+-------------------------+
| d | i | g | i | t | a | l |
+-------------------------+

interoffice

memorandum

To: Peter Hurley
Jan Jaferian

Date: 05 Jan 83
From: Mike Uhler
Dept: L.S.E.G.
DTN: (8-)231-6448
Loc/Mail stop: MR01-2/E85
Net mail: UHLER at IO

Subject: A Systems Concepts PDP-10

1.0 Executive summary

The possibility that Systems Concepts will design a PDP-10 is interesting but has some problems associated with it. Among these problems are:

- Defining what is to be implemented in the absence of a complete architectural description of a PDP-10.
- Providing them with enough documentation so that they can be successful in implementing the architecture.
- Defining a set of tests that can be used to measure correct implementation before the monitor and user-mode software run on the machine.

The next three sections of this memo discuss these problems. CPU requirements are suggested based on several sources of available documentation. Tests are identified (primarily based on work done for the Jupiter project) that could be used to measure correct implementation.

Subsequent sections discuss areas that are not well-defined and may benefit from further analysis. These areas include performance implications, the I/O structure of the machine, RAMP, resolution of architectural and implementation problems, etc.

2.0 Defining the CPU requirements

When trying to define the instruction set requirements for a new processor, one typically uses an existing processor as the model to say "this is how it should work". Given the numerous problems with the KL10 implementation of the PDP-10 architecture, I believe that we should use the Jupiter as the basis for stating the architectural requirements for the Systems Concepts design. Such a decision benefits from the following:

- The learning process involved in the implementation of another PDP-10.
- Bug fixes made in Jupiter that were not made on the KL10 for some reason.
- Resolution of the correct definition of the extended addressing architecture.
- Implementation of the full 30 bits of virtual address space.
- Additional microcode information provided on MUUO and page fail traps to the monitor.
- Improved methods for handling LUUOs, MUUOs, traps, page fails, interrupts, etc.
- New instructions implemented in Jupiter but not in the KL10.

If we indeed use the Jupiter design as the basis for defining requirements for the Systems Concepts design, there are two simple statements that can be made:

- A non-privileged user-mode program should be able to detect no differences when run on a Jupiter and on the Systems Concepts design.
- Ideally, an exec-mode program should be able to detect no differences when run on a Jupiter and on the Systems Concepts design. In the past, we have made changes to the exec mode environment with every processor implementation. This process resulted from a close coupling between the hardware designers and monitor programmers. Since the coupling is much looser in this case, it is more important to make the exec-mode environments identical.

3.0 Available documentation

The primary source of documentation on both the instruction set and the PDP-10 architecture is the Processor Reference Manual. There are, however, some serious deficiencies in certain areas.

Therefore, I suggest defining the instruction set and architectural requirements based on the Processor Reference Manual, as amended by the following:

- The annotations in my copy of the Processor Reference Manual. My copy contains corrections for some of the more serious problems. These pages can be copied and inserted in another copy of the manual.

- My memo on Extended Addressing. This memo has been reviewed by the Architecture Committee and it represents a "definition" of the extended addressing architecture.

- KCIO.MEM. This document is the Jupiter exec-mode spec. It defines all exec-mode instructions, and other processor dependent information.

4.0 Verifying the correctness of the implementation

To fully verify that the implementation complies with the architecture, we obviously need to run the monitor and existing user programs. However, other tests can be run before the monitor to give us a first-order indication that the machine correctly implements the instruction set. This set of tests consists of Jupiter CPU diagnostics and other programs that test significant pieces of the architecture.

The following list consists of basic instruction tests that have run successfully on Jupiter hardware and are known to work:

DCKAA	MOVE,SKIP,AND,XOR,EQV,Boole
DCKAB	MOVE,Compare,Test,HWtest,Boole,ADD
DCKAC	Logical,HWtest
DCKAD	Reg addr,JFCL,AR flags,AOS,SOS,JRST,AOBJ,JSP, XCT,indirect and indexed addressing
DCKAE	FWtest,ADD,SUB,PC change,Compare
DCKAF	Boole,HWtest,Test
DCKAG	PUSH,POP,XCT,Shift,Rotate
DCKAH	PI system,Interrupts,LUUOs,Processor I/O
DCKAI	Shift,Rotate
DCKAJ	Shift,Rotate,Arithmetic shift
DCKAK	Basic multiply (part 1)
DCKAL	Basic multiply (part 2)
DCKAM	General multiply
DCKAN	Multiply,Divide
DCKAO	Byte,BLT,JFFO,misc.
DCKBA	Basic instruction exerciser
DCKCA	FSC,Floating add,Floating subtract,normalize,round
DCKCB	Floating multiply,Floating divide
DCKCC	FIX,FIXR,FLTR,DMOVE,DMOVN,DMOVEM,DMOVNM
DCKCD	Double floating register set integrity
DCKCE	DFAD,DFSB,DFMP,DFDV

DCKGA	DADD,DSUB,DMUL,DDIV

The following list consists of extended functionality diagnostics that are either not complete or not debugged as of this date. They should be run if they are available:

DCKDA	TOPS-20 paging
DCKDB	Extended addressing
DCKEB	PXCT
DCKFA	Timer instructions
DCKGB	Extended exponent
DCKGC	Extended instruction set
DCKGD	I/O instructions
DCKGE	New (non I/O) instructions

The following list consists of programs that were written or adapted to assist in debugging Jupiter microcode and hardware. They are known to work and should all be run:

EAT	Extended addressing test
TPAGER	Paging tests
TUUO	UUO tests
TLUUO	LUUO tests
TTRAPS	Trap 1,2,3 tests
TPXCT	PXCT tests
PITST	PI system tests
KCBP	Byte and string instruction tests

In addition to the tests listed above, all examples in my Extended Addressing memo should be run on the machine and yield the results indicated. We should probably write a simple test containing all the examples.

