RW 1008

SCIENCE RESEARCH COUNCIL

ATLAS COMPUTER LABORATORY

P D P 1 5 U S E R N O T E 7

Addressing Mechanisms

issued by
R W Witty

11 April 1975

DISTRIBUTION: FRAH RB AWB PAD AHF JRG PMN
JMR DVR WDS RET RWW PB TRA
GVD JAH AJH MBK

1. SUMMARY

This paper details the PDP 15 mechanisms for calculating operand addresses and discusses some of the implications arising. The paper assumes familiarity with the principles of indirection, indexing and base-limit hardware relocation. The relevant sections of several PDP15 manuals are referenced and the paper tries to compensate for the gaps in the Digital documentation. Brief mention is made of the FR80 addressing scheme.

2. OPERAND ADDRESS CALCULATION

The algorithm by which the operand address is calculated is shown in figure 2.1. The flowchart is a 'program-like' model of the address evaluation process, and not a direct analog of the action of the addressing hardware.

← is the assignment operator

+ is the 2s complement addition operator

Δ is the concatenation operator

Subscripts on variables refer to bit subfields. The most significant bit in a word is bit 0. For example, if

PC = 274567 and INSTR = 203212

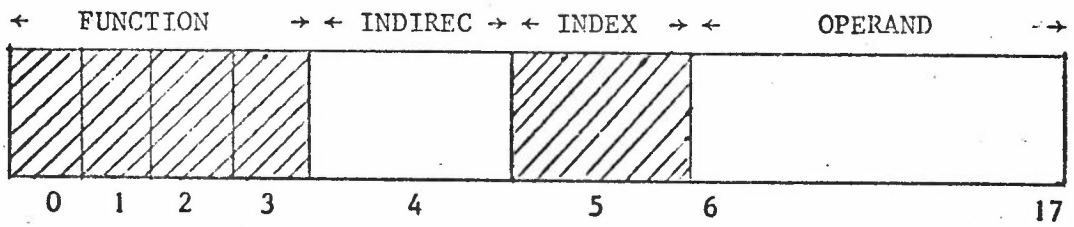
then

PC_{0-5} Δ $INSTR_{6-17}$ gives 273212

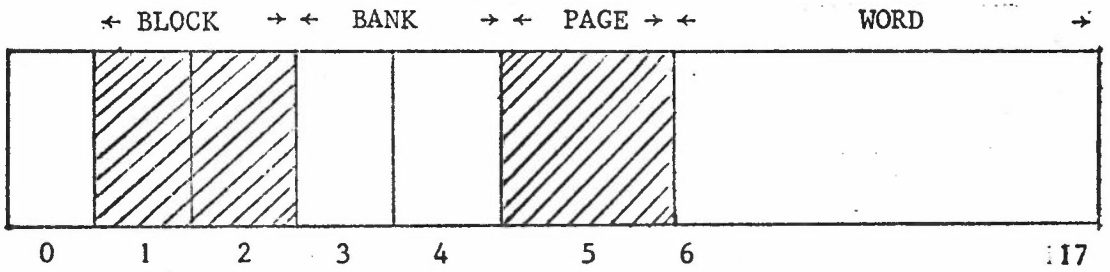
INSTR is the instruction currently being executed. BBP represents the upper bits of the program counter (PC) which give the current instruction's

block, bank and page location. A direct reference within the current block is formed when BBP is concatenated with OPAND, the lower bits of the instruction. BBP and OPAND vary with the addressing mode. RL is the hardware relocation base register. EA is the effective operand address, the address which is actually sent down the memory bus. AI is an auto-increment register. Sections 2.1 through 2.5 are useful tables when dealing with addressing at the machine code level.

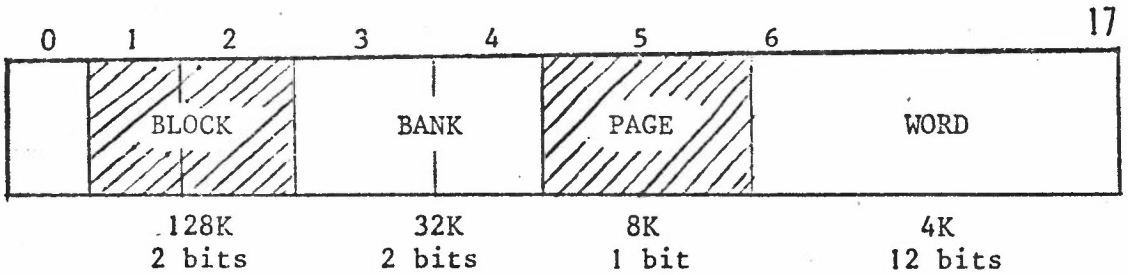
2.1 WORD AS A MEMORY REFERENCE INSTRUCTION



