

# Memo

Writings  
17+18

Found.: Compat.

What the machine is?

Cpu, fp, Access via VAX

→ Arithmetic Group: When

MJ Did memo go to invites,  
Conklin on VAXA?

## Groups

Todo: get Fort people  
to us

VAXA: 28.

- Mem. Mgmt. ??

- Multiprocessor:

→ Process Group 11-13, 16, 26,  
28,

→ Authentic. - 20, 24

→ I/O - 23

→ Compatibility 17, 58, (5)?  
+Phaseover

- Small systems Validation.

3/26

PMS issues  
(Stecher/Bell)

- Strecker Model. - for balance.
- Kio vs Pio vs Cio vs C.m's
- Multi-Pc
- multi-C - *computer modules*
- Nets including Front ends.
- model(s) range
- S (multi-drop)

Implementation  
(Rothman)

- Components (eg. CCD)
- range, ~~or~~ #, perf.

PSP  
(Stecher)

- added state ✓
- encoding + extra-word ✓
- new instruction + Data-types. ✓ (dec, string, call,
- Process + context switch
- Memory Mgmt/Mapping.
- I/O

Op. Sys. - what are primitives?  
(Lorig)

- structure across models.
- user environment across fens.
- *What is structure of current op. sys.? what do we want?*

Software - policies vis vis compatibility (HL, convention, macros)  
(Wade)

gen  
Call to buy Mgrs for ideas.

3/25

ISP+ Biki, BCT

Arch. Alt's (ISP)

CM-Seq.

St/W

S/hinan

Cult. - 11

Radical - Jega - 11 - Single for interpreter + Map for speed.

" - Jega - 10+11

On Reassignment on codes - cost/benefit - issue

Bull/stock

Process

PMS

Range issue

Nets.

Pio vs Kid<sup>(Pio)</sup>

vs Cio<sup>outside</sup>

vs Kid<sup>(Comm. speed)</sup>

Multidrop?

Software

Def. of user interface -

Sys. impl. language.

Cost to convert, x.

Impl.

Rothman - (on what basis?) - small ~ large

CCN's / Bubbles are inevitable; must be automated.

Need some assumptions about range - and bus -

Use

Transaction processing - speed. - Need measures from (Turner) on 10 vs 11

Rothman  
Denner

3/25

77 Boggly - CMudge.

lots of upgr.

lots of local storage (context)

internal Bus structure

gds

DSP enhances

I/O Process

PMS

mult-PC

~~3/25 Mtg Agenda~~

~~8:30 - 9:30 JB - Mem. model + & constants~~

~~9:30 - Doc. Control~~

~~10: Mudge impl.~~

~~10:30 Demmer other~~

~~11: Mel. Conqet.~~

D. ALTERNATIVES CONSIDERED (Contd)

2. Expansion at Northbrook - It is possible to continue the approach of obtaining satellite space around the present location at Northbrook. This would be the low cost solution for the short-term if only the tangible costs of real estate are considered. These cost savings would occur primarily because we would not be carrying any excess space. It appears that a better approach is to provide for at least two and one half to three years growth in a new facility. The Rolling Meadows facility does this, Northbrook would not.

Operationally, separate office buildings housing different functions does not favor operation of a coordinated field team. The Team Concept of field management has been actively pushed by Ted Johnson. We should not inhibit this by short-term considerations in our real estate planning.

3. Design Alternatives

First Proposal: Office Building - 42,168 sq.ft.  
Logistics Depot - 12,800 sq.ft.  
Training Center - 12,800 sq.ft.

Advantages: less expensive than multi-story by approximately \$300 K. The opportunity cost of the higher land usage has not been deducted from this (\$72K)  
  
easy for expansion

Disadvantages: less visibility than multi-story.  
  
higher land usage  
  
longer walking distances

Second Proposal: One 3 story building of 70,500 sq.ft.

Advantages: higher visibility  
  
lower land usage  
  
expandable  
  
short walking distances and good communications.  
  
good land usage

To do

Arch. Size

Compat. & extend. ~~address.~~

length with xlate

15 May

ISP (8+8 or 16 registers)?

- HLL - ~~strings~~ strings - convert
- Vector
- Lists
- State mapping into registers
- EC operation
- Indefinite Value (eg -1)

Modes why...

- Call - details will get done at  $inpl(t)$  + check arglist use.
- Exchange inst. this pair in wrong place
- Xvs useful?

Mem. reqt.

Complete it  
Problem of page size? is contiguous solve it.

How efficient is ISP? Can we use dual and address length to shorter streams?

Process starter

- primitives
- structures.
- interrupts & connection to process.
- context switching mechanism
- Clock.

Implementations

- Multi-Pc — unibus  
                          other bus
- Which machine.
- alignment on words?
- Purity data in Memory

How much does  
Sep. kbd add?  
higher up take?  
Portable & cheap?

I/O

- mapping of unibus
- " " " " into large bus.
- I/O instructions:
  - 1 instruction interrupt. BLX i/o
  - Comm.

Software (ground) - Larry - Wolcott M.

- I.L.
- Conventions for 2
- which op. sys. + utilities
- Simulator

Concerns

- Is it implementable at high speed
- " " " " low end

→ Get statement from Breadboard re Multi Pc's

Overall Structure

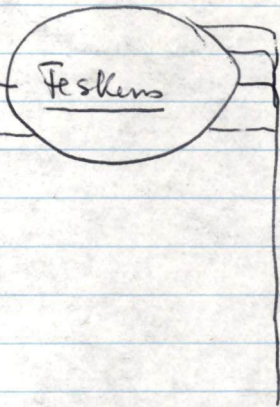
- Serial Bus.
- Comm in general.

Interface to Sp. Soft

Emul. for unimplemented instrns

Tuesday

- ISP notations
- Status of Questions
- VA mechanism
- Structural control
- Fields.
- Call/Return
- Group structures.
- Assembler Syntax.



Status

**digital**

## INTEROFFICE MEMORANDUM

0278

TO: Distribution

DATE: October 20, 1971

FROM: Peter van Roekens

DEPT: Programming

SUBJ: OS/45 Proposal Meeting

There will be a meeting on October 28 at 9:30 a.m. in conference room 12-2 to discuss the attached proposal.

It is being circulated for your review and comments.

Attachment: Proposal

Distribution

L. Wade	P. Conklin
H. Spencer	J. Mittell
D. Stone	OS/45 Group
B. Delagi	
R. Clayton	CC: A. Ryder
R. Frith	D. Schroeder
J. Lombardo	G. Thissell
A. Devault	PDP-11 Coordinating Committee



I. Introduction

On 17 Sept 71 Dick Clayton and Robin Frith presented their views on OS/45 to the OS/45 group.

Since that time we have reviewed the notes of the 17 Sept 71 meeting, added new inputs, and have investigated competitive systems. As a result, we have begun to acquire a bias as to the organization of OS/45. This paper presents the market orientation that has resulted from this bias and details the most current definition of OS/45.

## II. Programmers Overview of the PDP-11/45

The system programmer contemplating the design of an operating system for the 11/45 has two classes of problems to resolve:

- 1) the price range of possible configurations, and
- 2) several new hardware capabilities

The 11/45 has a remarkable range of potential prices: \$20,000-\$300,000. We cannot ignore the low end because we can expect competition in this range from IBM's first mini-computer, the System/7. At the upper end, even though our price/performance ratio humbles our competitors, we must deal with the IBM 1130 and 1800 and their immense software library. In addition to the challenge of devising a system which has upward compatibility over a price range which varies by an order of magnitude, the systems designer must contend with three new hardware options:

- \*Memory hierarchies (memories with different speeds);
- \*Independence of instruction and data space, and
- \*Segmentation.

Complete understanding of how to properly use these features has barely emerged from a research environment, yet we must make intelligent use of them in a production system.

To provide reasonable solution to these challenges will require a design phase pursued to an unusual depth of detail; else we run the risk of rendering the new hardware features either unuseable or not cost effective. Providing a software system whose facilities compliment those of the machine itself will depend on development of a design which demonstrates we have indeed mastered the requirements of configuration flexibility and innovative hardware.

### III. The 11/45 Marketplace - Developing a Workable Image of our Customers

A designer of any product must have prior to any design activity a distinct image of the individuals to whom he expects to sell his product. The data processing marketplace has five identifiable concentrations which reflect market needs\*.

- 1) Real Time
- 2) Scientific Batch
- 3) Time Sharing
- 4) Commercial Batch
- 5) Number Crunching

Let us eliminate item 5) from consideration immediately as inappropriate to the 11/45. The remaining four items represent the order of market priorities as specified by Dick Clayton and Robin Frith during our 17 Sept 71 meeting. During the same meeting Dick Clayton specified the following framework within which we should define OS/45:

- 1) Software support for the floating point unit and the segmentation unit should exist by July 1972.
- 2) We should announce OS/45 by June 1972 and deliver it during the second quarter of calendar year 1973.
- 3) OS/45 should unify PDP-11 software.

In addition, we have assumed that OS/45 will consume between 18 and 25 man-years of effort. Using the assumed manpower estimates it does not seem possible to attempt to satisfy the needs of the time sharing or commercial batch marketplace.

In commercial batch, DEC must compete directly against IBM and in a marketplace where IBM has no peer. We simply do not have the time or resources to design and implement an operating system for the 11/45 that would compete effectively against IBM's offerings. And even if we could produce such a system, does DEC presently have the sales and system force necessary to sell and service the commercial market? Current inputs indicate in the negative, and, hence, we recommend rejection of orienting OS/45 toward commercial batch.

---

\*Identifiable in the sense that their overlap does not result in a complete merging of the end user needs.

From the above we can extract the following composite of the OS/45 customer population:

He either already uses or has under consideration an IBM System/7, 1130, or 1800. His application requires only a subset of IBM's software for these machines. And finally, the existence of competitive equipment which has between three and five times the cost performance of the equivalent IBM system provides our hypothetical customer with sufficient reason not to remain with or choose IBM.

Of course, the computer marketplace does not exist entirely under the aegis of IBM and DEC, but we contend that if we can produce a software system which significantly impacts System/7, 1130, and 1800 sales, then we will have more than nullified the offerings of XDS, Hewlett-Packard, Data General, SEL, EMR, Varian, and Interdata all of whom offer real-time systems in the 16 bit class.

All these facts lead us to one inevitable conclusion: make sure OS/45 provides a set of scientific batch facilities that makes it possible for us to capture 10% of the potential 1130 market.

This brings us to the top priority on the list - Real Time. As with the 1130, we will confine ourselves to IBM. In real time, IBM offers the system/7, the 1800, and the 360/44. The 360/44 really belongs in a different class of equipment (Decsystem 10) so we will concentrate on System/7 and the 1800. First the 1800.

IBM has a total of 563 1800's installed or on order at an average system price of \$300,000. The 1800 hardware represents little or no competition for the 11/45. IBM does, however, tend to overwhelm their competition with software, which includes two systems (MPX and TSX) capable of running, simultaneously, real time in the foreground and batch in the background. As with the 1130, even though we can't hope to provide all the software IBM does, the task of selecting a competitive subset appears achievable. And even 10% of a \$200,000,000 market would handsomely repay our investment (and hopefully we would capture much more than 10% of this market).

Unlike the 1800, System/7 represents an insidious rather than a direct challenge to the 11/45. Only the smallest System/7 configurations offer any competition for the 11/45. In most of these small configurations, we suspect, an 11/20 would provide a more cost effective solution. Regardless of how DEC counters the threat of System/7, counter-it it must. IBMs track record for customer loyalty provides little comfort to the DEC salesman attempting to replace a System/7\* with a PDP-11. Once in the door with a system/7, add-on equipment and growth to larger systems will go to IBM by default; System/7 will lead to 1130's and 1800's. (Indeed, the initial System/7 marketing thrust practically requires that the user already have an 1130, 1800 or 360).

To compete with IBM in the real time market we suggest that OS/45 provide a real time capability that spans the entire price range of the 11/45 with upward compatibility of object programs provided across the entire range of possible configurations.

\* We should not delude ourselves regarding the potential of System/7. IBM has consistently enhanced products to meet the threat of competition - How about segmentation registers on System/7?

A time sharing orientation also seems unachievable on schedule with available resources. The pursuit of an 11/45 time sharing system also seems unadvisable if DEC decides to produce a small version of the 10. Any attempt to provide a multi-language time sharing system for the 11/45 (we already have a single language system in RSTS) runs the risk of colliding with the introduction of the small 10. Software production costs continue to rise and hardware costs continue to decline. And we have little assurance that the total cost of an 11/45 time sharing system (hardware plus software) will not exceed the total cost of the small 10.

The existence of RSTS and the relatively modest cost involved in altering RSTS to take advantage of the FPP and segmentation provides additional reason for avoiding a time sharing orientation for OS/45. If the future of time sharing depends on applications packages, and if BASIC Plus has sufficient language constructs to build most applications packages suited to the 11/45, then what incremental gain can we expect by producing a multi-language system? We cannot answer this question factually, but doubt that the incremental gain can offset the software development costs. Thus, we recommend rejection of a multi-language time sharing organization for OS/45.

We now arrive at scientific batch. We define this as batch streaming of FORTRAN programs and cite the 1130 Disk Monitor as the type of facility against which we can expect to compete.

IBM has installed or on order 3800 1130s at an average price of \$90,000.\* It would not surprise us if the 11/45 has a cost/performance ratio five times that of the equivalent 1130.

Dick Clayton projects 1000 11/45 sales over the life of the system. If we can capture 10% of IBM's 1130s as a result of providing a competitive scientific batch system, then we would help him achieve 1/3 of his goal.

Furthermore, the 1130 customer does not need the practically dimensionless volumes of software of the commercial market. These users rely on FORTRAN heavily (making it possible for him to convert at modest cost). And the size and cost of the 1130 itself places a practical limit on type of applications it can support.

---

\*September 1971 issue of Computer and Automation.

In summary, marketing will have support for the 11/45 FPP and segmentation by July '72, but at the cost of some additional software proliferation; they will have an announceable definition of OS/45 by June '72, and the announced OS/45 will take significant steps toward unifying PDP-11 software.

Let us return now to a more explicit definition of OS/45.

#### IV. Satisfying Marketing's Requirements

Dick Clayton and Robin Frith specified three requirements for OS/45:

- 1) Software support for FPP & Segmentation Unit by July 1972.
- 2) Announcement of OS/45 by June '72 for delivery during the second quarter of 1973.
- 3) Unification of PDP-11 software.

Before discussing details of OS/45 (in section III we established a customer profile; we have yet to describe how we intend to satisfy their requirements) let's examine each of these points and how we can satisfy them.

- 1) Local modification to existing software represents the most reasonable approach to meeting this requirement. RSTS can at modest cost make use of both the Segmentation Unit and the FPP. DOS plans also exist to make use of the segmentation unit.

Attempting to rush the design and implementation of the OS/45 Kernel in order to permit DOS & RSTS to convert in time to meet the July '72 deadline seems unwarranted; such haste will jeopardize both the July '72 date and the consistency and coherence of OS/45 over the longer term. Of course, by making 11/45 oriented modifications to DOS and RSTS, we pay the price of continuing the proliferation of software systems for the 11 line.

- 2) Without a doubt the OS/45 group will have a system defined for announcement by June '72, but the scope of the system depends on available resources (Dick Clayton has already specified delivery requirements)
- 3) Initially the objective of unifying PDP-11 software will not happen. DOS and RSTS will evolve independently, and it does not seem advisable to attempt to prevent this.

But as we will describe shortly, OS/45 as it evolves will make every attempt to reclaim as much existing 11 software as possible. The OS/45 design will provide users with a system covering a broad range of configurations, programming facilities for real time and batch, and will disrupt existing user interfaces only where absolutely essential.



We have identified our users in Section III. They have real time requirements and scientific batch requirements. At the low end we must block the purchase of a System/7; at the high end we must overcome the presumed user benefits of IBM's vast software library.

We believe we can satisfy these requirements, by offering a system which provides upward compatibility across the entire price range which is available on the 11/45 (20,000-300,000). This IBM cannot do; movement within System/7, 1800, and 1130 requires a conversion effort; OS/45 will not.

Now, it turns out, that Segmentation provides an efficient hardware mechanism for implementing a system which can cover the configuration range under discussion. With segmentation hardware and a set of software standards, we can specify a cascade of hardware configurations each of which requires additional hardware, in order to acquire more elaborate services; all the while we guarantee complete upward compatibility. The success of this approach depends on a careful definition of the user virtual machine.

Basically this means that the user of an OS/45 system has a well defined set of facilities he may use. These facilities consist of a subset of the PDP-11 instruction set and a collection of service routines. As configurations grow in complexity, the set of services expands correspondingly, but we always provide complete upward compatibility. This scheme implies that every OS/45 configuration operates with a set of supervisory code. This code, of course, will vary considerably on different configuration classes and in every case appears transparent to the user. Let's examine some possible configurations:\*

Hardware

- 1) 11/45\*\*  
4K-12K  
TTY

Software

Foreground only.

Small systems suffer from the lack of adequate program preparation facilities. If you can prepare your programs on a larger system often 4K suffices to meet the needs of the application. IBM has solved this problem for System/7 by providing host preparation facilities on larger equipment (1130, 1800, 360). With this technique they provide a Macro system called MSP/7. We see no reason why we cannot do the same. Indeed, if we intend to meet the threat of System/7, host preparation facilities seem essential.

\*These represent examples of possible configurations; the reader should not accept them literally.

\*\*We intend to investigate full 11/20 compatibility and issue a memo describing our conclusions.

2) Hardware  
11/45  
12K-16K  
TTY

Software  
Disk, tape, etc.  
DOS subset  
INDAC/11  
Overlay Facilities  
Fortran  
Host preparation eliminated  
Foreground only.

3) Hardware  
11/45  
16K-24K  
TTY

Disk, tape  
Segmentation

Software  
Same as 2, plus:  
Foreground single background stream. System  
maintains complete isolation between foreground  
and background.  
Foreground and background operate in fixed  
partitions.

4) Hardware  
Same hardware as  
3) but with 28K

Software  
same as 3) plus  
support of background  
jobs whose size ex-  
ceeds that of physically  
available core.

5) Hardware  
same as 3 but with 32K

Software  
4) plus  
shared code  
Multiple background jobs.  
Index Sequential file system.

6) Hardware  
5 but with 40K

Software  
5) plus  
Multiple Foreground jobs (Individually protected).

Once we define the basic user virtual machine we can determine exactly which particular configuration classes available resources permit us to produce. Furthermore, with well defined configuration classes, the product manager has available to him a shopping list that enables him to make cost trade-offs far more reasonably than he can at present. It also makes the programming department more aware of the incremental costs involved as you move up the scale in system complexity.

VI. Summary and Conclusions

If DEC continues to grow it must eventually increase its business at the expense of IBM. To accomplish this traditionally unaccomplishable feat, we have suggested a software plan for OS/45 which confronts IBM where they appear most vulnerable - real time and scientific batch.

IBM, with present offerings, provides zero hardware competition for the 11/45. To counter IBM's software libraries, OS/45 takes an approach IBM cannot easily counter: upward compatibility within a price range that completely covers IBM's real time offerings in the 16 bit class.

Rulben, Habermann

0269

Subject: Memory Protection and Relocation Scheme for the 11/25 and 11/40

To: Jim Bell, Roger Cady, Bruce Delagi, Ad van de Goor, Hank Spencer, Larry Wade

cc: Dick Clayton, Andy Knowles, Nick Mazaresse

From: Gordon Bell

Attached is a very rough description of the scheme that I hope can be used for protection and relocation on both of the above machines. It seems crucial that we use such a scheme, or continue to look for such a scheme in order to minimize the possible proliferation of unborn, unspecified inevitable monitors that will result. As I have indicated before, the decision on this scheme is only about 20 times more important than the instruction set, and a floating point format, so I hope we can all stay loose for the search.

Right now, the meeting on the scheme is to take place on next Friday, Oct 9 at Carnegie. It is imperative that representatives (eg. Bruce, Ad, Larry had perhaps Roger and Hank if they can spare the time) of the various groups attend. Bill Wulf, Dave Parnas, I and perhaps Nico Habermann will attend from here.

#### The Scheme

The attached scheme is a proposal by David Parnas. It isn't out of the blue, since he's been concerned with operating system design for the last 4 years, so it reflects these ideas, plus more recent concern based on work he did with the Philips company last year. I'm attracted to the scheme because:

1. It has a small amount of hardware (roughly the amount proposed by Bruce).  
(It has no associative registers)
2. It does not rely on paging or need it, although possibly a very large machine might want it.
3. It has a clean method for allowing a program operating in one address space to communicate easily with a program operating in another address space.
4. The complete system and user programs use it. Among other things it has a means of letting users control i/o, ... devices if necessary.
5. Since it is basically simple, it can be used on both, and subsequent machines.

