BY THE DTE20) THE RESULT IS AN "OR" OF THE BITS SET IN THE DTE20 AND THE CPU STATUS.
06	TO11	TO-11 TRANSFER MAJOR STATE
		READ: A 1 MEANS INTERFACE MAJOR STATE IS TO-11 TRANSFER
		WRITE: MBZ
05	D1011	DIAGNOSE 10/11 INTERFACE
		READ; IF A 1, THE DTE20 IS IN 10/11 DOAGNOSTIC MODE, I.E., IT WILL DIAGNOSE ITSELF. IF A 0, IT IS NOT IN 10/11 DIAGNOSTIC MODE.
		WRITE: IF A 1, SET DTE20 TO 10/11 DIAGNOSTIC MODE. THIS MODE IS USED TO DIAGNOSE THE DTE20 ITSELF. IF A 0, LEAVE 10/11 DIAGNOSTIC MODE.
		WRITE: MBZ
04		VECTOR INTERRUPT ADDRESS BIT 4
	VEC04	READ: VECTOR INTERRUPT ADDRESS BIT 4
	PULSE	WRITE: IF A 1, GENERATES A SINGLE CLOCK CYCLE IF 10/11 DIAGNOSTIC MODE STATUS[D1011] IS ALSO SET.
03		READ: 0
	DIKL10	WRITE: IF A 1 AND DTE20 IS SWITCHED TO PRIVILEGED, PUT THE DTE20 INTO KL10 DIAGNOSTIC DATA TRANSFER MODE. SUBSEQUENT EXAMINES AND DEPOSITS BECOME DIAGNOSTIC FUNCTIONS INSTEAD OF ACCESSING KL10 MEMORY. IF A 0, PUT THE DTE20 IN NORMAL DATA TRANSFER MODE. SUBSEQUENT EXAMINES AND DEPOSITS WILL REFER TO KL10 MEMORY.

02		READ: 0
	DSEND	WRITE: IF A 1, SEND DATA (TO-10) DURING A DIAGNOSTIC BUS TRANSFER, IF A 0, RECEIVE DATA (TO-11) DURING A DIAGNOSTIC BUS TRANSFER
01		UNUSED
00	DCOMST	DIAGNOSTIC COMMAND START
		READ: IF A 1, A DIAGNOSTIC COMMAND IS IN PROGRESS
		WRITE: IF A 1 AND DTE20 IS SWITCHED TO PRIVILEGED, SETS DIAGNOSTIC COMMAND START, A 0 CLEARS DIAGNOSTIC COMMAND START.

DIAG2: DIAGNOSTIC WORD 2 (174032 + 40*N)

THESE BITS ARE FOR DIAGNOSTIC USE ONLY MAINLY IN SINGLE STEP, SEE PRINT SET.

-11 BIT	SYMBOL	NAME
15	RFMAD0	RFM ADDRESS BIT 0
		READ: CONTENTS OF RFM ADDRESS BIT 0
		WRITE: MBZ
14		RFM ADDRESS BIT 1
	RFMAD1	READ: CONTENTS OF RFM ADDRESS BIT 1
	EDONES	WRITE: IF A 1, SET EBUS DONE. IF A 0, CLEAR EBUS DONE
13	RFMAD2	RFM ADDRESS BIT 2
		READ: CONTENTS OF RFM ADDRESS BIT 2
		WRITE: MBZ
12	RFMAD3	RFM ADDRESS BIT 3
		READ: CONTENTS OF RFM ADDRESS BIT 3
		WRITE: MBZ
11-07		UNUSED
		READ: 0S; WRITE: MBZ
06	DRESET	DTE20 RESET
		READ: 0
	DRESET	WRITE: IF A 1, RESET THE DTE20
05		WRITE: MBZ
		READ: 0'S
04-01		WRITE: LOADS 04,03,02,01 INTO MINOR STATE COUNTER 8,4,12,1. FOR DIAGNOSTIC USE ONLY. DURING NORMAL OPERATION MBZ>
		READ: 0'S

00	UNUSED
	READ: OS; WRITE: MBZ

DIAG3: DIAGNOSTIC WORD 3 (174036 + 40*N)

-11 BIT	SYMBOL	NAME
15	SWSLLT	SWAP SELECT LEFT
		READ: CNT1[N] SWAP SEL LT
		WRITE: MBZ
14	DPS4[N]	PARITY (1) H
		READ: DPS4 [N] PARITY FLOP IS ON A ONE. DIAGNOSTIC USE ONLY.
		WRITE: MBZ
13-08		CAPTURED UNIBUS PARITY ERROR INFO
		READ: ANN MEANS UNIBUS REGISTER ADDRESS BIT, DNN MEANS UNIBUS DATA BIT WHEN UNIBUS PARITY ERROR DETECTED.

	UNIBUS DATA BITS
INITIAL	D15 D14 D13 D12 D11 A00
1ST SHIFT	D10 D09 D08 D07 D06 A00
2ND SHIFT	D05 D04 D03 D02 D01 A00
3RD SHIFT	D00 A04 A03 A02 A01 A00
4TH SHIFT	D15 D14 D13 D12 D11 A00
. .	
. .	
. .	

		WRITE: MBZ
07-06		UNUSED
		READ: 0
		WRITE: MBZ
05	SCD	SHIFT CAPTURED DATA
		READ: 0
		WRITE: SHIFT CAPTURED DATA SO NEXT READ OF DIAG3 WILL CHANGE BITS 13-08
04	DUPE	DATO UNIBUS PARITY ERROR

READ: IF 1, A DATO UNIBUS PARITY ERROR HAS BEEN DETECTED BY THE DTE20

CDD WRITE: CLEAR DUPE AND DURE ERROR FLAGS

03 WEP WRITE EVEN (BAD) PARITY

READ: READ STATUS OF WRITE EVEN UNIBUS PARITY FLIP-FLOP

WRITE: IF 1, WRITE EVEN UNIBUS PARITY, RESULTS IN DTE20 GENERATING EVEN (BAD) PAITY ON ALL UNIBUS TRANSFERS WHICH HAVE PARITY. IF 0, THE DTE20 WILL GENERATE ODD (GOOD) PARITY ON ALL SUBSEQUENT UNIBUS TRANSFERS WHICH HAVE PARITY. THIS BIT IS PROVIDED FOR DIAGNOSTIC PURPOSES TO CHECK THE PARITY NETWORKS.

02 DURE DATO UNIBUS RECEIVE ERROR

READ: A UNIBUS RECEIVER ERROR HAS OCCURRED.

WRITE: MBZ

01 NUPE NPR UNIBUS PARITY ERROR

READ: IF 1, A UNIBUS PARITY ERROR HAS OCCURRED ON AN NPR (BYTE) TRANSFER.

CNUPE WRITE: CLEAR NUPE

00 TO10BM TO-10 BYTE TRANSFER MODE

READ: 0

WRITE: IF 1, TO-10 BYTE TRANSFERS ARE TO BE DONE IN BYTE MODE FROM -11 MEMORY. IF 0, TO-10 BYTE TRANSFERS ARE TO BE DONE IN WORD MODE FROM -11 MEMORY.

DLYCNT: DELAY COUNT WORD (174000 + 40*N)

-11 BITS NAME

15-14 UNIBUS ADDRESS BITS 17-16 SPECIFIES HIGH ORDER 2 BITS OF 18 BIT PDP-11 ADDRESS USED IN 18 BIT BYTE TRANSFER ADDRESSES. TRANSFER CANNOT CROSS A 32K

BOUNDARY. TO-10 AND TO-11 TRANSFERS MUST BE IN THE SAME 32K BANK.

13-00 NEGATIVE DELAY COUNT

THE SOFTWARE SPECIFIES HOW MANY 500 NANOSECOND UNITS OF DELAY ARE TO OCCUR BETWEEN EACH BYTE ON BYTE TRANSFERS IN EITHER DIRECTION. THE DELAY ALSO APPLIES BEFORE THE FIRST BYTE. THE HARDWARE UPCOUNTS A COPY OF THIS NEGATIVE NUMBER ONCE EACH 500 NANO SECONDS, UNTIL BIT 13 = 0. [BREADBOARD: BITS 12-00] THIS LOCATION IS NEVER RESET BY HARDWARE, INCLUDING RESET ISSUED BY THE -10 OR THE -11. THEREFORE SOFTWARE MUST SET IT TO A PARTICULAR VALUE AT SYSTEM STARTUP BEFORE DOING THE FIRST TRANSFER.

STATUS: STATUS WORD (174034 + 40*N)

DIFFERENT BITS IN THIS WORD ARE USED FOR ALL OPERATIONS.

-11 BIT	SYMBOL	NAME (+ FUNCTION)
15	TO10DN	TO-10 NORMAL TERMINATION READ: THE TO-10 BYTE COUNT WENT TO 0 OR THE -11 PROGRAM SET THE BIT STATUS [DON10S], TO10DN WILL NOT BE SET IF AN ERROR TERMINATION OCCURRED, SEE STATUS [TO10ER].
	DON10S	WRITE: IF 1, SET TO-10 NORMAL TERMINATION STATUS [TO10DN], THIS BIT ON A WRITE IS PROVIDED FOR DIAGNOSTIC PURPOSES ONLY, WRITING A 1 DOES NOT TERMINATE A TRANSFER IN PROGRESS.
14		READ: UNUSED
	DON10C	WRITE: IF 1, CLEAR TO-10 NORMAL TERMINATION STATUS [TO10DN]
13	TO10ER	TO-10 ERROR TERMINATION READ: AN NPR UNIBUS PARITY ERROR (DIAG3[NUPE]), PDP-11 MEMORY PARITY (STATUS[11MPE]), OR A UNIBUS TIMEOUT (NO BIT) OCCURRED DURING A TO-10 BYTE TRANSFER, OR THE -11 PROGRAM SET THE BIT STATUS [ERR10S], STATUS [TO10DN] WILL NOT BE SET, IF AN ERROR TERMINATION OCCURRED, THUS -11 PROGRAMS MUST TEST FOR BOTH TO10DN AND TO10ER.
	ERR10S	WRITE: IF 1, SET TO-10 ERROR TERMINATION STATUS [TO10ER], THIS BIT ON A WRITE IS PROVIDED FOR DIAGNOSTIC PURPOSES ONLY, WRITING A 1 DOES NOT TERMINATE A TRANSFER IN PROGRESS
12	RAMIS0	RAM IS ZEROS READ: THE DATA OUT OF A RAM LOCATION IS ALL 0'S, THIS BIT IS PROVIDED ON A READ FOR DIAGNOSTIC PURPOSES ONLY, IT HAS NO MEANING AND IS UNPREDICTABLE UNLESS THE DTE20 IS BEING SINGLE STEPPED.

	ERR10C	WRITE: IF 1, CLEAR TO-10 ERROR TERMINATION [TO10ER]
11	TO11DB	-10 REQUESTED -11 INTERRUPT
		READ: THE -10 HAS REQUESTED (CONO DTEN, TO11DB) AN -11 DOORBELL INTERRUPT.
	INT11S	WRITE: IF 1, SET -10 REQUESTS -11 INTERRUPT STATUS [TO11DB], THIS RESULTS IN AN -11 VECTOR INTERRUPT.
10	DXWRD1	DEXWORD 1. THIS BIT IS PROVIDED ON A READ FOR DIAGNOSTIC PURPOSES ONLY. IT HAS NO MEANING AND IS UNPREDICTABLE UNLESS THE DTE20 IS BEING SINGLE STEPPED.
	INT11C	WRITE: IF 1, CLEAR -10 REQUESTS -11 INTERRUPT STATUS [TO11DB]. THIS ENABLES MORE DOORBELL INTERRUPTS TO THE -11 TO BE GENERATED.
09	MPE11	-11 MEMORY PARITY ERROR
		READ: INDICATES THAT THE -11 MEMORY HAD A PARITY ERROR DURING A DATA FETCH FOR A TO-10 BYTE TRANSFER. PARITY ERRORS ARE DETECTED ONLY IF THE -11 HAS THE MF11UP OR MF11LP MEMORY PARITY OPTION.
	PERCLR	WRITE: IF 1, CLEAR THE -11 MEMORY PARITY ERROR FLAG STATUS [11MPE].
08	TO10DB	-11 REQUEST -10 INTERRUPT
		READ: THE -11 HAS REQUESTED A -10 DOORBELL INTERRUPT STATUS [INT10S] AND THE -10 HAS NOT YET CLEARED THE BIT (CONO DTEN, CL11PI)
	INT10S	WRITE: IF 1, SET REQUEST -10 INTERRUPT STATUS [TO10DB] AND CONI[TO10DB]. THIS RESULTS IN A VECTORED INTERRUPT TO EPT LOCATION 104 + 8*N IN KL10 EPT.
07	TO11DN	TO-11 TRANSFER DONE
		READ: THE TO-11 BYTE COUNT BECAME EQUAL TO 0 (TO11BC = 0), THE TRANSFER STOPPED ON A NULL CHARACTER (STATUS [NULSTP] = 1), OR THE -11 PROGRAM SET

THE BIT (STATUS [DON11S])

DON11S WRITE: IF 1, SET TO-11 NORMAL TERMINATION FLAG STATUS [TO11DN]. THIS BIT ON A WRITE IS PROVIDED FOR DIAGNOSTIC PURPOSES ONLY. WRITING A 1 DOES NOT TERMINATE A TRANSFER IN PROGRESS.

06 EBSEL E BUFFER SELECT. THIS BIT IS PROVIDED ON A READ FOR DIAGNOSTIC PURPOSES ONLY. IT HAS NO MEANING AND IS UNPREDICTABLE UNLESS THE DTE20 IS BEING SINGLE STEPPED.

DON11C WRITE: IF 1, CLEAR TO-11 NORMAL TERMINATION FLAG STATUS [TO11DN]

05 NULSTP NULL STOP

READ: THE TO-11 TRANSFER STOPPED BECAUSE THE STOP BIT WAS SET (TO11BC[ZSTOP] = 1).

INTRON WRITE: IF 1, ENABLE DTE20 TO GENERATE PDP-11 BR REQUESTS. WRITING 0 OR 1 DOES NOT CLEAR ANY INTERRUPTS WAITING. (INTROF DISABLES, INTSON READS)

04 BPARER EBUS PARITY ERROR [BREADBOARD: UNUSED]

READ: THE DTE20 DETECTED AN EBUS PARITY ERROR DURING A TO-11 DTE20 BYTE TRANSFER OR EXAMINE TRANSFER.

EBUSPC WRITE: IF A 1, CLEAR EBUS PARITY ERROR

03 RESTRICTED MODE

RM READ: IF 1, THE ATTACHED PDP-11 IS IN RESTRICTED MODE. IF 0, THE ATTACHED PDP-11 IS IN PRIVILEGED MODE. THE VALUE OF THIS BIT IS DETERMINED BY THE SETTING OF THE PRIVILEGED SWITCH ON THE DTE20. [BREADBOARD: SWITCH IS NOT PRESENT AND FRONT END IS ALWAYS PRIVILEGED]

INTROF WRITE: IF 1, DISABLE DTE20 FROM GENERATING PDP-11 BR REQUESTS. WRITING 0 OR 1 DOES NOT CLEAR ANY INTERRUPTS WAITING. (INTRON ENABLES.)

02 DEXDON DEPOSIT/EXAMINE DONE

READ: THE LAST DEPOSIT OR EXAMINE OPERATION HAS COMPLETED, NO INTERRUPT OCCURS, THE -11 MUST WATCH FOR THIS BIT TO BE SET AFTER EVERY DEPOSIT OR EXAMINE, THE DTE20 CLEARS STATUS [DEXDON] WHENEVER A DEPOSIT OR EXAMINE IS STARTED (BY WRITING TENAD2).