2.2 WORD AS AN ADDRESS



2.3 MEMORY ADDRESS RANGE



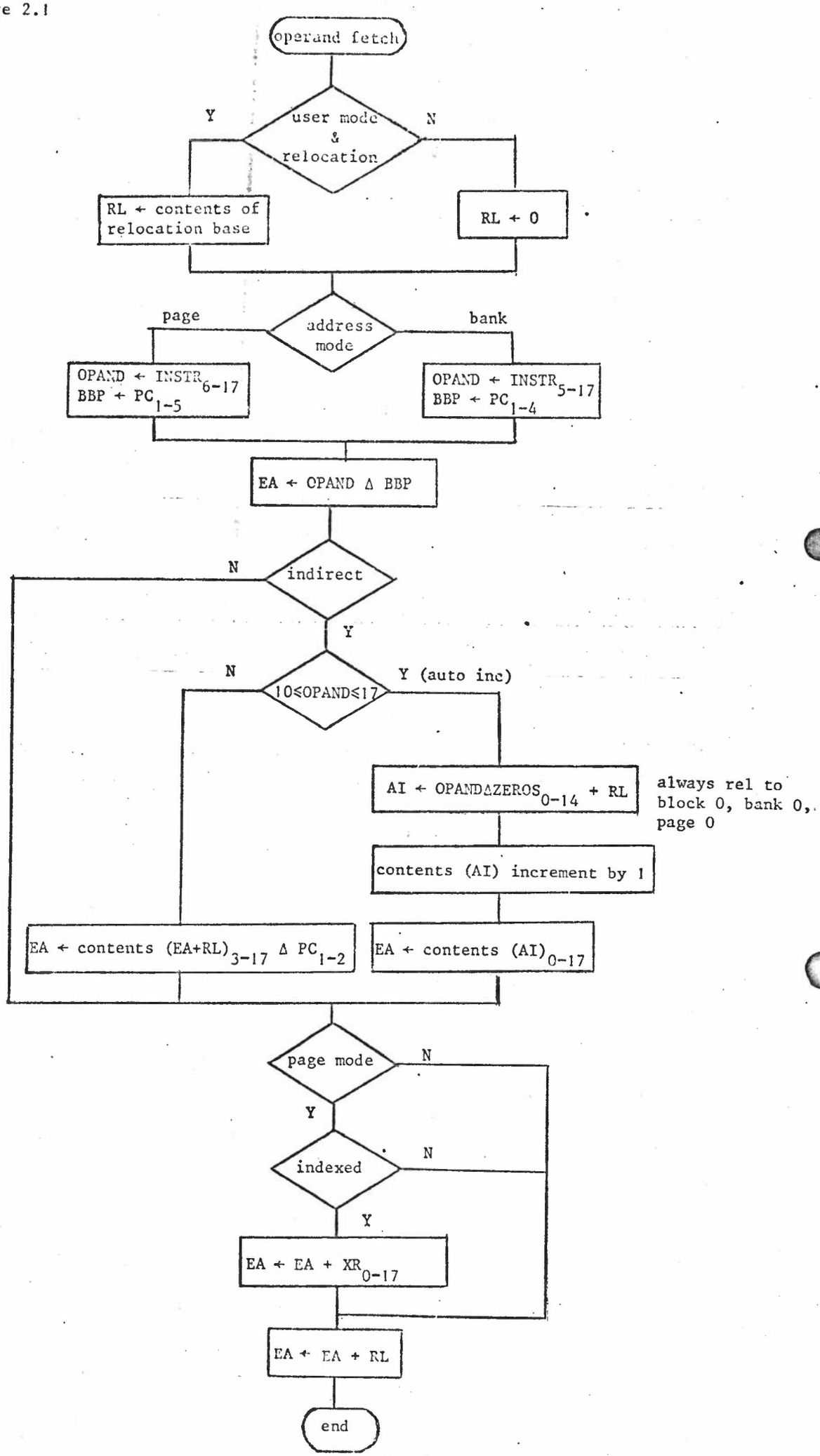
2.4 OCTAL - PAGE, BANK, BLOCK CONVERSION TABLE