Sept. 30, 1970

ADDRESSING SCHEME PROPOSAL

0270

Introduction and Motivation

After a serious study of some of the difficulties involved in the construction of software systems in general and hardware systems in particular, I have reached the conclusion that the construction of highly secure and highly reliable software systems is greatly ~~and~~ aided by a memory mapping device ~~that~~ <sup>which</sup> allows a program ~~and~~ its data to be divided into many segments, with the access of a program strictly limited to those segments which it is expected to use. The memory mapping devices now coming into vogue for automated memory management have this property as a side effect of their memory management purposes, but ~~at~~ times they have other properties ~~at~~ which make them difficult to use for this purpose. For the purposes that I have noted however automated memory management is not essential and a device designed for ~~a~~ such a purpose can be much simpler than the usual "paging box". In particular it is not necessary to resort to ~~any~~ use of an associative memory or hardware initiated references to core segment tables.

The particular scheme that I wish to describe here is designed specifically for the problems of protection and reliability and has the following unusual characteristics:

1. There is never any need for restricting a subset of the instructions and calling them a privileged ~~a~~ instruction set.
2. It is possible to share a machine <sup>among several</sup> ~~between~~ to subsystems ~~and~~ which are completely isolated from each other and the completeness of that isolation ~~does~~ does not depend on the correctness of any software. In other words there does not exist a software module which, if it failed, would result in programs from one system having access to data or programs from the other. [Note: there is of course a price, - in such a complete isolation no resource sharing is possible on a dynamic basis. One can always find a less extreme case in which some resource sharing is possible and a small amount of code must be trusted. However the ability to provide such complete, software independent isolation is in my opinion an essential ~~feature~~ <sup>feature</sup> of the power of the scheme.]

The scheme was originally designed in order to make it efficient to operate programs consisting of small segments of both code and data. However, given a machine with a 16 bit address size, relatively large segments are also possible. Although the use of such segments is contrary to the principles which I believe should be followed in software design, their availability leaves the programmer unrestricted. In fact, by making all of ~~yt~~ his segments the maximum possible size, the programmer may completely ignore the existence of the device.

I developed this scheme without specific interest in a particular machine. It is, infact, a fairly special case of a far more general scheme which I have been considering over a period of years. This rather generalized scheme was highly parameterized for adjusting to the size of the machine and the memory size. At the request of ~~KXHX~~ C.G. Bell, I am presenting only that subset of the rather large class of ~~XXXXXXXX~~ schemes which appear to be suitable for application to a PDP 11. This particular version of the scheme is still rather rough in that a number of the essential details remain to be worked out for this special case. I have available several tables from which I believe I can derive the appropriate values of the parameters for this case, but there are at least several days of work to be done, and I hesitate to spend such time without ~~xx~~ some assurance that there is interest in seeing the results. Further, I should have some additional information about the processors in order to do a good job of finding the appropriate solutions.

I believe that in the following, I have developed sufficient information to allow the scheme to be evaluated and a preliminary decision about the value of the scheme to be made.

The fact that I have not provided the additional details does not mean that they are unimportant. The scheme is sensitive <sup>to them</sup> in the sense that there are a number of details with the property that-if an incorrect value is chosen, the value of the mechanism is sharply degraded. It is the importance of these remaining details which cause their derivation to require careful thought and time. On the other hand I believe the ~~following information~~ following descriptions ~~x~~ makes it clear that appropriate values for the remaining parameters as well as appropriate encodings for certain status bits can be found.

0272

Notes: A number of aspects of the system are highly familiar. There are however several unique features which I believe are essential.

### The basic facilities provided

The mechanism provides for programs which operate in two name spaces or virtual memory spaces, the information in one being a subset of the information in the other. Programs may move freely about in the smaller of these two spaces subject only to the restrictions placed on the use of each of these segments. Programs may also pass segments of data or program from the larger to the smaller of these spaces and vice versa. No access <sup>to</sup> any data or program element in the larger space is possible unless it is also in the smaller space. The moving of segments between the two spaces is not automated and is done under direct program control. The movement of segments inside of a physical memory is independent of the movement between the two virtual memory spaces, and is invisible to the program except for real time considerations. Although the most basic versions of this scheme do not make use of this independence, it is possible to conceive of larger processors which do so to automate a segment migration without any loss of compatibility.

In addition to the program's ability to move about within the smaller address space and to move items from its larger space into the smaller one, the program has the ability to call other programs which are not in the address space and which operate in an address space which is distinct from that of the calling program. The set of programs which may be called in this way can be set up in such a way that they are predefined and the program may not alter the list. On the other hand, where desired, it is possible to have this list extendable and alterable by the program itself. If desired two lists, one fixed and one alterable may be set up.

The effect of this feature is to allow programs to be called in such a way that the calling program has no access to the code or private data of the called program and further that the called program has only restricted access to the code and data of the calling program, exactly that access which the calling program chooses to allow. I consider this feature highly important in allowing the efficient implementation of highly secure and reliable software.



For future use we now define the following names

Program Virtual Memory (PVM) = the large space mentioned above

Program Working Virtual Memory PWM = the small space mentioned above.

Program Segment Table (PST) = the table defining the PVM, kept in core.

PST base register = a register containing the location of the base of the current PST. NOT DIRECTLY ADDRESSABLE

Working Segment Table = the table defining the PWM stored in special registers (NST) mentioned below.

Working Segment Registers = a set of ~~xpm~~ registers containing the WST

(NSR)

They are not addressable by normal instructions.

Extended Call Space = the set of programs operating in other spaces which may be called by a given program. This may be seen to consist of three sub spaces

Fixed ECS = an Extended call space which may not be modified by the program itself.

Free ECS = an extended call space which the program may modify using programs and address spaces which it itself has constructed. THIS IS OPTIONAL AND OF QUESTIONABLE UTILITY IN MY MIND (for a machine of this size).

Return ECS = an extended address space which is defined by a calling program for the called program. As a rule it contains 1 entry.

*ECS registers = registers containing base(s) of ECS*

In the sequel I will describe the characteristics of these various items as I expect them for the PDP 11. ~~Several~~ of the numbers given are rough estimates based on inadequate procedures and I would expect them to be seriously reconsidered since there are ~~many~~ methods of determining appropriate values which could be used. Other figures are minimum figures ~~if~~ appropriate for a relatively small machine. I can imagine changing them considerably for larger processors, but would require ~~much~~ more time to calculate appropriate values. I will try to point out the variability of these figures as I give them. I provide them mainly as a guide to the size of the hardware that I have in mind.

The PVM is a segmented address space. Each segment would have a maximum size of ~~82k~~<sup>8k</sup> bytes (initial figure to be recalculated) but I would expect the average size to be much smaller, possibly on the order of 256 bytes. The segments would be identified by an integer between 0 and ~~2k~~<sup>2k</sup>. The most efficient use would keep ~~this~~ the maximum segment number much lower, perhaps under 256 or 128. If segment I is of the maximum possible size then the highest byte in the segment is considered virtually adjacent to the lowest byte of segment I+1. In all other cases indexing or other operations referring past the end of a segment would be considered errors. Each segment is characterized by a state (read only, write only, etc.) the number of states must yet be studied and a precise definition of their meaning made up.

The PST is a table with a maximum length of 1 segment containing ~~1~~<sup>4</sup> bytes for each segment. It is kept in core when in use and is then pointed to by the PST~~ix~~ base register. Under normal usage the segment containing the PST is not in the PVM of that program. There is of course no reason why this cannot be done if desired. *Segment i is described by the i<sup>th</sup> entry in the table.*

The PWM. is a segmented address space. The characteristics of the segments are exactly those of the PVM segments. In fact every segment in the PWM is in the PVM. I expect that 7 or 8 segments is a reasonable size for the PVM for this machine. If ~~8~~<sup>8</sup> is chosen then the maximum size of the PWM is the current core size. For this reason ~~axpragram~~, with suitable segment definitions, ~~can~~ a current program could be run without change entirely in the PWM.

The calling tables should probably contain 256 entries. Each entry would indicate a value for the PST base register, and an address in the ~~PVM~~ PVM defining the first segment to be executed and the first location in that segment.

#### ESSENTIALS

The registers and tables mentioned ~~ab~~<sup>above</sup> will ~~not~~ contain physical addresses. The following conditions are necessary for the success of the scheme.

1. There must exist no instruction which refers directly to one of the above tables and registers and places the values contained in a place accessible to the program.

0275

2. The only way to access or modify these tables and registers is either by means of the special instructions to be described below or if the tables or registers are placed in the PVM of the program.
3. It must be possible to place the PST in a segment included in a PVM. This is no problem since the ~~PST~~ PST is in core. This is also true for the ECS lists.
4. <sup>a</sup> ~~a~~ It must be possible to place the PST base register, the WSR registers, and the ECS registers ~~xxxxxxx~~ in a segment which may be included in a PVM. For example a facility which allowed a given segment ~~xxx k~~ to contain the PST base register in the first word or two, the WSR ~~xxxx~~ registers in the next 16 words and the ECS registers afterward.
- b. An alternative to 4 a above is a complete facility for loading all the registers by a single hardware instruction using information in fixed format core tables. I believe this is a better alternative. It is instruction 3 below

## SPECIAL INSTRUCTIONS

These instructions are not privileged. They are special ~~only~~ only in that they are not needed without the mechanism we are describing.

1. Load Working Memory (i,j) segment i from the PVM becomes segment j ~~from the~~ PVM. The old segment j is forgotten.
2. Save Working Memory (I,j). Segment j from the PVM is made segment i in the PVM. If ~~there~~ already was a segment there it is forgotten.

NOTE THAT these instructions shift information between the two tables but never change or alter the contents and never reveal the c)

3. Extended Call (i)\* to i<sup>th</sup> entry on the e.c.s list is loaded <sup>in the PST</sup> and the specified first instruction executed.

— Any of these ~~segments~~ <sup>instructions</sup> may specify a segment not in core. In that case a trap must call a system routine. Perhaps this can be extended call (0).

\* additional parameter could specify which of the ~~the~~ ECS lists.

0276

## Expected NORMAL USAGE.

In the expected normal usage a program is set up by system programs (or by initial load) so that the tables defining its address space are inaccessible to it. It then operates primarily within the PVM <sup>(lower case)</sup> ~~THEREBY~~ making all references to core using the PWR as base registers, never using the PST. Only the load and save memory instructions use the PST; only these go through the core tables. In proper usage these instructions should be rare compared to the normal instructions. All references out of the PVM by normal instructions are considered errors and trapped. All references out of the PVM by load and store memory instructions are similarly trapped. Similarly an attempted to use an extended call with the parameter too high is trapped.

The PST will contain bits indicating if a segment is not in core and the address in either case. If the segment is not in core a load working memory or an Extended Call will trap.

The ECS may contain routines with ~~XXXXXXXXXX~~ access to the tables defining the address space of the calling routine. Through these tables extensions or contractions to the PVM may be made by 1 or more ECS programs.

## Hidden Information.

I expect that hidden from all users will be the fact that the segment tables contain a number of extra bits such as an indirect bit allowing the entry to refer elsewhere for the actual entry. Note that the indirect bit is only used during a load working memory instruction sequence, not in the normal use of the segment. The number and interpretation of the status bits are essential information which must be carefully worked out, but which, as I mentioned earlier I have a basis for. In that I have some more general tables from which they can be derived.

## Larger Devices.

If in the load working memory instruction we specify more than the segment to be loaded by indicating ~~an~~ an address or locality within the segment, a later, larger device which partially loads segments can be made <sup>comparable</sup> ~~comparable~~ with this one. The number of bits available for this must yet be worked out.

The possibility of having portions of the PVM not in core might also be added to a device for a large processor that is still comparable. I PERSONALLY expect <sup>(lower case)</sup> ~~this~~ <sup>that</sup> this is not worth it, but it is worthwhile noting for a future investigation.

Appendix: Address interpretation

Physical Address := WSR<sup>12 0</sup>[Address<sup>15 0</sup><0:2>] + Address <3:15>;

**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 Hasset  
Pete Van Roekens  
Garth Wolfendale  
Denny Pavlock

## 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 bill* The new address length should be between *16 million* 24 and *4 billion* 32 bits, not just an extra bit or two over today's 16 bits.

#### 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.

- 7. via assembler*
- 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.

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:

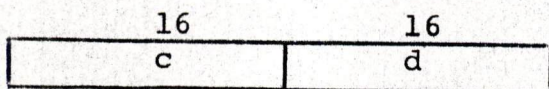
- a) Program modularity.
- b) Varying-size data structures.
- c) Protection.
- d) Sharing, without the conflict which derives from the 8-segment KT11.

## 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<sup>1</sup> address space: 2<sup>16</sup> chapters of 2<sup>16</sup> 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 FORTRN 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 K11 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  $X^5$  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 ~~algorithm~~ *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 RIX 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 interrupt. All interrupts are returned to Process 0, Chapter 0. The interrupt vector here, in particular PS <09>, controls the stacking and unstacking of PCX.

This allows the placing of a current 16-bit supervisor in P<sub>0</sub>C<sub>0</sub>, 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

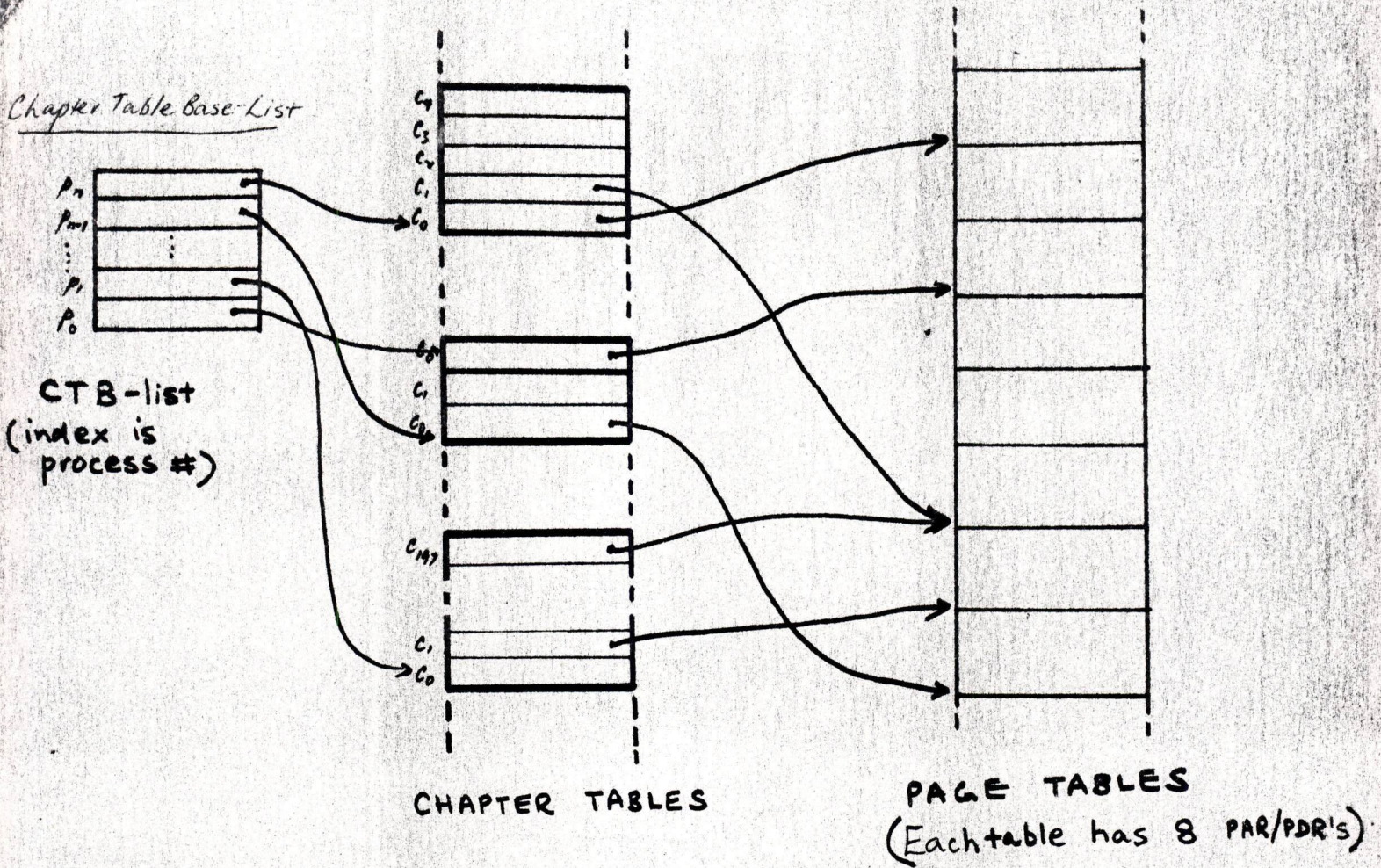
If we were designing PDP next, i.e., designing the size and structure of a 32-bit virtual address space from scratch, we would not propose a classic segmentation scheme, but the segment size would be 24, not 16. Other less-than-ideal properties of the chapter scheme, which derive from our strict compatibility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

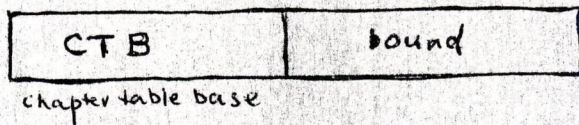
FIG 1

MAP TABLES IN MEMORY

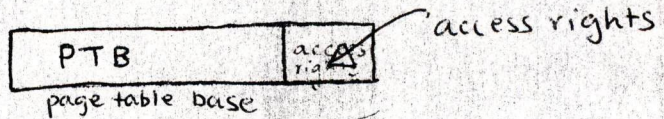


Format of table entries:

CTB-list entry



chapter table entry



page table entry

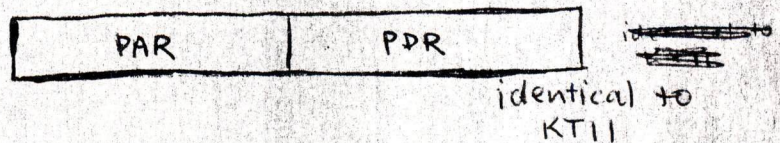
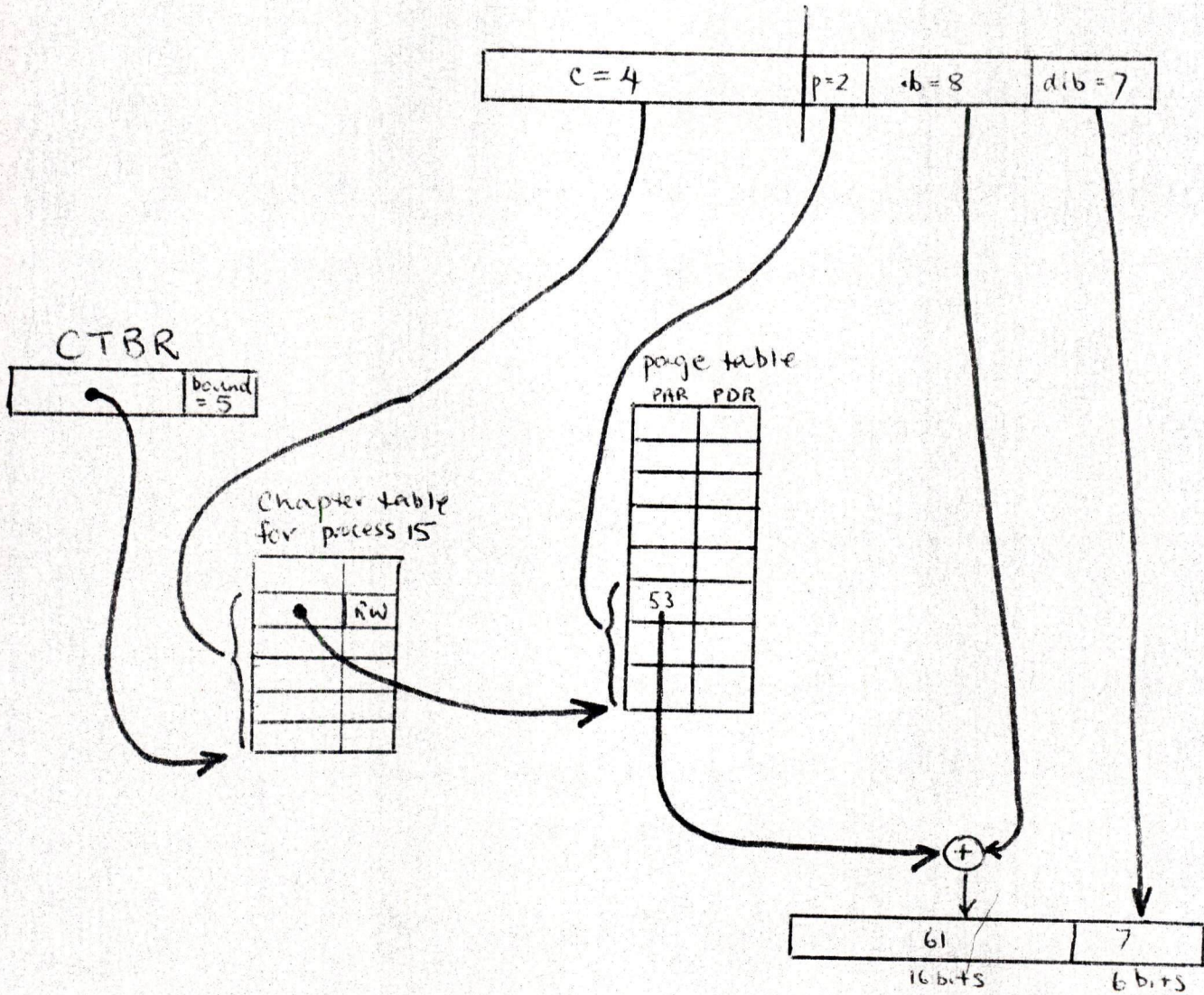


FIG 2

ADDRESS TRANSLATION EXAMPLE

32-bit Virtual to 22-bit physical



Chapter table lookup

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

Appendix 1Possible 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 today's set of KTI1 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

More follows on

- 1) Shared Segments
- 2) Required Hardware Support
- 3) Extra Hardware "to make life easier" like the MULTICS . it's and it's instruction
- 4) Motivation on Why Segmentation

⑤ - Paging

Ced

Carnegie-Mellon University

Department of Computer Science  
Schenley Park  
Pittsburgh, Pennsylvania 15213  
[412] 621-2600  
[412] 683-7000

October 21, 1970

Mr. Ad van de Goor  
Digital Equipment Corporation  
Maynard, Massachusetts

Dear Ad:

Since the busy schedule of meetings at Maynard yesterday did not provide me with an adequate chance to communicate my current opinions on the segmenting mechanism that we were discussing, I would like to place some observations on paper.