EBUSPS WRITE: IF A 1, SET EBUS PARITY ERROR

01 TO11ER TO-11 BYTE ERROR TERMINATION

READ: AN ERROR OCCURRED DURING A TO-11 BYTE TRANSFER, OR THE -11 PROGRAM SET THE BIT STATUS [ERR11S]. STATUS [TO11DN] WILL NOT BE SET IF AN ERROR TERMINATION OCCURRED, THUS PROGRAMS MUST TEST FOR BOTH TO11DN AND TO11ER.

ERR11S WRITE: IF 1, SET TO-11 ERROR TERMINATION FLAG STATUS [TO11ER]. THIS BIT ON A WRITE IS PROVIDED FOR DIAGNOSTIC PURPOSES ONLY, WRITING A 1 DOES NOT TERMINATE A TRANSFER IN PROGRESS.

00 INTSON INTERRUPTS ON

READ: IF 1, THE DTE20 IS ENABLED TO GENERATE -11 BR REQUESTS, IF 0, IT IS DISABLED, (INTRON ENABLES, INTROF DISABLES.)

ERR11C WRITE: IF 1, CLEAR TO-11 ERROR TERMINATION FLAG STATUS [TO11ER]

TENAD1: TEN ADDRESS WORD 1 (TENAD1 = 174010 + 40*N)

EXCEPT PRTOFF, THIS WORD IS NOT RESET SO THAT THE -11 PROGRAM NEED NOT WRITE IT FOR EACH OPERATION.

-11 BIT	SYMBOL	NAME
15-13	SPACE	ADDRESS SPACE

READ:

THIS FIELD IS USED FOR UNPROTECTED EXAMINES AND DEPOSITS ONLY. THEREFORE THE DTE20 INTERPRETS THIS FIELD ONLY ON PRIVILEGED FRONT ENDS WHEN TENAD1[PRTOFF] = 1.

0	EXEC PROCESS TABLE (EPT)
1	EXEC VIRTUAL
2	USER PROCESS TABLE (UPT)
3	USER VIRTUAL
4	PHYSICAL
5-7	UNDEFINED - RESERVED TO DEC

12	DEP	DEPOSIT (DEP)

READ:

WRITE: -11 SOFTWARE SPECIFIES WHETHER A DEPOSIT (1) OR AN EXAMINE (0) FUNCTION IS TO BE DONE WHENEVER THE -11 WRITES WORD TENAD2. ON A READ OF TENAD1, DEP IS A 1 IF THE LAST DEPOSIT/EXAMINE WAS A DEPOSIT OR A DIAGNOSTIC DEPOSIT; IT IS A 0 IF THE LAST DEPOSIT/EXAMINE WAS AN EXAMINE OR A DIAGNOSTIC EXAMINE.

11	PRTOFF	EXAMINE/DEPOSIT PROTECT OFF

[BREADBOARD: NOT PRESENT] THIS BIT IS IGNORED IF THE DTE20 IS SWITCHED TO RESTRICTED MODE. FOR A PRIVILEGED FRONT END, IF A 1, EXAMINES AND DEPOSITS ARE NOT RELOCATED AND NO PROTECTION CHECK IS MADE ON THEM. INSTEAD THE ADDRESS SPACE DESIRED IS INDICATED BY THE SPACE FIELD. THE

DTE20 RESETS THE ASSOCIATED HARDWARE BIT AFTER EVERY EXAMINE AND DEPOSIT TO REDUCE THE CHANCE THAT A RUNAWAY PROGRAM IN A PRIVILEGED FRONT END -11 WILL CLOBBER THE -10. A RESTRICTED FRONT END CAN ONLY EXAMINE OR DEPOSIT IN THE COMMUNICATION REGIONS DEFINED BY THE -10. A READ OF TENAD1 RETURNS THE VALUE WRITTEN BY THE LAST WRITE OF TENAD1 AND SO MAY DIFFER FROM THE ASSOCIATED HARDWARE BIT WHICH CONTROLS THE PROTECTION. IF PRTOFF = 0, EXAMINES ARE RELOCATED BY THE EPT.

10-09 UNUSED

READ: SHOULD BE 0

WRITE: MBZ THESE BITS DO NOT GO OUT ON EBUS.

08-07 UNIMPLEMENTED

READ: SHOULD BE 0

WRITE: MBZ THESE BITS GO OUT ON EBUS.

06-00 HIGH ORDER KL10 ADDRESS BITS

THE -11 PROGRAM WRITES THE DESIRED KL10 ADDRESS BITS 13-19 ON EXAMINES OR DEPOSIT FUNCTIONS.

TENAD2: TEN ADDRESS WORD 2 (174012 + 40*N)

THE DTE20 PERFORMS AN EXAMINE OR DEPOSIT EVERY TIME TENAD2 IS WRITTEN AS SPECIFIED BY THE TENAD1 [DEP] BIT.

-11 BIT NAME

15-00 LOW ORDER KL10 ADDRESS BITS

THE -11 PROGRAM MUST WRITE TENAD2 IN ORDER TO START AN EXAMINE OR DEPOSIT. WRITING TENAD2 CLEARS THE STATUS [DEXDON] BIT. STATUS [DEXDON] IS SET TO 1 WHEN THE EXAMINE OR DEPOSIT IS FINISHED. THE INTERPRETATION OF THE ADDRESS BITS DEPENDS ON THE FIELD TENAD1[SPACE] AND TENAD1[PRTOFF].

TO10AD: TO-10 PDP-11 MEMORY ADDRESS (174020 + 40*N)

-11 BITS	NAME
15-00	BYTE ADDRESS OF SOURCE STRING [BREADBOARD: BIT 00 INDICATES WORD OR BYTE MODE IN TO-11 TRANSFERS, BITS 15-01 ARE WORD ADDRESS OF STRING], THIS WORD IS UPDATED AS EACH BYTE (WORD) IS TRANSFERRED, AT THE END OF A TRANSFER, IT POINTS TO THE BYTE (WORD) WHICH WOULD HAVE BEEN TRANSFERRED NEXT.

TO10BC: TO-10 BYTE COUNT (174014 + 40*N)

THIS LOCATION IS PROVIDED FOR MONITORING PURPOSES ONLY, IT IS READABLE AND WRITEABLE BY THE PDP-11, HOWEVER THE DTE20 WILL NOT START THE TRANSFER UNTIL THE -10 SETS THIS REGISTER WITH A DATAO DTEN,

-11 BITS	NAME
15	INTERRUPT BOTH PROCESSORS AT THE COMPLETION OF THE CURRENT TO-10 TRANSFER
14-12	0'S
11-00	NEGATIVE BYTE COUNT

TO10DT: TO-10 PDP-11 DATA WORD (174024 + 40*N)

THIS WORD IS PROVIDED FOR MONITORING PURPOSES ONLY,

-11 BITS	NAME
15-08	HIGH ORDER BYTE -11 BYTE MODE: 0S -11 WORD MODE: -11 BITS WHICH WILL BECOME KL10 BITS 20-27
07-00	LOW ORDER BYTE KL10 BITS 28-35

TO11AD: TO-11 PDP-11 MEMORY ADDRESS (174022 + 40*N)

-11 BITS	NAME
15-00	READ: NEXT -11 BYTE ADDRESS IN MEMORY TO BE TRANSFERRED TO. MAY BE READ DURING A TRANSFER. WRITE: BYTE ADDRESS IN -11 MEMORY OF WHERE TO STORE NEXT BYTE RECEIVED FROM -10. [BREADBOARD: BIT 00 IS 1 IF BYTE MODE, 0 IF WORD MODE, BITS 15-01 ARE WORD ADDRESS]. THIS WORD IS UPDATED AS EACH BYTE (WORD) IS TRANSFERRED. AT THE END OF A TRANSFER, IT POINTS TO THE BYTE (WORD) WHICH WOULD HAVE BEEN TRANSFERRED NEXT.

TO11BC: TO-11 BYTE COUNT (174016 + 0*N)

-11 BITS	NAME
15	I READ: VALUE OF LAST WRITE TO THE REGISTER BY THE SOFTWARE AS STORED IN THE RAM. NOTE: THE I FLIP-FLOP IS RESET TO 0 ON TERMINATIONS AND THE ASSOCIATED RAM BIT IS NOT USED. WRITE: IF 0, ON NORMAL TERMINATION INTERRUPT ONLY THE -11. IF 1, ON NORMAL TERMINATION INTERRUPT BOTH CPUS. IF AN ERROR OCCURS, THE I BIT IS IGNORED AND BOTH CPUS ALWAYS GET AN ERROR TERMINATION INTERRUPT. A PRIVILEGED FRONT END CLEAR ISSUED BY A -10 RESET OR -11 MSTCLR RESETS THE ASSOCIATED HARDWARE BIT. A READ OF TO11BC RETURNS THE LAST VALUE WRITTEN BY SOFTWARE AND MAY DIFFER FROM THE ASSOCIATED HARDWARE BIT IF A PRIVILEGED FRONT END CLEAR HAS OCCURRED.
14	ZSTOP READ: VALUE OF LAST WRITE TO THE REGISTER BY THE SOFTWARE AS STORED IN THE RAM. NOTE: THE ZSTOP FLIP-FLOP IS RESET TO 0 ON TERMINATIONS AND THE ASSOCIATED RAM BIT IS NOT RESET.

WRITE: IF 1, STOP ON NULL CHARACTER RECEIVED FROM -10 AFTER STORING IT IN THE -11, TO11AD IS NOT INCREMENTED SO THAT THE NEXT TRANSFER CAN START BY OVERWRITING THE NULL CHARACTER IF THE SOFTWARE WISHES. A PRIVILEGED FRONT END CLEAR ISSUED BY A -10 RESET OR -11 MSTCLR RESETS THE ASSOCIATED HARDWARE BIT. A READ OF TO11BC RETURNS THE LAST VALUE WRITTEN BY SOFTWARE AND MAY DIFFER FROM THE ASSOCIATED HARDWARE BIT IF A PRIVILEGED FRONT END CLEAR HAS OCCURRED.

13 TO11BM

READ: VALUE OF LAST WRITE TO THE REGISTER BY THE SOFTWARE AS STORED IN THE RAM. NOTE: THE TO11BM FLIP-FLOP IS RESET TO 0 ON TERMINATION AND THE ASSOCIATED RAM BIT IS NOT RESET.

WRITE: IF 1, SET BYTE MODE, IF 0, SET WORD MODE FOR TO-11 TRANSFER. SEE TO10BC [TO10BM] FOR TO-10 DIRECTION.

12 UNUSED

11-00 NEGATIVE BYTE COUNT

READ: THIS LOCATION IS MODIFIED BY HARDWARE ONLY DURING TO 11 TRANSFERS. IT MAY BE READ DURING TRANSFERS. IT IS NOT AFFECTED ON TO-10 TRANSFERS

WRITE: THE TRANSFER WILL BEGIN WHEN BOTH TO11AD AND TO11BC HAS BEEN WRITTEN SINCE LAST TERMINATION.

TO11DT: TO-11 PDP-11 DATA WORD (174026 + 40*N)

THIS WORD IS PROVIDED FOR MONITORING AND ERROR REPORTING PURPOSES ONLY. ON TERMINATION IT CONTAINS THE LAST BYTE (WORD) OF DATA TRANSFERRED.

-11 BITS	NAME
15-08	HIGH ORDER BYTE -11 BYTE MODE: KL10 BITS 28-35 OR KL10 BITS 20-27 -11 WORD MODE: KL10 BITS 20-27

07-00 LOW ORDER BYTE

-11 WORD MODE: KL10 BITS 28-35 -11
BYTE MODE: KL10 BITS 20-27 OR 28-35

19, APPENDIX - PDP-11 UNIBUS PARITY SPECIFICATION

19,1 FORWARD

THIS SPECIFICATION FOR UNIBUS PARITY WAS REVIEWED AND ACCEPTED AT A MEETING APRIL 29TH BY PDP-11 CENTRAL ENGINEERING AND WILL BE INCORPORATED INTO THE FORMAL UNIBUS SPECIFICATION CURRENTLY BEING DEVELOPED BY THAT GROUP,

19,2 UNIBUS PARITY SPECIFICATION GOALS

19,2,1 INCREASE OF DATA INTEGRITY - THOSE SYSTEMS WHICH IMPLEMENT THIS SPEC, WILL REALIZE AN INCREASE IN DATA INTEGRITY SINCE DETECTION OF ANY SINGLE BIT FAILURE IN EITHER ADDRESS OR DATA ON THE UNIBUS WILL FLAG THE SOFTWARE THAT AN ERROR HAS OCCURRED, ADDITIONALLY, SYSTEM THROUGHPUT WILL INCREASE BECAUSE OF A REDUCTION IN THE AMOUNT OF SOFTWARE ERROR CHECKING REQUIRED,

19,2,2 BACKWARD COMPATABILITY - THE SPECIFICATION MUST TAKE INTO CONSIDERATION THAT NOT ALL DEVICES ON THE UNIBUS WILL INCORPORATE THIS SPEC, AND THEREFORE MUST BE CREATED TO FREELY ALLOW THE MIXING OF PARITY CHECKING AND NON-PARITY CHECKING DEVICES ON THE UNIBUS,

19,2,3 ERROR DETECTION - THE HARDWARE MUST DETECT THE PICKING OR DROPPING OF ANY SINGLE BIT ON EITHER THE ADDRESS OR DATA LINES ON THE UNIBUS, THE EXTENT OF THE RECOVERY CAPABILITIES IS LIMITED ONLY BY THE EFFORT UNDERTAKEN BY THE SOFTWARE RUNNING AT THAT TIME,

19,2,4 DECREASE MEAN TIME TO REPAIR - THIS SPECIFICATION WILL SIGNIFICANTLY REDUCE THE MEAN TIME TO REAIR THE OPTIONS AND, IN GENERAL, THE SYSTEMS WHICH INCORPORATE IT, THE REDUCTION WILL BE GREATER IN THOSE SYSTEMS WHICH IMPLEMENT THIS SPECIFICATION ON ALL DEVICES,

19,3 GENERAL DESCRIPTION OF UNIBUS PARITY

THE STATES OF THE PA AND PB LINES ARE DEFINES AS FOLLOWS:

PA	PB	DEFINITION
0	0	NO CHECKING, NO ERROR
0	1	CORE MEMORY PARITY ERROR, NO UNIBUS PARITY CHECKING IN EFFECT
1	0	THE DEVICE TRANSMITTING THE DATA
1	1	BITS HAS GENERATED PARITY AND PB IS THE PARITY BIT,

THE SIGNALS BUS PA L AND BUS PB L WILL BE USED FOR UNIBUS PARITY INFORMATION. PA ASSERTED BY THE DEVICE ASSERTING THE DATA BITS WILL INDICATE THAT PARITY IS BEING TRANSMITTED BY THAT DEVICE. THE PARITY BIT WILL BE PB AND COMPUTED PARITY WILL BE ODD. IF AN ERROR IS DETECTED, THE DEVICE DETECTING THE ERROR WILL SET A BIT NAMED "UNIBUS PARITY ERROR" IN ITS STATUS REGISTER AND GENERATE AN INTERRUPT. PARITY WILL BE COMPUTED FOR THE COMBINED ADDRESS AND DATA BITS.

19.3.1 DATO AND DATOB

1. MASTER - THE MASTER COMPUTES ODD PARITY ON THE COMBINED ADDRESS AND DATA. IT ASSERTS THE PA LINE INDICATING THAT THE PB LINE CONTAINS THE STATE OF THE PARITY BIT. IT ASSERTS THE STATE OF THE COMPUTED PARITY BIT ON THE PB LINE. FOR TIMING PURPOSES, THE PA AND PB LINES WILL BE TREATED AS "DATA" LINES.

2. SLAVE - THE SLAVE COMPUTES THE PARITY ON THE RECEIVED COMBINED ADDRESS AND DATA. IT THEN COMPARES IT WITH THE STATE OF THE RECEIVED PARITY ON THE PB LINE. IF THE COMPUTED PARITY ISDIFFERENT FROM THE RECEIVED PARITY, A BUS PARITY ERROR HAS OCCURRED. THE SLAVE WILL SET A "BUS PARITY ERROR" BIT IN ITS CONTROL AND STATUS REGISTER AND GENERATE AN INTERRUPT.