5.0 Areas requiring additional thought

When considering the possibility of obtaining a PDP-10 design from a source outside the company, topics other than simply defining the instruction set to be implemented come to mind. This section contains some thoughts in these areas. It is not intended to be a thorough review of the topics mentioned. Rather, it is intended to provoke some thought. Most of these items are not specific to the Systems Concepts design but are things that should be considered in any future PDP-10 design.

5.1 Measuring the performance of the machine

We are currently in the process of analyzing the performance of the Jupiter CPU. This project has demonstrated that there is no simple way to calculate the performance of a machine. Some of the methodologies that are developed during the Jupiter performance analysis project may be of use in measuring the performance of the

Systems Concepts machine. Unfortunately, these methodologies may not be available for several months. One way of making a gross estimate of machine performance may be applicable, and it is described below.

Some amount of opcode histogram data (also known as workload data) is available from sites that represent Fortran, Cobol, and general timesharing loads. This data gives the frequency of occurrence of each opcode measured during normal machine operation. To obtain an estimate of machine performance, one must know the time to execute each instruction in the instruction set on both the new machine and on the machine being used as a baseline (presumably the KL10 in this case). The time to execute the sample workload is then calculated by the summation of the products of instruction frequency and instruction time for each instruction. The ratio of times to execute the workload on the new machine and the baseline machine is then an estimate of machine performance.

Workload data for various job mixes and timing information for the KL10 is available today. In addition, a measurement program exists that may be useful in measuring the speed of each instruction on the new machine.

5.2 Effects of extended addressing

The use of extended addressing will increase significantly over the next few years. We know already that extended addressing causes an increased use of indexing and indirect addressing modes and this should be emphasized. We have no existing tests that measure the effect of increased use of extended addressing, so any evaluation of the performance of a new machine running extended addressing will be somewhat ad hoc.

5.3 Translation buffer organization and size

There have been some major performance problems concerning the size and organization of the KL10 translation buffer. While these problems have occurred in pathological cases up to now, the increased use of extended addressing could make the pathological case more common in the future. On Jupiter, we significantly increased the size of the translation buffer, but the organization remains 1-way associative. There is some belief that a multi-way associative translation buffer might have a major impact on the performance of the system, and no future PDP-10 processor should be built without understanding the pros and cons of this decision.

5.4 Cache organization and size

The KL10 contains a 2K, 4-way associative data cache which typically gives hit rates around 92%. The Jupiter contains an 8K, 4-way associative data cache which is expected to give a hit rate around 98%. If any cache analysis is done during the Jupiter performance project, this data should be considered for future designs.

5.5 Performance instrumentation

When our machines don't perform well, we tend to look for software solutions first. This makes sense since software is easier to change than hardware (and in some ways, microcode). However, when the problem really is a hardware bottleneck, we have no good tools with which to find the bottlenecks because our hardware has insufficient instrumentation points. It should be stated as a goal that all future machines have hardware and microcode instrumentation points so that hardware bottlenecks can be detected.

5.6 I/O structure of the machine

How does the machine do I/O? It is apparent that all future PDP-10 designs must have a CI and NI interface in order to make use of the new I/O architectures that are being developed.

What is the -10 interface to the console and what are the capabilities of the console? It should be a goal to make the protocol between the -10 and the console similar, if not identical, to that used by the Jupiter console.

5.7 RAMP

What are the RAMP characteristics of the new machine? Is error detection and recovery better than that of the KL10? Is there an increased requirement that the monitor handle hardware exception conditions?

5.8 Microcode space requirements

On all previous microcoded processors, we have quickly consumed all of the available microcode space. This situation has caused numerous problems including the inability to fix bugs. It should be a stated goal that there be at least 10% unused space remaining in the microcode when all instructions are implemented.

5.9 Problem resolution

How are we to resolve implementation and architectural problems with the new design? It has been difficult to do this in the past even when we all worked for the same company. Who has final approval that the design correctly implements the PDP-10 architecture? If the implementation doesn't comply with the architecture, how do we get it changed? How does Systems Concepts request approval for a change to the architecture or clarification on a point of architecture?

6.0 Conclusions

Specifying the requirements for a PDP-10 hardware design is not a simple task, as we've seen in the past. This task is made more difficult by the lack of a formal and complete definition of the PDP-10 architecture. If we are to attempt to specify such requirements, we should start with the architectural design for Jupiter rather than that for the KL10.

Some documentation is available to define the requirements for the design, but it is not complete. Programs are available against which the design could be measured for correct implementation. However, we all know that the only true measure is a running monitor and user-mode software.

There are additional areas of concern that should be considered in more detail. These area include performance characteristics of the machine, I/O structure, RAMP, problem resolution, etc. These are important aspects to overall system design and should not be ignored.

+-------------------------+
| d | i | g | i | t | a | l |
+-------------------------+

interoffice

memorandum

To: Jan Jaferian

CC: Peter Hurley

Date: 17 Jan 83
From: Mike Uhler
Dept: L.S.E.G.
DTN: (8-)231-6448
Loc/Mail stop: MR01-2/E85
Net mail: UHLER at IO

Subject: Documentation for Systems Concepts

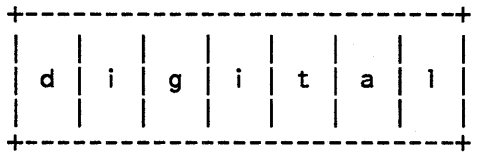
Attached are copies of the Systems Concepts documentation as mentioned in my memo of 05 Jan 83. The state of this documentation is as follows:

1. KCIO. This is the Jupiter exec mode spec. It contains information on exec mode instructions, new I/O operations, the Jupiter paging data structures, trap, UUO, and interrupt handling, etc. This spec contains information about an unannounced product, so it is technically company confidential. However, since it contains only hardware/software interface descriptions, I believe that it contains minimal (if any) proprietary information.

2. The memo entitled "Extended addressing". This memo is the result of some work that we did during the Jupiter microcode implementation. All sections except for section 12 (PXCT) have been reviewed and approved by the Architecture Committee. Section 12 has had a minimal review and has been updated based on comments from that review. Full review of section 12 should be complete within two weeks.

