74CM373-0141

digital INTEROFFICE MEMORANDUM

TO: Dick Clayton
Jega Arulpragasam

DATE: September 13, 1974

FROM: Craig Mudge

DEPT: 11 Engineering

EXT: 5064 LOC: 5-5

SUBJ: Summary of 11/VAX Architecture.

Enclosed is the report you requested. Detailed schedule for VAX is the following:

Overall Summary	9/13
Architectural Spec	9/27
Software & other Implications	10/15
Effect of Implementation on 11/44	9/27

CC:

Engineering Mgrs.
Gordon Bell
Roger Cady
Dick Clayton
Bruce Delagi
Bill Demmer
Robin Frith
Andy Knowles
Phil Laut
Al Sharon
Steve Teicher

DRAGON Engrs.
Sas Durvasula
Bob Giggi
Bob Gray
Kent Griggs
Dave Ives
John Levy

Product Line Managers
Irwin Jacobs
Ed Kramer
Bill Long
Julius Marcus
Brad Vachon

Software
Ron Brender
Dave Cutler
Frank Hassett
Pete Van Roekens
Garth Wolfendale
Denny Pavlock

Craig Mudge
9/13/74

SUMMARY OF 11/VAX ARCHITECTURE

I. DESIGN GOALS

1. Implementable over a range.

The architecture must be efficiently implementable over a cost and performance range. The range should span from an 11/05-type-cost machine to an 11/55-successor-type-performance machine. In addition the hooks necessary on the Basic Machine should be minimal - no more than a few IC's. The option itself may exceed the current KT in cost.

2. A substantial increase in virtual address.

16 million 4 billion

The new address length should be between 24 and 32 bits, not just an extra bit or two over today's 16 bits.

16 million 4 billion

3. Use known art.

Segmentation, whose strengths and limitations are known, should be used. New methods, domains and capabilities, for example, should not be explored.

4. Compatible with today's PDP-11.

a) Existing user programs must run unmodified.
b) Existing user subroutines must be callable from new programs which exploit extended addressing.
c) Existing system code, except for the code that loads the KT11 mapping registers, must run unmodified.
d) The scheme must be compatible with the KT11 memory management unit.

? via assembler

The goal of running most system code as well as all user code is unusually strong.

5. No loss of performance.

a) I-stream
Within a loop, the number of I-stream bits passed must be no more than in today's 11. Extra I-stream bits for loop set up are allowed.

b) Address translation
No more time added above conventional dynamic address translation schemes.

6. Flexible name space (program space) management.

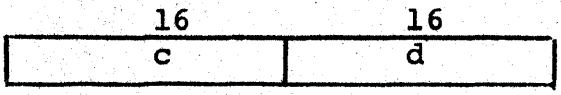
The following programming needs must be met:
a) Program modularity.
b) Varying-size data structures.
c) Protection.
d) Sharing, without the conflict which derives from the 8-segment KT11.

II. THE 11/VAX ARCHITECTURE

1. Extended addresses.

In today's 11 a processor generates a 16-bit virtual address. A register always takes part in this address calculation. For example, in the instruction CLR (R4) + , the address mode is 2 (auto-increment) and the contents of R4 is the operand address. In MOV B, -(SP), the source operand addressing is by mode 6, register 7 and the destination operand addressing is by mode 4, register 6.

11/VAX exploits this fact that a general register always takes part in address formation and simply extends each register to 32 bits. The 32-bit address has two components:



c= chapter number
d= displacement with chapter

A single chapter is exactly equivalent in size and structure to the 64K bytes of today's 11 virtual address space.

The 16 bit register extension of register Ri is called RiX. The definition of the address mode is as in today's 11.

Now consider the 11/VAX instructions needed to manipulate 32-bit addresses. The instructions to manipulate the d part of the address (c,d) are exactly today's 11 instructions. New instructions to load and store RiX, i.e., load and store chapter number, have been added. There are also new instructions to do interchapter jumps (JMPX), and JSRX and RTSX for subroutine invocation and return.

The virtual address space presented to the programmer is a classic segmented[1] address space: 2^{16} chapters of 2^{16} bytes. The two component addressing will be exploited by programmers: logically related entities will be grouped and assigned separate chapter numbers. For example, a separate chapter number could be assigned to each of a) a matrix, b) a row of a large matrix, c) a large main program, d) a large subroutine, and e) a group of subroutines e.g., the FORTRAN object time system. The chapter is thus the logical unit of allocation for modularity, sharing, and protection in the programmer's logical address space.

Address specification is efficient. Full 32-bit addresses will appear in the instruction stream much less frequently than 16-bit addresses, which, in turn, appear much less frequently than 3-bit register addresses (specifying address-holding registers).