First, I wish to make it clear that, with some exceptions which are mentioned below, the scheme that we have discussed is well represented by your memorandum dated October 14, 1970. The mechanism appears to be sufficient to allow all system and user code to run in a virtual address space. Further, if the appropriate instruction capabilities are provided, it will allow the assignment of an input-output device to a specific program without allowing that program to access other devices or to be physical address dependent in any way. My recent studies have indicated that these features are essential to the development of highly reliable software systems. In this connection I wish to emphasize that the instructions for adjusting the working segment registers must be carefully specified so that these properties are maintained. At least one of the current proposals allows the construction of programs whose behavior may vary if the physical address assigned to a segment varies. I believe that it will be highly valuable if such an arrangement is avoided.

With respect to the general contents of your memorandum, I believe that any rewrite should clearly separate the two types of information provided therein. Your memorandum contains both a description of some hardware features and a description of one way in which they may be used. I believe it to be dangerous in that, in further work, some of the assumptions about possible usage may be confused with restrictions imposed by the scheme. The result could be that the final product would be unnecessarily restricted in its manner of use. This is especially important since, in my own opinion, the method of use assumed in your paper does not make maximum use of the capabilities of the mechanism to provide highly reliable software. The method of use which you assume appears to be

perfectly feasible and is quite likely to be the optimal method under some circumstances. On the other hand, it would be unfortunate if later decisions made other forms of use impossible.

I am of the opinion that the major rough edges left on the scheme are consequences of the fact that we have not yet provided a convenient means of passing parameters when transferring control to another virtual memory space. The current scheme in which parameters must be passed either in a shared segment or by use of an intermediary imposes restrictions on the form of address spaces and introduces the possibility of timing problems in the event of simultaneous calls to some shared routine. You will recall that in my original discussions of the subject I had an "ECS" mechanism which is now a UVO but which provided for parameter passing. It now appears that the decision to leave out that feature causes more problems than it saves. I had an extended discussion with Bill Wulf on this matter on the trip home and he now agrees that some form of parameter passing on UVO or ECS is highly desirable.

The scheme as it is now described provides for a Working Address Space of up to eight 8k segments. Thus it is possible to imitate exactly the current 64k core. If necessary this can be reduced to 7 to provide an extra bit combination for special address space oriented operations, such as adjusting the WSRs or referring outside the Working Address Space for brief periods without loading one of the registers. Each of these effects can equivalently be gained by means of a new instruction code (at least for the larger machines). I believe that, where possible, the second type of solution is preferable. An asymmetry in the way that one segment (7) is handled can lead to a number of difficulties (e.g., strange effects on the occurrence of program errors).

We have specified that the smaller segments must be in low core. I am told that in some cases the i/o devices, trap vectors, etc. will be elsewhere. There is no doubt that in such a case smaller segments should be restricted to an area of the same size as specified, but including that "external" segment of physical core. In other words, we should shift the area.

It is important to specify that the use bit is only changed on completion of a segment usage. In the indirect referencing cases a segment may be referenced but the use not completed if one of the segments referenced is not in core. In that case the use bit should remain unchanged. The 360/67 suffers from this error and prevented a perfect "holiness" state as a result.

The remarks on the SUF are incorrect and unnecessary.



With respect to the validity states: First it is important to note that the "valid copy" need not be on backup store; it can also be in core (not a very common case but one that will occur). The motivation for the choice of the eight states is twofold. First we wish to prevent reads and writes in cases where segment swapping has made such actions impossible or incorrect. Second we wish to have sufficient information about a segment to know if it must be swapped out or not when space is needed. These states were derived by reducing a larger table (combining all states which were equivalent for these two purposes). The only cases which are not obvious are, for some people 4 and 5, but for others 6 and 7. All four of these are cases in which the write action makes the "valid" copy invalid. When that happens there are three possibilities:

1. the action was an error
2. the action was correct, the backup space may now be released, but we are in a hurry and will take care of that later
3. the action was correct, backup store is scarce and we want to release it immediately.

In states 4 and 5 we assume that condition 2 holds. In states 6 and 7 we assume either 1 or 3, or that it takes software to find out which case holds. In state 7 possibility 1 above is irrelevant. In considering 4 and 6 we find that 6 is the "normal" case with 4 being reserved for the situation in which a relatively small segment is being completely rewritten. The choice between 5 and 7 depends on the availability of backup store space. Choose 7 if it is a scarce resource.

With regard to the state transitions: For the multiprocessor case only it is important that the state changes be determined finally on the basis of the core copy and not the WSR copy. However, the core need not be consulted each time because all hardware produced transitions move to a stable state. Actually it is only necessary that the "change possibilities" for the destination state are a subset of those for the original state. The diagram given has this property.

The incompatibility in the trap vectors has to be studied carefully. I think it correct to say that the difficulty arose because the current PDP-11 is wrong in this area. I will justify that statement by claiming that the current scheme assumes a "last to be interrupted is the first to be reawakened" discipline for processes in an operating system. There are numerous situations in which such a discipline is inappropriate and in those cases the assumption of LIFO or stack behavior leads to inefficiencies and a more complex interrupt handler than should be needed. You will find those difficulties in the MULTICS interrupt handler for just those reasons. The stack assumption could have been avoided by stacking PC and PS and then using R6 (SP) in the dope vector or scheduling tables. Instead PC and PS

are used in the dope vector and the stack is common. One result of this will be unnecessary stacking and popping for real time systems, but, more important, in the old scheme the probability of stack overflows will be greatly increased over the scheme we propose because all processes in the old scheme use the same stack. I believe that this type of arrangement is incorrect even when there is no segmenting scheme. In our case the extra stack operations are even more important because there is extra state to deal with. It is perfectly feasible to keep the PC and PS in the dope vector as Delagi suggested, but the price will definitely be decreased performance because we are carrying an earlier mistake forward. I believe that Wulf and Bell now agree on this.

In this regard we must be aware that by keeping R6 and W0-W7 in the ST as shown we eliminate the possibility of sharing a segment table for several processes which operate in the same address space. Note that we can do that now with a physical space. To avoid duplicate tables we can separate the values of W0 through W10 from the segment table. Since they represent addressable data they can be kept and used in the Virtual Address Space. In particular the values of these registers + the base of a segment table determine a "process" (term used for lack of a better one). I see very little cost resulting from moving this data out of the PST.

With the exception of one paragraph on page 13, the material on 13 to 16 should be eliminated on the basis of my remark about treating a possible use as the "only way".

The trap vectors must be fixed by convention in virtual address space; the external interrupt vectors must be tied to physical space.

I believe it would be nice if the UUO interpreted the extra eight bits as an index into an array of names. This is done now in software anyway. We could then eliminate the intermediary program easily.

Steps 1 and 3 of the restore sequence on page 21 are highly questionable. A minor matter is that you never use or refer to W12 and W13 elsewhere. That is probably just an oversight. On the other hand, this decision assumes the same "last interrupted, first awoken" discipline which we criticized the current 11 for. I believe that in most cases we must rely on the routine which first awoke the interrupted process (scheduler) to have retained a VMD or process pointer and to use it again at the appropriate time. This should be thought out very carefully. The parameter passing possibility will help in many cases.

Above all we must resist the tendency to follow some suggestions and try to make the mechanism a "cure all". It allows all code to

run in virtual space. That solves a lot of problems but not all of them. It alleviates the stack overflow problem but will not provide a magic complete cure; it will not allow a FORTRAN compiler to be able to ignore the fact that memory is not a random access memory. If we follow the MULTICS route and try to take all the special cases into account, the thing will get beyond our understanding (just as MULTICS did for its designers).

Yours truly,

D. L. Parnas  
Associate Professor  
Computer Science

DLP/dmj

cc: same mailing list as your memo.

TO: Distribution

DATE: March 21, 1974

0442

FROM: Craig Mudge

DEPT: 11 Engineering

EXT: 5064      LOC: 1-2

SUBJ: Initial Feedback on Chapter Scheme

VAS MEMO #2

1. Problems

- (1) External representation of a process's loaded image will need to be an encoding of the internal representation because of the tagged stack. Needed, for example, in swapping out a process.
- (2) References to the tagged stack will not necessarily be through R6, e.g.,

```
MOV  SP, R0
ADD  (R0)+, B
```

Hence, the implementation must be able to recognize this.

2. Suggestions

- (1) The tag needed for stack entries could be bit 0 (PC <0> is redundant on the stack).
- (2) Allow 32-bit addresses in indirect addressing and use bit 0 as tag.
- (3) Make mode 5 do something useful.

3. Clarifications

- (1) Index mode, X(R6) goes through the stack entries one by one to the X'th one.
- (2) Rules for addresses in the registers:

- a. registers always hold 32 bit address
- b. loading a register always fills 32 bits

(a) Memory to register

- i. MOV:            c ← current chapter
- ii. LA:            c ← high 16 bits of operand

(b) Stack to register

- i. MOV when stack entry is short: c ← current chapter  
" " " " long: c ← high 16 bits of entry
- ii. LA " " " " long: "  
" " " " short: error

(c) Register to register

Long address to long address

Distribution

- Bruce Delagi
- Bill Strecker
- Dave Rodgers
- Ron Brender
- Ed Marison
- Jega Arulpragasam
- John Levy
- Bill Demmer
- Len Hughes
- Bob Gray

digital

## INTEROFFICE MEMORANDUM

SUBJECT: Protection and Relocation for the 11/25      DATE: September 28, 1970

TO: PDP-11 Coordinating Committee      FROM: Gordon Bell

cc: Bruce Delagi      DEPARTMENT:

Let me thank Bruce for getting this proposal memo out into the open. Three comments (so far):

1. This subject (I believe) is a lot more important than an issue like floating point data format or a calling sequence for subroutines because it affects all software (monitor, I/O, translators, utilities, and all user-written programs). Therefore, can we hurry and get a small group together to really consider it and make sure it's right in the same way the two committees worked on floating point? Getting a group together won't harm the 11/25 schedule if, say they're given the guideline of having about the same amount of hardware ( $\pm 2$  registers). In fact, I believe it will speed up the 11/25's by about 6 months, because it will force the monitor structure to be outlined - and thus the software will be able to use the hardware instead of having to be written in spite of it.
2. At least make the 3 segments (register pairs) have control bits to indicate whether a segment is read-only, read-write, execute-only, or stack. In this way you aren't stuck with the program organization Bruce is dictating by his hardware registers. Since Bruce's comments deal with time-sharing, I assume that's the program structure on the PDP-10. We've gone through a fair amount of pain to modify the structure to allow several independent programs to access common data. Also, I would hope the problems on the 10, like not being able to swap a program doing I/O, are solved with this organization. (For process control this seems very important because it allows programs to be brought into core and executed only when there's data ready in an I/O area.) Finally, the biggest single problem of the 10 monitor is its size. This is partially caused by the fact that I/O can only be done in the monitor (in monitor mode). Therefore I would hope that user written I/O control programs (e.g., disk, special I/O) are permitted. These routines do work for other user programs, placing results in the calling user's area.
3. I hope to have an extensive alternative proposal which uses the same amount of hardware.

bwf

**digital** INTEROFFICE MEMORANDUM

DATE: October 14, 1970

SUBJECT: Protection and Relocation for the 11/25

TO: Gordon Bell ✓  
 Bruce Delagi  
 PDP-11 Coordinating Committee  
 Ken Stapleford

FROM: Van Diehl

I am very much in favor to get together and discuss the above subject, fundamental for the structure of the real-time monitor we will be writing for the PDP-11/25. In fact, a real-time multiprogramming system with background-foreground capabilities that does not have a good hardware protection scheme, i.e., protects any task or executive from being destroyed by the running task and at the same time does allow a good management of Common's and Global Common has very restricted market potentials.

The subject of data queuing in a multitask system is also a very important subject that has not been solved by the competition, exclusion perhaps of the IBM-1800 MPX.

Because the architecture of the PDP-11/25 real time monitor that we are presently specifying is so dependent in these ideas, I will like very much that we get together ASAP to make sure we are going in the right direction.

WD: cs

INTEROFFICE MEMORANDUM

0342

TO: Gordon Bell ✓  
Dick Clayton  
Julius Marcus  
Bruce Delagi  
Ken Hughes  
Peter Van Roekens  
Hank Krejci

DATE: July 16, 1973  
FROM: Al Avery AC  
DEPT: 11/45 Marketing  
EXT : 2543

JUL 17 1973

188

PDP-11

Memory Management

or 11/45

SUBJ: PDP-11/45 PROCESSOR AS A VIRTUAL MACHINE

Attached is a proposal from the Computer Science Department at UCLA, outlining the virtues of the 11/45 as a virtual machine. Also attached is a quote from CSS, which was completed after a meeting with Dr. Popek and Lou Nelson of UCLA.

The major question here is, as you can see from Dr. Popek's letter, is Digital interested in some kind of joint venture? If yes, to what (financial) extent? We should really consider this from a programming as well as hardware standpoint. Dr. Popek feels that the modification which they have proposed will greatly aid in the programming of such a system.

Please let me know your views quickly, as Dr. Popek has to make a decision. Their Sigma 7 has been cancelled and will be removed in late December. They have a slot reserved with the communications product line so that the time they are without a machine is minimized.

mc  
Attachments





EINGEGANGEN  
16. JUL 1973  
A. AVERY

COMPUTER SCIENCE DEPARTMENT  
SCHOOL OF ENGINEERING AND APPLIED SCIENCE  
LOS ANGELES, CALIFORNIA 90024  
3532 Boelter Hall

July 12, 1973

Mr. Al Avery  
Digital Equipment Corporation  
Maynard, Massachusetts

Dear Al:

I have enclosed some material regarding our recent discussions concerning the PDP-11. One of the papers is a general discussion of the modifications that we had proposed. The note concerning the value of security and virtual machines will, I hope, be of use to you in your internal marketing discussions. The paper by Buzen and Gagliardi discusses a number of the desirable features of virtual machine architectures that are mentioned but not elaborated upon in my note.

I might point out the context in which this effort is taking place at UCLA. The project is under my direction. I am a newly arrived faculty member. It is being initially funded in a limited fashion by seed money in the existing ARPA contract here. As a result, large sums cannot be justified until we have shown enough progress to submit new funding requests. So for example, even a quarter of your quotation would greatly strain my resources.

I feel quite strongly, however, that we are providing you with rather well thought out and viable ideas. A not insignificant gesture on the part of DEC would be appropriate in return. Suppose we paid for the actual modification cost, plus a small fraction of the design overhead. Then for example, extra storage would allow us to develop the teaching application mentioned in my note.

It would be unfortunate if we must look elsewhere, since our progress would be set back by the task of recruiting new funding as well as establishing a rapport with another manufacturer. I look forward to your reply.

Sincerely,

  
Gerald J. Popek  
Assistant Professor

cc: L. Nelson  
R. Kahn  
L. Kleinrock  
I. Cohen



Computer  
Special  
Systems

QUOTATION No. : W-821-0

0344

Date: M 6, D 27, Y 73

Doug Forsberg *D.F.*

CUSTOMER: UCLA DEPT./RFQ/CONTACT: Dr. Gerry Popek

SALES OFFICE: West Los Angeles SALESMAN: Tom Sherman

INQUIRY BY: SALESMAN  CUSTOMER  OTHER

MARKET AREA: Communications/Lab COMPUTER LINE 11/45

INTENDED SYSTEM APPLICATION Computer Science

INQUIRY:

Modify PDP-11/45 Processor to Virtual Machine per attached technical description. Reference meeting June 12, 1973 between Dr. Gerry Popek and Lou Nelson of UCLA and DEC CSS Engineering.

QUOTATION:

- 1) \$28,700 1st unit
- \$ 4,700 Each additional unit

Delivery requested in December 1973

PREREQUISITES: PDP-11/45 to be at CSS by November 16, 1973

DELIVERY ESTIMATION: Delivery is estimated <sup>AS</sup> above months following formal acceptance of customer purchase order; firm commitments are established upon acceptance of P.O.

CSS QUOTE EXPIRATION: All quotation commitments to both the salesman and customer expire M 8 D 27, Y 1973. Extensions must be requested.

TERMS & CONDITIONS: DEC standard terms and conditions as modified by CSS rider apply unless otherwise specifically negotiated.

WHITE - CUSTOMER FILE

CANARY - SALES OFFICE

PINK - COPY 1

GOLDENROD - COPY 2

The following list briefly describes the PDP-11/45 modifications which DEC will supply to U.C.L.A.

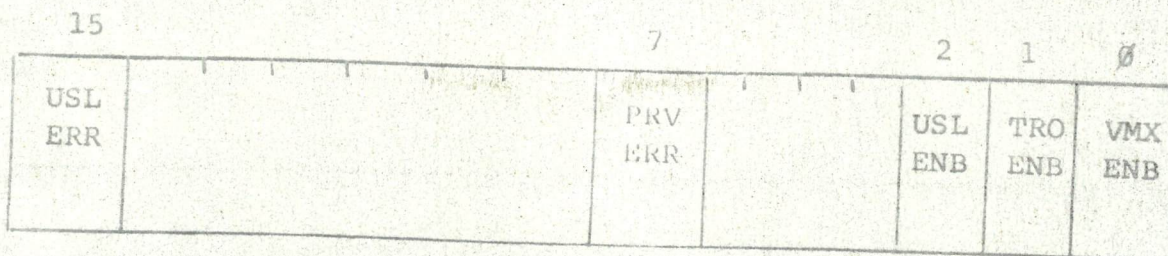
1. A programmable register will be supplied to enable/disable the following features: A switch to over-ride the register will also be supplied. When the special features are disabled, the CPU will run like standard PDP-11/45.
2. The following instructions will cause reserved instruction traps when executed in user or supervisor mode:

RESET  
WAIT  
RPL  
RTP  
SPL  
MTP1  
MTPD  
MFP1  
MFPD

(Note that "HALT" Already traps)

3. A register will be provided such that the kernel mode program servicing the traps described in (2) above can read the encoded reserved instruction as a 4-bit word index, such that immediated dispatch to the proper service routine can be made. This will save the software from having to decode the instructions.

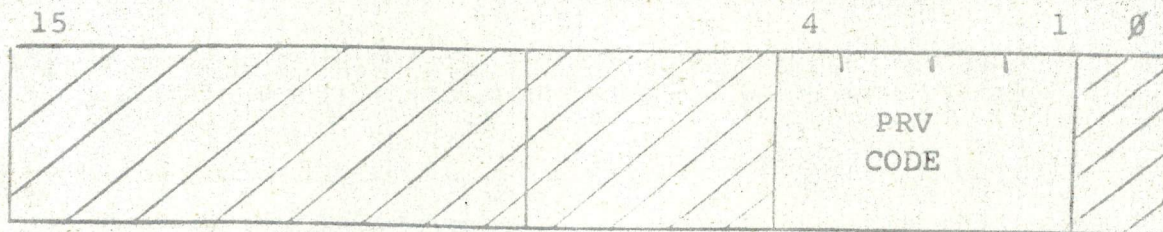
4. An offset for vectors used for traps and interrupts from user mode will be provided, such that traps and interrupts from kernel or supervisor mode will trap to the normal vectors and traps and interrupts from user mode will trap to the normal vector plus the offset. The offset will be set at 1000 (octal) unless otherwise specified. Example: the supervisor tries to execute a privileged instruction and so will trap using vector 10 in kernel space. A user program trying to execute the same instruction would trap using a vector at 1010 in kernel space.
5. A "User Stack Limit" register will be provided to operate similarly to the kernel stack limit register. Any user-mode stack operation below the limit will be allowed to complete but will cause a "Memory Management" (segmentation) trap (as opposed to an abort).
6. The registers supplied on the Unibus for control of the option are shown in figures 1, 2 and 3.