19.3.2 DATI AND DATIP

1. SLAVE - THE SLAVE WILL ASSERT THE PA LINE INDICATING THAT THE PB LINE CONTAINS THE STATE OF THE PARITY BIT. THE SLAVE WILL ASSERT THE STATE OF THE COMPUTED ADDRESS AND DATA PARITY BIT ON THE PB LINE. IF, HOWEVER, THE SLAVE IS A DEVICE EQUIPPED WITH MEMORY PARITY, (MEMORY) AND IT HAS ALSO DETECTED A MEMORY PARITY ERROR, THE SLAVE WILL FOLLOW THE RULES FOR MEMORY PARITY. THAT IS, IT WILL NOT ASSERT THE PA LINE AND IT WILL SIGNAL THE MEMORY PARITY ERROR BY ASSERTING THE PB LINE. DEVICES WHICH IMPLEMENT THIS SPEC. MUST RECOGNIZE THIS MEMORY PARITY ERROR INDICATION.

2. MASTER - THE MASTER COMPUTES THE PARITY ON THE COMBINED TRANSMITTED ADDRESS AND RECEIVED DATA. IT THEN COMPARES IT WITH THE RECEIVED PARITY ON THE PB LINE. NOTE THAT THE RECEIVED PARITY HAS BEEN CREATED BY THE SLAVE USING THE ADDRESS IT RECEIVED AND DATA IS TRANSMITTED. AGAIN, PARITY HAS BEEN CHECKED ON THE COMBINED ADDRESS AND DATA. EITHER A CHANGE IN AN ADDRESS BIT - AS RECEIVED BY THE SLAVE, OR A CHANGE IN DATA BIT - AS RECEIVED BY THE MASTER WILL RESULT IN A BUS PARITY ERROR.

MAINTENANCE FEATURE - EACH DEVICE IMPLEMENTING THIS SPEC. MUST ALSO HAVE THE CAPABILITY TO GENERATE BAD (EVEN) PARITY ON COMMAND. THIS ALLOWS THE PARITY LOGIC THROUGHOUT THE SYSTEM TO

BE CHECKED FOR CORRECT OPERATION. ALSO, THE "BAD" DATA MUST BE CAPTURED BY A DEVICE WHEN AN ERROR OCCURS TO ASSIST IN FURTHER ISOLATION OF THE FAILURE.

TIMING CONSIDERATIONS - SINCE THE PRIMARY GOAL OF THIS SPECIFICATION IS AN INCREASE IN DATA INTEGRITY PROVIDED FOR SYSTEM SOFTWARE, ALL PARITY CHECKING AND ERROR DETECTION MAY BE ACCOMPLISHED "AFTER THE FACT." THIS MEANS THE UNIBUS TRANSFER RATE IS NOT HINDERED AND "BAD" DATA MAY BE USED BUT THE SYSTEM WILL BE FLAGGED THAT AN ERROR HAS OCCURRED AND MAY TAKE APPROPRIATE MEASURES TO RECOVER.

ADDITIONAL CONTROL CONSIDERATIONS - ALTHOUGH NOT REQUIRED BY THIS SPECIFICATION VARIOUS DEVICES MAY INCORPORATE ADDITIONAL FEATURES. IF DEVICE TIMING PERMITS, THE PARITY CHECKING MAY BE ACCOMPLISHED BEFORE THE DATA IS USED AND THE OPERATION IN PROGRESS ABORTED IS AN ERROR IS DETECTED. AGAIN, THESE ARE NOT STRICT REQUIREMENTS OF THIS SPECIFICATION BUT INCLUSION OF FEATURES OF THIS TYPE WOULD FURTHERSTRENGTHEN THE DEVICE'S SUPPORT OF THE GOALS OF THIS SPECIFICATION. IT IS EXPECTED THAT DESIGNERS OF NEW EQUIPMENT WOULD CONSIDER INCLUSION OF THIS EXTENDED CONTROL FACILITY IN THEIR EQUIPMENT.

19.4 ADVATAGES OF THIS SPECIFICATION

1. DATA INTEGRITY OF TRANSFERS ON THE UNIBUS.

2. NO NEW UNIBUS SIGNAL LINES NEEDED AND THEREFORE NO ECOS ETC. TO EXISTING EQUIPMENT.

3. DECREASE IN MTTR BECAUSE ERRORS ARE NOW DETECTED.

19.5 IMPLEMENTATION

APPROVAL OF THIS SPECIFICATION IS NOT INTENDED TO TRIGGER THE REWORK OF ALL EXISTING OPTIONS BUT WOULD AID IN DIRECTION OF FUTURE DESIGN EFFORTS, HOWEVER REWORK IS NOT TOTALLY RULED OUT. THOSE GROUPS SEEING A NEED TO REWORK AN EXISTING DEVICE SHOULD NOT MAKE THE CHAGES IN THE FORM OF ECO BUT RATHER GENERATE A NEW OPTION DESIGNATION MODIFIER SUCH AS -P INDICATING THE PARITY CAPABILITIES. THESE MODIFIED OPTIONS MIGHT BE SOLD AT A SLIGHTLY HIGHER PRICE TO CUSTOMERS DESIRING THE DATA INTEGRITY TO HELP DEFER THE DEVICE MODIFICATION COSTS.

THESE SALES WOULD MOST LIKELY OCCUR WHEN THE SYSTEM IS SOLD INTO AN ENVIRONMENT WHERE ERRORS CANNOT GO UNDETECTED SUCH AS ANY COMMERCIAL ENVIRONMENT.

20. APPENDIX - LOOSE ENDS

1. ASSIGNMENT OF DTE20 ADDRESSES WITH -11 ENGINEERING SO THAT THEY CONSIDER THE DTE20 A DIGITAL DEVICE RATHER

THAN JUST A -10 SPECIAL.

2. WHAT'S ON THE 2 DS LINES WHICH ARE NOW 0 WHILE SYSTEM RUNS?
HOW ABOUT USER MODE AND CONCEALED MODE?

21. INDEX

[END OF CH4S03,SPC]

1080, 2040, 2060 ENGINEERING FUNCTIONAL SPEC - CHAP 5.1

TO: KL10 LIST

TITLE: CHANNEL BUS (CBUS) - MBOX/RH20 INTERFACE - REV 0

STATUS: THIS CHAPTER IS ESSENTIALLY THE SAME AS THE SUPERSEDED MEMO. PARITY WILL BE ADDED AS A REVISION. AT WHICH TIME THERE WILL BE A REVIEW.

FILE: [EFS]CH5S01.SPC

PDM#: 200-200-033-00

DATE: 25 FEB 74

SUPERSEDED MEMOS: M BUS [SIC] FUNCTIONAL SPECIFICATION, V. KU, 25 APRIL 73

ENGINEER: V. KU

APPROVED:

EDITOR: T. HASTINGS

TYPIST: M. PROUTY

REVIEWED:

ABSTRACT

THE CHANNEL BUS (CBUS) CONNECTS THE 8 BUILT-IN CHANNELS IN THE MBOX WITH THE UP TO 8 MASSBUS CONTROLLERS (RH20). THE CBUS PASSES BOTH CONTROL INFORMATION AND DATA AT HIGH SPEED.

REVISION HISTORY

REV DESCRIPTION CHG NO ORIG DATE APPD BY DATE

1.0 INTRODUCTION

THE CBUS IS A SYNCHRONOUS BUS SYSTEM CONNECTING THE MBOX TO A MAXIMUM OF EIGHT MASSBUS CONTROLLERS (SEE FIG. 1). THE FIRST FOUR HANDLE A 1 MICROSECOND DATA RATE AND THE SECOND FOUR A 2 MICROSECOND DATA RATE.THE MBOX IS A LOGICAL UNIT WHICH, AMONG OTHER FUNCTIONS, COMMUNICATES DIRECTLY WITH THE MEMORY SYSTEM. EACH MASSBUS CONTROLLER CAN CONTROL UP TO A MAXIMUM OF EIGHT DRIVES (FIXED-HEAD DISK, MOVING-HEAD DISK, MAGTAPE DRIVE). THE PURPOSE OF THE CBUS IS TO TRANSMIT HIGH SPEED DATA AND CONTROL INFORMATION BETWEEN THE MBOX AND THE MASSBUS CONTROLLERS.

2.0 REFERENCE

1. "EBUS SPECIFICATION" AS REVISED, VIC KU, APRIL 17, 1973
2. "WIRING RULES FOR M&E BUS AND CLOCK DISTRIBUTION" AS REVISED, SULTAN ZIA, FEBRUARY 7, 1973
3. MASSBUS CONTROLLER HARDWARE SPECIFICATION" AS REVISED, VIC KU, FEB. 6, 1973
4. "MASSBUS INTERFACE STANDARD" AS REVISED, OCTOBER 17, 1972

3.0 TERMINOLOGY

1. CONTROLLER - MASSBUS CONTROLLER
2. INPUT DATA TRANSFER - ONE WORD TRANSFERRED FROM A CONTROLLER TO THE MBOX VIA THE CBUS
3. OUTPUT DATA TRANSFER - ONE WORD TRANSFERRED FROM THE MBOX TO A CONTROLLER VIA THE CBUS
4. INPUT BLOCK TRANSFER - A SERIES (ONE OR MORE) OF INPUT DATA TRANSFERS
5. OUTPUT BLOCK TRANSFER - A SERIES (ONE OR MORE) OF OUTPUT DATA TRANSFERS

4.0 GENERAL THEORY OF OPERATION - SEE FIG. 4 FOR THE TIMING OF THE INPUT/OUTPUT BLOCK TRANSFER.

CLOCK-TIME-DIVISION - MULTIPLEXING TECHNIQUE IS USED TO CONTROL THE CBUS OPERATIONS. A FREE RUNNING CLOCK (8 MHZ)

EXISTS IN THE EBOX AND THE CLOCK IS SENT TO EACH CONTROLLER VIA THE EBUS AND IS SENT TO THE MBOX BY INTERNAL CONNECTIONS. A DELAY LINE PER CONTROLLER IS USED TO SYNCHRONIZE (DESKEW) THE SIGNALS BETWEEN EACH CONTROLLER AND THE MBOX.

THE MBOX CONTINUOUSLY SELECTS ONE OF THE EIGHT CONTROLLERS BY GENERATING EIGHT CONTROLLER SELECTION LINES IN THIS SEQUENCE: 0,1,2,3,4,5,0,1,2,3,6,7;0,1,2,3,4,5,... (SEE FIG. 3). THE SEQUENCE IS STEPPED WITH THE LEADING EDGE OF THE CLOCK SIGNAL.

A CONTROLLER IS ALLOWED TO TRANSMIT OR RECEIVE DATA AND CONTROL INFORMATIONS ONLY WHEN IT HAS BEEN SELECTED BY THE MBOX. FIG. 3 SHOWS THE FOUR CYCLES USED BY THE MBOX AND A CONTROLLER DURING A DATA TRANSFER OPERATION. EACH CYCLE IS ASSERTED BY THE LEADING EDGE OF A CLOCK PULSE AND IS NEGATED BY THE LEADING EDGE OF THE NEXT CLOCK PULSE.

1. SELECT CYCLE - THE "SELECT" LINE OF A PARTICULAR CONTROLLER IS ASSERTED THROUGHOUT THIS CYCLE.

2. REQUEST CYCLE - THE SELECTED CONTROLLER WILL ASSERT THE "REQUEST" LINE (IF DATA REQUEST IS NEEDED) THROUGHOUT THIS CYCLE.

3. WAIT CYCLE - THIS CYCLE IS USED BY THE MBOX TO PREPARE DATA AND STATUS FOR TRANSMISSION. NEITHER DATA NOR STATUS IS ASSERTED DURING THIS CYCLE.

4. DATA CYCLE - DATA ARE PUT ON THE "DATA" LINES EITHER BY THE MBOX (OUTPUT DATA TRANSFER) OR BY THE CONTROLLER (INPUT DATA TRANSFER) DURING THIS CYCLE. THE RECIPIENT OF THE DATA WILL STROBE THE DATA LINES AT THE TRAILING EDGE OF THE DATA CYCLE. ALL CONTROL LINES (ERRORS, READY, LAST WORD, CTOM, START, RESET, DONE AND STORE) EXCEPT THE "REQUEST" LINE ARE ALLOWED TO BE ASSERTED DURING THIS CYCLE ONLY.

TRANSFER RATE - CONTROLLERS 0,1,2, AND 3 ARE SELECTED TWICE AS OFTEN BY THE MBOX'S SELECTION SEQUENCE AS CONTROLLERS 4,5,6 AND 7.

1. THE MAXIMUM TRANSFER RATE OF CONTROLLER 0,1,2,3 IS 1 USEC/WORD.

2. THE MAXIMUM TRANSFER RATE OF CONTROLLER 4,5,6 AND 7 IS 2 US/WORD.

5.0 CBUS SIGNALS (SEE FIG. 2) - ALL SIGNALS EXCEPT THE EIGHT RADIAL "SELECT" LINES ARE DAISY-CHAIN LINES.

1. CATA (D00-D35); BIDIRECTIONAL

THESE 36 LINES CARRY THE HIGH SPEED DATA AND ARE VALID ONLY DURING "DATA CYCLE" (SEE FIG. 4.2). THE MBOX SHALL APPLY ZEROS ON THE"DATA" LINES FOR A CONTROLLER (DURING ITS DATA CYCLE) WHENEVER THERE IS NO DATA TRANSFER REQUEST FROM THE CONTROLLER.

2. SECLECT (SEL 0 - SEL 7); MBOX TO CONTROLLER.

THESE EIGHT RADIAL LINES ARE CONTINUOUSLY SENT TO SELECT ONE CONTROLLER AT A TIME EVERY 125NS (SEE FIG. 3 FOR TIMING INFORMATION). THE SELECT LINE OF A CONTROLLER DEFINES THE BEGINNING OF ITS FOUR DATA TRANSFER CYCLES (SELECT CYCLE, REQUEST CYCLE, WAIT CYCLE AND DATA CYCLE).

3. ERROR; MBOX TO CONTROLLER

THE MBOX SHALL ASSERT THIS LINE (DURING DATA CYCLE ONLY) TO INFORM THE CONTROLLER THAT THE CURRENT INPUT/OUTPUT BLOCK TRANSFER MUST NOT CONTINUE DUE TO ERROR CONDITIONS DETECTED IN THE MBOX (SEE 7.0). THE CONTROLLER, UPON SENSING THE ERROR SIGNAL, SHALL TERMINATE THE BLOCK TRANSFER BY NOT MAKING ANY MORE DATA REQUEST AND SHALL ASSERT "DONE" (SEE 5.10) EXACTLY ONCE DURING A SUBSEQUENT DATA CYCLE. THE ERROR LINE SHALL BE NEGATED BEFORE THE MBOX NEGATES THE "READY" LINE (5.4.1). IF THE ERROR LINE IS DETECTED AFTER THE "READY" LINE IS NEGATD IT MAY BE INTERPRETED BY A CONTROLLER TO BE ERROR ASSOCIATED WITH THE NEXT BLOCK TRANSFER.

4. READY; MBOX TO CONTROLLER

THE MBOX SHALL ASSERT THIS LINE (DURING THE DATA CYCLE ONLY) AFTER IT DETECTS A "START" SIGNAL SENT BY A CONTROLLER AND AFTER THE MBOX IS READY FOR DATA TRANSFER. FOR OUTPUT BLOCK TRANSFER, THE MBOX SHALL HAVE AT LEAST TWO WORDS OF DATA FROM MEMORY BEFORE ASSERTING "READY". THE "READY" SIGNAL, ONCE ASSERTED, SHALL BE NEGATED ONLY AFTER SENSING THE "DONE" SIGNAL AND AFTER THE MBOX IS PREPARED TO START ANOTHER BLOCK TRANSFER OPERATION.