---

1. I have used the term chapter instead of segment because KT11 documentation has sometimes used the terms segment and page interchangeably.

2. Mapping.

Every address generated by the processor is mapped to a physical address. Map tables in memory define the mapping for each process, or task, known to the operating system. See Figure 1. Suppose process #15 executes the instruction INC (R3) and that R3 holds [c=4 | d=41007] . Then Figure 2 shows the address translation.

This address translation takes 4 memory references. Because it is a serial delay which must occur before the processor can issue a memory reference, it must be speeded up(to about 150 nsec. on an 11/44 type of machine). Thus a "DAT box" for dynamic address translation will be used in each implementation of 11/VAX. This will hold a subset of the map table in fast registers. The goal of a DAT box is to make this subset the most frequently used parts of the total map.

A range of implementation of DAT is possible: from a one-register implementation(slow, cheap - 11/05) to one that has many registers, associative look-up, and elaborate replacement ~~allowing~~ algorithms (fast, expensive -11/95) such as the "translation buffer memory" on the S/370.

III. COMPATIBILITY

To ensure that user programs will run unmodified, a spare PS bit, PS <08>, is used to indicate X or non-X mode. A program written for today's machine does not know about RiX. The mode bit when zero forces the program to run as it was intended, i.e., as a one-chapter program, by using R7X (PCX) as the value for R1X through R6X. With this mechanism an extended program may call a "16-bit" subroutine by a normal JSR, ~~as follows~~.

The call itself is issued from a chapter whose page table is identical with that required by the subroutine. This is possible since by definition the subroutine was written to exist in a 16-bit VAS, and there is no restriction against different chapters having identical pages.

Note that with a normal JSR the PCX is not stacked and the called program, therefore, faces no ambiguity.

The 11/VAX working notes (Version 1, 5/2/74) gives full details of how the X-mode bit, together with the general mapping concept works for all calls, including examples with the appropriate "linking code" (a couple of instructions), where necessary.

To ensure that our compatibility goals for system programs are met, another spare bit, PS <09>, is used to control the stacking and unstacking of PCX on interrupts and RTI, return from interupt. All interrupts are returned to Process Ø, Chapter Ø. The interrupt vector here, in particular PS <Ø9> , controls the stacking and unstacking of PCX.

This allows the placing of a current 16-bit supervisor in P_0C_0, or allows a Super Supervisor, OSX, to reside here and forward interruptions to several different "16-bit" Supervisors existing simultaneously in different individual chapters.

Reference is again made to the Working Notes for full details of how this actually works. Note that Version 2 of the Working Notes eliminates a deficiency/limitation in this area, which had been caused by attempting to get by with a single mode bit. Version 2 recognizes that adding a second mode bit to remove that deficiency is a trade-off with a high payback.

IV. WEAKNESSES

If we were designing PDP next, i.e., designing the size and structure of a 32-bit virtual address space from scratch, we would not propose the chapter scheme. I would propose a classic segmentation scheme, but the segment size would be 24, not 16. Other less-than-ideal properties of the chapter scheme, which derive from our strict compatibility goal are:

a) Access rights appear at two levels - at the chapter level and the page level - the latter is not only redundant, but is the wrong place because a page is the unit of allocation in the physical space.