<u>Bit</u>	<u>Mnemonic</u>	<u>Description</u>
0	VMX ENB	Virtual Machines Extensions Enable -- When set, enables the trapping of the instructions specified in 2 above.
1	TRO ENB	Trap Offset Enable -- Enables the special user trap offset.
2	USL ENB	User Stack Limit Enable
7	PRV ERR	Privileged Instruction Error -- One of the special privileged instructions caused the trap to 10.
15	USL ERR	User Stack Limit Error -- User Stack Limit caused segmentation trap.

FIGURE 1

VMX CONTROL STATUS REGISTER



<u>Instruction</u>	<u>Code*</u>
RESET	1
WAIT	2
RTZ	3
RTT	4
SPL	5
MTPI	6
MTPD	7
MFPI	10
MFPD	11
Unassigned	0, 12-17

\*(Assignments Tentative)

FIGURE 2

PRIVILEGED INSTRUCTION ENCODING REGISTER

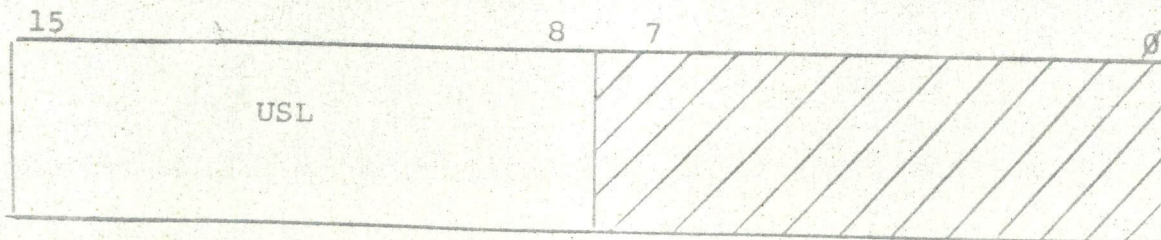


FIGURE 3

USER STACK LIMIT REGISTER

Gerald J. Popek

UCLA, June 29, 1973

(The following discussion assumes the reader is familiar with the introductory material presented in "Part of a Proposal for the Design of a Certifiably Secure Multiuser Computer Facility".) These remarks are intended to more firmly specify our proposed modifications to the PDP-11/45.

Several hardware modifications are necessary to the PDP-11/45 in order to make it virtualizable. The sensitive instructions that must trap are the following:

- RTI
- RTT
- WAIT
- RESET
- SPL
- MFPI
- MTPI
- MFPD
- MTPD

That is, an attempt to execute these instructions in other than kernel mode should cause a privileged instruction trap to one of the reserved locations, similar for example to the way HALT is treated. These modifications are absolutely necessary in order to construct a virtual machine monitor.

From the point of view of efficiency, it would also be useful to have a stack limit register operative in supervisor and user

modes, protected by the memory segmentation hardware. If such a register existed, it would not be necessary for the VMM to simulate one, likely to be an awkward, inefficient and time consuming effort.

One might consider, as a fall back option, simply making the current register active in supervisor and user mode rather than kernel mode, and set a fixed boundary for kernel mode. By providing such a register, the task of simulating its existence essentially vanishes, leading to a simpler VMM.

The third modification is also related to efficiency. In the standard architecture, the state (user, supervisor, kernel) in which the processor is found after a trap occurs is taken from the appropriate trap vector, but there is no way that the new state can be a function of the old state. That is, the state of the processor after trap is set by the single appropriate vector, regardless of whether the machine had been in supervisor or user state when the trap occurred.

In order to make the new state depend on the old, we propose that there be a second set of trap vectors: one set to be used by the hardware when a trap occurs in user state, the other when it occurs in supervisor state. It was mentioned at the meeting that a simple implementational approach would be to OR a bit into a high order position in the trap address if the trap originated in one of the states; with no new action if it originated in the other state. Such an implementation implies that all trap vectors will be located in kernel virtual space, which seems like a reasonable decision. The utility of this change is discussed in the next section.

It was also mentioned by W. Weiske that adding a second set of trap vectors and switching the action of the stack limit register



from kernel to supervisor/user modes requires another minor change. The fixed stack limit in kernel mode should no longer be 400<sub>g</sub>, since at least 500<sub>g</sub> locations have already been used for traps. Hence that fixed boundary should be higher, probably double the old.

The last major aspect of these modifications is the following. They should all be activated by a programmable switch, a bit perhaps that is located in a place where it can be simply protected. Depending on the value of that switch, the machine either acts as an unmodified or modified PDP-11/45: all the modifications are either disabled or active.

The possibility of a manual toggle switch override was mentioned at the meeting. We suggest that it be a three position switch: standard machine, extended machine, under program control.

In this fashion, software for standard machines, including test and diagnostics, can be run without any changes to them. It also allows us to run standard software during the development of the kernel and VMM.

To help the modification of the hardware along, it seemed useful to set down more specifically in everyone's mind our proposal as we see it. I hope that the preceding discussion helps to serve that purpose.

The following remarks attempt to motivate some of these changes in a general, conceptual fashion by sketching the intended structure of the software to be built.

#### Remarks on the Hardware and System Software Design

As noted in the proposal, the goal is a secure multiuser system, and the approach includes the construction of an operating system

subnucleus, called a kernel, and a virtual machine monitor (VMM) layered over that kernel. The VMM will produce virtual PDP-11/40 environments, in which standard software can be run. Intended uses include ANTS, Digital's DOS, and other operating systems. The practicalities of making virtual machines available to students at UCLA for teaching purposes has also be raised.

Here "virtual machine" is being used in the sense of the hyper-visor monitors of IBM's 370 or CP-67. The decision to produce PDP-11/40s rather than PDP-11/45s is based on the relative ease of the former task compared to the latter. The PDP-11/40 is a relatively simple architecture, with two states ("kernel" and "user") and no segmentation hardware for the time being. In contrast, the PDP-11/45 has three states and rather complex memory management hardware. Just switching from one user to another in order to multiprogram, for example, requires saving a great deal of information.

In order to virtualize the memory management hardware, in addition to having facilities to save virtual memory management information, it is necessary for the sake of efficiency to have the actual hardware contain the result of the composition of two memory maps: one performed by the VMM and the other by the currently running operating system. Managing these fairly formidable details was considered to be a relatively low priority item. While practical and desirable, it is not necessary at first, since PDP-11/40 environments are quite acceptable for the currently envisioned applications.

Extending the VMM to yield PDP-11/45 environments will be contemplated at a later date. The recursion that can result (running

copies of the kernel and VMM in a particular VM) is of use for several reasons. Debugging new VMMs and other such advantages are mentioned in the literature. In our case there may also be the value of a finer division of protection, since the kernel is now running in a given VM, and can in a reliable fashion divide up and control the information units allocated to it by a kernel running above it. The structure is illustrated in the accompanying figure.

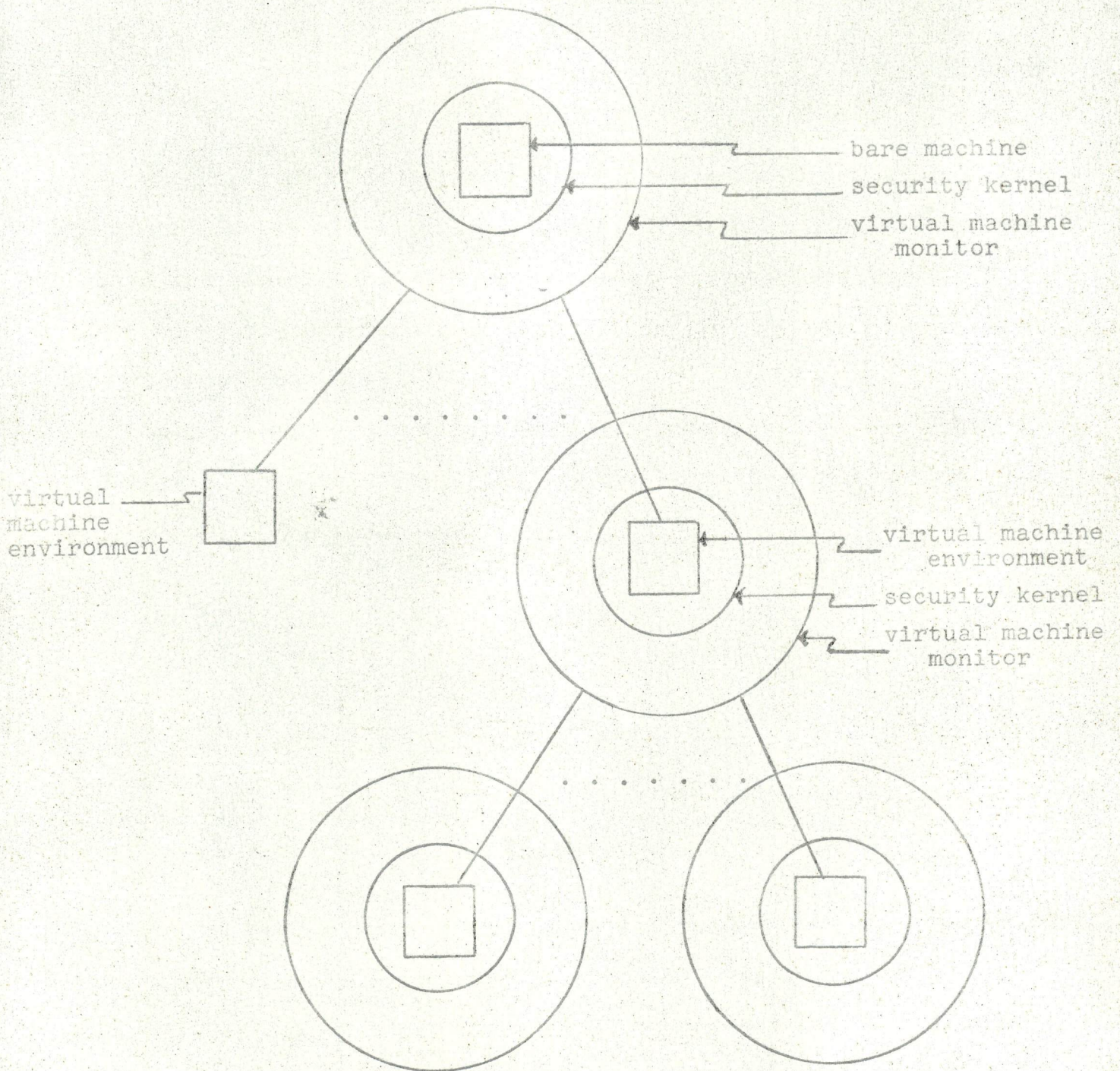
The three states of the PDP-11/45 will be used as follows. The kernel program will run in kernel state, the VMM in supervisor state, and all other programs in user state.

The kernel program will be the object of a great deal of attention if it is to be proven correct. It is critical that the kernel be as small as possible if a reliable correctness proof is to be constructed. As much as possible then the kernel will contain code relevant only to protection. Other concerns will be relegated to the VMM.

For this reason for example, one does not want all traps to go to the kernel but rather only those relevant to access control. In our system, practically all traps generated by an operating system (running in real user mode) will be handled by the VMM. This view motivates the desirability of having the destination of a trap depend on its origin. If separate sets of trap vectors were not available, then all traps would go directly to the kernel program, and it would then be necessary to have kernel code to reflect some of the traps out to supervisor state to be handled by the VMM. The presence of that code unnecessarily complicates the kernel and makes the task of proving correctness more difficult. It also decreases the efficiency of the system.

Recursive Virtual Machine and Security System Structure

0354



## Notes on the Value of Security and Virtual Machines

Gerald J. Popek

July 11, 1973

In other notes [1], a research effort to produce a provably correct operating system subnucleus has been proposed at UCLA. Briefly, the goal of that research is a general examination of the task of constructing secure multiuser systems, but for the short term, this view is much more specific: developing a simple multiuser computer system with elementary sharing facilities that can be certified secure. The approach is described elsewhere, [1,2], but basically the idea is to design and prove correct an operating system subnucleus, called a kernel. Then a virtual machine monitor will be layered over it. In this fashion, the amount of code to be produced and verified is small relative to an operating system, and hence a practical task. Since the virtual machines produced can host operating systems, all of the functionalities provided by such systems, user services and the like, are also available. Hence the result is expected to be a relatively inexpensive, highly secure, useful system.

The virtual machine approach has several values apart from its simplicity, however. General values such as allowing the development of operating systems; providing for concurrent running of multiple operating systems, of many test and diagnostic programs; and aiding in program transferability are commercially recognized; and are discussed for example in a paper by Buzen and Gagliardi [3]. That these points of view apply specifically to the "smaller machine industry" is illustrated

by the following examples.

Jim Hart at the NASA Ames Research Center at Moffet Field has expressed an interest in running multiple operating systems, ANTS (the ARPA Network Terminal System) in one virtual machine and the software for a graphics station in another. A virtual machine approach would allow him to do so without expensive adaptations of ANTS or the graphics software, especially considering that at least one of these is purchased and maintained commercially.

In some university environments, as at the University of Waterloo, there is also a teaching interest. Because of costs, protection problems, and the like, it is generally not possible to give students "hands on" experience with real hardware of significant complexity--that is, of hardware for example which can host modern operating systems. The PDP11/40, the goal virtual machine in the current UCLA proposal, has essentially those characteristics. It is a multistate machine, to which some form of memory management is easily added. Hence the virtual machine system could be made available to students for the purpose of teaching operating systems ideas for example, at the same time that other production operations are going on. For example, both ANTS and student applications are expected to make relatively light demands on CPU time; in the case of teaching, it is primarily the logical completeness that is required. Hence the practical nature of such an application seems very likely.

The preceding sketch illustrates another facet of the utility of "virtual small machines". The incremental cost to provide virtual machines for teaching purposes is likely to be quite small. More executable memory, some secondary storage, some terminals are likely all that is necessary. It might be possible for a computer science department to raise the modest sum necessary to piggyback its needs, while the purchase of any complete system remotely similar in pedagogical value is out of reach. This ability to intimately share the computer resources concurrently without logically interfering may well allow several users to band together to share costs, who could not have justified the purchase of hardware that would otherwise only be theirs for say six hours per day. The logically complete machine has effectively had its cost cut significantly, putting it within reach of more people.

This commercial viability of the virtual machine design is emphasized by IBM's VS1 and VS2 operating systems for VM370.

However, the motivating force behind the UCLA research is not primarily in VM design, but rather concerns security. Interest in secure systems has of course grown significantly in recent years; the military, commercial, and academic markets all show concern. Research is currently being carried out at a significant level. IBM has committed \$40 million, and various government agencies are also spending sizable amounts. Some of this attention has been directed at large centralized operating systems, but as networking continues to grow, the problem proliferates. A major network (WWMCCS) being designed for the

military has security as one of its major concerns, and includes a number of small machines. Commercial networks (and concepts of legal liability for leaks) are already beginning to develop.

To aid in the broader aspects of our research, the design of the security kernel at UCLA is intended to be fairly general, although for the time being some of that generality will not be exploited by the virtual machine monitor. Hopefully, it will shed light on and be adaptable for other security problems.

In closing, note that the hardware modification are necessary for virtualization, not security, although in our case the two are closely tied.



Bibliography

1. Popek, G. J., "Part of a Proposal for the Design of a Certifiably Secure Multiuser Computer Facility" UCLA, May 1973
2. Popek, G. J., "Correctness in Access Control", ACM National Conference, August 1973 (Atlanta, Georgia)
3. Buzen, J., and Gagliardi, U., "The Evolution of Virtual Machine Architecture", AFIPS National Computer Conference, June 1973 (New York, New York)

① → addressing large segments

DRAFT

- Interoffice Memorandum

② virtual ~~mem~~ machines  
③ run as base 11/20

did not acct for protection in state diagram 0243

SUBJECT: SEGMENTATION

DATE: October 14, 1970

TO: Gordon Bell  
Dick Clayton  
Bruce Delagi  
Dave Parnas  
Bill Wulf

FROM: Ad van de Goor  
Larry Wade

25

This memo contains a very preliminary description of a segmentation scheme for the PDP-11 family.

The scheme attempts to accomplish the following:

- 1) Increase the user's virtual address space to  $2^{24}$  bytes = 4 million bytes.
- 2) Give a hardware definition of the "working set" model.
- 3) Implement "sharing" and "protection".
- 4) Allow processes to handle private I/O devices.
- 5) The scheme is usable with or without paging.
- 6) Provide efficient protection between processes and different segments in a process.
- 7) Provide storage efficiency by allowing a large range of segment sizes.
- 8) Allow the user to work in virtual space only.
- 9) Provide a physical address space of  $2^{25}$  bytes = 8 million bytes.

- 10) Provides all programs, <sup>especially p system</sup> to work in <sup>to work in</sup> virtual space
- 11) Allows programs, larger than physical space to be run on small machines
- 12) No loading of process needed.

1.0 Basic Solutions

The address generated by the PDP-11/20 is a 16-bit byte address.

The bigger members of the PDP-11 family will interpret this address (now a virtual address) as a two dimensional (segmented) address as shown in Figure 1. The 16-bit virtual address "VA" is divided into two fields.

- 1) The Working Segment Field "WSF". This 3-bit field determines which of the 8 working segment registers "WSR's" has to be used to form the physical address of the data or instruction. The WSR's contain, among other things, pointers to the beginning (i.e. word 0) of a segment. Appendix A lists some reasons for considering 8 WSR's adequate.
- 2) The Displacement Field "DF". This is a 13-bit field which contains an address relative to the beginning of a segment. This allows for segment sizes of up to 8K bytes.

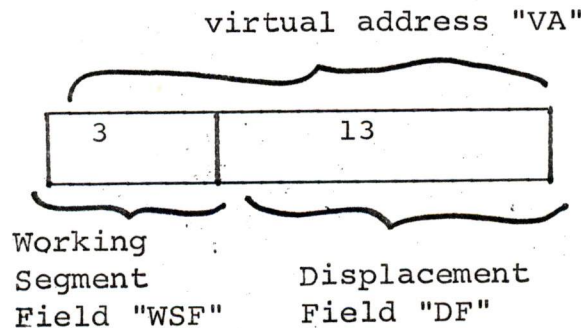


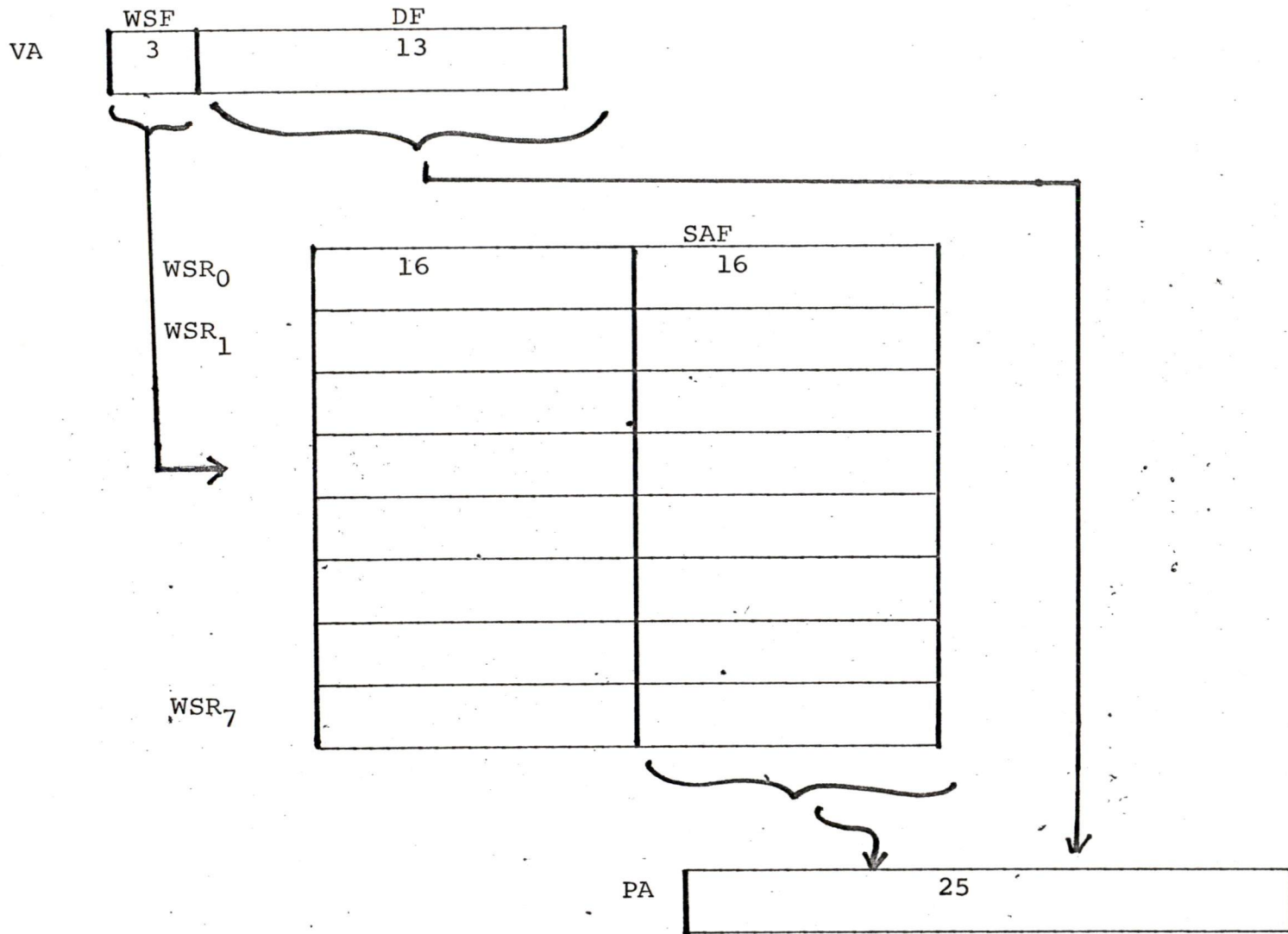
FIGURE 1. Intepretation of a Virtual Address.