5. LAST WORD; MBOX TO CONTROLLER

THE MBOX (FOR AN OUTPUT BLOCK TRANSFER ONLY) SHALL ASSERT THIS LINE (DURING THE DATA CYCLE) AT THE SAME TIME THE LAST DATA WORD IS SENT TO A CONTROLLER. NO MORE DATA REQUEST SHALL BE MADE BY THE CONTROLLER AFTER DETECTING "LAST WORD".

6. REQUEST; CONTROLLER TOMBOX

A CONTROLLER SHALL ASSERT REQUEST, DURING ITS REQUEST CYCLE, WHEN:

1. ONE OF ITS DATA BUFFER IF FULL (FOR AN INPUT BLOCK TRANSFER OPERATION).

2. ONE OF ITS DATA BUFFER IS EMPTY (FOR AN OUTPUT BLOCK TRANSFER OPERATION).

A CONTROLLER SHALL NOT ASSERT REQUEST IF:

1. "READY" LINE IS NOT ASSERTED BY THE MBOX

2. "ERROR" LINE HAS BEEN ASSERTED BY THE MBOX DURING THE CURRENT BLOCK TRANSFER.

3. "LAST WORD" HAS BEEN ASSERTED BY THE MBOX DURING THE CURRENT BLOCK TRANSFER.

4. "DONE" HAS BEEN ASSERTED BY THE CONTROLLER DURING THE CURRENT BLOCK TRANSFER.

FOR AN INPUT DATA TRANSFER, THE CONTROLLER SHALL PLACE DATA (THROUGHOUT ITS DATA CYCLE) ON THE "DATA" LINES AND THE MBOX SHALL STROBE THE "DATA" LINES AT THE TRAILING EDGE OF THE SAME DATA CYCLE. FOR AN OUTPUT DATA TRANSFER, THE ABOVE OPERATION IS REVERSED.

7. CTOM (CONTROLLER TO MBOX); CONTROLLER TO MBOX

A CONTROLLER BEGINS A BLOCK TRANSFER BY ASSERTING "START" FOR EXACTLY ONE DATA CYCLE. THE CONTROLLER SHALL INFORM THE MBOX DURING THE SAME CYCLE, THE DIRECTION OF THE BLOCK TRANSFER BY:

1. ASSERTING CTOM FOR AN INPUT BLOCK TRANSFER

2. NEGATING CTOM FOR AN OUTPUT BLOCK TRANSFER

8. START; CONTROLLER TO MBO

A CONTROLLER SHALL ALWAYS BEGIN A BLOCK TRANSFER BY ASSERTING THIS LINE ONCE DURING ITS DATA CYCLE. THE LINE SHALL BE ASSERTED ONLY WHEN "READY" IS NEGATED. THE CBUS SHALL ASSERT "READY" WHEN IT IS PREPARED FOR DATA TRANSFER.

9. RESET; CONTROLLER TO MBOX

THIE SIGNAL MAY BE ASSERTED BY A CONTROLLER DURING ANY OF ITS DATA CYCLE. THE MBOX, UPON DETECTING THIS SIGNAL, SHALL:

1. CLEAR THE DATA BUFFERS ASSOCIATED WITH THE CONTROLLER

2. RESET THE COMMAND LIST POINTER ASSOCIATED WITH THE CONTROLLER TO THE INITIAL ADDRESS.

3. NEGATE ALL STATUS AND DATA LINES ASSOCIATED WITH THE CONTROLLER AFTER 1 AND 2 ARE DONE.

10. DONE; CONTROLLER TO MBOX

THE CONTROLLER SHALL TERMINATE A BLOCK TRANSFER BY ASSERTING THIS SIGNAL ONCE DURING ITS DATA CYCLE. NO MORE DATA REQUEST SHALL BE MADE AFTER "DONE" IS ASSERTED. THE MBOX, AFTER DETECTING "DONE", SHALL GET READY FOR A NEW BLOCK TRANSFER (EMPTY THE INPUT DATA BUFFERS, ETC.). THE ERROR LINE CAN STILL BE USED TO INFORM THE CONTROLLER THAT ERROR HAS BEEN DETECTED IN THE CURRENT BLOCK TRANSFER AS LONG AS THE "READY" LINE IS NOT NEGATED.

THE MBOX SHALL NEGATE THE "READY" LINE ONLY WHEN IT IS PREPARED FOR A NEW BLOCK TRANSFER.

11. STORE; CONTROLLER TO MBOX

THE CONTROLLER SHALL SEND "STORE" TO THE MBOX ONCE (AT THE SAME TIME THE CONTROLLER SENDS "DONE") WHEN:

1. THE CURRENT BLOCK TRANSFER IS TERMINATED DUE TO ERRORS DETECTED IN THE CONTROLLER AND/OR

2. THE CURRENT BLOCK TRANSFER COMMAND IN THE CONTROLLER SPECIFIES THAT "STORE" BE SENT TO THE MBOX AT THE CONCLUSION OF THE BLOCK TRANSFER.

   THE MBOX, UPON DETECTING "STORE", SHALL WRITE ALL STATUS INFORMATIONS ASSOCIATED WTH THE CONTROLLER INTO MEMORY. DETAILED STATUS DESCRIPTIONS ARE NOT COMPLETELY DEFINED AND WILL BE PUBLISHED LATER BY PAUL GUGLIELMI AND ALAN KOTOK.

   THE MBOX SHALL KEEP "READY" ASSERTED UNTIL IT IS PREPARED TO DO ANOTHER BLOCK TRANSFER.

7.0 MBOX ERROR CONDITIONS

UPON DETECTING AN ERROR CONDITION, THE MBOX SHALL ASSERT THE ERROR SIGNAL DURING A DATA CYCLE.

ERROR CONDITIONS DETECTED BY THE MBOC SHALL INCLUDE BUT NOT BE LIMITED TO:

1. MEMORY PARITY ERROR (COMMAND WORD AND DATA WORD)

2. INPUT DATA OVERRUN - THIS ERROR OCCURS WHEN THE CONTROLLER IS INPUTINGDATA (INPUTBLOCK TRANSFER) FASTER THAN THE MBOX CAN WRITE INTO MEMORY.

3. OUTPUT DATA OVERRUN - THIS ERROR OCCURS WHEN THE CONTROLLER IS REQUESTING DATA (OUTPUT BLOCK TRANSFER) FASTER THAN THE MBOX CAN READ FROM MEMORY.

4. WORD COUNTER TOO SHORT - THIS ERROR OCCURS WHEN THE WORD COUNTER SPECIFIED IN THE CHANNEL COMMAND WORD FOR AN INPUT BLOCK TRANSFER OPERATION IS SMALLER THAN THE NUMBER OF DATA WORDS SENT IN FROM A CONTROLLER. THIS ERROR IS DETECTED WHEN THE MBOX FETCHES AND DETECTS A "STOP CHANNEL COMMAND WORD AND (A) THERE IS MORE DATA IN ITS INPUT DATA BUFFER AND (B) HAS NOT RECEIVED A "DONE" SIGNAL FROM THE CONTROLLER.

   WORD COUNTER TOO SHORT ERROR FOR AN OUTPUT BLOCK TRANSFER IS DETECTED BY THE CONTROLLER (SEE 8.2).

5. WORD COUNTER TOO LONG - FOR AN INPUT BLOCK TRANSFER, THIS ERROR IS DETECTED WHEN THE WORD COUNTER IN THE MBOX ASSOCIATED WITH THE CONTROLLER IS NOT AT ZERO WHEN (A) DATA IS RECEIVED, AND (B) ALL THE INPUT DATA BUFFERS IN THE MBOX ARE EMPTIED.

   FOR AN OUTPUT BLOCK TRANSFER THIS IS DETECTED WHEN "DONE" IS RECEIVED AND (A) THE OUTPUT DATA BUFFERS IN THE MBOX ARE NOT EMPTIED AND/OR (B) THE WORD COUNTER IN THE MBOX IS NOT AT ZERO.

8.0 CONTROLLER ERROR CONDITIONS

UPON DETECTING AN ERROR CONDITION, THE CONTROLLER SHALL:

1. STOP ANY MORE DATA TRANSFER REQUEST

2. TERMINATE THE BLOCK TRANSFER BY SENDING "DONE" AND "STONE" TO MBOX

3. NOT EXECUTE ANOTHER BLOCK TRANSFER UNTIL ALL ERROR CONDITIONS HAVE BEEN CLEARED BY PROGRAM.

ERROR CONDITIONS DETECTED BY THE CONTROLLER SHALL INCLUDE BUT NOT LIMITED TO:

1. ALL MASSBUS ERROR CONDITIONS

2. WORD COUNTER TOO SHORT - THIS ERROR OCCURS WHEN A MASSBUS DEVICE REQUESTS DATA (OUTPUT BLOCK TRANSFER) FROM THE CONTROLLER AND THERE IS NO MORE DATA IN THE CONTROLLER DUE TO THE FACT THAT "LAST WORD" HAS BEEN SENT BY THE MBOX TO INDICATE THAT THE LAST WORD OF THE BLOCK TRANSFER IS SENT.

3. WORD COUNTER TOO LONG - THIS OCCURS WHEN A MASSBUS DEVICE HAS REQUESTED ALL THE DATA IT NEEDED (OUTPUT BLOCK TRANSFER) BUT THE CONTROLLER (A) HAS MORE DATA IN ITS OUTPUT DATA BUFFERS AND/OR (B) HAS NOT RECEIVED A "LAST WORD" SIGNAL FROM THE MBOX.

4. AN "ERROR" SIGNAL IS RECEIVED (SENT BY THE MBOX).

5. INPUT DATA OVERRUN - THIS ERROR OCCURS WHEN A MASSBUS DEVICE IS INPUTTING DATA FASTER THAN THE CONTROLLER IS ABLE TO TRANSFER INTO THE MBOX.

6. OUTPUT DATA OVERRUN - THIS ERROR OCCURS WHEN A MASSBUS DEVICE IS REQUESTING DATA FASTER THAN THE CONTROLLER IS ABLE TO OBTAIN FROM THE MBOX.

9. WIRING RULES

REFER TO "WIRING RULES FOR M & E BUS AND CLOCK DISTRIBUTION" (SEE 2.2) FOR ALL PROPAGATION DELAYS, IC TYPES AND CABLE INFORMATIONS.

10. WORD COUNTER REQUIREMENT

"WORD COUNTER TOO SHORT" (7.4 AND 8.2) AND "WORD COUNTER TOO LONG" (7.4 AND 8.3) ERROR CONDITIONS ARE MEANINGFUL ONLY IF THE WORD COUNTER SPECIFIED IN A CHANNEL COMMAND WORD FOR A BLOCK TRANSFER IS EXACTLY N TIMES (N = 0,1,...) THE BLOCK SIZE OF A MASSBUS DEVICE (64 WORDS PER SECTOR FOR THE RS04 DRIVE, ETC.).

THE CONTROLLER WILL NOT EXECUTE THE BACKUP BLOCK TRANSFER COMMAND STORED IN THE CONTROLLER WHENEVER THE CURRENT BLOCK TRANSFER COMMAND IS TERMINATED DUE TO TRANSFER ERROR (SEE 7.0 AND 8.0). THEREFORE, THE ABOVE MENTIONED WORD COUNTER REQUIREMENT SHOULD BE USED IN WRITING SOFTWARE IN ORDER TO KEEP SYSTEM THROUGHPUT HIGH BY AVOIDING THROUGHPUT DELAYS CAUSED BY UNNECESSARY AND MISLEADING ERROR CONDITIONS.

[END OF CH5S01.SPC]

1080,2040,2060 ENGINEERING FUNCTIONAL SPEC - CHAP 5,2

TO: KL10 LIST

TITLE: EBUS - EBOX TO RH20, DTE20, DIA20 INTERFACE-REV 0

STATUS: THIS SPEC IS THE SAME AS THE SUPERSEDED ONE. A BRIEF SPEC REVIEW WILL BE HELD ON FROM TO . THIS SPEC DOES NOT INCLUDE ANY PARITY ADDITIONS. IT WILL BE REVISED AND REVIEWED AT THAT TIME ALSO.

FILE: [EFS]CH5S02.SPC

PDM # 200-200-004-00

DATE: 26 FEB 74

SUPERSEDED SPECS: "E" BUS SPECIFICATION, V. KU, 17 APRIL 73

ENGINEER: V. KU

APPROVED:

EDITOR: T. HASTINGS

TYPIST: M. PROUTY

REVIEWED:

ABSTRACT

THE EBUS IS USED TO CONNECT ALL KL10 CONTROLLERS, IN PARALLEL, WITH ONE END AT THE EBOX. A MAXIMUM OF THIRTEEN KL10 CONTROLLERS ARE IMPLEMENTED: UP TO 4 10/11 INTERFACES (DTE20), UP TO 8 MASSBUS CONTROLLERS (RH20), AND 1 I/O BUS INTERFACE (DIA20).

REVISION HISTORY

REV DESCRIPTION CHG NO ORIG DATE APPD BY DATE

1. INTRODUCTION

THE EBUS IS USED TO CONNECT ALL KL10 CONTROLLERS, IN PARALLEL, WITH ONE END AT THE EBOX. A MAXIMUM OF THIRTEEN KL10 CONTROLLERS ARE IMPLEMENTED: UP TO FOUR "10/11 INTERFACES" (DTE20)(2 HEX- BOARDS EACH), UP TO 8 "MASSBUS CONTROLLERS" (RH20)(3 HEX-BOARDS EACH) AND 1 "I/O BUS INTERFACE" (3 HEX-BOARDS).

2. FUNCTION OF THE KL10 CONTROLLERS

2.1 10/11 INTERFACE (DTE20)

10/11 INTERFACE IS A LINK BETWEEN THE KL10 PROCESSOR AND A PDP-11 SYSTEM.

2.2 MASSBUS CONTROLLER (RH20)

EACH MASSBUS CONTROLLER CAN CONTROL, IN SERIAL, UP TO EIGHT DEVICES (FIXED-HEAD DISKS, MOVING HEAD DISKS, DRUMS, AND MAGTAPE DRIVES) PROVIDED THAT EACH DEVICE MEETS THE "MASSBUS INTERFACE STANDARD".

2.3 I/O BUS INTERFACE (DIA20)

I/O BUS INTERFACE IS USED BY THE KL10 PROCESSOR TO COMMUNICATE WITH ALL KA10/KI10 PERIPHERAL CONTROLLERS (TM10, RP10, ETC.) AND DEVICES (TU40, ETC.). KA10/KI10 I/O CONTROL AND VOLTAGE LEVEL CONVERSIONS ARE ITS MAIN FUNCTION.

3. KL10 CONTROLLER'S PHYSICAL NUMBER AND DEVICE CODE

3.1 PHYSICAL NUMBER

THE BACK PANEL WILL BE HARD-WIRED IN SUCH A WAY THAT EACH CONTROLLER'S OCTALPHYSICAL NUMBER (0 THROUGH 7 FOR THE MASSBUS CONTROLLERS, 10 THROUGH 13 FOR THE 10/11 INTERFACES AND 14 FOR THE I/O BUS INTERFACE IS MODULE SLOT DEPENDENT. THE FOUR 10/11 INTERFACES ARE INTERCHANGEABLE AND THE 8 MASSBUS CONTROLLERS ARE INTERCHANGEABLE.

3.2 DEVICE CODE

EACH KL10 CONTROLLER (EXCEPT THE I/O BUS INTERFACE) WILL BE ASSIGNED A DEVICE CODE EXACTLY THE SAME WAY ALL KA10/KI10 CONTROLLER'S DEVICE CODES WERE ASSIGNED. THE DEVICE CODES WILL BE HARD-WIRED IN ON THE BACK PANEL. SEE CHAPTER 2.12, DEVICE CODE AND OPCODE ASSIGNMENT.