3. Pages from my annotated copy of the Processor Reference Manual (AA-H391A-TK with June 1982 update AD-H391A-T1 installed). These pages correct some errors in the existing manual. They have not been reviewed by anyone else, but I believe that they are accurate based on past Architecture Committee meetings.

If you have additional questions, I'd be happy to answer them.



interoffice

memorandum

To: Jan Jaferian

CC: Peter Hurley

Date: 17 Jan 83
From: Mike Uhler
Dept: L.S.E.G.
DTN: (8-)231-6448
Loc/Mail stop: MR01-2/E85
Net mail: UHLER at IO

Subject: Verification programs for Systems Concepts

The attached tape contains 30 verification programs as mentioned in section 4.0 of my memo of 05 Jan 83. The programs are as follows:

DCKAA-DCKAO
DCKBA
DCKCA-DCKCE
DCKGA
EAT
TPAGER
TUUO
TLUUO
TTRAPS
TPXCT
PITST
KCBP

Note that these programs use Jupiter instructions and data structures that are incompatible with a KL10 and will not run correctly on a KL10. In addition, most of the DCKxx programs use the Jupiter console protocol to type messages on the console. If the Systems Concepts machine implements the Jupiter instructions, data structure and console protocol, these programs should run unmodified. If not, modifications to the programs will be required and any modifications should be approved by us.

PERFORMANCE VERIFICATION

The following programs must run in stand alone mode on a System Concepts system with an execution time less than or equal to that of KL10E system. The two systems would be running either TOPS-10 (7.02) or TOPS-20 (5.1) and would have the same amount of memory and disk space.

COBOL PROGRAMS

COMPUT	Decimal Arithmetic Packed Decimal Arithmetic Integer Arithmetic Floating Point Arithmetic
MOVIT	String manipulation
NPRF1	General Cobol
USTEEL	General Cobol
TIMTST	General Cobol

FORTRAN PROGRAMS

SPEED	Floating point Single precision
WHETS	Single precision
WHETD	Double precision, floating point
FFT45	Fast Fourrier Transform
HANOI	Integer Arithmetic
The RARICH Tests	(A-W) General Fortran
MODCUT	Extended Addressing Fortran
SP1111	General Fortran

MISCELLANEOUS PROGRAMS

SP1111 Compilation (Fortran)	Logic Mix
MR1CD1 Compilation (Cobol)	Logic Mix
Building TOPS-20 5.1 (Macro/Link)	Logic Mix
SANDIA Tests	Fortran and Pascal Extended Addressing
Rutgers LISP Test	Extended Addressing

PERFORMANCE VERIFICATION

As a basic verification of the S.C. CPU performance, the S.C. CPU instruction timing results from "SPEEDY" will be compared to the KL10E (with MF20 memory and cache) instruction times. The general mix of instructions and their appropriate weighting (as has already been given to S.C.) must yield an aggregate performance value equal to or better than that of the KL10E. Also, no S.C. instruction timings should be longer than 1.2 times the KL10E timings and the floating point instructions and the byte and string manipulation instructions timings must all be as good as or better than those of the KL10E.

CPU

1. Acceptance Criteria:

1.1 A non-privileged user-mode program running under TOPS-10 or TOPS-20 should not be able to detect that it is running on a SYSTEMS CONCEPTS Unit instead of a KL System through the execution of any PDP-10 instructions.

Some differences may be allowed if they are deemed by DIGITAL to be desirable improvements to the PDP-10 architecture. Such waivers must be made by DIGITAL in writing.

1.2 The definition of the PDP-10 Architecture is contained in an attached copy of the Processor Reference Manual which has been annotated with the latest DIGITAL approved updates to the PDP-10 Architecture and in an attached copy of a memo from Mike Uhler describing the PDP-10 Extended Addressing Architecture. A third document, KC10.MEM, also attached, represents the current directions in which the PDP-10 Architecture is headed for EXEC Mode instructions. Adherence to this specification is mandatory.

1.3 As a partial demonstration of the conformance of the SYSTEMS CONCEPTS Unit, DIGITAL is supplying a set of programs and examples which must be run without unauthorized modification and which must complete without errors and produce the correct results. The list of programs is as follows:

DCKAA	MOVE, SKIP, AND, XOR, EQV, Boole
DCKAB	MOVE, Compare, Test, HW test, Boole, ADD
DCKAC	Logical, HWtest
DCKAD	Reg addr, JFCL, AR flags, AOS, SOS, JRst, AOBJ, JSP, XCT, indirect and indexed addressing
DCKAE	FWtest, ADD, SUB, PC change, Compare
DCKAF	Booke, HWtest, Test
DCKAG	PUSH, POP, XCT, Shift, Rotate
DCKAH	PI system, Interrupts, LUUOs, Processor I/O
DCKAI	Shift, Rotate

EXHIBIT C

DCKAJ	Shift, Rotate, Arithmetic shift
DCKAK	Basic multiply, (part 1)
DCKAL	Basic multiply (part 2)
DCKAM	General multiply
DCKAN	Multiply, Divide
DCKAO	Byte, BLT, JFFO, misc.
DCKBA	Basic instruction exerciser
DCKCA	FSC, Floating add, Floating subtract, normalize, round
DCKCB	Floating mltiply, Floating divide
DCKCC	FIX, FIXR, FLTR, DMOVE, DMOVN, DMOVEM, DMOVNM
DCKCD	Double floating register set integrity
DCKCE	DFAD, DFSB, DFMP, DFDV
DCKGA	DADD, DSUB, DMUL, DDIV
EAT	Extended addressing test
TPAGER	Paging tests
TUUO	UUO tests
TLUUO	LUUO tests
TTRAPS	Trap 1,2,3 tests
TPXCT	PXCT tests
PITST	PI system tests
KCBP	Byte and string instruction tests

In addition to the tests listed above, all examples in the Extended Addressing memo should be run on the Unit and yield the results indicated.