The formation of a Physical Address "PA" is shown in Figure 1A.

The (WSF) of the VA is used to address one of the WSR's. The Segment Address Field "SAF" of the addressed WSR is used together with the (DF)<sup>1</sup> to form the PA. The PA is, as will be shown later, a 25-bit byte address.

The 8 WSR's can be loaded with Segment Descriptor Words "SDW's" from the Segment Table "ST" under control of the process.

<sup>1</sup> Note: (X) means "the contents of X".



VA = Virtual Address  
 PA = Physical Address  
 WSF = Working Segment Field  
 DF = Displacement Field  
 WSR<sub>i</sub> = Working Segment Register i  
 SAF = Segment Address Field

Figure 1A. Formation of a Physical Address.

### 1.1 The Segment Descriptor Word

The Segment Descriptor Word is a double word (32 bits) containing information relevant to a particular segment. It contains 5 fields as shown in Figure 2. Detailed descriptions are given in subsequent sections.

1) Use and Validity Field "UVF". This 4-bit field is the only field which is subject to change during execution of a segment.<sup>1</sup> The UVF consists of two sub-fields:

- a) The Use Field "UF". This is a 1-bit field which indicates whether the segment has ever been used.
- b) The Validity Field "VF". This is a 3-bit field describing the validity state of segment. These states will be discussed further on.

2) Software Use Field "SUF". This 4-bit field allows for 16 encoded states four of which are assigned already and describe the movability and size flexibility of a segment.

- a) Move Freely (can be swapped out).
- b) Move in core only (cannot be swapped out).
- c) Do not move (specifically for segments containing I/O addresses).
- d) Do allow size changes (e.g. for the stack segment).

3) Access Control Field "ACF".

This is a 3-bit field describing whether Read, Write and Execute are allowed.

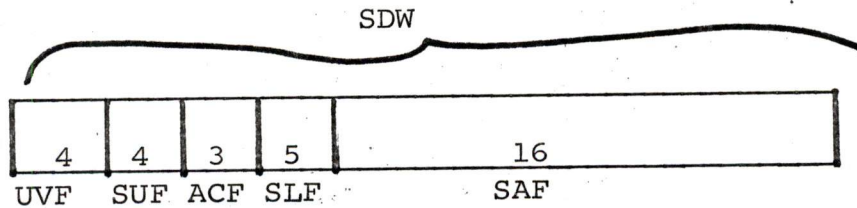
<sup>1</sup> All changes are done under hardware control.

- 4) Segment Length Field "SLF". This is a 5-bit field describing the length of the segment.
- 5) Segment Address Field "SAF". This 16-bit field contains a pointer to physical word 0 of the segment.

1.1.1 The Validity States

The 8 possible validity states are encoded in the validity field VF. These 8 states are listed in Table 1 below. The column "core assigned" indicates whether any physical core has been reserved. The column "core valid" indicates whether the assigned section of core contains valid information. The column "valid copy" indicates whether a backup copy (on the disk/drum) is available.

In order to get a better understanding of Table 1, the state transitions of Table 2 should be consulted.



- SDW = Segment Descriptor Word
- UVF = Use and Validity Field
- SUF = Software Use Field
- ACF = Access Control Field
- SLF = Segment Length Field
- SAF = Segment Address Field

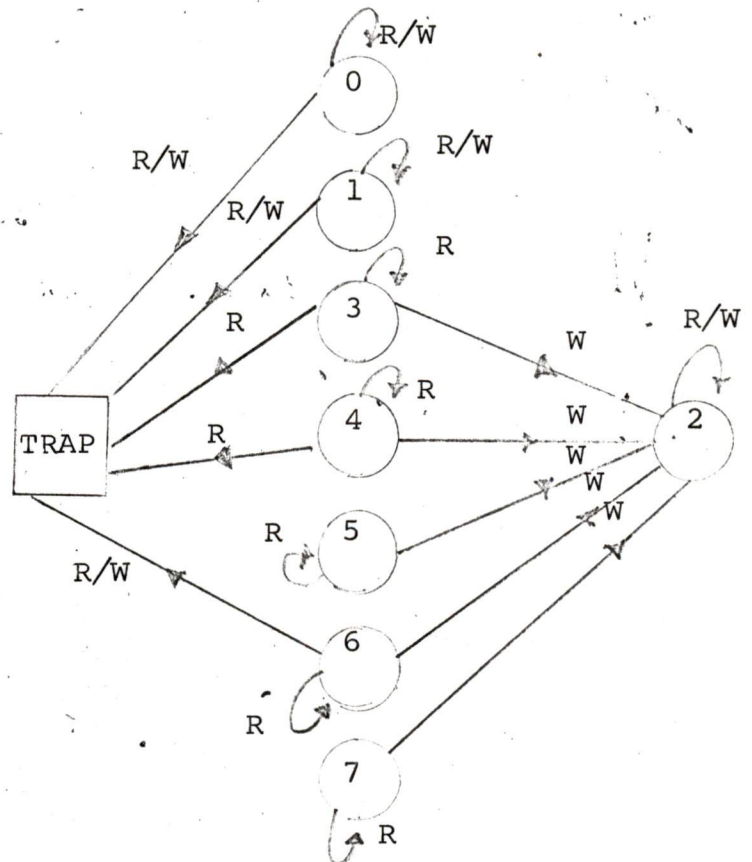
Figure 2. Layout of a Segment Descriptor Word.

	CORE ASSIGNED	CORE VALID	VALID COPY	TRAP AFTER WRITE	COMMENT
0	No	<del>    </del>	No	<del>    </del>	Empty Segment
1	No	<del>    </del>	Yes	<del>    </del>	Segment on backup storage
2	Yes	Yes	No	No	Core copy only
3	Yes	No	No	No	Core reserved for segment
4	Yes	No	Yes	No	Core reserved & backup copy available
5	Yes	Yes	Yes	No	State after transfer from backup
6	Yes	No	Yes	Yes	
7	Yes	Yes	Yes	Yes	

Table 1. The 8 Validity States



	READ OR EXECUTE	WRITE
0	0 Trap	0 Trap
1	1 Trap	1 Trap
2	2	2
3	3 Trap	2
4	4 Trap	2
5	5	2
6	6 Trap	2 Trap
7	7	2 Trap



NOTE: R = Read or Execute  
 W = Write  
 Trap is an ACTION - not a state.

Table 2. Validity State Transition Table & Flow Diagram

1.1.2 The Access Control States

The access control state of a segment is described in the 3-bit access control field "ACF". Table 3, below, shows the 8 states.

	READ	WRITE	EXECUTE	COMMENT
0	/	/	/	This state allows for passing segments
1	/	/	X	Execute only segment
2	/	W	/	Write only segment
3	/	W	X	Useful?
4	R	/	/	Read only data segment
5	R	/	X	"Normal" shared segment
6	R	W	/	R/W Data Segment
7	R	W	X	"Garden Variety" Segment

Table 3. Access Control States

### 1.1.3 The Segment Length

This is described in the 5-bit segment length field "SLF". Small segments are incorporated for storage efficiency and to allow for "private I/O", e.g. to allow users in a time-sharing system to have their own I/O devices. The meaning of the encoded bits is as shown in Table 4 below.

(SLF) = 0 means that the segment descriptor word is void, i.e. it does not describe a segment.

(SLF) = 15 indicating a shared segment, means that the SDW points to a string of SDW's (of length 1 or more) the last one of which contains the actual length of the segment.

For the smaller segments the length is a power of 2. The bigger segments, however, have a size which is a multiple of 256 words for storage efficiency reasons. (See Section 5.0)

The maximum size of a segment can be derived from the 13-bit displacement field of Figure 1. By requiring that any item in the segment be direct, byte addressable the 13-bit displacement has to be interpreted as a 13-bit, byte address, limiting the maximum segment size to  $2^{12} = 4096$  words.

(SLF)	LENGTH OF SEGMENT IN WORDS
0	invalid segment
1	1
2	2
3	4
4	8
5	16
6	32
7	64
8	128
9	} NOT USED
10	
11	
12	
13	
14	
15	shared segment

The numbers 16-31 in the SLF indicate the following lengths:

$$(X-15) * 256 \text{ where } 16 \leq X \leq 31$$

This allows for 256 word pages.

Table 4. Interpretation of the Segment Length.

#### 1.1.4 The Segment Address

The physical address of the first word (i.e. word 0) of the segment is contained in the 16-bit segment address field "SAF". The interpretation of this 16-bit quantity is as follows.

- 1) If  $1 \leq (\text{SLF}) \leq 8$  then the 16-bit quantity is interpreted as a word address. This means that "small" segments (i.e. those with a length between 1 and 128 words) have to be located in the first 65K words of core memory.
- 2) If  $(\text{SLF}) > 15$  then the 16-bit quantity is interpreted as a "page address", i.e. an address of a 256 word quantity. This allows for a maximum physical word address of  $2 \uparrow 16 * 2 \uparrow 8 = 2 \uparrow 24$  words or  $2 \uparrow 25$  bytes.

#### 2.0 Layout of the Segment Table

The Segment Table "ST" contains all the segment descriptor words "SDW's" belonging to a certain process.

The ST, itself, is a segment and its maximum size, therefore, is limited to  $2 \uparrow 13$  bytes = 4K words. Considering the length of a SDW (4 bytes) the ST can contain  $2 \uparrow 11 = 2\text{K}$  SDW's maximally.

This gives a maximum virtual memory per process of: (max. segment size) \* (max. # of segments) =  $2 \uparrow 13 * 2 \uparrow 11 = 2 \uparrow 24$  bytes.

The layout of the ST is shown in Figure 3. The top 16 words of the ST are not used to store SDW's for reasons to be explained later. Currently these words are used as follows.

- 1) The first 8 words (W0-W7) are used to contain Segment Numbers "S#'s". A S# of j in  $W_i$  of Figure 3 indicates that  $WSR_i$  is loaded with  $SDW\#j$ . So the S#'s loaded into W0-W7 of the ST indicate the SDW's the WSR's are loaded with. Because the maximum # of SDW's in a ST is  $2^{11}$  a S# does not have to be bigger than 11 bits.
- 2) Word  $\#10_8$  (W10) contains the stack pointer (R6) when the process is inactive.
- 3) The remainder of the words (W11-W17) are reserved for software use.

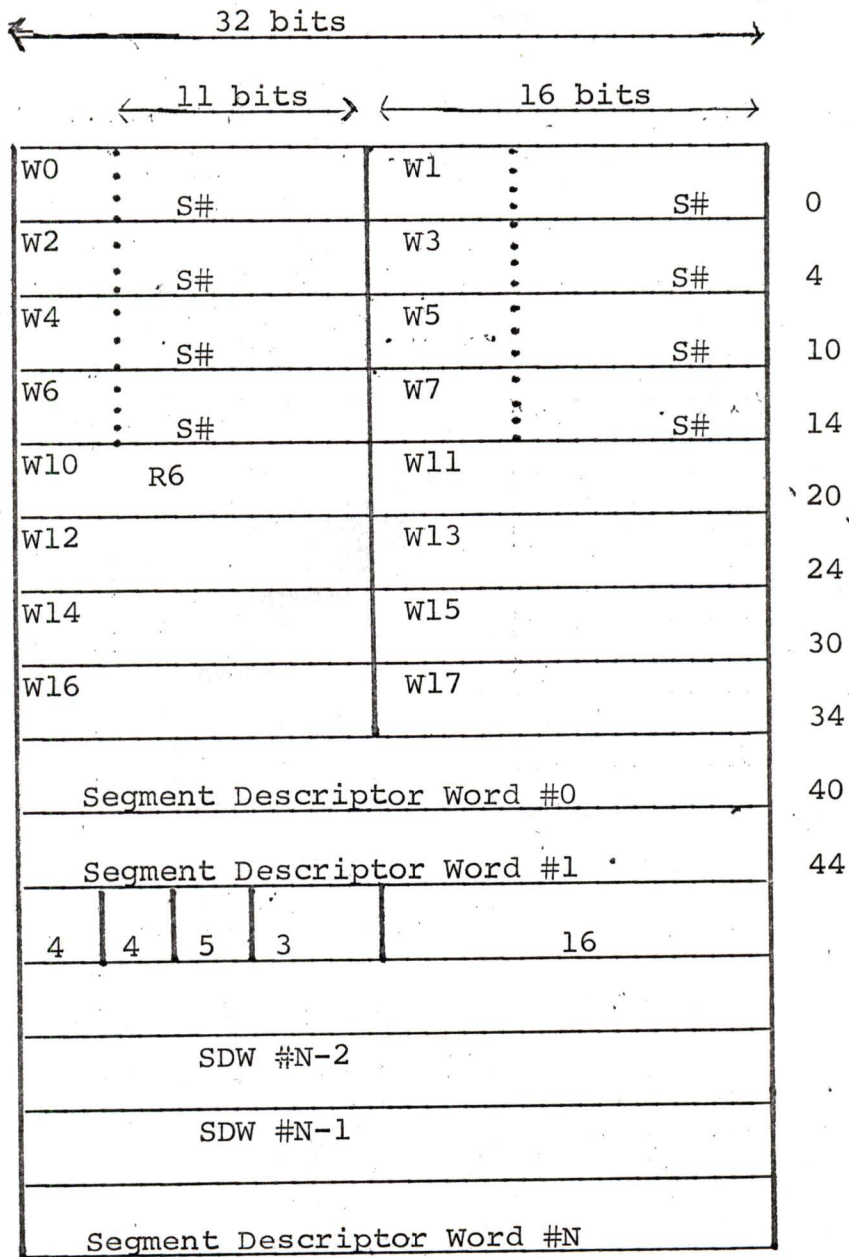
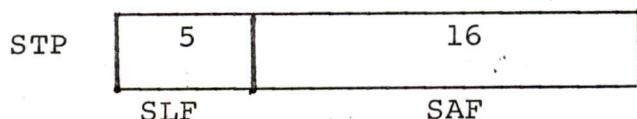


Figure 3. Layout of Segment Table

3.0 Master Control Process "MCP"

This process has the authority to allocate and de-allocate resources in the system. Core management and the creation and deletion of segments belong to its responsibility. This is the only process which is allowed to add, delete, or modify ST's and SDW's. Other processes have no control over their ST and SDW's.

Every process in the system is completely specified by its ST. When a process is in control, a hardware register, the Segment Table Pointer "STP", points to the ST of the process. The STP is a 21-bit register (see Figure 4) and has the same layout as the low order 21 bits of the SDW of Figure 2.



SLF = Segment Length Field  
 STP = Segment Table Pointer  
 SAF = Segment Address Field

Figure 4. Layout of the Segment Table Pointer "STP"

The MCP has a special segment, called the Segment-Segment Table, "SST", which contains Segment Table Descriptor Words "STDW's" pointing to all processes in the system, including the MCP. The location of the SST is known to the MCP because



it is one of its segments. Figure 5 shows the layouts of the different tables. The SST contains M STDW's indicating that there are M processes in the system.

Process 0 "Pr.0" is the MCP. Note that the SST is the first segment in the MCP's ST and is therefore under complete control of the MCP. It is quite obvious that the SST should not be a shared segment. The process in control is Pr. 1 because the STP (Segment Table Pointer) points to it.

it is one of its segments. Figure 5 shows the layouts of the different tables. The SST contains M STDW's indicating that there are M processes in the system.

Process 0 "Pr.0" is the MCP. Note that the SST is the first segment in the MCP's ST and is therefore under complete control of the MCP. It is quite obvious that the SST should not be a shared segment. The process in control is Pr. 1 because the STP (Segment Table Pointer) points to it.

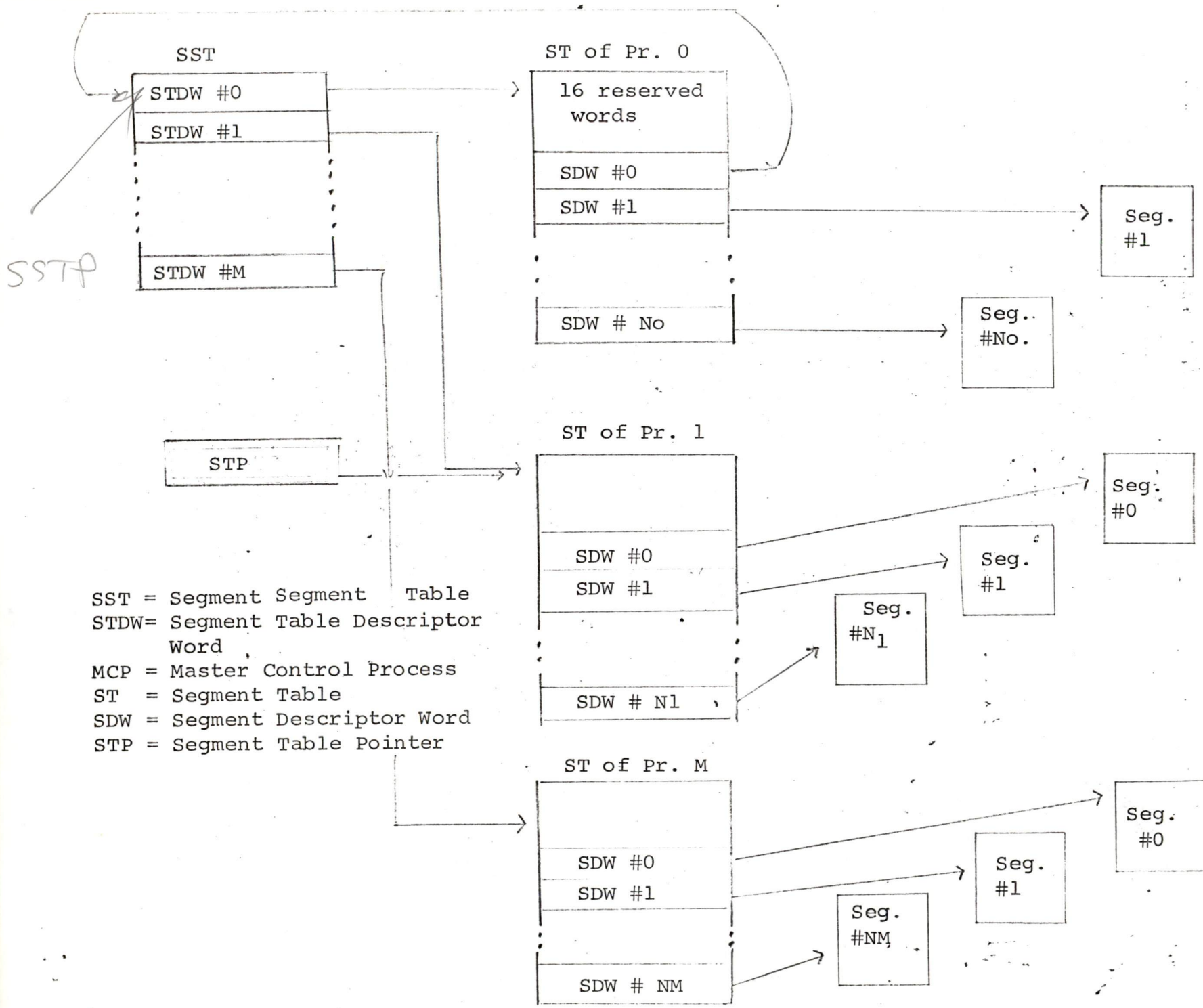


Figure 5. Layout of SST and ST Tables.

#### 4.0 Interruptability

Fast interrupt response is a requirement especially because the machine might be used in real-time applications. The state of a running process is determined by the following:

- 1) The program counter "PC"
- 2) The stack pointer "SP"
- 3) The program status word "PS"
- 4) The location of the ST which is the (STP)
- 5) The contents of the 8 WSR's
- 6) The contents of the accumulators "AC's"

The interrupt response time can be divided into two groups:

- 1) The time needed to save the current status, called "Save Time".
- 2) The time needed to set up the new status, called the "Restore Time".

#### 4.1 Reduce Save Time

In order to reduce the Save Time, the following two facilities are introduced:

- 1) The saving of the AC's is done optionally through Save AC's bit "SAC" in the PS word (see Figure ).

2) The saving of the WSR's is made not necessary because of the scheme discussed below. In order to allow for this, two requirements have to be satisfied.

- a) Duplicate copies of the contents of the WSR's have to be available in core memory.
- b) Knowledge as to which SDW's are loaded in which WSR's has to be available to allow for a correct restore operation.