4. EBUS SIGNALS

FIG. 1 SHOWS THE EBUS SIGNAL DIAGRAM. A TOTAL OF 60 SIGNALS ARE USED.

5. EBUS OPERATIONS

ALL SIGNAL DESKEWING WILL BE DONE IN THE EBOX TO MINIMIZE LOGIC GATES.

5.1 OUTPUT (DATAO, CONO) OPERATION - SEE FIG. 2 FOR TIMING INFORMATIONS

THE EBOX PLACES A DEVICE CODE ON "CONTROLLER SELECT", OUTPUT DATA ON "DATA", ENCODED DATAO OR CONO ON "FUNCTION", WAITS AT LEAST 200 NS AND ASSERTS "DEMAND".

EACH 10/11 INTERFACE AND MASSBUS CONTROLLER, ON DETECTING THE LEADING EDGE OF "DEMAND", COMPARES "CONTROLLER SELECT" WITH ITS HARD-WIRED DEVICE CODE. IF A TRUE COMPARISON RESULTS, THE CONTROLLER ASSERTS "ACKNOWLEDGE" WITHIN 150 NS. THE SELECTED CONTROLLER, AT THE SAME TIME, DECODES "FUNCTION" TO BE DATAO LINES INTO THE APPROPRIATED REGISTER.

SINCE THE KA10 KI10 ADAPTER DOES NOT HAVE A DEVICE CODE OF ITS OWN AND DOES NOT KNOW THE DEVICE CODES OF THE KA10/KI10 CONTROLLERS (RP10, ETC.) ATTACHED TO IT, IT DOES NOT PERFORM THE DEVICE CODE COMPARISON. THE I/O BUS INTERFACE WILL:

5.1.1 SIMULATE THE KA10/KI10 I/O BUS TIMING AND CONTROL SIGNALS IF AFTER 250 NS AFTER DETECTING THE LEADING EDGE OF DEMAND, NO "ACKNOWLEDGE" SIGNAL IS GENERATED BY ANY 10/11 INTERFACE OR MASSBUS CONTROLLER. "TRANSFER" WILL BE GENERATED BY THE I/O BUS INTERFACE WHEN THE EBUS COMMAND IS DONE. NO "ACKNOWLEDGE" SIGNAL IS GENERATED.

5.1.2 REMAIN IDLE IF AN "ACKNOWLEDGE" IS DETECTED WITH 250 NS AFTER THE LEADING EDGE OF "DEMAND" IS DETECTED IN THE I/O BUS INTERFACE.

THE EBOX WILL RESET "DEMAND" WHEN ONE OF THE FOLLOWING CONDITIONS OCCURS:

5.1.3 A "TRANSFER" SIGNAL IS RECEIVED WITHIN 6 USEC AFTER "DEMAND" IS ASSERTED.

THIS IS A NORMAL CONDITION.

5.1.4 NO "TRANSFER" SIGNAL IS RECEIVED WITHIN 6 USEC AFTER "DEMAND" IS ASSERTED. THIS IMPLIES THAT THE DEVICE CODE IS NOT RECOGNIZED BY A KL10 CONTROLLER AND NO

I/O BUS INTERFACE EXISTS. THIS IS AN ERROR CONDITION.

THE EBOX SHALL NOT CHANGE THE "CONTROLLER SELECT", "DATA" AND "FUNCTION" LINES FOR AT LEAST 150 NS AFTER RESETTING "DEMAND".

THE TRAILING EDGE OF "DEMAND" WILL CAUSE "TRANSFER" (IF ANY) AND "ACKNOWLEDGE" (IF ANY) TO RESET.

5.2 INPUT (DATAI, CONI) OPERATION - SEE FIG. 3 FOR TIMING INFORMATION.

THE EBOX PLACES A DEVICE CODE ON "CONTROLLER SELECT", ENCODED DATAI OR CONI ON "FUNCTION", WAITS AT LEAST 200 NS AND ASSERTS "DEMAND".

EACH 10/11 INTERFACE AND MASSBUS CONTROLLER PERFORMS THE DEVICE CODE COMPARISON, "FUNCTION DECODING AND ASSERTS "ACKNOWLEDGE" AS IN 5.1. THE SELECTED CONTROLLER WILL ASSERT "TRANSFER" AT THE SAME TIME IT PUTS THE INPUT DATA ON "DATA".

THE KA10/KI10 ADAPTER DOES NOT DO THE DEVICE CODE COMPARISON BUT DOES DECODE THE "FUNCTION" TO BE DATAI OR CONI. AFTER 250 NS, IF NO "ACKNOWLEDGE" SIGNAL IS DETECTED, THE I/O BUS INTERFACE WILL SIMULATE THE KA10/KI10 BUS OPERATION, PUTS DATA (SUPPLIED BY THE SELECTED KA10/KI10 CONTROLLER) ON "DATA" LINES AND ASSERTS "TRANSFER".

THE EBOX STROBES "DATA" AND RESETS "DEMAND", 250 NS (NEEDED TO DESKEW THE "DATA" LINES) AFTER THE LEADING EDGE OF "TRANSFER" IS DETECTED PROVIDED THAT "TRANSFER" IS DETECTED WITHIN 6 USEC AFTER "DEMAND" IS SENT. THE EBOX WILL RESET "DEMAND" IF NO "TRANSFER" IS DETECTED WITHIN 6 USEC AFTER ASSERTING "DEMAND". THE EBOX SHALL NOT CHANGE THE "CONTROLLER SELECT" AND "FUNCTION" LINES FOR AT LEAST 150 NS AFTER RESETTING "DEMAND". THE ABSENCE OF "DEMAND" CAUSES "ACKNOWLEDGE" (IF ANY), "TRANSFER" (IF ANY) AND "DATA" TO RESET.

5.3 INTERRUPT OPERATION

PART OF THE STATUS INFORMATION SENT FROM THE EBOX TO A CONTROLLER IS A 3-BIT INTERRUPT CHANNEL NUMBER. WHEN THE CONDITIONS FOR INITIATING AN INTERRUPT REQUEST ARE MET IN A CONTROLLER, THE CONTROLLER WILL APPLY A SIGNAL TO ONE OF THE SEVEN "PRIORITY INTERRUPT" LINES. THE EBOX DETECTS THE PRIORITY INTERRUPT AND RESOLVES THE INTERRUPT PRIORITY. WHEN THE EBOX IS READY TO SERVE ONE OF THE SEVEN INTERRUPT REQUESTS, IT PUTS THE INTERRUPT CHANNEL NUMBER, IN BINARY, ON THE LOW ORDER THREE BITS OF "CONTROLLER SELECT" (E.G. CS04, CS05, CS06 = 110 FOR PRIORITY CHANNEL 6) ENCODED "PI SERVED" ON THE "FUNCTION" LINES, WAIT 200 USEC, AND ASSERT

"DEMAND".

EVERY CONTROLLER (INCLUDING THE I/O BUS INTERFACE), UPON DETECTING THE LEADING EDGE OF "DEMAND", DECODES "FUNCTION" TO BE "PI SERVED", COMPARES THE LOW ORDER THREE BITS OF "CONTROLLER SELECT" WITH THE CHANNEL ON WHICH IT IS INTERRUPTING. IF A TRUE COMPARISON RESULTS (ONE OR MORE CONTROLLERS MAY HAVE TRUE COMPARISON), THE CONTROLLER WILL PUT A "1" BIT ON THE "DATA" LINE ACCORDING TO ITS PHYSICAL CONTROLLER #. CONTROLLER 0 AND CONTROLLER 15 (PHYSICAL #0 AND #15) WILL PUT A "1" BIT ON BITS 0 AND 15 OF THE "DATA" LINES, ETC. NO "ACKNOWLEDGE" OF "TRANSFER" IS ASSERTED BY ANY CONTROLLER.

THE EBOX, AFTER ASSERTING "DEMAND", WAITS AT LEAST 400 NS, STROBES THE "DATA" LINES AND RESETS "DEMAND". THE "DATA" WILL REMAIN VALID UNTIL "DEMAND" RESETS. THE EBOX SHALL NOT CHANGE THE "CONTROLLER SELECT" AND "FUNCTION" LINES FOR 150 NS AFTER RESETTING "DEMAND".

THE EBOX, AFTER DETERMINING WHICH PHYSICAL CONTROLLER'S INTERRUPT REQUEST IT WANTS TO SERVE, PUTS THE INTERRUPT CHANNEL NUMBER, IN BINARY, ON THE LOW ORDER THREE BITS OF "CONTROLLER SELECT", THE CONTROLLER'S PHYSICAL NUMBER, IN BINARY, ON THE NEXT FOUR BITS OF "CONTROLLER SELECT" (CS00, CS01, CS02, CS03 = 1010 FOR PHYSICAL CONTROLLER 10 ETC.) ENCODED "PI ADDRESS IN" ON "FUNCTION" LINES, WAITS 200 NS, AND ASSERTS "DEMAND".

EVERY CONTROLLER (INCLUDING THE KI10/KA10 ADAPTER) DECODES "FUNCTION" TO BE "PI ADDRESS IN", COMPARES THE INTERRUPT CHANNEL NUMBER AND PHYSICAL NUMBER WITH ITS OWN. IF TRUE COMPARISON RESULTS (ONLY ONE CONTROLLER WILL HAVE TRUE COMPARISON), THE CONTROLLER ASSERTS "ACKNOWLEDGE" WITHIN 150 NS.

IF THE CONTROLLER IS A 10/11 INTERFACE OR A MASSBUS CONTROLLER IT PLACES THE INTERRUPT FUNCTION, THE INDEX (IF ANY) AND THE INTERRUPT ADDRESS ON "DATA" AND ASSERTS "TRANSFER" (SEE FIG. 5).

IF THE CONTROLLER IS A I/O BUS INTERFACE IT SIMULATES THE KA10/ KI10 I/O BUS INTERRUPT SEQUENCE AND PLACES THE INTERRUPT FUNCTION, THE INDEX (IF ANY), AND THE INTERRUPT ADDRESS (IF ANY) ON "DATA" AND ASSERTS "TRANSFER" AT THE END OF THE INTERRUPT SEQUENCE (SEE FIG. 5).

THE EBOX STROBES "DATA" AND RESETS "DEMAND":

5.3.1 6 USEC AFTER "DEMAND" IS ASSERTED IF NO "TRANSFER" IS DETECTED. THE EBOX SHOULD NOTE THAT THIS IS AN ERROR CONDITION.

5.3.2 250 NS AFTER "TRANSFER" IS DETECTED PROVIDED

"TRANSFER" IS DETECTED WITHIN 6 USEC AFTER "DEMAND" IS ASSERTED. THIS IS THE NORMAL CONDITION.

5.4 READ IN OPERATION

THIS OPERATION IS NOT PERFORMED BY EITHER THE 10/11 INTERFACES OR THE MASSBUS CONTROLLER. THE I/O BUS INTERFACE WILL SIMULATE THE KA10/KI10 I/O BUS READ IN OPERATION WHEN "READ IN" IS DECODED. THE I/O BUS INTERFACE WILL ASSERT "TRANSFER" WHEN THE OPERATION IS DONE.

5.5 "READ IN DATA" AND "DRUM SPLIT"

SIGNALS "READ IN DATA" AND "DRUM SPLIT" ARE PASSED FROM THE KA10/KI10 CONTROLLER THROUGH THE I/O BUS INTERFACE TO THE EBOX AND SERVE THE SAME FUNCTIONS AS IN THE KA10/KI10 SYSTEMS. NEITHER THE 10/11 INTERFACE NOR THE MASSBUS CONTROLLER USE THESE SIGNALS.

5.6 SYN CLOCK THE "SYN CLOCK" IS AN 8 MHZ TIMING TRAIN USED BY SOME CONTROLLER TO PERFORM LOGICAL FUNCTIONS AND IS ALSO USED BY THE MASSBUS CONTROLLER TO SYNCHRONIZE DATA TRANSFER WITH THE MBOX.

5.7 RESET

THIS SIGNAL IS USED BY EACH CONTROLLER TO CLEAR ITS CONTROL LOGIC TO THE INITIAL STATE.

6.0 ELECTRICAL SPECIFICATION

TRANSCEIVER 8838 (DEC/911117) IS USED TO DRIVE AND RECEIVE THE BUS. REFER TO MEMO "WIRING RULES FOR M&E BUS AND CLOCK DISTRIBUTION" (SULTAN ZIA, FEB. 7, 1973).

1080,2040,2060 ENGINEERING FUNCTIONAL SPEC - CHAP

TO: KL10 LIST, J. PARSLOW (200 FILE)

TITLE: EBOX/MBOX INTERFACE

STATUS: THIS CHAPTER CONTAINS THE CHANGES IN SIGNALS REQUIRED BY ADDING PARITY. THE PARITY IS NOT IN THE BREADBOARD MACHINE AND WILL NOT BE ECOED TO IT.

FILE: [EFS]CH5S04.SPC

PDM #: 200-200-014-00

DATE: 27 MARCH 74

SUPERSEDED MEMOS: EBOX/MBOX INTERFACE, P. GUGLIELMI, 3 OCT 73

SUPERSEDED SPECS:

ENGINEER: P. GUGLIELMI

APPROVED:

EDITOR: T. HASTINGS

TYPIST: M. PROUTY

REVIEWED:

ABSTRACT

THIS CHAPTER DESCRIBES ALL OF THE SIGNALS WHICH PASS BETWEEN THE EBOX AND THE MBOX. A SHORT DESCRIPTION OF EACH SIGNAL IS GIVEN ALONG WITH TIMING AND LOADING INFORMATION.

REVISION HISTORY

REV	DESCRIPTION	CHG NO	ORIG	DATE	APPD BY	DATE

1. INTRODUCTION

THE FOLLOWING MEMORANDUM CONTAINS A LIST OF ALL THE CURRENTLY DEFINED SIGNALS WHICH GO BETWEEN THE EBOX AND MBOX. A SHORT WRITEUP ACCOMPANIES EACH SIGNAL ALONG WITH TIMING AND LOADING INFORMATION.

THIS LIST OF SIGNALS IS NOT COMPLETE BECAUSE IT DOES NOT DESCRIBE THE HANDLING OF PAGE FAILS NXM, AND PARITY ERRORS. THIS INFORMATION WILL BE ADDED WHEN THE LOGIC IS DESIGNED.

2. TIMING OF EBOX REQUESTS TO MBOX

EBOX REQUESTS TO THE MBOX ARE INITIATED BY THE EBOX ASSERTING "EBOX REQ IN" DURING THE CLOCK PERIOD BEFORE THE REQUEST IS TO BE MADE. THIS CAUSES THE "EBOX REQUEST FLIP-FLOP" IN THE MBOX TO BECOME TRUE ON THE NEXT CLOCK TICK (SEE FIGURE 1A). ON THE TICK FOLLOWING THE SETTING OF THE "EBOX REQ FLIP-FLOP" THE VMA AND ALL CONTROL SIGNALS GOING FROM EBOX TO MBOX MUST BE VALID. (EXACT TIMING IS SPECIFIED IN THE TABLES WHICH SHOW SIGNAL LOADING.) THESE SIGNALS MUST REMAIN TRUE UNTIL THE CURRENT MBOX REQUEST HAS EITHER BEEN PROCESSED TO COMPLETION OR ABORTED.

DURING THE SECOND CLOCK PERIOD, DURING WHICH THE VMA IS VALID, THE CURRENT EBOX REQUEST FOR AN MBOX CYCLE CAN BE ABORTED IF THE EBOX SENDS "EBOX AC REF" TO THE MBOX (SEE FIGURE 1C). IF THIS SIGNAL IS SENT THEN THE EBOX SHOULD REMOVE "EBOX REQ IN" AT THE SAME TIME SO THAT THE MBOX WILL NOT FALSELY START A CYCLE ON A SUBSEQUENT CLOCK TICK.