1.4 Acceptance shall be considered complete when the programs listed in 1.3 have been demonstrated to run properly; the diagnostics supplied by SYSTEMS CONCEPTS are verified; user mode compatibility has been satisfactorily verified by DIGITAL; design modification responsibility has been assigned and accepeted by either DIGITAL or SYSTEMS CONCEPTS as appropriate, and DIGITAL has written an evaluation report, which has been presented to both DIGITAL and SYSTEMS CONCEPTS. This Acceptance shall take place within six (6) months of the Unit having been delivered to DIGITAL and having demonstrated to run the designated conformance tests.

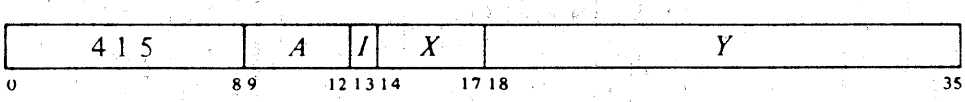
1.5 Successful Demonstration of Remote Serviceability capability as outlined in attached Documents, and referred appendicies.

2. Performance Criteria:

ANNOTATED PAGES OF THE PROCESSOR REFERENCE MANUAL GIVEN TO SYSTEMS CONCEPTS

The following instruction makes the result of an effective address calculation available for use as a global address, even for accessing a fast memory location from any section.

XMOVEI Extended Move Immediate



If the program is running in a nonzero section, do one or the other of the following.

If E is not a local AC address, clear AC bits 0–5 and place the global effective address E in AC bits 6–35.

If E is a local AC address, put 1 in AC left and E in AC right.

IN A NON-ZERO SECTION

If the program is running in section 0, this instruction is called SETMI, a Boolean instruction that performs an analogous function for section 0 (§2.4).

Notes. The form given a local AC address is that of a global AC address, which therefore still refers to fast memory no matter what section the address may be moved to or used in. Giving XMOVEI with an address 20 or greater without indexing or indirection places the current PC section number in AC left, and it can thus be used to determine what section the program is in.

E.G.

$1_{11}100$/XMOVEI AC,@[$200000_{11}202$]

$0_{11}200$/$0_{11}2$

STORES $0_{11}2$ IN AC, NOT $1_{11}2$

Double Move Instructions[1]

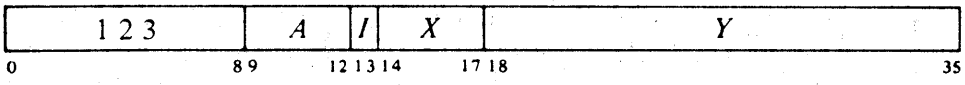
These four instructions are principally for manipulating the double length operands used in double precision arithmetic, fixed or floating. But they may be used to move or negate any doubleword, i.e. the contents of a pair of adjacent accumulators or memory locations. Two of the instructions are simple extensions of MOVE and MOVEM to doublewords, and for them the configuration of the operands is irrelevant. The other two are extensions of MOVN and MOVNM, with the operand interpreted as a double precision floating point number. They can just as well be used for fixed point numbers, but with a slight variation in the format. Namely a negative result has a 0 in bit 0 of the low order word instead of a copy of the sign. For arithmetic operations per se this difference is inconsequential, as all arithmetic instructions ignore bit 0 of all low order words. However it could cause a comparison of two equal double precision numbers to fail.

All of these instructions address a pair of adjacent accumulators and a pair of adjacent memory locations. The accumulators have addresses A and A+1 (mod 20_8), the memory locations have addresses E and E+1.

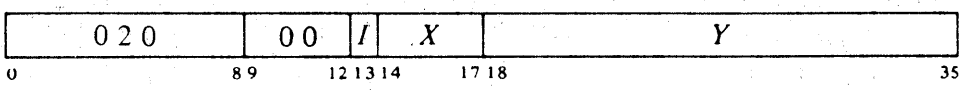
---

[1] In the KA10 these instructions are trapped as unassigned codes (§2.16).

XBLT Extended Block Transfer

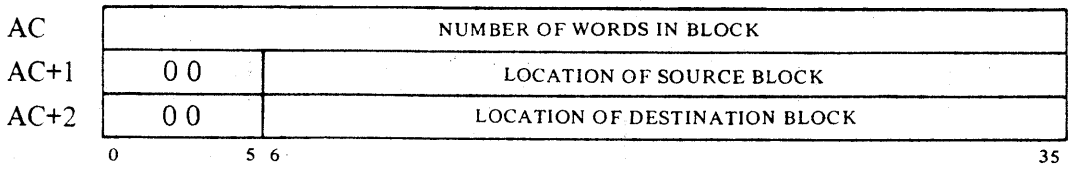


E0



E1 is not used.[3]

Move a block of words from one area of memory to another. The block size and the locations of the source and destination areas are defined by the contents of a block of three accumulators.



~~If this instruction is given in section 0, execute it as an MUUO. Otherwise~~ perform a forward or backward block transfer as follows.

If AC contains a positive number N, move a block of N words from a source area beginning at the location specified by AC+1, to a destination area beginning at the location specified by AC+2, and extending through increasing addresses. At the end AC is clear, and AC+1 and AC+2 respectively contain addresses one greater than those of the final source and destination locations referenced.

If AC contains a negative number $-N$, move a block of N words from a source area beginning at a location one less than that specified by AC+1, to a destination area beginning at a location one less than that specified by AC+2, and extending through decreasing addresses. At the end AC is clear, and AC+1 and AC+2 respectively contain the addresses of the final source and destination locations referenced.

CAUTION

This instruction uses three accumulators, and under no circumstances should any of these three be part of either the source or destination block. Because of the possibility of an interrupt or page failure, the contents of these accumulators even as a source cannot be guaranteed. And in any event, use of XBLT for moving an AC block is quite unnecessary, as a simple BLT can move fast memory to any section.