Part a) is satisfied by guaranteeing that the (WSR's) are always the same as the corresponding SDW's in the ST. This can be done relatively easily at the expense of very little overhead because the SDW's do not change very often when they are loaded in the WSR's. Only 4 bits of the SDW can change while a "segment is working" (i.e. loaded in a WSR). These are the Use and Validity bits (see Section 1.1)

- a1) The Use bit changes, at most, once while the segment is working, namely when it is used the first time.
- a2) The validity bits can change only a few times after which they end up in a stable state or cause a trap (see Table 2).

Because the changes mentioned under a1 and a2 are so infrequent, they are made in the WSR and the corresponding SDW simultaneously (i.e. in a non-interruptable sequence).

Part b) is satisfied by reserving in the ST 8 words which contain the SDW #'s loaded in the corresponding WSR's (see Figure 3).

The additional requirement in loading a WSR is that the SDW # has to be loaded in the corresponding entry in the ST.

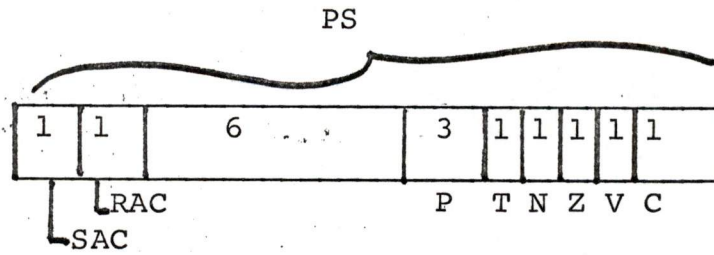
#### 4.2 Reduce the Restore Time

The restore time can be reduced by

- 1) Conditionally restore the AC's. This is done through the Restore AC bit "RAC" in the PS word (see Figure 4).
- 2) Selectively restore the WSR's. This is done through an 8-bit mask, the Restore WSR mask "RWSR", in the Virtual Memory Descriptor "VMD" of Figure 5.
- 3) Conditionally Change Address Space.

This is done through the change address space bit "CAS" in the VMD. The exact operation of this bit needs some more work.

- 19 -



SAC = Save AC's

RAC = Restore AC's

P = Priority

T = Trace

N = Negative

Z = Zero

V = Overflow

C = Carry

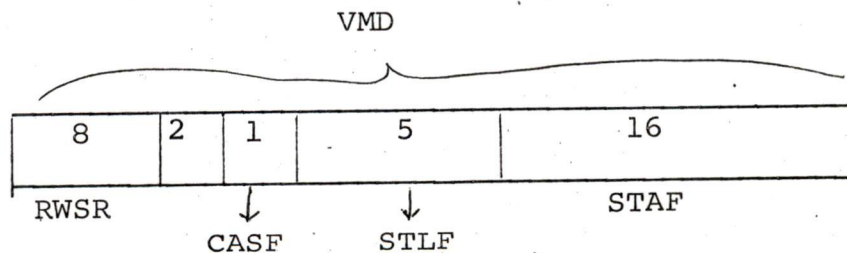
PS = Program Status Word

Figure 4. Layout of the Program Status Word

5.0 Interrupts and Traps

The interrupt and trap vectors, as currently exist on the PDP-11/20, have to be redefined in order to guarantee efficient operation. Instead of the "old" interrupt/trap vectors consisting of a PC and a PS word, we will now have a Virtual Memory Descriptor "VMD", see Figure 5. These VMD's are located in physical core, they are also 2 words long, and can therefore replace the interrupt/trap vectors.

The VMD contains all the information necessary to start a process operating in virtual memory, rather than interrupt/trap handlers operating in physical address space. The above feature allows processes to handle their own interrupts/traps.



- VMD = Virtual Memory Descriptor
- RWSR = Restore Working Segment Register Mask
- CASF = Change Address Space Field
- STLF = Segment Table Length Field
- STAF = Segment Table Address Field

Note: The STLF is similar to the SLF.  
The STAF is similar to the SAF.

Figure 5. Layout of the Virtual Memory Descriptor



The saving of the state of an interrupted process consists of the following steps.

- 1) Test the SAC bit in the PS (see Figure 4) and conditionally push the AC's on the stack of the interrupted process.
- 2) Push the PC and PS on the stack of the interrupted process.
- 3) Store SP in W10<sub>8</sub> of the ST of the interrupted process.
- 4) Invalidate the WSR's by clearing them. This is to safeguard the interrupting process from accidentally being able to access the interrupted process's VM space.

Restoring the state of the interrupting process consists of the following steps.

- 1) Store (STP) in a temporary location "TSTP".
- 2) Pick up VMD from interrupt/trap vector and store it in the STP.
- 3) Store (TSTP) in W12 and W13 of the ST of the interrupting VM space.
- 4) Restore R6 from W10 of the ST of the interrupting process.

- 5) Pop PS and PC.
- 6) Test RAC bit of popped PS and conditionally restore the AC's by popping them from the stack.
- 7) Selectively restore the WSR's under control of the RWSR mask in the VMD.

## 6.0 Indirect Addressing and Shared Segments

Instruction as well as data addresses are virtual addresses "VA's". Because the (WSF's)<sup>1</sup> can be different for instructions and data, instructions and addresses can come from different segments. The two possible cases for direct addressing are shown in Figure 6.

### 6.1 Indirect Addressing

Indirect Addressing is handled in a way very similar to direct addressing. Now, however, three VA's are generated: 1 for the instruction; 1 for the indirect address; and 1 for the data. This leads to the five possible addressing cases shown in Figure 7.

<sup>1</sup> See Figure 1 and 1A.

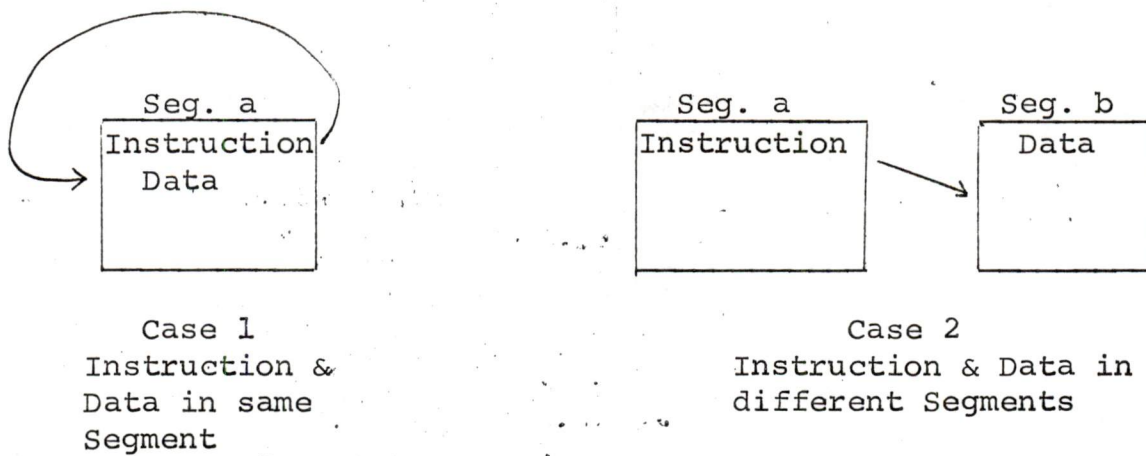


Figure 6. Direct Addressing Cases

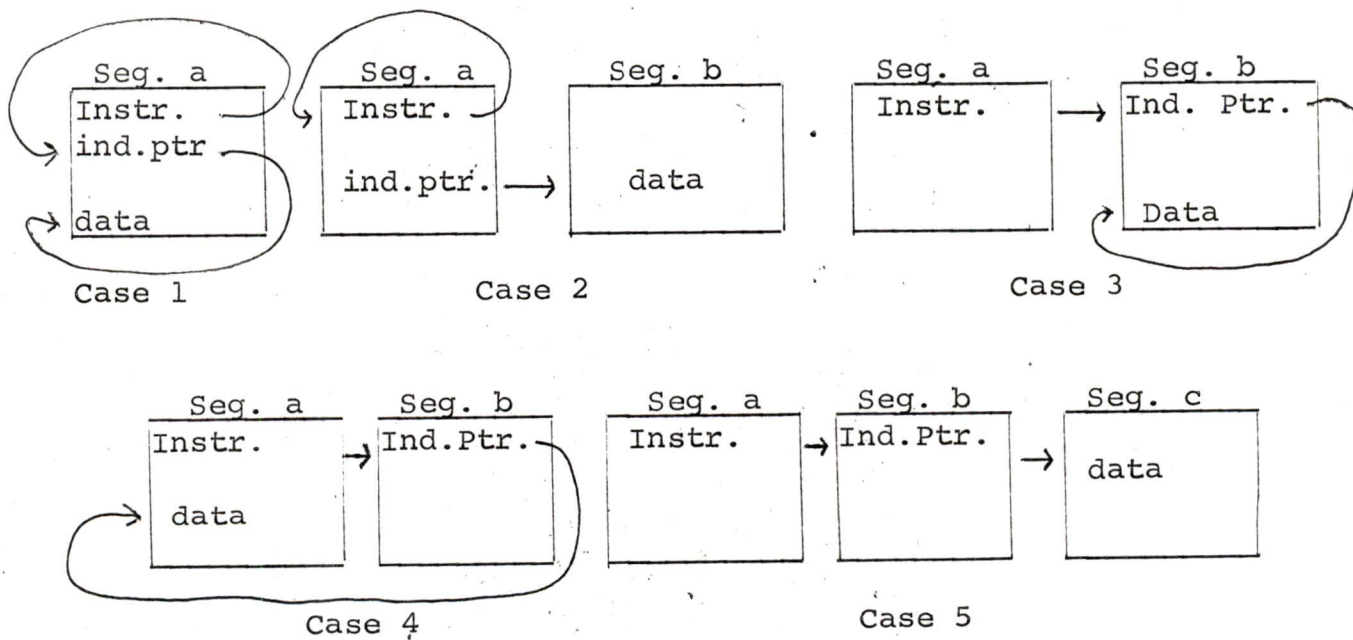


Figure 7. Indirect Addressing Cases

0291

TITLE: Protection & Relocation for the PDP-11/40

PDP-11/40 Technical Memo #29

AUTHOR: Ad Van de Goor  
Robert Gray  
K.C. Huang

DATE: March 8, 1971

REVISION: C OBSOLETE: Memo #29 Feb 24, 1971

INDEX KEY: Relocate/Protect  
Segmentation

DISTRIBUTION: PDP-11/40 Group  
Van Diehl  
Ken Stapleford  
Hank Spencer  
John Hittel  
Gordon Bell: CMU  
David Parnas: CMU  
Nick Pappas

## ABSTRACT

The memory mapping or relocate protect scheme proposed for the 11/40 is essentially a segmentation scheme. It is designed in such a way that it is upwards compatible with the 11/60 scheme and does not presume or dictate a particular use.

The scheme provides for a physical address space of  $2^{18}$  bytes and a maximum active virtual address space of  $2^{16}$  bytes. The total virtual address space (i.e. the length of the segment table) is determined under software control.

The active virtual address space can be divided into 8 segments. The size of each segment can vary from 1 to 16 pages (1 page = 256 words). Protection and relocation are provided on the segment level.

An implementation of the relocate/protect option is proposed. It will fit on two QUAD boards. 16 ACTIVE SEGMENT registers and 4 STATUS registers will be provided. A hardware aid is proposed to help recover from NON-RESIDENT faults. Appendices contain suggested recovery routines.

## PREFACE

This document is intended to provide a detailed description of the relocate/protect scheme being designed for the PDP-11/40. It further is being used as a "working" set of engineering specifications. As such it will ultimately become the text for the "Engineering Specifications" and the "Maintenance Manual".

New In Revision C are:

1. Segment Length Field and Segment Address Field descriptions.
2. Section 1.2 "Segment Fault Action"
3. Use of SSR0 bit 7 to enable/disable Memory Management Trapping.
4. Storing of Trap Vectors in SSR2
5. Section 7.2 Address bits 16 and 17
6. Sections 9.14 and 9.15 Console communication
7. Section 7.3 Address Assignments

## 1.0 INTRODUCTION

This memo is intended to be a preliminary description. Many details have yet to be worked out. It is our intention to revise this document from time to time with the additional detail.

Because this is a living document, it is extremely important that we know about errors in writing, detail, or most importantly, in plan. We expect there will be questions and contentions raised by this technical description. We remain always ready to listen and try to understand questions you may raise. If we are going a wrong direction, NOW is the time to change; we earnestly request your criticisms and suggestion, all deserve an honest reply.

## 1.0 BASIC SOLUTION

The addresses generated by the PDP-11/20 are 16-bit byte addresses. On the 11/40 with segmentation these addresses are considered Virtual Address "VA's". A VA is considered to be a two dimensional address as shown in Figure 1. It consists of:

1. The Active Segment Field "ASF". This 3-bit field determines which of 8 Active Segment Registers "ASR's" has to be used to form the Physical Address "PA".
2. The Displacement Field "DF". This is a 13-bit field which contains an address relative to the beginning of a segment. This allows for segment sizes of  $2^{13}=8k$  bytes.

The formation of a physical address "PA" is shown in Figure 2. The Active Segment Field "ASF" of the Virtual Address "VA" is used to address one of the eight Active Segment Registers "ASR's". The Segment Address Field "SAF" of the addressed ASR is used together with the Displacement Field "DF" to form the PA.

The ASR's can be loaded with Segment Descriptor Words "SDW's" under program control.

### 1.1 THE SEGMENT DESCRIPTOR WORD "SDW"

A SDW is a 16-bit word containing information relevant to a particular segment. A SDW consists of 3 fields, see Figure 3, which are described below.

1. The Access Control Field "ACF". This 3-bit field contains the access rights, called KEYS, of a process with respect to a particular segment. The following keys have been assigned already.
  - a. Non Resident "NR" (key=0). Any access to such a segment will cause an abort. The reasons why a segment is

March 8, 1971

non-resident can be many fold, e.g. not swapped in yet, segment does not exist, etc.

- b. Resident Read Only and Trap "RROT" (key=4). This is essentially a read only segment, a trap upon read could be desired to gather statistics about the use of the segment, etc.
- c. Resident Read Only "RRO" (key=5). An attempt to write in an RRO segment will cause an abort.
- d. Resident Read Write and Trap "RRWT" (key=1). In this segment essentially Read and Write operations are allowed, the trap upon a read or a write access could be desired for statistics gathering (see b above).
- e. Resident Read Write and Trap when Write "RRWTW" (key=2). This is a segment where essentially read and write operations are allowed. When a write operation is done, a trap will occur. This could be used to indicate that no valid backup copy exists any more.
- f. Resident Read Write and Written Into "RRWW" (key=3). This is a segment in which read and write operations are allowed. The "written into" could indicate that no backup copy is available.

## 2. SEGMENT LENGTH FIELD (SLF)

This four bit field specifies the number of 256 word pages in the segment. From 1 to 16 pages may be specified in this field.

Note that "0" in the SLF specifies 1 page and 15 in the SLF specifies 16 pages.

The four bits of the SLF are compared with the four high order address bits in the DISPLACEMENT FIELD from the processor to detect SEGMENT LENGTH errors. A SEGMENT LENGTH error exists if the SLF is smaller than the four high order bits of the DISPLACEMENT FIELD.

## 3. SEGMENT ADDRESS FIELD (SAF)

This field of 9 bits in combination with the 13 bit DISPLACEMENT FIELD from the processor form the 18 bit physical address. This process is shown schematically in Figure 3.

Since the 11/40 segmentation scheme allows segments of different sizes (256 word to 4096 words in 256 word increments), a method must be provided for creating segment boundaries at 256 word intervals rather than at 4K intervals

March 8, 1971

If memory is to be fully used. This is implemented by adding the 4 high order address bits in the DISPLACEMENT FIELD into the SAF as shown in Figure 3. This technique allows a VIRTUAL address 0 of a 256 to 4K word segment to be physically located at any 256 word boundary in the memory system. This allows segments of varying lengths to be placed in physical core with no "gaps" between them.

## 1.2 SEGMENTATION FAULT ACTION

If the processor sends an address that is: non-resident, exceeds segment length or violates "read only" restrictions, the operation is aborted before the memory operation occurs. No memory reference occurs and an ABORT signal is sent to the processor. No further memory references can occur until the processor acknowledges the ABORT signal with a segmentation fault acknowledge (SEGACK) signal. This process is handled automatically in the 11/40 hardware.

If a segment access is made that requires a memory management trap, the processor is notified by a signal (MEM MGMT FLAG). The processor acknowledges this flag only at the end of an instruction. The acknowledgement is signal SEG ACK. See section 6.0 for a full discussion of this operation.

The MEM MGMT FLAG has priority over "T bit" traps.

## 2.0 DEGREE OF HARDWARE AIDS ON SEGMENTATION FAULTS

When a segment fault occurs, if it is a "non-resident" fault, the system must bring the missing segment into core then restart the task. Since such "non-resident" faults can occur within instructions, some means of either restarting the instruction in the middle or backing the instruction up and restarting at the beginning must be provided.

Restarting the instruction in the middle is rejected as there are a number of internal inaccessible registers whose value can be neither saved nor restored for "middle of instruction" restarts.

An investigation of backing up the effects of a partially executed instruction shows that only auto-increment and auto-decrement operations effect the registers and that to restart, it is sufficient to reverse the effects of any auto decrement/incrementing done by the partially completed instruction before restarting at the "instruction address." Further it has been determined that a maximum of two registers are changed during a given instruction.



March 8, 1971

The next decision must determine the degree of help provided by the segmentation hardware in correcting register values. Clearly a range of help is possible—from fully automatic correction to merely indicating how far the instruction got in SOURCE/DESTINATION calculations.

Previous experience with the KT11 (11/20 Paging option) [which merely provided the EXEC with a bit indicating whether the SRC/DST had been completed.] indicated additional help was needed.

A fully automatic scheme was rejected because it was regarded as "overkill" and the intimate connection to the processor ROM that would have been required was not thought desirable.

The proposed scheme is implemented in Segment Status Register "SSR #1". It allows the EXEC to correct all registers modified in less than 40 memory references. It's hardware implementation is straight forward and does not control the processor ROM. Appendix A is a sample recovery routine. Basically it provides the EXEC with the register number changed and a description of how much and in what direction it was changed.

### 3.0 SEGMENT STATUS REGISTER #0 (Segmentation status and error indicators)

SSR0 will contain error flags and the "virtual segment number" causing the error as well as other status flags. The register will be organized as in Figure 4.

Bits 15-12 are the error flags. They may be considered to be in a "priority queue" in that "flags to the right" are ignored! That is a "non resident" fault service routine would ignore segment length, access, and memory management flags. A "segment length" service routine would ignore access and memory management faults, etc. Note that the word format is convenient for "ROTATE" and "BRANCH" breakout sequences.

Bits 11-8 are presently spares. They may be assigned uses in a "Debugging Option" allowing "hardware breakpoints."

Bit 7 enables MEMORY MANAGEMENT trapping. If bit 7 is 0, Segment Status Register 3 will keep track of which segment references requested memory management traps, but the "trap signal (MGMT TRAP FLAG) is not sent to the processor. When bit 7 is made 1, the next time a segment whose ACF calls for a MEM MGMT trap is

March 8, 1971

referenced, MGMT TRAP FLAG is sent to the processor.

Bit 6 specifies a MAINTENANCE mode in which only the DESTINATION fetch/store is relocated and protected. It is expected to be useful for diagnostic program development.

Bit 5 indicates that the instruction has completed. It will be set when non-instruction (traps) memory references are made. This provides the error handling routine a way of finding that the last instruction will not have to be repeated on restart.

Bits 4-1 give the virtual segment number of the reference causing a fault. Bit 4 on a "1" indicates USER segments. Note that this field is positioned for convenient relative addressing on the segment number.

Bit 0 this bit controls whether virtual addresses are operated upon by the segmentation hardware. This bit will be cleared by the processor signal INIT.

SSR0 bits 0,5,6,12-15 can be written into as a word. Other bits (1-4) will not contain valid information after writing into SSR0 until the next Trap occurs.

### 3.5 DESTINATION ONLY SEGMENTATION

Experience with debugging KT11 diagnostics has shown that a DESTINATION MODE ONLY relocation & protection is desirable. It is proposed that SSR1 bit 6 be used for that purpose, and that segmentation be controlled by the following Boolean equation:

$$\text{segmentation} = \text{SSR1}\langle 0 \rangle (1) + \text{SSR1}\langle 6 \rangle (1) * \text{DST}$$

The amount of logic required to implement destination only segmentation is estimated to be one 14 pin chip.

### 4.0 SEGMENT STATUS REGISTER #1

This register keeps track of any AUTO INCREMENTING/DECREMENTING of the general registers, i.e. PUSHES and POPS. The register is cleared at the beginning of each instruction fetch. Whenever a general register is either pushed or popped, the register number and the 2's complement number of bytes the register got modified are written into SSR1. The low order byte is written first. See

## Figure 4.

When the instruction is completed, there is no need to restart the instruction even if there may be segment faults before next instruction fetch. The register SSR1 is cleared and any stack modification from that time till next instruction fetch is recorded.