LOWER		UPPER		PAGE	BANK	BLOCK
0	000000	007777	4K	0	0	0
4K	010000	017777	8K	1	0	0
8K	020000	037777	16K	0,1	1	0
16K	040000	057777	24K	0,1	2	0
24K	060000	077777	32K	0,1	3	0
32K	100000	117777	40K	0,1	0	1
40K	120000	137777	48K	0,1	1	1
48K	140000	157777	56K	0,1	2	1
56K	160000	177777	64K	0,1	3	1

2.5 ADDRESS SPACES

BITS	OCTAL RANGE	PAGES	BANKS	BLOCKS	K
12	000000-007777	1	0	0	4
13	000000-017777	2	1	0	8
15	000000-077777	8	4	1	32
17	000000-377777	32	16	4	128

Figure 2.1


operand fetch
user mode & relocation
Y
N
RL ← contents of relocation base
RL ← 0
address mode
page
bank
OPAND ← INSTR 6-17
BBP ← PC 1-5
OPAND ← INSTR 5-17
BBP ← PC 1-4
EA ← OPAND Δ BBP
indirect
N
Y
10≤OPAND≤17
N
Y (auto inc)
AI ← OPANDΔZEROS 0-14 + RL
always rel to block 0, bank 0, page 0
contents (AI) increment by 1
EA ← contents (EA+RL) 3-17 Δ PC 1-2
EA ← contents (AI) 0-17
page mode
N
Y
indexed
N
Y
EA ← EA + XR 0-17
EA ← EA + RL
end

3. ADDRESSING MODES

PAGE MODE	BANK MODE
(a) direct (4K)	(a) direct (8K)
(b) indirect (32K)	(b) indirect (32K)
(c) auto-increment (128K)	(c) auto-increment (128K)
(d) indexed (128K)	
(e) auto-increment indexed (128K)	
(f) indirect indexed (128K)	

Either mode may or may not be hardware relocated, hence there are at least

(6 page mode + 3 bank mode)*2 = 18 modes.

3.1 DIRECT ADDRESSING

Direct addressing in page (bank) mode accesses the current page (bank) ie that page (bank) containing the instruction being executed. This is guaranteed by concatenating the lower 12 (13) bits of the instruction (OPAND) with the upper 6 (5) bits of the program counter (BBP).

3.2 INDIRECT ADDRESSING

Only the lower 15 bits of an indirect address are significant, the block number being automatically taken to be the current block containing the instruction (PC_{1-2}), hence indirection enables addressing within a block (32K). Indirect jump instructions are therefore capable of switching execution between banks.

3.3 AUTO-INCREMENT REGISTERS

Auto-increment registers (AIs) are just ordinary words in core. An instruction is only deemed to be referring to an AI if its OPAND is in the range 10 through 17 and its bit 4 is set. In calculating the actual address of an AI the hardware does not include the upper bits of the PC (OPANDΔZEROS) so AIs are always locations 10 through 17 of block 0, bank 0, page 0, irrespective of the bank of the current instruction. Thus an indirect technique must be used to access words 10 through 17 from a bank other than bank 0 when bit 4 is not set ie when 10 to 17 are not treated as AIs, see PDP15 User Handbook vol 1, section 4.3.3.

In contrast to ordinary indirection, the contents of an AI are not concatenated with PC_{1-2} but are treated as a full 17 bit address, thereby allowing the AIs to access the complete 128K address space.

AIs, being ordinary core words, are subject to hardware relocation. If a program is running in user mode with hardware memory relocation operative, then that program has its own (virtual) set of AIs at absolute addresses RL+10 to RL+17, which are distinct from the monitor mode set at absolute locations 10-17, unless of course in the silly situation where $0 \leq RL \leq 17$!

3.4 INDEXING

Indexing is the two's complement addition of the 18 bit signed index register to the effective operand address. Negative values may be usefully employed in the index register. Note when indexing that even if the instruction operand (OPAND) is zero, the block, bank and page address (PC_{1-5}) is automatically added to the contents of the index register to form the effective operand address. Hence to address an absolute core location, the index register must contain the required absolute address minus the base address of the current bank and page. This assumes the program to be running in Monitor mode. If the program is running in User mode with memory relocation operative then absolute addressing is impossible unless the operating system informs the user program of the value of the relocation register (RL). This is likely to be a time varying quantity anyway.