---

[3] *I*, *X* and *Y* are reserved and should be zero.

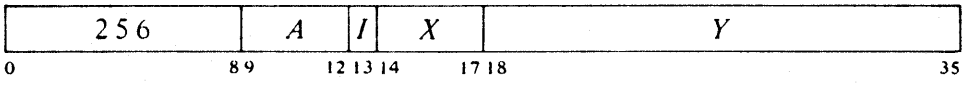
which the disruption in the normal sequence occurred. Saving the program position is referred to as "saving PC," although the quantity actually saved may be the value currently contained in PC or an address one greater than that, depending on the circumstances. For example, the same instruction may be used to call a subroutine in a program or to call a service routine in an interrupt. When a return is later made using the saved address in the subroutine case, the instruction that saved PC should not be repeated — the return should be made instead to the instruction following it in normal sequence, i.e. the instruction at the address one greater than that originally in PC. In the interrupt case, on the other hand, a subsequent return has nothing to do with the instruction that saved PC — the return should be made to the interrupted instruction, the one PC pointed at when the interrupt occurred. Both cases are covered in the instruction descriptions by the phrase "save PC," and it is to be assumed that the address saved is the one appropriate to the situation in which the instruction is given.

Sometimes regarded as program control, in a somewhat trivial sense, are those instructions that do nothing. The most commonly used no-op is JFCL, which is described here. Other no-ops are among the testing and Boolean instructions discussed previously: SETA, SETAI, SETMM, CAI, CAM, JUMP, TRN, TLN, TDN, TSN.[17] Of these, SETA, SETAI, CAI, JUMP, TRN and TLN are preferred because they do not use the calculated effective address to reference memory.

The Execute Instruction

This instruction allows the programmer to execute the contents of any memory location as an instruction without altering the normal program counting sequence to do it.

XCT Execute



If *A* is zero or the processor is in user mode or is a KA10, execute the contents of location *E* as an instruction.[18] Any instruction may be executed, including another XCT. If an XCT executes a skip instruction, the skip is relative to the location of the XCT (the first XCT if there are several in a chain). If an XCT executes a jump, program flow is altered as specified by the jump (no matter how many XCTs precede a jump instruction, when PC is saved it contains an address one greater than the location of the first XCT in the chain).

NOTE THAT EA-CALC OF XCTED INSTRUCTION IS RELATIVE TO THE SECTION FROM WHICH IT WAS FETCHED AND NOT NECESSARILY TO THE SECTION CONTAINING THE XCT

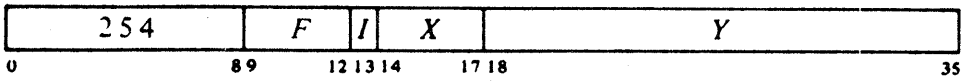
[17] KA10 instruction codes 247 and 257 are reserved for instructions installed specially for a particular system. They execute as no-ops when run on a KA10 that contains no special hardware for them, but for program compatibility it is advised that they not be used regularly as no-ops.

[18] *Caution:* In a private program (concealed or kernel mode) on the KI10, never give an XCT that executes an instruction in a public page. It does not work.

		Extended KL10	*Single-section KL10*	*KS10*	*KI10*	*KA10*	KC10
JRST 0,	JRST	Yes	Yes	Yes	Yes	Yes	YES
JRST 1,	PORTAL	Yes	Yes	Yes	Yes	Yes	YES
JRST 2,	JRSTF	Z	Yes	Yes	Yes	Yes	Z
JRST 3,		No	No	No	Yes	Yes	NO
JRST 4,	HALT	K–H	K–H	K–H	K–H	IO–H	K-H
JRST 5,	XJRSTF	Yes	No	Yes	K–H	IO–H	YES
JRST 6,	XJEN	IO	No	K	K–H	IO–H	K
JRST 7,	XPCW	IO	No	K	K–H	IO–H	K
JRST 10,		IO	IO	IO	K	IO	NO
JRST 11,		No	No	No	K	IO	NO
JRST 12,	JEN	Z ∧ IO*	IO	IO	K	IO	Z∧IO
JRST 13,		No YES	No	No	K	IO	NO
JRST 14,	SFM	~~NZ V IO~~*	No	K	K–H	IO–H	YES
JRST 15,	XJRST	~~No~~ YES	No	No	K–H	IO–H	YES
JRST 16,		No	No	No	K–H	IO–H	NO
JRST 17,		No	No	No	K–H	IO–H	NO

* JEN is legal only where IO is legal in section 0; SFM is legal anywhere in a nonzero section and also where IO is legal in section 0.

JRST Jump and Restore (KI10–KA10)



Perform the functions specified by *F* if they are legal; then if the function was performed and the processor is not halted, take the next instruction from location *E* and continue sequential operation from there. Bits 9–12 are programmed as follows.

Bit *Function Produced by a 1 if Legal*

9 Restore the level on which the highest priority interrupt is currently being held (dismiss the interrupt (§§5.2, 5.5)).

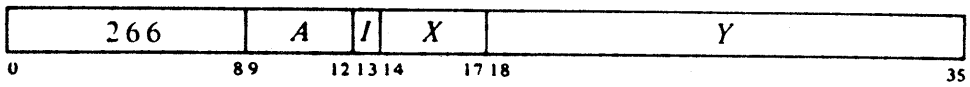
10 Halt the processor. When it stops, the AR lights on the KI10 and the MA lights on the KA10 display an address one greater than that of the location containing the instruction that caused the halt, and PC displays the jump address (the location from which the next instruction will be taken if the operator causes the processor to resume operation without changing PC).