Register numbers will be recorded "MOD 8". It will be up to the recovery service routine to determine which set of registers was modified by the state of the processor Status Word at the time of trapping.

The existence of SSR1 will speed up the recovery from a non-fatal segment fault by 2-3 times (a saving 100-200 uS) and requires only 1/5 to 1/10 of core (75 words) of the case where the only information available to the EXEC are the SRC/DST bits. Although non-resident segment faults are not expected to occur at a rate more than one per two mill-seconds, thus making the 100-200 uS saving in overhead perhaps seem insignificant; the implementation of SSR1 can probably be justified in terms of the amount of core saved (500 words) and the small amount of hardware that has to be added. It now seems that the 4\*256 ROM is required in any case to eliminate some of the logic needed to decode the relevant CP states. The register itself requires nothing extra and the control logic is estimated to be five 14-16 pin chips. A read only memory (ROM) will be used to detect processor states during which AUTO INCREMENTING or DECREMENTING occur. The ROM will "track" the processor Control Memory ROM. That is, the address being sent to the processor ROM will be bussed to the segmentation modules, where it will be used to select the same numbered location in a ROM located in the segmentation option. One bit of the ROM will indicate that an "AUTO" change is occurring. Another bit will indicate the direction (INC/DEC) of the change.

SSR1 is READ ONLY. It cannot be written into.

## 5.0 STATUS REGISTER #2

SSR2 will contain the 16 bit virtual instruction address. It will be loaded at the beginning of each instruction fetch. It will be loaded with the Trap Vector (TV) address at the beginning of an interrupt or a "T bit" trap.

SSR2 is READ ONLY. It cannot be written into.

March 8, 1971

## 6.0 MEMORY MANAGEMENT AND SSR #3

Three of the ACF keys cause a trap under the condition that the segment was referenced. This will be used for swapping control and other memory management functions. Since these are not "error" faults, there is no reason to "abort" the operation when a MEMORY MANAGEMENT trap occurs. It is sufficient to merely "note" that the trap occurred. A convenient "unit sampling period" is the "instruction." During a single instruction several (5 max in an indirect BINARY instruction) different segments may cause STATISTICAL traps. Each of these must be retained until the trap is serviced. In addition since the move to user and move from user instructions operate in both USER and EXEC virtual address space, some combination of 16 different segments can be at fault. A MEMORY MANAGEMENT trap causes a bit in the SSR3 to be set. EXEC Segment 0 will set bit 0, USER segment 7 will set bit 15 of the register. See Figure 4.

Once a bit is set in SSR3 signal MEMORY MANAGEMENT TRAP will send MEM MGMT TRAP to the processor. This will cause a trap at the end of the current instruction. The request will be signified by bit #12 in SSR0 becoming a "1". During the service routine the bits in the SSR3 must be reset. (Note additional bits may be set during the service routine.) At the end of the service routine, SSR0 bit 12 must be cleared. This "rearms" the MEMORY MANAGEMENT error logic and enables the "T" bits to be set again. An "ABORT" error occurring on a fetch or the location being a register that is internal to the segmentation option will prevent a Memory Management bit from being set, even if the access key calls for a MEM. MGMT. TRAP. SSR3 can be written into as a word.

Note that if bit 7 in SSR0 is "0" [ENABLE MEMORY MANAGEMENT TRAPS], no flag is sent to the processor when the MEM MGMT access keys are detected. The proper bit in SSR3 is set however. This allows "periodic" EXEC memory use checking at EXEC discession rather than checking forced by hardware.

## 7.0 DETAILED SYSTEM SPECIFICATIONS

## 7.1 CLEARING SEGMENT STATUS REGISTERS FOLLOWING TRAP

At the end of the segmentation fault service routine certain bits in SSR0 must be cleared to "rearm" the segmentation trap logic.

Bits 12-15 and 1-5 must be cleared to resume segmentation error checking. On the next memory reference following clearing these bits the other SSR's will continue monitoring the computer operation. SSR2 will be loaded with the next instruction address. SSR1 will get register information. If a new

statistical trap were detected, SSR3 will be loaded.

## 7.2 ADDRESS BITS 16 AND 17

- a. SEGMENTED-bits 16 and 17 are specified by the contents of the Active Segment Register. (Max Memory=128K)
- b. UNSEGMENTED-bits 16 and 17 are equal to 0 unless 13,14,15 are 1, then bits 16 and 17 are automatically made 1. (Max Memory = 32k)
- c. CONSOLE ACCESS-when console ADDRESS SELECT switch is in "PHYSICAL" position, bits 16 and 17 are controlled explicitly by switches 16 and 17 respectively. (examines and deposits only.)

## 7.3 REGISTER ADDRESS ASSIGNMENTS

777630	SSR0
777632	SSR1
777634	SSR2
777636	SSR3
777640	USER ASR 0
777642	USER ASR 1
777644	USER ASR 2
777646	USER ASR 3
777650	USER ASR 4
777652	USER ASR 5
777654	USER ASR 6
777656	USER ASR 7
777660	EXEC ASR 0
777662	EXEC ASR 1
777664	EXEC ASR 2
777666	EXEC ASR 3
777670	EXEC ASR 4
777672	EXEC ASR 5
777674	EXEC ASR 6
777676	EXEC ASR 7

## 8.0 FLOATING POINT PROCESSOR

Several considerations regarding the interaction with the Floating Point Unit (FPU) remain unresolved.

Floating point instructions cause general registers to be auto Inc/dec by 4 or 8 bytes. These values will be stored in SSR1.

March 8, 1971

## 9.0 INTERFACE SIGNAL SPECIFICATIONS

## Introduction

The internal registers of the segmentation option will interface with the processor through the "fast bus". In addition a number of other signals will be passed between the two units. It is intended that agreements on the characteristics of these interfacing signals will be catalogued in this section.

## 9.1 PROCESSOR/SEGMENTATION INTERFACE

## 9.1.1 Fast Bus Interface

## 1. VIRTUAL ADDRESS LINES - [VA0 through VA15] (16 lines)

16 lines sending the virtual address from the processor or floating point processor to the segmentation option. A true signal will be ground. These signals will begin and end with the leading edge of "T1".

## 2. BUS START - BUST (1 line)

A pulse indicating an address is on the VA's. A true condition will be ground. This signal will begin at the leading edge of "T2".

## 3. FAST (1 line)

A signal coming from "memory" that can respond in 150ns from BUS START at processor (250ns in segmented mode). The segmentation will "pass" FAST from the semiconductor memory and generate FAST when one of the 20 internal registers of the segmentation option is addressed by the processor. A true condition will be ground. The pulse will have a nominal width of n.s.

## 4. CONTROL (C1) (1 line)

This will be used to differentiate READ and WRITE memory cycles. [No byte operations will be allowed in registers internal to this option.]

## 5. INTERNAL BUS DATA (16 lines)

Data to be read from the 4 status registers in the segmentation option will be placed on these lines. True value is ground.

## 6. BBR DATA (16 lines)

These are used by the processor to transfer data to one of the 20 internal registers. True value is ground. Signals

are made valid at leading edge of "T1".

7. BUS END - BEND (1 line)

Within 125ns (225ns with segmentation) after BUST, This tells to stop processing the address. No flag/error bits should be set when BEND is decoded as it indicated an aborted fetch. True value is ground.

8. PHYSICAL ADDRESS OUT (18 lines)

The physical address generated by segmentation option.

9. BUST OUT

A delayed (100 ns) and enabled version of Bus Start received from the processor. This signal is generated only if no error occurred on the access attempt.

9.1,2 PROCESSOR SIGNALS NEEDED BY SEGMENTATION

1. ROM ADDRESS (8 lines)

The 8 bit ROM ADDRESS will be needed by segmentation to decode "pushes",+"pops" to registers for SSR 2. It will also be used to decode DESTINATION FETCH, INSTRUCTION DONE. All signals will be true on a +3V logic level.

2. REGISTER NUMBER (3 lines)

The 3 bit register number of the register being pushed or popped.

3. CONSTANT (4 lines)

A 4 bit positive number that is to be added or subtracted from the register.

4. USER/EXEC Address (1 line)

Not Status Register bit 15, but a signal that indicates whether the address on the fast bus is a USER or EXEC address. Required because of INTERmode instructions!

5. CLOCK - (several lines)

To be used for various timing and synchronizing functions within the segmentation option.

6. LOAD IR - signal indicating this is an INSTRUCTION FEICH.

7. SEG ACK (1 line)

March 8, 1971

A pulse from the processor that lowers the ABORT and MEM MGMT trap flags. Once ABORT or MEM MGMT is true no further memory references can occur until SEG ACK is received.

8. INIT (1 line)

A power clear signal that occurs during powerup and when the console START switch is depressed.

9.1.3 SIGNALS NEEDED BY THE PROCESSOR

1. ABORT (1 line)

Made true when segmentation fault occurs. Will be cleared by SEG ACK.

2. MEM MGMT TRAP (1 line)

A level will be used for MEM MGMT traps. It will be cleared by SEG ACK.

3. SEGMENTED/UNSEGMENTED

TRUE LEVEL SIGNAL WHEN SEGMENTATION OPTION IS IN PLACE AND BIT 0 OF SSR 1 is a 1.

9.1.4 CONSOLE SIGNALS REQUIRED BY SEGMENTATION

1. EX ADDRESS 16 and 17 (2 lines)

These are the output of console switches 16 and 17.

2. CONSOLE INHIBIT SEGMENTATION (1 line)

Tells the segmentation option to use the values of EX ADDRESS 16 and 17 to form the high order physical address bits.

3. VIRTUAL (1 line)

When True sends the VIRTUAL address to the console ADDRESS lamps. When false sends the physical address.

9.1.5 SEGMENTATION SIGNALS REQUIRED BY CONSOLE

1. CONSOLE ADDRESS (18 lines)

Output of a multiplexer which provides the console with either the PHYSICAL or VIRTUAL address.

10.0 LOGIC REQUIREMENTS FOR SEGMENTATION



March 8, 1971

Our estimate is that the RELOCATION/PROTECTION can fit on two "QUAD" boards. The following chips are presently planned to be used:

DEC#	DEV	PINS	DESC	QUANTITY	COST	TOTAL
19-09930	7405	14	Hex INVT OPEN COL	3	.22	
19-10155	7408	14	Quad AND	2		
19-10091	7437	14	Quad NAND Buf	1		
19-09050	7475	14	Quad D FLIP FLOP	4	1.00	
19-09937	74153	16	Dual 4 to 1 Mux	8	.97	
19-09814	74154	16	4 to 16	1	1.91	
	74157	16	Quad 2 to 1 Mux	3		
	74174	16	Hex D Flip Flop	2		
	74175	16	Quad D Flip Flop	3		
	74187	16	ROM	1		
19-09056	74H00	14	4 2IN NAND	4	.24	
19-09931	74H04	14	6 INVT	8	.28	
19-09057	74H10	14	3 3IN NAND	2	.24	
19-09267	74H11	14	3 3IN AND	3	.24	
19-09058	74H21	14	2 4IN AND	1	.24	
19-09059	74H30	14	8 IN NAND	1	.24	
19-05586	74H40	14	2 4IN NAND BUF	6	.26	
19-09063	74H55	14	AND OR INV	1	.24	
19-09667	74H74	14	2 D FLIP FLOPS	1	.48	
	74s03	14	INVT OPER COL	1		
	74S04	14	INVT	8		
	74S15	14	3 3 IN NAND OPER COL	1		
	74S64	14	AND OR INVT	2		
	74S158	16	Quad 2 to 1 Mux	4		
	74S181	24	ALU	4		
	74182	16	LOOK AHEAD	1		
19-10087	4015	16	4 bit D with Set	5	.87	
	3101(Intel)	16	35 ns Max	4		

-----  
Total 84

March 8, 1971

## APPENDIX A: EXAMPLE OF "NON-RESIDENT" RECOVERY PROGRAM

## TRAPVECTOR:

```

:EXEC,REGSET=0, NONINTERRUPTIBLE
HERE:  BIT      #B14,2(SP)      /WHICH REG WAS IN USE?
      BEQ      PUSHLDS        /WAS NORMAL
      BIS      #B14,STATUS     /WAS DEDICATED, ACTIVATE
                                      /DEDICATED
PUSHLDS: PLODM      ,R6-R0     /PUT OLD REG ON STACK

      MOV      SSR2,R2
AUTOCK: BIT      #B7,6,5,4,3,R2 /TEST FOR A REGISTER
                                      /AUTO INC/DEC
      BEQ      DONE           /NONE CHANGED
      MOVVB   R2,R1          /WAS CHANGED, GET
                                      /PARAMETERS
      MOVVB   R1,R0          /SAVE ADDITIONAL COPY
      ASL     R1             /MOVE REG# TO WORD
                                      /BOUNDRY
      BIC     #377741,R1     /CLEAR OUT OTHER BITS 8,
      ASH     R0,#3         /MOVE CONSTANT INTO
                                      /PROPER POSITION
      ADD     R6,R1         /R1 GETS LOC IN STACK
                                      /WITH REG. VALUE
      ADD     R0,(R1)       /CORRECT OLD REG VALUE
      CLRB   R2
      SWAB   R2            /SETUP FOR SECOND TS?
      BR     AUTOCK
ERRRYP: MOV      SSR0,R0
      BIT      #B15,R0
      BEQ     ,+2
      JSR     R7, NONRES    /NONRESIDENT ROUTINE
      BIT      #B14,R0
      BEQ     ,+2
      JSR     R7, PROTECT  /BOUNDRY ROUTINE
      JSR     R7, ACCESS   /WRITE ERROR ROUTINE

TBIT:  BIT      #B5,R0      /
      BEQ     INST        /INSTRUCTION COMPLETED
EMUL:  MOV      #12,R5     /TRAP DIDN'T COMPLETE,
      ADD     R6,R5       /EMULATE.
      MOV     (R5)+,R0     /OLD USER PC
      MOV     (R5)+,R1     /OLD USER PS
      MOV     @#16,-(R5)   /PUT TRAP VECTOR
      MOV     @#14,-(R5)   /MONITOR STACK
      MOV     #10,R5      /LOC OF USER R6
      ADD     R6,R5       /
      MTU     @-(R5)      /PUSH PC TO USER STACK
      MTU     @-(R5)      /PUSH PS TO USER STACK

INST:  MOV      #,R5       /INST TO COMPLETE
      ADD     R6,R5       /SET PROPER STARTING
      MOV     SSR2,(R5)   /ADDRESS IN STACK

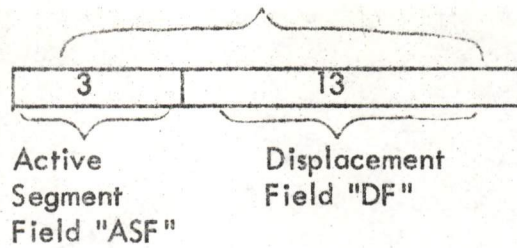
```

Protection & Relocation for the PDP-11/40  
Implementation proposal March 8, 1971.

PAGE 16

RESTART: POPM      R0-R6      /MACRO REG RESTORE  
          BIS        #000001,SSR0    /REARM SEG CHECKING  
          RTI

↑C



The DF contains a 13-bit positive number.

FIGURE 1. Interpretation of a Virtual Address

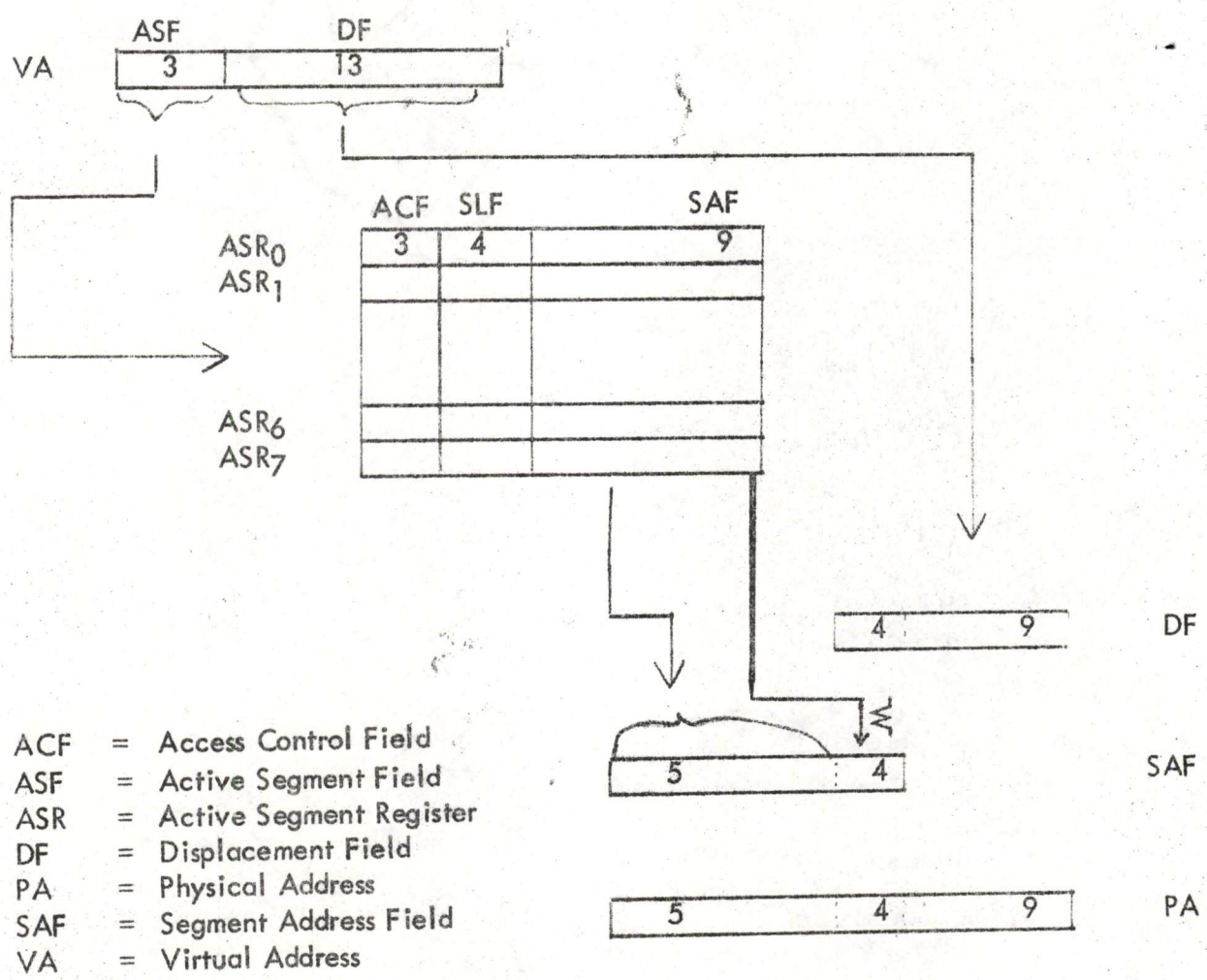
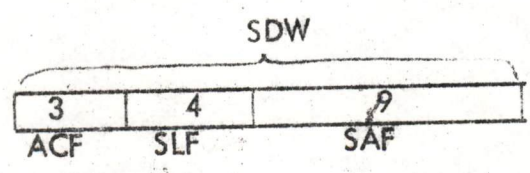


FIGURE 2. Formation of a Physical Address



- ACF = Access Control Field
- SAF = Segment Address Field
- SDW = Segment Descriptor Word
- SLF = Segment Length Field

FIGURE 3. Layout of a Segment

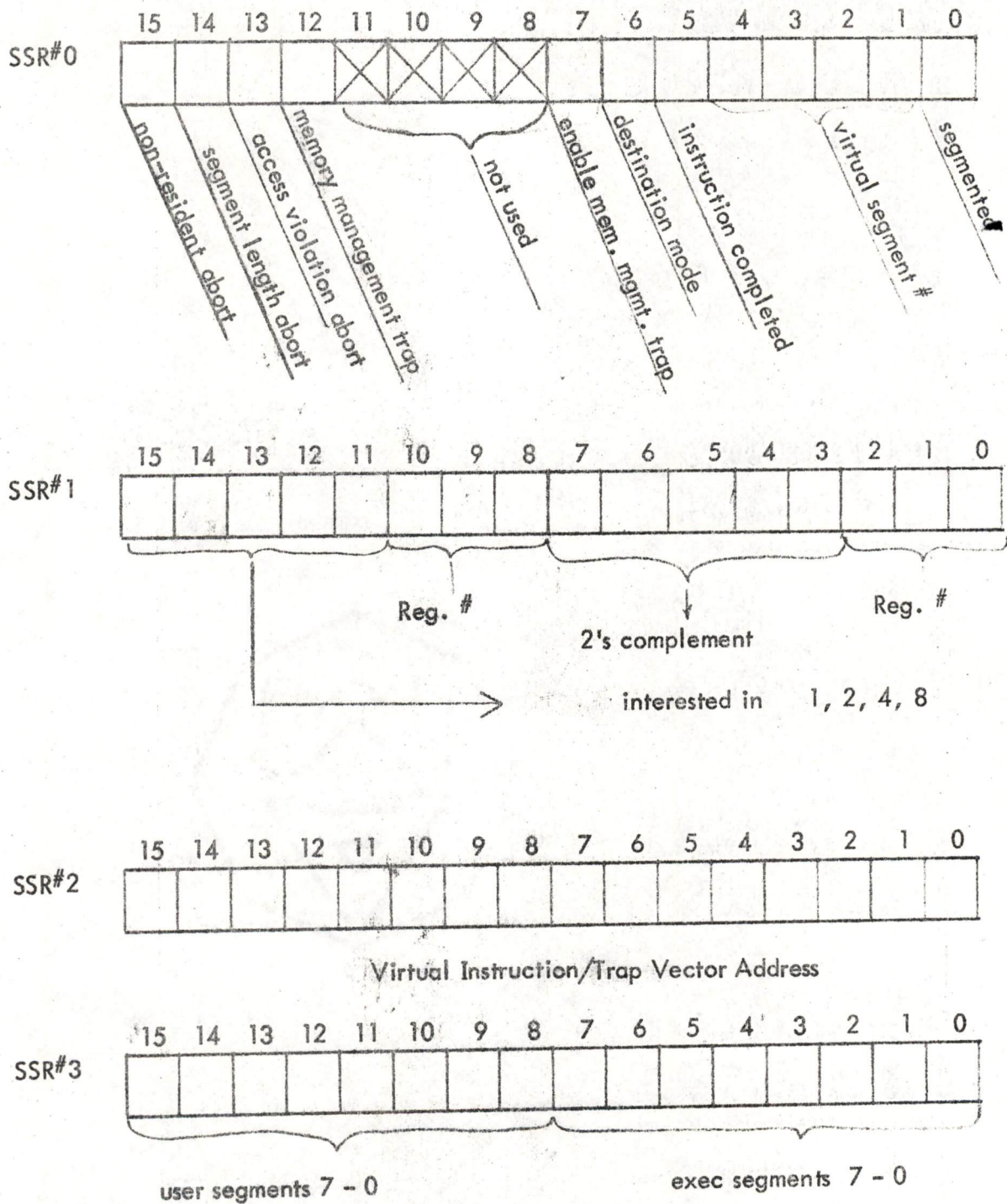


FIGURE 4. Segment Status Register Formats

*Gordon Bell*

TITLE:

The PDP-11 Floating Point Unit

0309

PDP-11/40 Technical Memo #31 35 pages

Authors: Ad van de Goor, Len Hughes

Date: February 1971

Revision: No. 2 Obsolete: 23 and 23A

*Confidential*

Index Key: Floating Point Instructions  
Virtual Address Space  
Physical Address Space  
Bus Option  
Internal Option

Distribution: PDP-11/40 Working List

ABSTRACT

The purpose of this memo is to describe the nature of the Floating Point Unit "FPU". The FPU is designed to be a Unibus option for the 11/05 and the 11/20. For the 11/40, the FPU is planned to be an internal option.