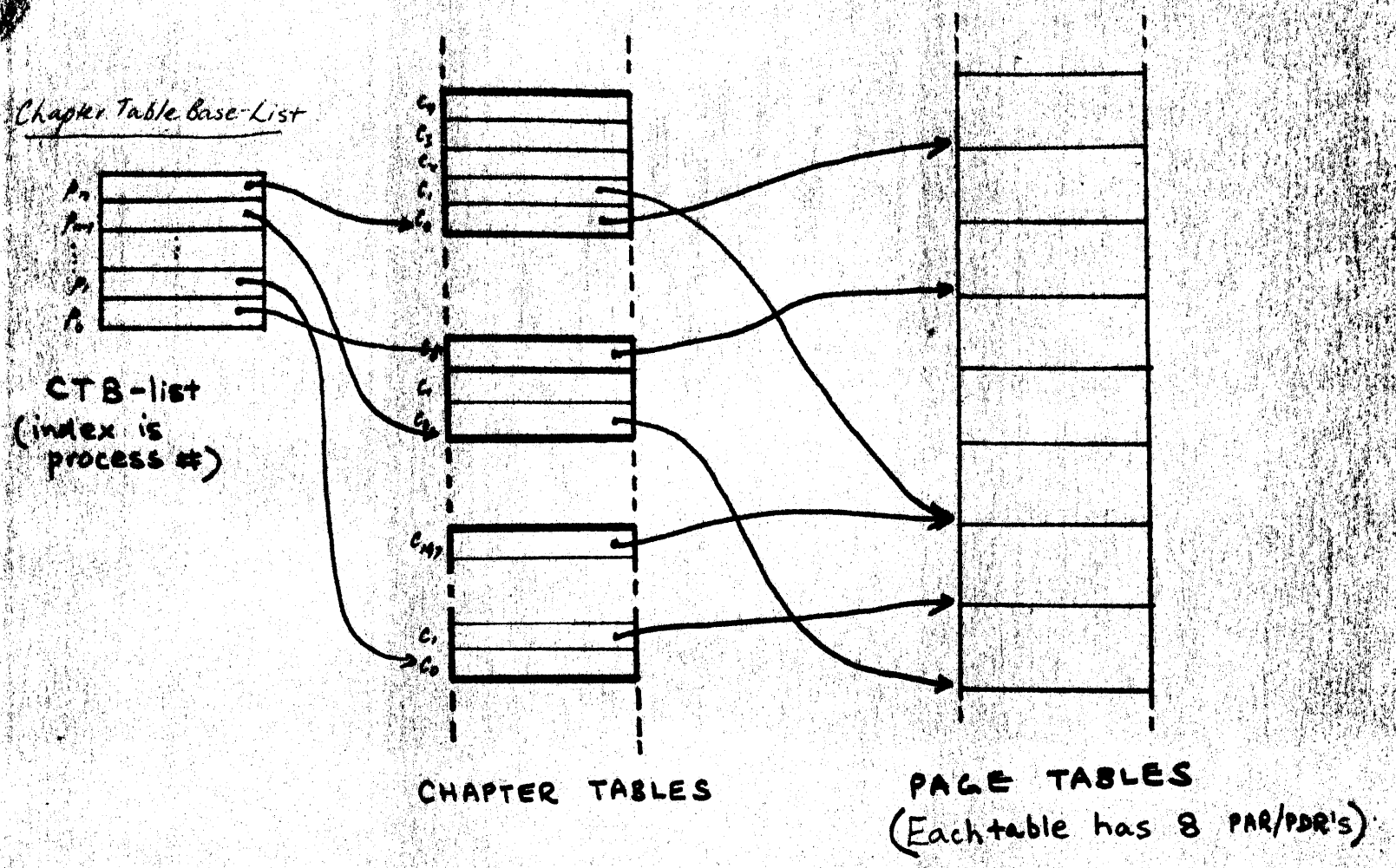
b) The page size dictated by the KT11 is 4K words, generally accepted to be too large.

c) 32-bit index words and 32-bit indirect addresses in memory are not provided.

The only one of concern is the KT11-derived page size. Operating systems which support 11/VAX on large systems will require hardware assistance for physical memory mangement. In that case the page size should be changed. The RSX11-D group claim that that part of the KT11 compatibility could be sacrificed at little cost. The other part of KT11 compatibility is the Kernel/User privilege structure. We could not change that without a large rewriting effort. We are currently getting a new set of figures to quantify "little" and "large".

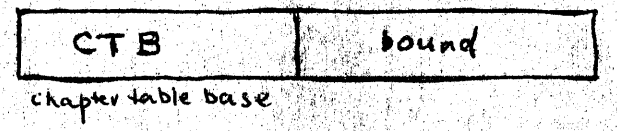
FIG 1

MAP TABLES IN MEMORY

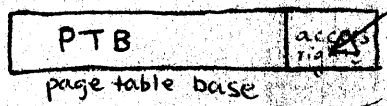


Format of table entries:

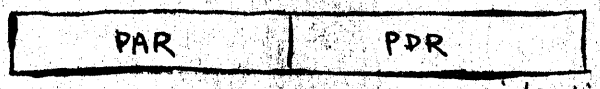
CTB-list entry



chapter table entry



page table entry

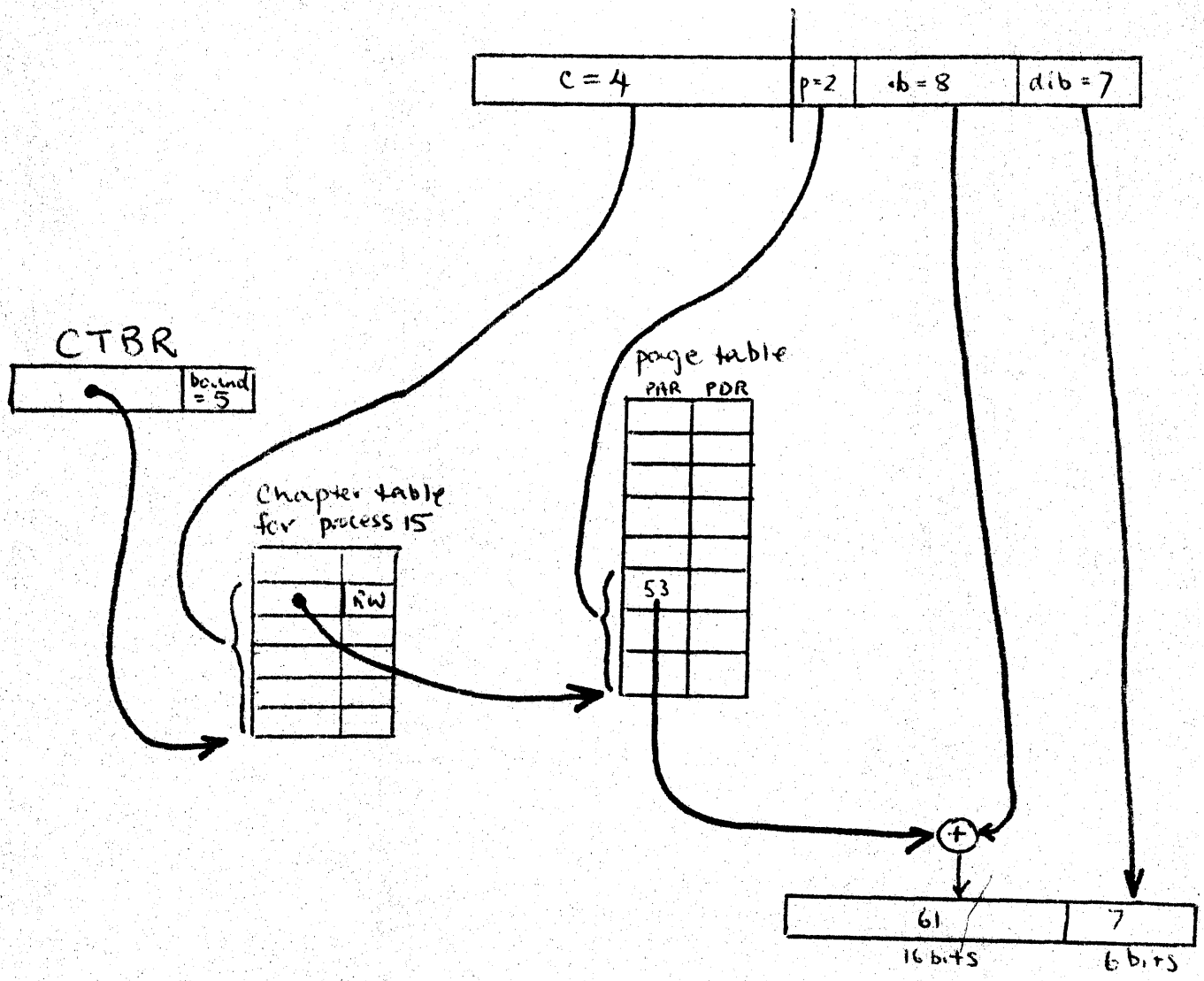


JCM

FIG 2.

ADDRESS TRANSLATION EXAMPLE

32-bit Virtual to 22-bit physical



Chapter table lookup | Page table lookup and translate (as for KT-11)

JCM
9/13/74

Appendix 1

Possible Implementation on a Medium Scale 11.

It will be noted that each RiX corresponds to a single chapter which will each have a set of Page Tables corresponding to it, identical in form to todays set of KT11 registers.

Therefore an implementation could be such that the KTX option itself would hold RiX and a limited set of KT type Register sets. Instead of 6 sets for Kernel/Supervisor/User times I/D space, we would have at most 16 sets (probably only 9) which would be Kernel/User times one for each of the RiX. (9 if the Kernel was essentially single chapter).

The "hooks" required can be seen to be only the provision of the Register used on each memory reference (4 lines: 8 registers plus Kernel/User) and the mechanism for setting RiX on a Load Address instruction. The latter hook can also be made simple if integrated at initial design time.

With these hooks accessing the required page table is clearly of the same order of speed as for the present KT. Further, replacement of the page tables could be driven from the KTX itself and would run "blinding fast" on a high bandwidth 32-bit wide memory bus, as loading/storing would be from/to contiguous memory locations. In particular, the DRAGON bus would be appropriate. The extra cost of the hooks on the basic DRAGON is estimated to be <$25, if it is specifically designed to shift the cost burden on to the KTX option itself, wherever possible.

/br