AR or MA actually displays the address of the location that would have been executed next had the JRST been replaced by a no-

```
PRINT:  HRLI    T,440700    ;Initialize left half of pointer for
                            ;size 7, position 36
        ILDB    CH,T        ;Increment pointer and load byte
        JUMPE   CH,1(T)     ;Upon reaching zero character
                            ;return to one beyond last data word
        .                   ;Print routine
        .
        .
        JRST    PRINT+1     ;Get next character
```

The next two instructions have no capacity for handling extended addresses. Hence their usefulness is limited to making intrasection subroutine calls. However most programmers regard them as obsolete anyway, as they have been supplanted entirely by the stack instructions. FOR JSA, IF THE EA-CALC RESULTS IN A GLOBAL ADDRESS IN ANOTHER SECTION, THE RESULTS ARE UNDEFINED.

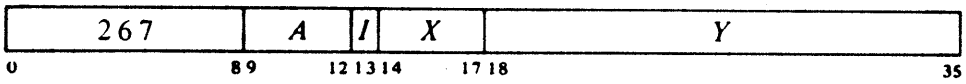
JSA Jump and Save AC



Save AC in location *E*, the in-section part of *E* in AC left, and the in-section part of PC in AC right. Then jump to location *E* + 1. The original contents of *E* are lost.

While the KA10 is in user mode, if this instruction is executed as an interrupt instruction or by an MUUO, the processor leaves user mode.

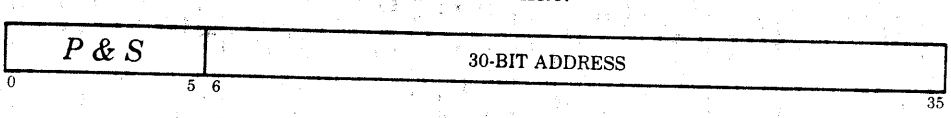
JRA Jump and Restore AC



Place the contents of the location addressed by AC left into AC, and jump to location *E*.

A JSA combines advantages of the JSR and JSP. JSA does modify memory, but it saves PC in an accumulator without losing its previous contents (at a cost of not saving the flags). It is thus convenient for multiple-entry subroutines. In a subroutine called by a JSR, the returning JRST must refer to the (single) entry point. Since a JRA can retrieve the original PC by addressing AC as an index register, it is independent of any entry point without tying up an accumulator to the extent a JSP would. The accumulator contents saved by a JSA are restored by a JRA paired with it despite intervening JSA-JRA pairs. Hence these instructions are especially useful for nesting subroutines.

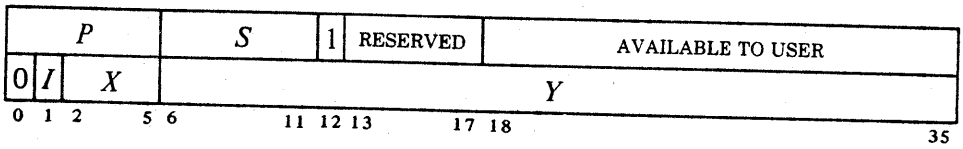
format. The one-word global pointer is available only with TOPS–20 microcode version 271 or greater, cannot use indirection, and provides for only the most common byte sizes via this format:



where the address can point to any section, and the left six bits specify both byte position and size by a number > 36 as follows.

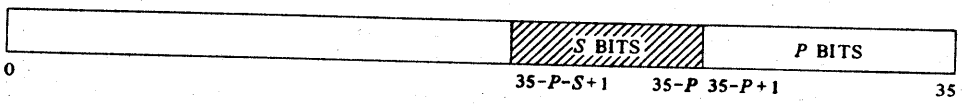
P&S	P	S	P&S	P	S
37	36	6	49	36	7
38	30	6	50	29	7
39	24	6	51	22	7
40	28	6	52	15	7
41	12	6	53	8	7
42	6	6	54	1	7
43	0	6			
			55	36	9
44	36	8	56	27	9
45	28	8	57	18	9
46	20	8	58	9	9
47	12	8	59	0	9
48	4	8			
			60	36	18
			61	18	18
			62	0	18

For unrestricted use in a nonzero section, the pointer can be a doubleword in location $E.E+1$ with this format:



2ND WORD MAY ALSO BE IN DIW FORMAT

which allows unlimited pointing, as P and S are independent, and the second word can be local or global, direct or indirect (see the discussion of indirect words in §1.6). The processor determines the number of words in a pointer with independent P and S by the state of bit 12 in the first word (in section 0 bit 12 is ignored and should be 0). Any type of pointer aims at a word whose format is



where the shaded area is the byte.

Bytes are always contiguous within a word, and the forward order is left to right in words and from low to high addresses. The position of the byte area in a word is called the "byte alignment." Let P be the position of a specified byte; $36 - P$ is then the number of bits in the left part of the word including the given byte and all byte positions at the left of it. Dividing

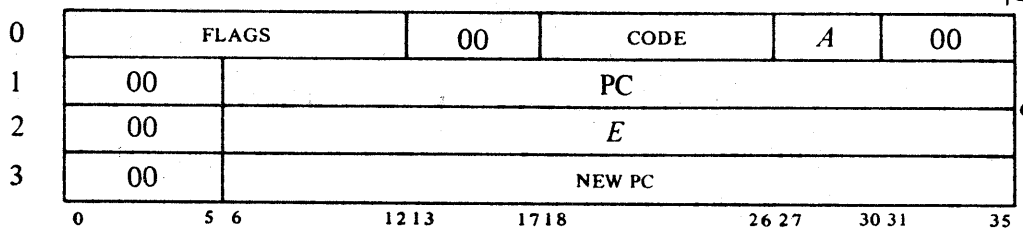
If the program is running in a nonzero section, take one of these two courses of action.

~~In executive mode perform an MUUO — not because the code is illegal, but because it is actually unassigned rather than an LUUO.~~

IN EXEC MODE, DO THE SAME AS FOR USER MODE BUT USE LOCATION 420 OF THE EPT

In user mode perform the following operations using a block of four locations beginning at that specified by bits 6–35 of location 420 in the user process table. In the first two locations save the program flags and PC in a flag-PC doubleword; in the rest of the flag word clear bits 13–17 and 31–35, and store the instruction code and *A* in bits 18–26 and 27–30. In the third location store *E* in bits 6–35 (clear bits 0–5).