IF THE REQUEST IS FOR A WRITE CYCLE OR AN SBUS DIAGNOSTIC CYCLE, THEN THE "AR" REGISTER MUST BE LOADED WITH THE WRITE DATA NO LATER THAN THREE CLOCK TICKS AFTER "EBOX REQ" BECAME TRUE.

WHEN THE MBOX STARTS TO PROCESS THE CURRENT EBOX REQUEST, IT SENDS "MBOX ACK" TO THE EBOX. THIS SIGNAL TELLS THE EBOX TO TAKE AWAY ITS REQUEST (SEE FIGURE 1A). THIS CAN OCCUR ON THE SAME CLOCK TICK ON WHICH THE VMA IS LOADED IF THE MBOX HAS NO HIGHER PRIORITY REQUESTS TO PROCESS WHEN THE EBOX MAKES ITS REQUEST. IF "EBOX AC REF" IS SENT TO THE MBOX BEFORE THE END OF THE SECOND CLOCK PERIOD THEN THE CURRENT CYCLE WILL BE ABORTED WITH NO FURTHER RESPONSES FROM THE MBOX. IF THE MBOX IS BUSY WHEN THE VMA IS LOADED THEN A NUMBER OF CLOCK TICKS MAY PASS BEFORE "MBOX ACK" IS SENT TO THE EBOX. NOTE THAT "MBOX ACK" WILL NOT BE SENT IF THE MBOX IS BUSY WHEN "EBOX AC REF" IS ASSERTED (SEE FIGURE 1A).

AFTER "MBOX ACK" IS SENT ONE OR MORE CLOCK TICKS MAY ELAPSE BEFORE THE MBOX COMPLETES PROCESSING OF THE REQUEST. THE MBOX NOTIFIES THE EBOX THAT IT HAS FINISHED PROCESSING THE REQUEST

BY ASSERTING "MBOX RESP". THIS SIGNAL STAS ASSERTED UNTIL THE MBOX SEES "EBOX SYNC" FROM THE EBOX, IF ANY DATA WAS TO BE SENT TO THE AR, ARX, OR IR BY THE REQUESTED CYCLE THEN IT WILL BE TRUE ON THE CACHE DATA LINES WHILE "MBOX RESP" IS ASSERTED. IF THE DATA WAS TO GO TO THE EBUS REGISTER THEN THE DATA WILL BE LOADED INTO THE REGISTER BY THE SAME CLOCK TICK THAT CAUSES "MBOX RESP" TO BE ASSERTED. THE "AND" OR "MBOX RESP" AND "EBOX SYNC" WILL CAUSE "MBOX RESP" TO BE CLEARED. THE EBOX CYCLE TYPE SIGNALS MAY ALSO BE CHANGED AT THIS TIME.

NOTE THAT CERTAIN OF THE CYCLE TYPE SIGNALS MUST BE VALID AT OR BEFORE THE SAME CLOCK TICK THAT THE "EBOX REQ FLIP-FLOP" BECOMES TURE AND MUST STAY VALID UNTIL THE MBOX PROCESSING IS COMPLETE. THIS MEANS THAT REQUESTS THAT ARE GOING TO CHAGE THESE SIGNALS CANNOT FOLLOW IMMEDIATELY BEHIND EACH OTHER WITHOUT TIMING PROBLEMS. THE SIGNALS IN QUESTION ARE "EBOX ERA", AND "EBOX CCA".

3. COMMON EBOX/MBOX SIGNALS

3.1 EBOX REQ IN

SIGNAL WHICH THE EBOX RAISES TO REQUEST AN MBOX CYCLE. THIS IS THE ONLY SIGNAL WHICH WILL CASE THE MBOX TO DO ANYTHING FOR THE EBOX. THIS SIGNAL IS THE INPUT TO A FLIP-FLOP WHICH HOLDS THE EBOX REQUEST. THE VMA AND ALMOST ALL OF THE SIGNALS LISTED BELOW MUST BECOME VALID ONE CLOCK TICK AFTER THE FLIP-FLOP WHICH HOLDS EBOX REQ BECOMES TRUE. "EBOX REQ IN" MUST BECOME FALSE DURING THE CLOCK PERIOD IN WHICH "MBOX ACK" IS TRUE IF THE EBOX DOES NOT HAVE A SECOND MBOX REQUEST CAMPING AT THE GATES. "EBOX REQ IN" SHOULD BE TAKEN AWAY IF THE EBOX DECIDES THAT IT HAS MADE A REQUEST FOR A WORD OF DATA WHICH IS IN ITS ACS.

3.2 EBOX AC REF

SIGNAL WHICH EBOX SENDS TO MBOX TO TELL THE MBOX TO ABORT THE CURRENT CYCLE BECAUSE THE EBOX HAS LOOKED AT THE VMA AND DECIDED THAT THE REQUESTED DATA IS HELD IN ITS ACS. "EBOX AC REF" MUST BE SENT DURING THE "EBOX REQ +2" CLOCK. "EBOX REQ IN" SHOULD BE MADE FALSE WHENEVER "EBOX AC REF" IS ASSERTED.

3.3 MBOX ACK

SIGNAL WHICH MBOX SENDS TO EBOX TO TELL THE EBOX TO TAKE AWAY ITS CURRENT REQUEST BECAUSE THE MBOX HAS STARTED PROCESSING IT. "MBOX ACK" IS ASSERTED FOR ONLY ONE CLOCK TICK. EBOX MUST TAKE AWAY ONLY THE "EBOX REQ IN" SIGNAL AND NOT CHANGE ANY OF THE OTHER EBOX TO MBOX SIGNALS UNTIL "MBOX RESP" IS ASSERTED.

3.4 PAGE FAIL HOLD

SENT FROM MBOX TO EBOX TWO CLOCK TICKS BEFORE "MBOX RESP" TO TELL EBOX THAT THE MBOX DETECTED A PAGE FAILURE ON THE CURRENT EBOX REQUEST.

3.5 MBOX RESP

SENT FROM MBOX TO EBOX TO NOTIFY THE EBOX THAT THE MBOX HAS COMPLETED PROCESSING OF THE CURRENT REQUEST. IF ANY DATA IS TO BE SENT FROM THE MBOX TO THE EBOX IT WILL BE VALID ON THE CACHE DATA LINES WHILE "MBOX RESP" IS TRUE. PAGE FAIL, MAP, AND REGISTER READ DATA WILL BE LOADED INTO THE "EBUS REQ" IN THE MBOX ON THE SAME CLOCK TICK THAT "MBOX BECOMES TRUE.

3.6 EBOX READ EBUX REG

DC LOGIC LEVEL WHICH ENABLES DATA IN THE "EBUS REG" IN THE MBOX ONTO THE EBUS. THE "EBUS REG" WILL HOLD PAGE FAIL, MAP, AND REGISTER READ DATA FROM THE MBOX.

4. EBOX MEMORY REFERENCES

THE EBOX CAN REQUEST FIVE TYPES OF MEMORY REFERENCE CYCLES USING THREE CONTROL LINES. THEY ARE:

1. READ - READ A WORD FROM MEMORY.
   READ CHECK THE PAGING

2. WRITE - WRITE THE WORD IN THE "AR" REGISTER INTO MEMORY.
   WRITE CHECK THE PAGING.

3. READ-WRITE - READ A WORD FROM MEMORY AND READ/WRITE CHECK
   THE PAGING.

4. READ-PAUSE-WRITE - DO THE READ PART OF A READ PAUSE WRITE
   CYCLE INCLUDING READ/WRITE CHECKING THE PAGING.

5. PAUSE-WRITE - WRITE CHECK THE PAGING FOR THE ADDRESS IN THE
   VMA.

THE CYCLE TYPE IS SPECIFIED BY ASSERTING "EBOX READ, EBOX WRITE, OR EBOX PAUSE" TO REQUEST THE APPROPRIATE TYPE OF CYCLE.

NOTE: EBOX WRITE CORRESPONDS TO THE KI10 "PAGE WRITING" SIGNAL.

5. PAGING + CACHE QUALIFIERS

5.1 EBOX SEC

IF FALSE USE KI10 STYLE PAGING. IF TRUE USE SECTION MODE PAGING.

5.2 DIA CACHE LOAD EN

USE THE CACHE FOR THIS REFERENCE. WHEN FALSE A REFERENCE WILL NEVER CAUSE DATA TO GET LOADED INTO THE CACHE FROM CORE. HOWEVER, IF THE DATA REQUESTED ON A READ IS FOND IN THE CACHE IT WILL BE TAKEN FROM THE CACHE ON WRITES. IF THE DESIRED LOCATION IS IN THE CACHE THEN THE DATA WILL BE WRITTEN INTO THE CACHE.

5.3 EBOX PAGED

IF TRUE, MAP THIS REFERENCE VIA THE PAGING. IF FALSE THIS IS AN ABSOLUTE CORE REFERENCE.

5.4 EBOX USER

IF TRUE, THIS IS A USER ADDRESS SPACE REFERENCE. IF FALSE, THIS IS AN EXEC ADDRESS SPACE REFERENCE.

5.5 DIA CACHE LOOK EN

IF TRUE CHECK THE CACHE TO SEE IF IT HAS THE DATA. IF FALSE, DO NOT CHECK THE CACHE. (NOTE+ IF DIA CACHE LOAD EN IS ALSO FALSE ALL REFERENCES GO DIRECTLY TO CORE.)

5.6 EBOXEPT

IF TRUE, THIS IS A REFERENCE TO THE EXEC PROCESS TABLE. THE MBOX WILL REPLACE BITS 13-26 OF THE VMA WITH THE CONTENTS OF THE EXEC BASE REGISTER (EBR).

5.7 EBOX UPT

IF TRUE, THIS IS A REFERENCE TO THE USER PROCESS TABLE. BITS 13-26 OF THE VMA WILL BE REPLACED BY THE CONTENTS OF THE USER BASE REGISTER (UBR).

5.8 PAGE UEBR REF

IF TRUE, THIS IS AN EBOX REFERENCE TO ONE OF THE PROCESS TABLES.

5.9 XCT PROT BYPASS

IF ASSERTED, THEN THE PAGING PROPRIETARY TEST IS BYPASSED.

5.10 PAGE ILL ENTRY

IF ASSERTED, THIS SIGNAL FORCES A PAGE FAIL IN THE MBOX. SET BY THE EBOX IF IT HAS JUST FETCHED AN INSTRUCTION FROM A PROPRIETARY AREA AND THE INSTRUCTION IS NOT A PARTAL INSTRUCTION OR THE EBOX HAS NOT DECIDED THAT THE INSTRUCTION IS A PORTAL INSTRUCTION YET.

5.11 PAGE TEST PRIVATE

THIS IS A NON-INSTRUCTION PUBLIC MODE EBOX REFERENCE.

5.12 PAGE ADDRESS COND

ASSERTED WHEN THE EBOX DETECTS AN ADDRESS BREAK CONDITION.

5.13 EBOX PT WRITE

WRITE PULSE FROM THE EBOX TO THE HARDWARE PAGE TABLE DIRECTORY. THIS SIGNAL IS USED TO WRITE THE PAGE TABLE CLEAR. THE EBOX DOES THIS BY COUNTING AN ADDRESS IN VMA BITS 18-23 AND GENERATING THIS SIGNAL FOR AT LEAST ONE CLOCK TICK FOR EACH VALUE OF ADDRESS. THIS WILL CAUSE THE VALID BIT IN THE PAGE TABLE DIRECTORY TO BE WRITTEN CLEAR. NO DISTINCTION IS MADE BETWEEN EXEC AND USER ENTRIES IN THE HARDWARE PAGE TABLE WHEN CLEARING IT.

NOTE: "-EBOX PAGED" CORRESPONDS TO THE KI10 "VMA EXEC UNPAGED" SIGNAL.

NOTE: "PAGE ILL ENTRY" MUST BE ASSERTED WHEN "PAGE ADDRESS COND" IS ASSERTED.

5.14 EBOX MAP

CAUSES THE MBOX TO MAP THE VIRTUAL ADDRESS IN THE VMA INTO A PHYSICAL ADDRESS. THE MAPPED ADDRESS IS PLACED IN THE VMA ALONG WITH THE MAP BITS SPECIFIED IN THE PDP10 SYSTEM REFERENCE MANUAL. TO REQUEST A MAP CYCLE, THE EBOX MUST ASSERT EBOX MAP AND EBOX READ REG. THE FOLLOWING QUALIFIERS ARE POSSIBLE FOR A MAP CYCLE:

QUALIFYING SIGNALS	COMMENTS
EBOX PAGED	
EBOX SEC	
EBOX USER	
XCT PROT BYPASS	
PAGE TEST PRIVATE	
PAGE ADDRESS CONT.	

EBOX WRITE	(TO CHECK PAGING)
EBOX AC REF	(IF ASSERTED MBOX WILL ABORT MAP REF)
EBOX READ	(MUST BE FALSE OR WON'T PAGE FAIL ON A WRITE PROTECTION VIOLATION)

MUST ALL BE CORRECTLY ASSERTED. MAP DATA IS READ BACK IN THE SAME WAY AS FOR THE "EBOX READ REG" OPERATION.

MAP WILL EITHER PERFORM THE READ REGISTER FUNCTION SUCCESSFULLY INCLUDING DOING A PAGE REFILL IF NECESSARY, OR IT WILL PAGE FAIL IF AN EBOX WRITE WOULD HAVE RESULTED IN A PAGE FAIL.

IN ANY CASSE, THE MBOX WILL LOAD THE EBUS REGISTER WITH THE DATA SHOWN BELOW:

0	EBOX USER
1-8	PAGE FAIL CODE OR MAP DATA
9-13	UNUSED - RESERVED FOR FUTURE HARDWARE
14-26	PHYSICAL ADDRESS REFERENCED

IF BIT 1 = 0 THEN MAP DID NOT PAGE FAIL.

BIT 2 = PT ACCESS

BIT 3 = PT WRITABLE

BIT 4 = PT SOFTWARE

BIT 5 = WRITE REFERENCE

BIT 6 = PT PUBLIC

BIT 7 = PT CACHE

BIT 8 = PT MATCH

IF BIT 1 = 1 THEN A PAGE FAIL CODE 2X WILL BE RETURNED IN BITS 2 THRU 5. BITS 6 + 7 + 8 CONTAIN GARBAGE.

PF CODES

20	NOT USED ON KL20 (KI10 SMALL USER)
21	PROPRIETARY VIOLATION
22	PAGE REFILL FAILURE (PAGING HARDWARE FAILURE

23 ADDRESS FAILURE

24 SECTION TABLE REFILL ACCESS FAILURE

25 PAGE TABLE ENTRY BAD (NEW PARITY)

6. REGISTER REFERENCES

THIS GROUP OF SIGNALS IS USED TO READ AND WRITE CERTAIN MBOX REGISTERS. SIGNALS METIONED UNDER "EBOX MEMORY REFERENCES" AND "PAGING + CACHE QUALIFIER" SECTIONS MUST NOT BE ASSERTED WHEN READING OR WRITING REGISTERS.

6.1 EBOX LOAD REG

WHEN TRUE CAUSES DATA IN VMA 14-26 TO BE WRITTEN INTO ONE OF THE REGISTERS SPECIFIED BELOW.

6.2 EBOX READ REG

WHEN TRUE CAUSES ONE OF THE REGISTERS SPECIFIED BELOW TO BE READ INTO THE EBUS REGISTER IN THE MBOX. THE EBOX CAN READ THIS REGISTER OVER THE EBUX BY ASSERTING "EBOX READ EBUS REG" AFTER THE "EBOX READ REGISTER" OPERATION HAS BEEN COMPLETED BY THE MBOX.

6.3 EBOX UBR

REFERENCE THE USER BASE REGISTER.

6.4 EBOX EBR

REFERENCE THE EXEC BASE REGISTER.

6.5 EBOX ERA

REFERENCE THE ERROR REGISTER (ERA). THIS REGISTER CAN ONLY BE READ. IT IS LOADED WHEN A MEMORY ADDRESS PARITY ERROR IS DETECTED BY A MEMORY, OR WHEN A NXM OCCURS, OR WHEN A DATA PARITY ERROR IS DETECTED. BITS 34 AND 35 ARE THE SAME AS SBUS 34 + 35 ON THE SBUS.