4. GENERAL NOTES

4.1 JMP, JMS, CAL INSTRUCTIONS

A simple 'JMP' instruction has an effective address of 12 (13) bits in page (bank) mode and may only transfer control within the current page (bank), ie over a range of 4K (8K).

An indirect jump, 'JMP', has an effective address of 15 bits and may transfer control anywhere within the current block (32K). This is the way to change banks. An indexed jump, 'JMP,X', has an effective address of 17 bits and may be used to transfer control over block boundaries ie anywhere within 128K.

'JMS' instructions only save the lower 15 bits of the return address, the block number is lost. Thus simple subroutine calls must refer to routines in the same block (32K). A non-standard call and return sequence using indexed jumps must be developed if calls outside the current block are required.

Ordinary user programs cannot be greater than 32K because API, JMS and CAL instructions are used by DOS,FORTRAN and RSX. As these instructions only save 15 bit return addresses they restrict programs to within one block (32K), absolute addresses 0-32K in Monitor mode (DOS). If hardware relocation is used programs may be situated anywhere within 128K of core but their sizes must still be less than 32K if supervisor calls and interrupt handlers are to work normally (RSX).

4.2 DOS

DOS and subservient user programs run only in Monitor mode. User mode and hardware relocation are not supported by DOS. This means that user programs share the same set of auto-increment registers with DOS and system software like DDT. Beware! DOS only recognises the first physical 32K of core. The Atlas upper 32K does not 'exist' under DOS, so user programs must be less than 32K, see figure 4.1. The loader cannot place programs in upper core. Upper core is accessible from a user program for data storage only through auto-increment or indexed

addressing as a block boundary is crossed. Executing code out of upper core is dangerous because API, CAL and JMS instructions will cause DOS to return to the lower 32K only, return addresses being truncated to 15 bits. When using upper core remember not to overwrite that portion of DOS that resides around 32K, and remember that a user program in the lower 32K may be loaded anywhere in that area, making the gap between the user program and the start of upper core a variable quantity, so address upper core absolutely.

4.3 RSX

RSX employs User mode and hardware relocation, so that it can use the entire 64K of our machine. However, individual user programs (tasks) still must be less than 32K due to API and CAL only saving 15 bit return addresses, see figure 4.2.

4.4 PROGRAM COUNTER - PC

The program counter is incremented modulo 4K (8K) in page (bank) mode. This means that instruction address wraparound can occur during sequential execution. PC_{0-4} never undergo addition.

4.5 MEMORY ADDRESSING ERRORS

In User mode with the appropriate hardware available, an address outside the scope defined by the base and limit registers is trapped and control returned to the operating system. If the address is logically valid but addresses a non-existent location, then again control is returned to the system.

In Monitor mode a non-existent memory address will cause the machine to hang.