IF E IS A LOCAL REFERENCE TO THE AC'S IN A NON-2020 SECTION, CONVERT IT TO THE GLOBAL AC FORM (1,,n) BEFORE STORING IT.



Then load bits 6–35 of the fourth location into PC, and continue performing instructions in normal sequence beginning at the location then addressed by PC. If *E* is a local AC address, store it in global form (i.e. with a section number of 1).

MUUOs

The actions of MUUOs depend to a considerable degree on the processor, and also on which Monitor is in use. These are the MUUO codes.

TOPS–20	104;040–051, 055–077 in section 0
TOPS–10 except KA10	040–051, 055–077
KA10	040–051, 055–100

MUUOs have considerable flexibility in the way they can alter the operating characteristics of the machine (mode, section). But the information that governs the alterations is contained in the user process table, and is therefore assumed to be under sole control of the kernel program.

The unassigned codes, which are listed in Appendix E, are not MUUOs, but the processor reacts to them in the same way in order to turn control over to the Monitor. (In the KA10 there are minor differences as explained below.) The processor also takes the same action if the program gives a JRST with an undefined function, an instruction that is illegal because of the context in which it is given, an extended instruction with incorrectly formatted accumulators, or code 000. The last is so that control returns to the Monitor should a user program wipe itself out or inadvertently attempt to execute a location that has been cleared.

The rest of this section is devoted to the different ways in which MUUOs are performed. Except in the KA10, all types use locations in the user process table to store similar information. Figure 2–3 shows what information is stored in which locations for each processor type.

Extended KL10 MUUOs. In locations 424–426 of the user process table, store the same information (as specified above) that is stored in the first three locations of an LUUO block by an LUUO given in a nonzero section, except that when the MUUO is given in executive mode, also save the previous context section in bits 31–35 of location 424. Store the "process context word" in location 427; this word saves information that partially defines the context in which the MUUO is given, and is exactly the information read by a DATAI PAG, (§3.5). Complete the specification of the MUUO context by setting up the previous context flags, and clear the rest of the flags to place the processor in kernel mode. Then load PC from bits 6–35 of the appropriate location in a PC list, and continue performing instructions in normal sequence beginning at the location then addressed by PC. (Note that the MUUO can change PC from any section to any other.) The new PC can be taken from among the eight locations in the user process table listed here depending upon the mode at the time the MUUO is given, and whether or not it is executed as the result of an overflow trap.

THE CONVERSION OF A LOCAL AC REFERENCE DESCRIBED FOR AN LUUO IS DONE FOR AN MUUO ALSO

Mode	*Execution*	*Location*
Kernel	No trap	430
Kernel	Trap	431
Supervisor	No trap	432
Supervisor	Trap	433
Concealed	No trap	434
Concealed	Trap	435
Public	No trap	436
Public	Trap	437

Single-section KL10 MUUOs. With either the TOPS-20 or TOPS-10 Monitor, MUUOs store the same information and take the same action, but they use a different set of three locations in the user process table. In the first location store the instruction code, *A* and the effective address *E* in bits 0–8, 9–12 and 18–35 respectively, and clear bits 13–17 (this is the same information as that stored by an LUUO given in section 0); save the flags and PC in a PC word in the second location; and save the process context word in the third location. Then set up the flags and PC according to the contents of the appropriate location in a PC word list, and continue performing instructions in normal sequence beginning at the location then addressed by PC. The PC word list occupies the same area as the PC list for an extended KL10, and it is organized and used (with respect to mode and trap) in the same way.

There are no restrictions on the manner in which the new PC word of an MUUO can set up the flags. It can switch the processor from any mode to any other.

- If your public program has the use of concealed programs, do not reference a location in a concealed page for any purpose except to fetch an instruction from a valid entry point, i.e. a location containing a PORTAL (JRST 1,).
- In an extended processor, do not use XBLT or SFM in section 0 or JEN in a nonzero section. Also be aware of the differences between running in section 0 and in other sections. Differences appear both in the execution of instructions, such as JSR and JSP, and in the format and handling of such quantities as index registers, indirect address words, and stack and byte pointers.
- Make sure to format the accumulators correctly in string instructions (§2.12).