6.6 EBOX CCA

REFERENCE THE CACHE CLEARER. ON CACHE CLEARER LOADS VMA 14-26 HOLD THE PHYSICAL PAGE NUMBER OF THE PAGE FOR WHICH THE CACHE SWEEP IS TO BE DONE. THE CACHE CLEARER STARTS AS SOON AS IT IS LOADED.

6.7 EBOX CCA INVAL CSH

TELLS THE CACHE CLEARER TO INVALIDATE ENTRIES IN THE CACHE FOR

THE SPECIFIED PAGE.

6.8 EBOX CCA VAL CORE

TELLS THE CACHE CLEARER TO VALIDATE CORE FROM THE CACHE FOR THE SPECIFIED PAGE.

6.9 EBOX CCA ONE PAGE

IF TRUE, DO THE ABOVE CCA OPERATIONS ONLY FOR THE SPECIFIED PAGE. IF FALSE, DO THE ABOVE OPERATIONS FOR ALL PAGES IN THE CACHE.

6.10 EBOX WR REFILL RAM

IF TRUE DURING AN EBOX READ REGISTER CYCLE ONE 3-BIT WORD OF DATA FROM VMA 18-20 WILL BE WRITTEN INTO THE CACHE REFILL ALGORITHM RAM IN THE LOCATION SPECIFIED BY BITS 27-33 OF THE VMA (SEE MEMO ON CACHE SWEEP INSTRUCTION).

6.11 CCA REQ L

TRUE ALL THE TIME THE CACHE CLEARER IS RUNNING. GOES FALSE WHEN CACHE CLEARER HAS COMPLETED A CACHE SWEEP.

7. SBUS DIAGNOSTIC CYCLES

7.1 EBOX/SBUS DIAG

THIS FUNCTION ALLOWS THE EBOX TO SEND AND RECEIVE DIAGNOSTIC INFORMATION FROM THE MEMORIES ON THE SBUS. TO USE THIS FUNCTION THE EBOX FIRST LOADS THE DIAGNOSTIC INFORMATION IT WATS TO SEND TO THE MEMORIES INTO ITS AR REGISTER.

THEN IT REQUESTS AN EBOX SBUS DIAGNOSTIC CYCLE BY ASSERTING "EBOX SBUS DIAG" AND "EBOX REQ" AT THE CORRECT TIMES. APPROXIMATELY 720 NSEC AFTER "EBOX SBUS DIAG" IS ASSERTED, ASSUMING THE MBOX IS NOT BUSY, A WORD OF DIAGNOSTIC INFORMATION WILL BE SENT TO THE EBOX FROM THE MBOX OVER THE CACHE DATA LINES. SIMULTANEOUS WITH THE DATA ARRIVAL "MBOX RESP" WILL BE ASSERTED FOR ONE CLOCK TICK. THE EBOX MUST TAKE THE DATA DURING THIS CLOCK TICK.

8. DATA AND ADDRESS SIGNALS

8.1 AR 00 - 35

DATA TO MBOX FROM EBOX (36 BITS)

8.2 CACHE DATA 00-35 B

MBOX DATA TO AR, ARX (36 BITS)

8,3 CACHE DATA 00-35 C

MBOX DATA TO IR (36 BITS)

8,4 VMA 27-35 G

ADDRESS LINES TO MBOX (9 BITS)

8,5 VMA 13-35 A

REGISTER LOAD DATA OR ADDRESS LINES TO MBOX (23 BITS)

NOTE: 1. THE VMA ADDRESS LINES MUST BECOME TRUE ON THE CLOCK TICK AFTER "EBOX REQ" IS ASSERTED.

2. DATA BEING SENT TO THE MBOX ON CORE WRITE CYCLES DOES NOT HAVE TO BE VALID IN THE AR UNTIL TWO CLOCK TICKS AFTER THE VMA BECOMES VALID.

9. ERROR SIGNALS [BREADBOARD ONLY]

IN THE KL10 THE MBOX DETECTS ERRORS ASSOCIATED WITH MEMORY TRANSFERS AND NOTIFIES THE EBOX OF THE OCCURRENCE OF ANY OF THREE TYPES OF ERRORS:

1. NON-EXISTENT-MEMORY (NXM) TIMEOUTS
2. MEMORY DATA PARITY ERROR
3. SBUS ERRORS

9,1 NON-EXISTENT MEMORY ERRORS (NXM)

NON-EXISTENT-MEMORY ERRORS OCCUR WHEN ONE OF THE MEMORIES WHICH HAS BEEN ADDRESSED DOES NOT RESPOND WITH AN SBUS ACKNOWLEDGE PULSE TO INDICATE ITS PRESENCE WITHIN 32 MICROSECONDS AFTER THE MBOX INITIATED A REQUEST ON THE SBUS. THE MBOX DISTINGUISHES BETWEEN CHANNEL AND EBOX INITIATED MEMORY CYCLES AND SENDS SEPARATE ERROR NOTIFICATIONS TO EACH ON NSMS. EBOX INITIATED MEMORY REFERENCES INCLUDE REFERENCES FOR PAGE REFILLS, CACHE WRITEBACKS, CACHE SWEEPS, EBOX READS, EBOX WRITES, AND EBOX READ-PAUSE-WRITE REFERENCES.

WHEN AN NXM TIMEOUT OCCURS THE MBOX HOLDS THE ERROR ADDRESS REGISTER (ERA) WHICH HOLDS THE ADDRESS OF THE FIRST REQUESTED WORD FOR WHICH THE REFERENCE WAS MADE. THE ERA IS NORMALLY LOADED AT THE START OF A MEMORY REFERECE. THIS IS NOT NECESSARILY THE ADDRESS OF THE WORD WHICH GOT THE NXM TIMEOUT BECAUSE ANY OF UP TO FOUR WORDS WITHIN A QUAD WORD MAY HAVE CAUSED THE ERROR. THE ERA MAY BE READ BY THE EBOX WITH A REGISTER READ OPERATION.

ON READS THE MBOX SUPPLIES FULL WORDS OF ZEROS FOR THE REQUESTED WORD WHICH GOT THE NXM ERROR AND ALL SUBSEQUENT REQUESTED WORDS WITHIN THAT QUAD WORD. ON WRITES THE MBOX THROWS AWAY THE WORDS WHICH GOT THE NXM ERROR AND ALL SUBSEQUENT REQUESTED WORDS WITHIN THE QUAD WORD.

9.2 MEMORY DATA PARITY ERRORS

THE PARITY OF ALL DATA READ FROM MEMROY IS CHECKED BY THE MBOX. SEPARATE ERROR SIGNALS FOR BOTH THE EBOX AND CHANNELS ARE PROVIDED SO THAT THE MBOX CAN NOTIFY EACH AS APPROPRIATE. THE MBOX INITIATES MEMORY READS FOR THE EBOX IN ONLY THREE CASES:

1. EBOX READ REQUESTS
2. EBOX READ-PAUSE-WRITE REQUESTS
3. PAGE REFILL CYCLES

IN ALL CASES WHEN A DATA PARITY ERROR IS DETECTED THE MBOX SETS "MBOX PAR ERR" TO NOTIFY THE EBOX OF THE OCCURRENCE OF THE ERROR AND HOLDS THE ERA WHICH HAS THE ADDRESS OF THE FIRST REQUESTED WORD OF THE QUAD WORD WITHIN WHICH THE ERROR OCCURRED. NO FURTHER ACTION IS TAKEN BY THE MBOX EXCEPT TO CLEAR ITS FLAG WHEN IT SEES "DIA PAR ERR" FROM THE EBOX. THE BAD DATA IS PASSED TO THE EBOX, STORED IN THE CACHE, OR STORED IN THE HARDWARE PAGE TABLE AS APPROPRIATE.

9.3 SBUS ERRORS

SBUS ERRORS ARE ERRORS THAT ARE DETECTED BY THE MEMORIES. WHEN A MEMORY DETECTS AN ERROR IT SENDS "SBUS ERROR" TO THE MBOX. THE MBOX SETS "MBOX SBUS ERR" TO NOTIFY THE EBOX THAT IT HAS SEEN "SBUS ERROR" AND IT HOLDS THE ADDRESS THAT IS CURRENTLY IN THE ERA. THIS ADDRESS CANNOT BE GUARANTEED TO BE A VALID ADDRESS IN ALL CASES BECAUSE OF LATENCY PROBLEMS ASSOCIATED WITH "SBUS ERR" COMING FROM THE MEMORIES. THIS SITUATION WILL BE FIXED ON THE KL10 PROTOTYPE. HOWEVER, EVEN THEN THE MEMORIES WILL HAVE TO INTERROGATE WITH SBUS DIAGNOSTIC FUNCTIONS TO DETERMINE WHAT TYPE OF ERROR WAS DETECTED BY THE MEMORY. THE ERA IS INTENDED TO ONLY BE VALID FOR SBUS ADDRESS PARITY ERRORS.

FOR ALL OF THE ERRORS DISCUSSED ABOVE THE CONTENTS OF THE ERA WILL BE HELD UNTIL ALL THE EBOX ERROR FLAGS (SBUS ERROR, PARITY ERROR, NXM) ARE CLEARED. THIS MEANS THAT THE SOFTWARE SHOULD READ THE ERA BEFORE IT CLEARS THESE FLAGS.

TO ALLOW THE EBOX TO WRITE EVEN ADDRESS AND DATA PARITY, TWO SIGNALS WHICH THE EBOX MAY ASSERT BEFORE STARTING A MEMORY REFERENCE ARE PROVIDED. THESE ARE "DIA WR EVEN ADR PAR" AND "DIA WR EVEN DATA PAR". WHEN ONE OR BOTH OF THESE IS ASSERTED AND IT IS AN EBOX INITIATED WRITE CYCLE (SEE DESCRIPTION OF NXM) EVEN ADDRESS OR DATA PARITY WILL BE SENT FROM THE MBOX TO THE MEMORIES.

MBOX	NXM	ERR	L	ASSERTED BY MBOX WHEN NXM TIMEOUT OCCURS IN THE PROCESSOR ON AN EBOX CAUSED MEMORY REFERENCE. AN NXM TIMEOUT OCCURS IF THE MBOX DOES NOT SEE ALL THE "SBUS ACKN" PULSES FROM THE MEMORIES WITHIN 32 SEC.
MBOX	PAR	ERR	L	ASSERTED BY MBOX WHEN A DATA PARITY ERROR IS DETECTED IN THE MBOX ON DATA RECEIVED AS A RESULT OF AN EBOX INITIATED MEMORY REFERENCE.
MBOX	SBUS	ERR	L	ASSERTED BY MBOX ANYTIME SBUS ERROR IS SENT BY ONE OF THE MEMORIES.
DIA	NXM	ERR	L	ASSERTED BY EBOX TO NOTIFY MBOX THAT THE EBOX HAS RECORDED THE FACT THAT THERE WAS A NXM TIMEOUT ON AN EBOX INITIATED CORE REFERENCE.
DIA	PAR	ERR	L	ASSERTED BY EBOX TO NOTIFY MBOX THAT THE EBOX HAS RECORDED THE FACT THAT THERE WAS A DATA PARITY ERROR ON AN EBOX INITIATED CORE REFERENCE.
DIA	SBUS	ERR	L	ASSERTED BY EBOX TO NOTIFY MBOX THAT THE EBOX HAS RECORDED THE FACT THAT AN SBUS ERROR WAS DETECTED ON A CORE REFERENCE. ERROR MAY HAVE BEEN CAUSED BY A CHANNEL OR EBOX INITIATED REFERENCE.
DIA	ANY EBOX ERR FLG		L	ASSERTED BY THE EBOX TO NOTIFY THE MBOX THAT ONE OR MORE OF THE EBOX MEMORY ERROR FLAGS IS SET. THIS SIGNAL IS ASSERTED ON THE SAME TICK THAT THE FIRST EBOX ERROR FLAG IS ASSERTED. AS LONG AS THIS SIGNAL IS ASSERTED THE CONTENTS OF THE ERROR ADDRESS' REGISTER IN THE MBOX IS HELD.
DIA	WR EVEN ADR PAR		L	WHEN TRUE THE MBOX CALCULATES EVEN ADDRESS PARITY ON EBOX INITIATED WRITE CYCLES

AND
ODD PARITY ON ALL OTHER MEMORY CYCLES.
WHEN FALSE MBOX CALCULATES ODD ADDRESS
PARITY ON ALL MEMORY CYCLES

DIA WR EVEN DATA PAR L — WHEN TRUE THE MBOX CALCULATES EVEN DATA PARITY ON EBOX INITIATED WRITE CYCLES AND ODD PARITY ON ALL OTHER MEMORY CYCLES.
WHEN FALSE MBOX CALCULATES ODD DATA PARITY
ON ALL MEMORY CYCLES.

TO REQUEST A MAP CYCLE, THE EBOX MUST ASSERT EBOX MAP AND EBOX READ
REG. THE FOLLOWING QUALIFIERS ARE POSSIBLE FOR A MAP CYCLE:

1080,2040,2060 ENGINEERING FUNCTIONAL SPEC CHAP A.1

TO: KL10 LIST

TITLE: DIFFERENCES BETWEEN BREADBOARD, PROTOTYPE, PRODUCTION MACHINES - REV 1

STATUS: THIS APPENDIX DESCRIBES THE CURRENT PLANNED DIFFERENCES BETWEEN THE BREADBOARD MACHINE AND THE PROTOTYPE MACHINE. IT IS A SUMMARY OF THE MEMO, LOOSE ENDS - PART 1, T. HASTINGS, 18 DEC 73. IT WILL BE REVIEWED 13 MAR 74 AT THE 1:00 P.M. KL10 STEERING COMMITTEE MEETING.

FILE: [EFS]CHAS01.SPC

PDM #: 200-200-007-00

DATE: 5 MAR 74

SUPERSEDED MEMOS: NONE

ENGINEER: A. KOTOK, T. EGGERS, R. REID, V. KU, P. SULLIVAN,B. WALTON, P. GUGLIELMI

APPROVED: T. EGGERS, R. REID, V. KU, P. SULLIVAN, P. GUGLIELMI, A. KOTOK

EDITOR: T. HASTINGS

TYPIST: L. NOWELL

REVIEWED:

ABSTRACT

THE BREADBOARD MACHINE IS BEING CHECKED OUT. THIS APPENDIX DESCRIBES THOSE FEATURES WHICH ARE NOT IN THE BREADBOARD BUT ARE PLANNED FOR THE NEXT LAYOUT WHICH IS THE PROTOTYPE MACHINE. THE PURPOSE OF THIS LIST IS TO MAKE SURE THAT WE HAVE AGREEMENT ON THE SCOPE OF THE PROTOTYPE BEFORE IT STARTS RELAYOUTS.

REVISION HISTORY

REV	DESCRIPTION	CHG NO	ORIG	DATE	APPD BY	DATE

1. BACKGROUND

THE KL ENGINEERING GROUP IS DESIGNING FOUR VERSIONS OF THE MACHINE:

1. 3 BREADBOARD - POWERED ON 26 FEB 74
2. 5 PROTOTYPE - NEXT RELAYOUT
3. 7 PRE-PRODUCTION MACHINES
4. N PRODUCTION MACHINES

ORIGINALLY ONLY THREE VERSIONS WERE PLANNED: PROTOTYPE, PRE-PRODUCTION, AND PRODUCTION. HOWEVER, THE PROTOTYPE WAS RENAMED BREADBOARD WHEN IT WAS DISCOVERED HOW MANY FEATURES AND CAPABILITIES WERE MISSING. THIS APPENDIX AND THE SUPPORTING CHAPTERS OF THE FUNCTIONAL SPECIFICATION WILL HELP PREVENT FURTHER SURPRISES AND RESULTING REVISED SCHEDULES FOR THE PROTOTYPE. IF WE DO A GOOD JOB OF WRITING DOWN OUR GOALS AND SPECIFICATIONS, WE SHOULD BE ABLE TO MINIMIZE THE FUNCTIONAL CHANGES BETWEEN THE PROTOTYPE AND SUBSEQUENT MACHINES. THE ONLY CHANGES WILL COME FROM SERIOUS PROBLEMS FOUND WHILE CHECKING OUT HARDWARE AND SOFTWARE.