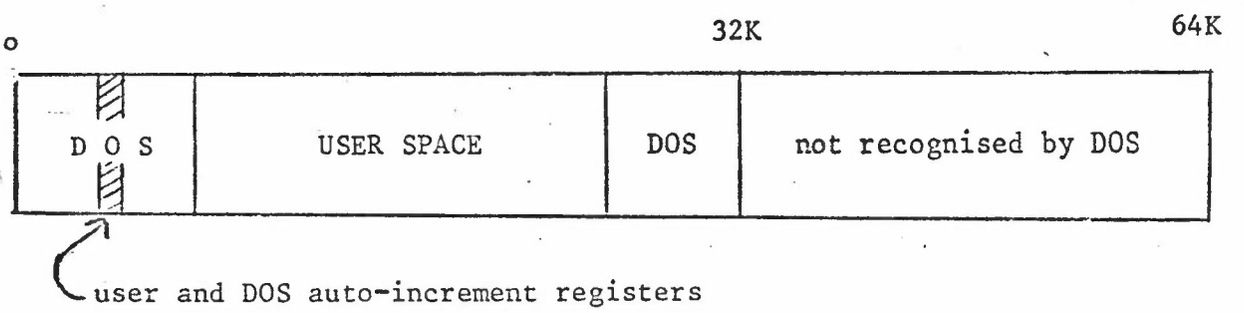
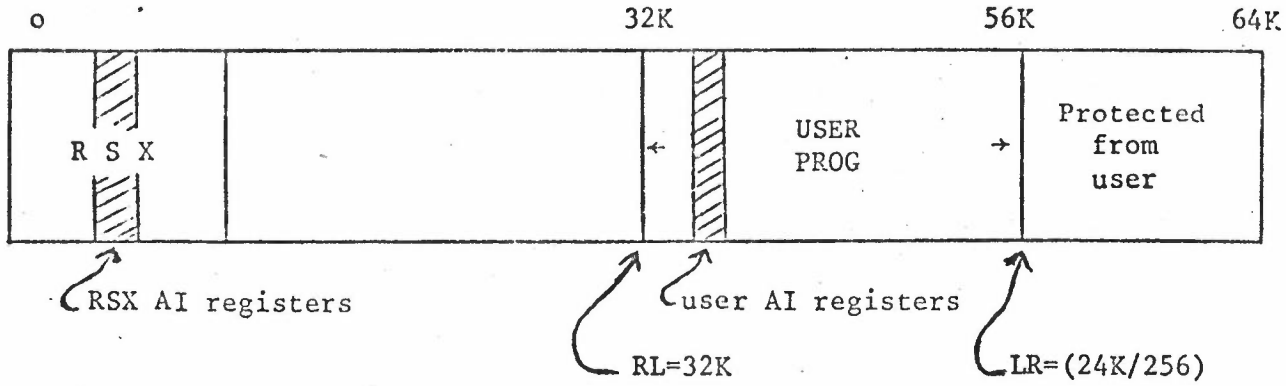


Figure 4.1 DOS CORE LAYOUT



User physical area 32K - 56K
User address space 0 - 24K
Relocation register 32K
Boundary register 24K/256 (number of 256 words in prog)

Protected from user 0 - 32K and 56 - 64K

Note separate auto-increment registers

Figure 4.2 RSX CORE LAYOUT (RELOCATION)

4.6 HARDWARE RELOCATION

A program running in User mode is allocated an integral number of contiguous chunks of 256 words each. The absolute address of the start of this area is the relocation constant (RL) which is placed in the relocation base register. The limit or boundary register (LR) contains the number of chunks of core allocated to the program ie program size /256. If the program size is say 24K, located at absolute locations 32K to 56K then the user's address space is 0 - 24K, the RL = 32K, the LR = 24K/256, see Figure 4.2. The action of the hardware may be formulated as

```
IF ((EA/256)-LR)≥0  THEN BEGIN

                    violation;
                    software interrupt;
                    END
                    ELSE BEGIN

                    EA:=EA+RL;
                    memory fetch;
                    END;
```

WARNING The description given in PDP15 Systems Reference Manual is wrong. The User's Handbook vol 1 is correct.

5. PDP15 EXTENDED MODE

There exist two IOTs (see Digital's extended memory test program)

EEM = 707702 / enter extended mode

LEM = 707704 / leave extended mode

which seem to be accepted by the machine but which have no effect on the addressing. It seems as though EM is permanently hardwired as 'entered'. The engineers do not know - the manual does not say. It might be a hangover from the PDP9. It would indicate that indirect addresses are always taken as 15 bits - contrast with the FR80, section 6.

6. FR80 ADDRESSING

Briefly the FR80 addressing is the same as the PDP15 except for extended memory mode. When extended mode is disabled, indirect addresses are 13 bits long. When enabled, indirect addresses are 16 bits long, thus allowing indirect addressing of 64K. It would appear that extended mode on the PDP15 is 15 bits and that the FR80 is modified to include the 16th bit that part of the address which on the PDP15 is used to save the User/Monitor mode setting. The FR80 does not have hardware relocation and User/Monitor modes. The FR80 normally runs in extended memory mode.

7. REFERENCES

This paper assumes familiarity with the following items

PDP15 SYSTEMS REFERENCE MANUAL

(a) Chapter 4, Memory Organisation

(b) Chapter 6, Addressing

(c) Chapter 8, Internal Options Instruction Set - KM15,KT15

PDP15 USER HANDBOOK VOL 1

(d) Chapter 4, Addressing Features

III15 PROGRAMMERS REFERENCE MANUAL, PUB NO 90437

(e) Chapter 2, Memory Addressing

1b