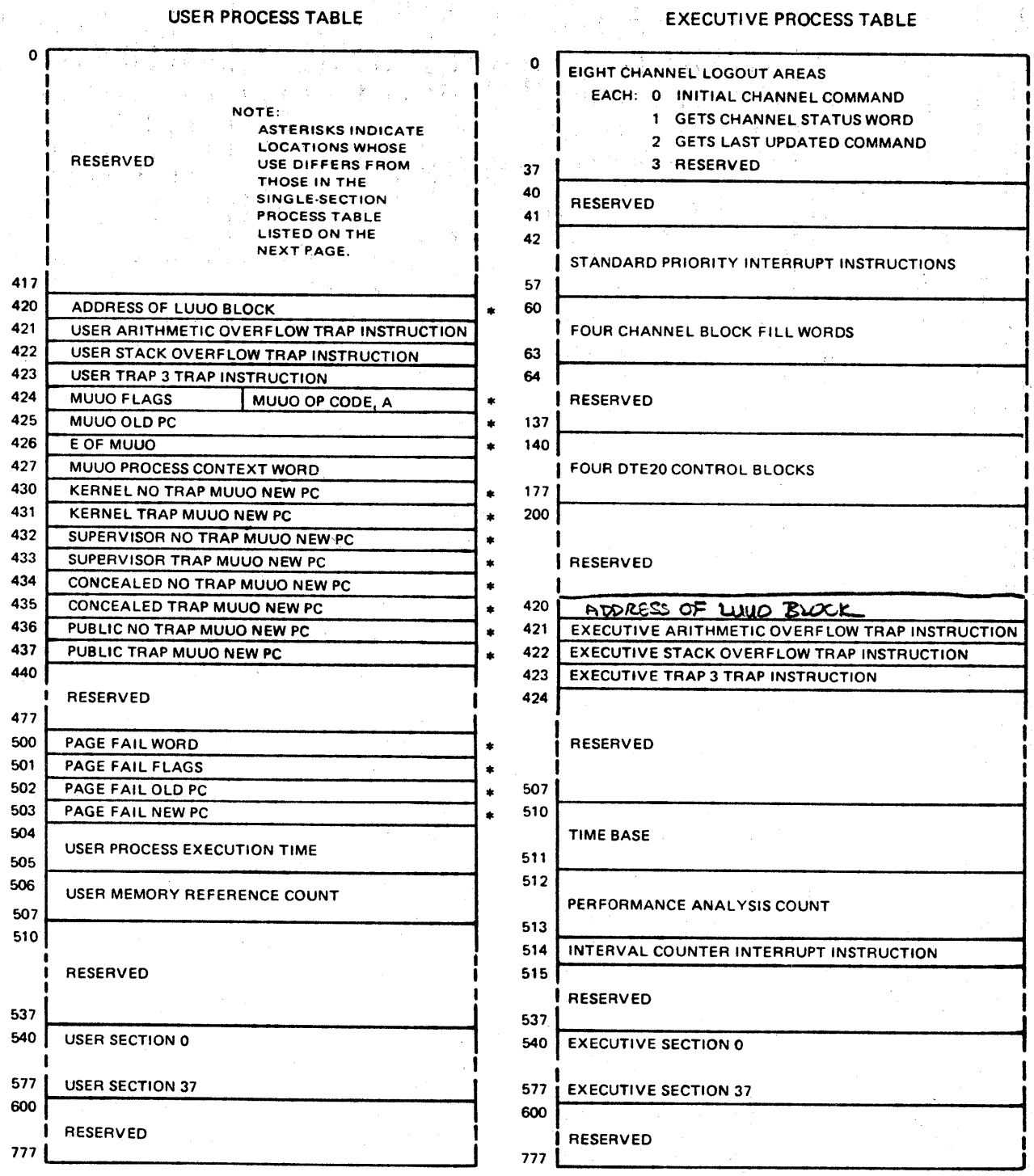
The user can give a JRSTF or XJRSTF but a 0 in bit 5 of the PC word or flag word does not clear User (a program cannot leave user mode this way); and a 1 in bit 6 does not set User In-out, so the user cannot void any of the instruction restrictions himself. Note that a 0 in bit 6 will clear User In-out, so a user can discard his own special privileges. Similiarly a 1 in bit 7 sets Public, but a 0 does not clear it, so a public program cannot enter concealed mode this way.

Many hardware characteristics however are actually transparent to the user, in particular the whole paging setup is invisible. Although the hardware allows for user virtual address spaces that are scattered or very large (even larger than available physical memory), the actual constraints will be dictated by the particular Monitor and the system manager. Most TOPS–10 Monitors enforce a two-segment virtual address space that mimics the one imposed by the KA10 hardware. In any case the user must write a sensible program, which can be handled easily and cheaply by the system; if he uses addresses a few to a page all over memory, his program can be run but will require a much larger amount of space than necessary or cause excessive page swapping.

The basic idea is to localize everything as much as possible. Do not spread parts of the program out through the address space leaving gaps. Put together whatever will be used together: divide a large program into smaller segments, and with each group of instructions put whatever pointers, data locations and the like that will be used with it. Group together subroutines that are called by the same programs. If a package is to be used at all frequently, take advantage of the various features, e.g. a core map, provided by the Digital software to determine just how the package was assembled, and if necessary revise it to reduce the working set of pages.

The rules given above apply generally to all systems, but there are minor differences from one to another, and a user who wishes to write programs to run on more than one type of processor must be aware of whatever incompatibilities exist. For example, the interrupt handling JRST functions are legal in user IO mode except on the KI10, where they are restricted to kernel mode. Because of the more restricting JRST decoding in the earlier processors, the KL10 and KS10 have more functions, and they produce quite different effects when given in a KI10 or KA10 program. The matter of unassigned codes works both ways with respect to different

Figure 3.4: Extended TOPS–20 Process Table Configuration



12 Byte data; ~~stack in PUSH or POP~~; source in BLT; destination in EXTEND; effective address calculation of EXTEND destination pointer if bit 9 is 1

Previous context referencing is useful and reasonable in some instructions but inapplicable to others. There is no trap of any kind, and the effect of using the feature with an instruction to which it does not apply is simply undefined.

Applicable	*Inapplicable*
Move, XMOVEI	LUUO, MUUO
EXCH, BLT, XBLT	AOBJN, AOBJP
Half word, XHLLI	JUMP, AOJ, SOJ
Arithmetic	JSR, JSP, JSA, JRA, JRST
Boolean	PUSHJ, POPJ
Double move	XCT, PXCT
CAI, CAM	Shift-rotate
SKIP, AOS, SOS	String (except MOVSLJ)
Logical test	IO
PUSH, POP, ~~ADJSP~~	ADJSP
Byte	
MOVSLJ (extended KL10 only)	
MAP	

Note that no jumps can use previous context referencing. Even among the instructions to which such referencing is applicable, only a limited number of the sixteen possible bit combinations is useful or meaningful. Doing an effective address calculation in the previous context (selected by bit 9 or 11) makes sense only if the corresponding data access is also in the previous context (as selected by bit 10 or 12 except 11 or 12 in EXTEND). Only these combinations are permitted.

Instructions	*9*	*10*	*11*	*12*	*References in Previous Context*
General	0	1	0	0	Data
	1	1	0	0	*E*, Data
Immediate	1	–	0	0	*E* (no data access)
BLT	0	0	0	1	Source
	0	1	0	0	Destination
	0	1	0	1	Source, destination
	1	1	0	0	*E*, destination
	1	1	0	1	*E*, source, destination
XBLT	0	0	1	0	Source
	0	0	0	1	Destination
	0	0	1	1	Source, destination

WRONGS FOR KC