2. FUNCTIONS TO BE INCLUDED

THE FOLLOWING LIST OF MAJOR LOOSE ENDS DESCRIBES THE ONES THAT WE ARE COMMITTING TO BE IN THE 1080, 2040, AND 2020. THE PURPOSE OF THIS SHORT LIST OF MAJOR ITEMS IS TO BE USED IN FIRMING UP THE REVISED SCHEDULE FOR DESIGN AND LAYOUT OF THE PROTOTYPE MACHINE. THIS LIST WILL BE GIVEN A WIDE CIRCULATION FOLLOWING REVIEW BY KL10 ENGINEERING SO THAT WE CAN AGREE ONCE AND FOR ALL WHAT MAJOR ITEMS ARE AND ARE NOT GOING TO BE DONE. THIS WILL PERMIT US TO MAKE UP A REALISTIC SCHEDULE WITH MINIMUM CHANCE FOR SURPRISES. MINOR LOOSE ENDS NOT APPEARING ON THIS LIST DO NOT MEAN THAT THEY WILL NOT BE DONE. FOR PURPOSES OF ESTIMATING MAN-POWER, THE SMALL ONES HAVE BEEN GROUPED TOGETHER ON THE FOLLOWING LIST.

THE FOLLOWING TABLE SHOWS FUNCTIONS THAT ARE NOT IN THE BREADBOARD MACHINE BUT ARE PLANNED FOR THE PROTOTYPE.

LOOSE END #	MAJOR LOOSE ENDS
1. A3	MICRO CODE INSTRUCTIONS LEFT TO BE CODED.
2. A8	DK20 (SLOT ALLOCATED)
3. A9,A11	IMPROVED PAGING.

4. A25,A35,C13 POWER FAIL, POWER SUPPLY REVIEW.

5. A30 MEMORY ERROR INFORMATION: [BREADBOARD: INCLUDED BUT SOME PROBLEMS]

ADDRESS PARITY ERROR ON:

INT. MEMORY READS

INT. MEMORY WRITES

EXT. MEMORY READS

EXT. MEMORY WRITES

DATA PARITY ERROR ON:

INT. MEMORY READS

INT. MEMORY WRITES

6. C1 PARITY

EBOX:

1. TRANSFERS FROM MBOX
2. TRANSFERS FROM FAST MEMORY (IE. AC'S)
3. TRANSFERS TO AR, ARX
4. DATAI, BYTE INPUTS FROM DTE20
5. DATAO, BYTE OUTPUT TO DTE20

MBOX:

1. CACHE DATA RAMS
2. CACHE ADDRESS STORAGE RAMS
3. PAGING RAMS
4. CHANNEL AND CBUS DATA

3. INTERNAL DESIGN LOOSE ENDS

1.	C2	CHANNEL DESIGN AND MBOX WORK TOGETHER.
2.	C3	MICRO CODE SIMULATOR.
3.	C4	MICRO CODE SIMULATOR - ADD INSTRUCTION TIMING. MARKETING SUPPLYING MANPOWER.
4.	C5	GATE LEVEL SIMULATOR
5.	C6	MASSBUS ELECTRICAL MARGINS.
6.	C10	BOARDS OVERCROWED; NO ROOM FOR ECOS.
7.	C11	STUBBING PROBLEMS.
8.	C12	EBUS OVERLOADED.
9.	A38	SIZE OF MICRO-CODE STORE 1024 OR 1280?
10.	XX	MISCELLANEOUS SMALL LOOSE ENDS. SEE LOOSE ENDS - PART 1, T. HASTINGS, 18 DEC 73.
11.	XX	2060 NO SPARE SLOTS LEFT.
12.	XX	PI 0 FOR DTE20 (RE-OPEN QUESTION NOW THAT RESCHEDULE HAS OCCURRED).

4. FUNCTIONS TO BE INCLUDED IF ROOM IN MACHINE AND/OR MICRO-CODE AND AFTER REVIEWED AND APPROVED

1.	A4	MICRO-CODE FOR SUBROUTINE CALLING INSTRUCTIONS.

5. FUNCTIONS NEVER TO BE INCLUDED

THE FOLLOWING TABLE SHOWS FUNCTIONS OR TASKS WHICH ARE NOT IN THE BREADBOARD MACHINE AND ARE NOT PLANNED FOR ANY SUBSEQUENT VERSION OF THE MACHINE. IT IS JUST AS IMPORTANT TO APPROVE THINGS THAT ARE NOT GOING TO BE DONE, AS IT IS TO APPROVE THINGS THAT WILL BE DONE.

[END OF CHAS01.SPC]

```
 **END** USER MCCARTHY  JOB  CH0S01 SEQ, 1949 DATE  3-DEC-76 15:39:21
MONITOR 555 LCEG SYSTEM, TOPS-20 MONITOR 1B **END**
```


```
 **END** USER MCCARTHY  JOB  CH0S01 SEQ, 1949 DATE  3-DEC-76 15:39:21
MONITOR 555 LCEG SYSTEM, TOPS-20 MONITOR 1B **END**
```


```
                    * * * L P T S P L   R U N   L O G * * *

15:12:54 LPDAT  [LPTLSJ  LPTSPL VERSION 102(2226) RUNNING ON PLPT0,  3-DEC-76
15:12:54 LPDAT  [LPTSJS  STARTING JOB CH0S01, SEQ #1949, REQUEST CREATED AT 1
15:15:43 LPMSG  [LPTSTF  STARTING FILE DSK:<EFS>CH0S01.MEM]
15:15:48 LPMSG  [LPTFPF  FINISHED PRINTING FILE DSK:<EFS>CH0S01.MEM]
15:15:49 LPMSG  [LPTSTF  STARTING FILE DSK:<EFS>CH0S02.MEM]
15:15:57 LPMSG  [LPTCPT  CHECKPOINT TAKEN DURING FILE DSK:<EFS>CH0S02.MEM, CO
15:16:31 LPMSG  [LPTFPF  FINISHED PRINTING FILE DSK:<EFS>CH0S02.MEM]
15:16:31 LPMSG  [LPTSTF  STARTING FILE DSK:<EFS>CH1S01.MEM]
15:16:52 LPMSG  [LPTFPF  FINISHED PRINTING FILE DSK:<EFS>CH1S01.MEM]
15:16:52 LPMSG  [LPTSTF  STARTING FILE DSK:<EFS>CH1S03.MEM]
15:17:38 LPMSG  [LPTFPF  FINISHED PRINTING FILE DSK:<EFS>CH1S03.MEM]
15:17:39 LPMSG  [LPTSTF  STARTING FILE DSK:<EFS>CH1S05.MEM]
15:17:50 LPMSG  [LPTFPF  FINISHED PRINTING FILE DSK:<EFS>CH1S05.MEM]
15:17:51 LPMSG  [LPTSTF  STARTING FILE DSK:<EFS>CH1S06.MEM]
15:19:00 LPMSG  [LPTCPT  CHECKPOINT TAKEN DURING FILE DSK:<EFS>CH1S06.MEM, CO
15:19:47 LPMSG  [LPTFPF  FINISHED PRINTING FILE DSK:<EFS>CH1S06.MEM]
15:19:48 LPMSG  [LPTSTF  STARTING FILE DSK:<EFS>CH2S02.MEM]
15:21:40 LPMSG  [LPTFPF  FINISHED PRINTING FILE DSK:<EFS>CH2S02.MEM]
15:21:42 LPMSG  [LPTSTF  STARTING FILE DSK:<EFS>CH2S03.MEM]
15:22:05 LPMSG  [LPTCPT  CHECKPOINT TAKEN DURING FILE DSK:<EFS>CH2S03.MEM, CO
15:22:31 LPMSG  [LPTFPF  FINISHED PRINTING FILE DSK:<EFS>CH2S03.MEM]
15:22:32 LPMSG  [LPTSTF  STARTING FILE DSK:<EFS>CH2S06.MEM]
15:23:35 LPMSG  [LPTFPF  FINISHED PRINTING FILE DSK:<EFS>CH2S06.MEM]
15:23:39 LPMSG  [LPTSTF  STARTING FILE DSK:<EFS>CH2S07.MEM]
15:25:05 LPMSG  [LPTFPF  FINISHED PRINTING FILE DSK:<EFS>CH2S07.MEM]
15:25:07 LPMSG  [LPTSTF  STARTING FILE DSK:<EFS>CH2S08.MEM]
15:25:13 LPMSG  [LPTCPT  CHECKPOINT TAKEN DURING FILE DSK:<EFS>CH2S08.MEM, CO
15:26:08 LPMSG  [LPTFPF  FINISHED PRINTING FILE DSK:<EFS>CH2S08.MEM]
15:26:09 LPMSG  [LPTSTF  STARTING FILE DSK:<EFS>CH2S09.MEM]
15:26:28 LPMSG  [LPTFPF  FINISHED PRINTING FILE DSK:<EFS>CH2S09.MEM]
15:26:29 LPMSG  [LPTSTF  STARTING FILE DSK:<EFS>CH2S10.MEM]
15:27:39 LPMSG  [LPTFPF  FINISHED PRINTING FILE DSK:<EFS>CH2S10.MEM]
15:27:39 LPMSG  [LPTSTF  STARTING FILE DSK:<EFS>CH2S12.MEM]
15:28:18 LPMSG  [LPTCPT  CHECKPOINT TAKEN DURING FILE DSK:<EFS>CH2S12.MEM, CO
15:29:19 LPMSG  [LPTFPF  FINISHED PRINTING FILE DSK:<EFS>CH2S12.MEM]
15:29:19 LPMSG  [LPTSTF  STARTING FILE DSK:<EFS>CH2S15.MEM]
15:30:08 LPMSG  [LPTFPF  FINISHED PRINTING FILE DSK:<EFS>CH2S15.MEM]
15:30:09 LPMSG  [LPTSTF  STARTING FILE DSK:<EFS>CH2S17.MEM]
15:31:24 LPMSG  [LPTCPT  CHECKPOINT TAKEN DURING FILE DSK:<EFS>CH2S17.MEM, CO
15:31:58 LPMSG  [LPTFPF  FINISHED PRINTING FILE DSK:<EFS>CH2S17.MEM]
15:31:59 LPMSG  [LPTSTF  STARTING FILE DSK:<EFS>CH3S04.MEM]
15:32:48 LPMSG  [LPTFPF  FINISHED PRINTING FILE DSK:<EFS>CH3S04.MEM]
15:32:48 LPMSG  [LPTSTF  STARTING FILE DSK:<EFS>CH3S06.MEM]
15:32:57 LPMSG  [LPTFPF  FINISHED PRINTING FILE DSK:<EFS>CH3S06.MEM]
15:32:57 LPMSG  [LPTSTF  STARTING FILE DSK:<EFS>CH3S07.MEM]
15:33:25 LPMSG  [LPTFPF  FINISHED PRINTING FILE DSK:<EFS>CH3S07.MEM]
15:33:25 LPMSG  [LPTSTF  STARTING FILE DSK:<EFS>CH4S01.MEM]
15:34:25 LPMSG  [LPTCPT  CHECKPOINT TAKEN DURING FILE DSK:<EFS>CH4S01.MEM, CO
15:35:12 LPMSG  [LPTFPF  FINISHED PRINTING FILE DSK:<EFS>CH4S01.MEM]
15:35:13 LPMSG  [LPTSTF  STARTING FILE DSK:<EFS>CH4S03.MEM]
```

MOORE BUSINESS FORMS, FORM 8510 PRINTED IN U.S.A.

```
15:37:33 LPMSG  [LPTCPT  CHECKPOINT TAKEN DURING FILE DSK:<EFS>CH4S03.MEM, CO
15:37:49 LPMSG  [LPTFPF  FINISHED PRINTING FILE DSK:<EFS>CH4S03.MEM]
15:37:50 LPMSG  [LPTSTF  STARTING FILE DSK:<EFS>CH5S01.MEM]
15:38:17 LPMSG  [LPTFPF  FINISHED PRINTING FILE DSK:<EFS>CH5S01.MEM]
15:38:18 LPMSG  [LPTSTF  STARTING FILE DSK:<EFS>CH5S02.MEM]
15:38:33 LPMSG  [LPTFPF  FINISHED PRINTING FILE DSK:<EFS>CH5S02.MEM]
15:38:35 LPMSG  [LPTSTF  STARTING FILE DSK:<EFS>CH5S04.MEM]
15:39:12 LPMSG  [LPTFPF  FINISHED PRINTING FILE DSK:<EFS>CH5S04.MEM]
15:39:12 LPMSG  [LPTSTF  STARTING FILE DSK:<EFS>CHAS01.MEM]
15:39:20 LPMSG  [LPTFPF  FINISHED PRINTING FILE DSK:<EFS>CHAS01.MEM]
15:39:21 LPSUM  SPOOLER RUNTIME 39 SECONDS, 592 PAGES PRINTED


000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000111111111122222222223333333333444444444455555555556666666666777777777
1234567890123456789012345678901234567890123456789012345678901234567890123467
 **END** USER MCCARTHY  JOB  CH0S01 SEQ. 1949 DATE  3-DEC-76 15:39:21
MONITOR 555 LCEG SYSTEM, TOPS-20 MONITOR 1B **END**
```


END USER MCCARTHY JOB CH0S01 SEQ. 1949 DATE 3-DEC-76 15:39:21
MONITOR 555 LCEG SYSTEM, TOPS-20 MONITOR 1B **END**

END USER MCCARTHY JOB CH0S01 SEQ. 1949 DATE 3-DEC-76 15:39:21
MONITOR 555 LCEG SYSTEM, TOPS-20 MONITOR 1B **END**

END USER MCCARTHY JOB CH0S01 SEQ. 1949 DATE 3-DEC-76 15:39:21
MONITOR 555 LCEG SYSTEM, TOPS-20 MONITOR 1B **END**

END USER MCCARTHY JOB CH0S01 SEQ. 1949 DATE 3-DEC-76 15:39:21
MONITOR 555 LCEG SYSTEM, TOPS-20 MONITOR 1B **END**

END USER MCCARTHY JOB CH0S01 SEQ. 1949 DATE 3-DEC-76 15:39:21
MONITOR 555 LCEG SYSTEM, TOPS-20 MONITOR 1B **END**

END USER MCCARTHY JOB CH0S01 SEQ. 1949 DATE 3-DEC-76 15:39:21
MONITOR 555 LCEG SYSTEM, TOPS-20 MONITOR 1B **END**

END USER MCCARTHY JOB CH0S01 SEQ. 1949 DATE 3-DEC-76 15:39:21
MONITOR 555 LCEG SYSTEM, TOPS-20 MONITOR 1B **END**

END USER MCCARTHY JOB CH0S01 SEQ. 1949 DATE 3-DEC-76 15:39:21
MONITOR 555 LCEG SYSTEM, TOPS-20 MONITOR 1B **END**

END USER MCCARTHY JOB CH0S01 SEQ. 1949 DATE 3-DEC-76 15:39:21
MONITOR 555 LCEG SYSTEM, TOPS-20 MONITOR 1B **END**

END USER MCCARTHY JOB CH0S01 SEQ. 1949 DATE 3-DEC-76 15:39:21
MONITOR 555 LCEG SYSTEM, TOPS-20 MONITOR 1B **END**

END USER MCCARTHY JOB CH0S01 SEQ. 1949 DATE 3-DEC-76 15:39:21
MONITOR 555 LCEG SYSTEM, TOPS-20 MONITOR 1B **END**