The FPU is capable of executing single and double precision (i.e. 32 and 64-bit) floating point instructions and is capable of reading and writing its own operands from and into memory. Once an FPU instruction has been started, it can continue without CPU intervention, leaving the CPU free to execute other (i.e. non-FPU) instructions.

0.0 INTRODUCTION

The position of the PDP-11 in the market is such that some floating point arithmetic capabilities are very desirable, if not necessary. Considering the complexity, and therefore the price of a floating point unit "FPU", it should be available as an option only.

Some questions to be answered concerning the FPU option are listed below and elaborated on in the following sections.

- 1) Internal versus Bus Option
- 2) FPU - CPU interaction
- 3) The FPU's Instruction and Data Formats
- 4) The FPU's Instruction Set

1.0 INTERNAL VERSUS BUS OPTION

The FPU is thought of as a fairly independent processor, i.e. when started it is supposed to finish the instruction independent of the CPU. This includes reading and writing data from and into memory. Therefore, the FPU has to be connected to the bus.

Because the 11/40 will have two memory buses (i.e. a fast synchronous one and the asynchronous Unibus) it should at least be connectable to the Unibus in order to make it acceptable for the 11/50 and 11/20.

0310

The next question to be solved in this section is whether the FPU should operate in virtual or physical address space. Figure 1-1 shows the configuration with the FPU operating in virtual address space. In Figure 1-2 the FPU operates in physical address space.

The virtual address space is defined as the address space the user runs in; the physical address space is defined as the set of core locations actually addressed. For a machine which does not have address mapping (e.g. relocate protect) the virtual address space is identical to the physical address space.

Looking at the solution of Figure 1-2, the following comments can be made:

- 1) The addresses of the operands have to be passed to the FPU as physical addresses. Looking at the Relocate Protect option, this means that it should recognize certain FPU addresses and not relocate them. Instead, it should take the data (on the data lines) which contain the virtual address and relocate it as if it were an address. This requires special controls and data paths in the relocate protect option.
- 2) The Relocate/Protect option might have Read/Write protect bits which might have to be duplicated in the FPU, or the Relocate/Protect option has to have knowledge relevant to the use of the virtual addresses it relocates.
- 3) In case of the modes (R)+ or -(R) address violations can occur which can be detected with difficulty by the Relocate/Protect option.

The solution of Figure 1-1, i.e. let the FPU operate in virtual memory, has none of the above disadvantages. In Figure 1-1 it is treated in the same way as the CPU for which the Relocate/Protect option is designed. Clearly, from the above it can be stated that the FPU should operate in virtual space.

Because of mechanical limitations and restrictions on the fast bus, it is most desirable that the FPU take up no more than one system unit if the solution of Figure 1-1 is implemented.

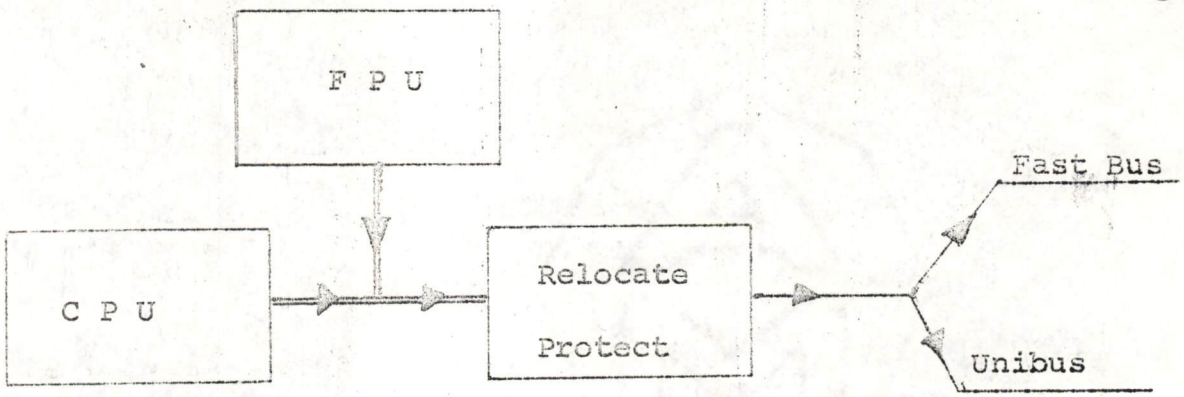


Figure 1-1 Configuration with FPU  
 Operating in VIRTUAL Address Space  
 (Presuming an open-collector internal bus.)

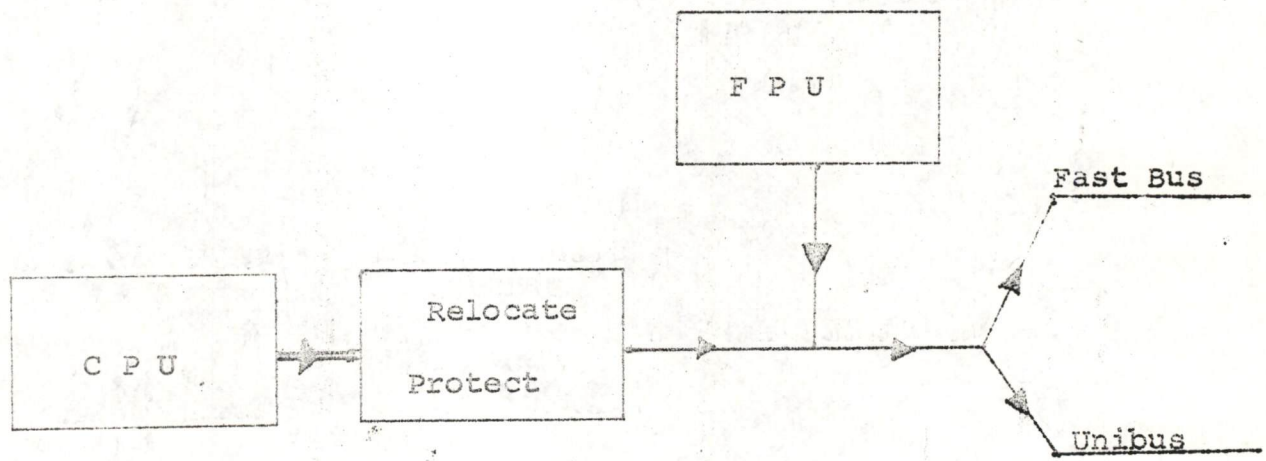




Figure 1-2 Configuration with FPU Operating in  
 PHYSICAL Address Space

NOTE:  = Virtual Address Lines  
 = Physical Address Lines



Because the FPU will be an independent processor, it is possible to allow it to finish a started instruction independent of the CPU. This allows the CPU to be handled in either of two ways.

- 1) The CPU is declared busy while the FPU carries out its operation. This has the advantage that it would deteriorate the interrupt response time (because floating point operations tend to take a relatively long execution time). It also prohibits the CPU from executing other, i.e. non-floating, instructions.
- 2) Allow the CPU to continue executing non-floating instructions once the FPU is set up (i.e. ready to start executing). This allows the CPU to carry out subscript computation, etc., in parallel with the execution of the floating point instruction thus improving the overall execution time. This method is selected because of its described advantages.

Because the FPU is a bus option on the 11/05 and 11/20, the FPU OP code and the source address have to be transferred to the FPU. The FPU-CPU interaction takes place for the 11/05, 11/20, and 11/40 as described in the next sections.

## 2.1 FPU-11/20 INTERACTION

In previous memos the FPU was activated by the 11/20 through a sequence of MOV instructions, as described in Technical Memos 23 and 23-A. A typical sequence looked like the one given below for the case of the instruction MULF A(Rx), AC1

```

ADD #A, Rx      ; compute address
SUB FBR, PC     ; test for FPU busy (FBR)=4 when FPU busy else 0
MOV PC, FPC    ; save PC
MOV Rx, FIR+FOC+AC ; move operand address and start FPU

```

The above sequence takes 8 words and has to be repeated for every FPU instruction of the above type.

The new scheme requires that every FPU instruction (like: MULF A(R2), AC1) is preceded by a JSR. The JSR allows the FPU to take control over the CPU. The FPU uses the CPU for address computation, stack pointer adjustments, etc., and acts like a hardwired interpreter. The JSR instruction has to be the following, "JSR R7, FPU" where FPU is an address in the I/O area. An example of this is given below. (next page)

JSR R7, FPU ; typical call sequence in

MULF A(R2), AC1 ; user's program

-----

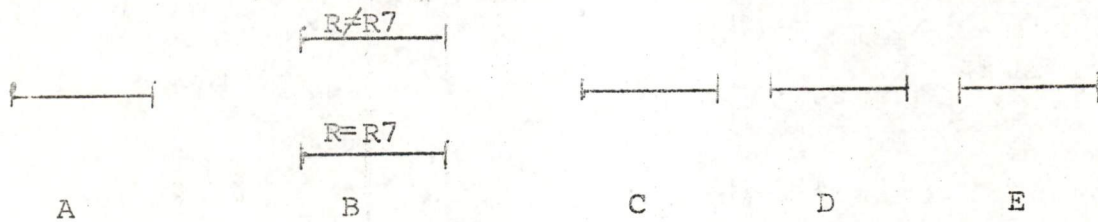
FPU  $\left\{ \begin{array}{l} \text{BR}_0 \\ \text{MOV (R6) + , FRA*} \end{array} \right.$  ; the I/O address FPU contains  
; "BR<sub>0</sub> when the FPU is busy  
; otherwise it contains "MOV (R6) + , FRA"  
; When the latter instruction is executed  
; the return address is popped off  
; the stack into the FPU's FRA register  
  
 $\text{Ⓒ(FRA)} \rightarrow \text{FIR*}$  ; The instruction is fetched, under  
 $(\text{FRA}) + 2 \rightarrow \text{FRA}$  ; hardware control, and loaded in the  
; FIR register and FRA is incremented  
  
MOV R2, FDA ; The CPU's register R2 is read  
 $(\text{FRA}) + (\text{FRA}) \rightarrow \text{FDA*}$  ; The index computation "A+(R2)"  
 $(\text{FRA}) + 2 \rightarrow \text{FRA}$  ; is done under hardware control  
; and FRA is incremented  
  
2 or 4 data fetches ; Depends on the mode of the FPU  
MOV FRA, PC ; Control is transferred back to the  
; CPU while the FPU does the  
; required operation

- \* FRA means Floating Return Address
- \* FIR means Floating Instruction Register
- \* FDA means Floating Data Address

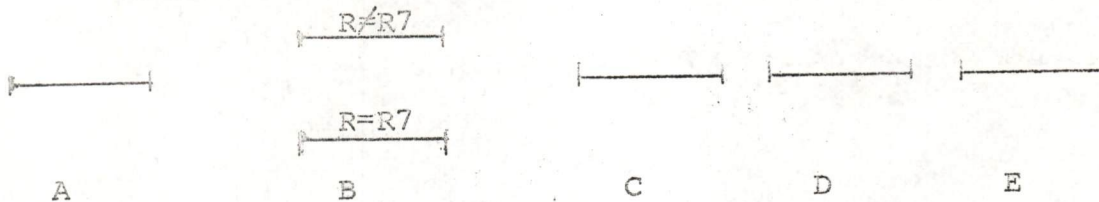
A summary of the different CPU instructions issued by the FPU is given below.

0314

This sequence of CPU instructions is interleaved with FPU data fetches/stores done under control of the FPU hardware for greater efficiency. A complete FPU instruction execution cycle can be divided into 6 sub-cycles as shown in Figure 2-1 below.



SEQUENCE FOR MOST INSTRUCTIONS



SEQUENCE FOR CERTAIN CONVERT INSTRUCTIONS

A = Instruction Fetch

B = Operand Address Computation, two paths depending on R=R7/R≠R7

C = Data Fetches/Stores

D = Transfer Control from FPU back to CPU

E = Execution

FIGURE 2-1, FPU instruction Subsequences

Below is the sequence of FPU issued instructions and FPU actions which are required for the address computation and final execution of most FPU instructions. The capital letters preceding the sections correspond to the subsequences of Figure 2-1.

```

A:  FPU: ← BR .           ; FPU busy loop
      MOV (R6) +, FRA     ; Get return address
      (FRA) → FIR        ; Get instruction, both parts
      (FRA) +2 → FRA     ; done by FPU hardware
  
```

- 5 @ -(R) \*(FRA)-2→FRA  
 \*@(FRA)→FDA
- 6 A(R) \*@(FRA)→FDA  
 \*(FRA)+2→FRA  
 \*(FRA)+(FDA)→FDA
- 7 @ A(R) \*@(FRA)→FDA  
 \*(FRA)+2→FRA  
 \*(FRA)+(FDA)→FDA  
 \*@(FDA)→FDA

## C. Possible data fetches/stores

These happen at priority level 7

---

- D. MOV FRA, PC ; transfer control back to CPU  
 ; to execute non FPU initiated  
 ; instructions
- 

E. Execute the FPU instruction, i.e. perform the actual multiplication, etc.

2.1.1 CPU CONDITION CODES

The CPU condition code bits C, N, Z, and V which existed just prior to the FPU instruction, are destroyed. This is caused by the instructions the FPU issues to the CPU.

The FPU has its own set of condition code bits, "FC, FN, FZ, and FV". These can be transferred to the CPU's condition code bits under control of a special instruction Copy Floating Condition "CFCC".

2.1.2 PRIORITY LEVEL OF THE FPU

On the 11/05 and the 11/20 the FPU will be a Unibus option. The priority level of the FPU will be 7.

In order not to increase NPR latency, the FPU will monitor the NPR line and give up the bus between memory cycles.

The 11/20 bus priority arbitrator requires a MSYN signal to transfer Bus Mastership between peripherals. In certain special cases this could lead to the execution of an instruction before the bus would be re-arbitrated to another requesting device (e.g. the FPU). The execution of an out of sequence instruction would be in conflict with the correct operation of the FPU, as will be clear from Section 3.0. This is prevented by "feeding" the CPU a "BR." instruction when the above condition occurs and the FPU is in control.

When the CPU wants to make use of the FPU, the CPU's priority level should be less than 7 (i.e.  $PR < 7$ ). When  $PR = 7$ , the FPU will not be able to become bus master, because the CPU's  $PR = 7$  is considered to be higher. This will cause the CPU to be in an infinite loop executing the "BR ." instruction as described above, once it tries to execute an FPU instruction.

### 2.1.3 ALTERNATIVE FPU-11/20 INTERACTION

The method of Section 2.1 requires every FPU instruction to be preceded by a JSR. An alternative method is to issue the FPU instruction "as is" and have a trap service routine to transfer control to the FPU. An example of such a routine is given below. (It should be noted that all FPU OP codes start with a "17".)

```

; Trap service routine to handle FPU instructions

SUB #2, @R6      ; decrement saved PC
CMP #1700000, @0(R6) ; test for FPU OP code
BLO NOTFPU
MOV @R6, -(R6)  ; make 3 top words of stack
MOV 4(R6), 2(R6) ; FPU, PS and PC
MOV R6, 4(R6)
MOV #FPU, @R6
RTI              ; end of FPU trap handler

```

NOT FPU: ADD #2, @R6

### 2.1.4 INTERRUPTABILITY

A special deadlock condition can arise when the CPU is executing FPU supplied instructions and an interrupt occurs by a device which also wants to make use of the FPU. At the time the CPU was executing FPU supplied instructions, it was considered "busy". The CPU is interruptable at that point because it is running at a priority level lower than 7. If the interrupting device would go off and use the FPU without testing, the CPU would start an infinite loop of "BR ." instructions because the FPU was busy.

This loop is executed at the priority level of the interrupting device. In order for the FPU to become free, it has to continue supplying CPU instructions until subsequence D of Figure 2-1 has been completed, i.e. when the FPU dismisses the CPU.

A special hardware aid is built into the FPU to discover this state. The FPU has a register called the Floating Interrupt Vector "FINTV" and a bit called the Floating Interrupt CPU Dismissed "FICD" in the Floating Program Status "FPS" word. The FICD bit is set whenever a non-zero value is loaded into FINTV. The operation is as follows: Whenever the subsequence D of Figure 2-1 is executed, and the FICD bit of the FPS is set, the FPU will cause an interrupt using as interrupt vector (FINTV).

A possible routine preventing the deadlock making use of the above hardware, is shown on the next page. This code is part of the interrupt service routine of the interrupting device which wants to use the FPU.

```

CMP R6, #172      ; did PC point to FPU
BLO FPUFREE

MOV FINTV, TEMP   ; save old FINTV
MOV NEW.INTV, FINTV ; set up new interrupt vector

*MOV 2(R6), TEMP1 ; save old PS
*MOV NEW.PS, 2(R6) ; install new PS

RTI               ; dismiss current interrupt
                 ; and start FPU

```

FPU FREE: SAVE FPU STATUS  
USE THE FPU

```

MOVE TEMP, FINTV ; restore old FINTV

*MOVE TEMP1, 2(R6) ; restore old PS

RTI               ; dismiss interrupt

```

It should be noted that any interrupt vector can be loaded into FINTV. If, for example, the interrupt vector of the interrupting device is loaded into FINTV, then upon the first RTI in the above code, the interrupt will be dismissed until the FPU has dismissed the CPU. At that point, the FPU will request an interrupt with the interrupt vector of the original interrupting device, thus simulating the old interrupt.

## 2.2 FPU-11/05 INTERACTION

The use of the FPU with the 11/05 is essentially the same as with the 11/20 except for the JSR preceding an FPU instruction, which is not required with the 11/05. The 11/05 will execute code making use of the JSR, however, for compatibility reasons.

When the 11/05 fetches an instruction which starts with a "17" (i.e. an FPU OP code) it will not trap, but execute the following sequence.

```

TST FPU05        ; test is FPU is busy
BEQ .-4          ; loop is busy

MOV PC, FPU05+2
MOV IR, FPU05+4
MOV FPU05+6, PC  ; start fetching instructions from the FPU

```

The above sequence is not executed with PDP-11 instructions as shown above, but in 11/05 micro code which is done at a much greater speed. This allows the FPU instructions to be given without a JSR, thus eliminating the space and time consuming JSR and the Instruction Fetch subsequence "A" of Figure 2-1.

\*These instructions are only necessary when the FPU has to proceed with the interrupted instruction at a different priority level.

The FPU will be connected to the 11/40 via a direct set of wires rather than via the Unibus. This is required for the proper operation of the segmentation option, see Section 1.0.

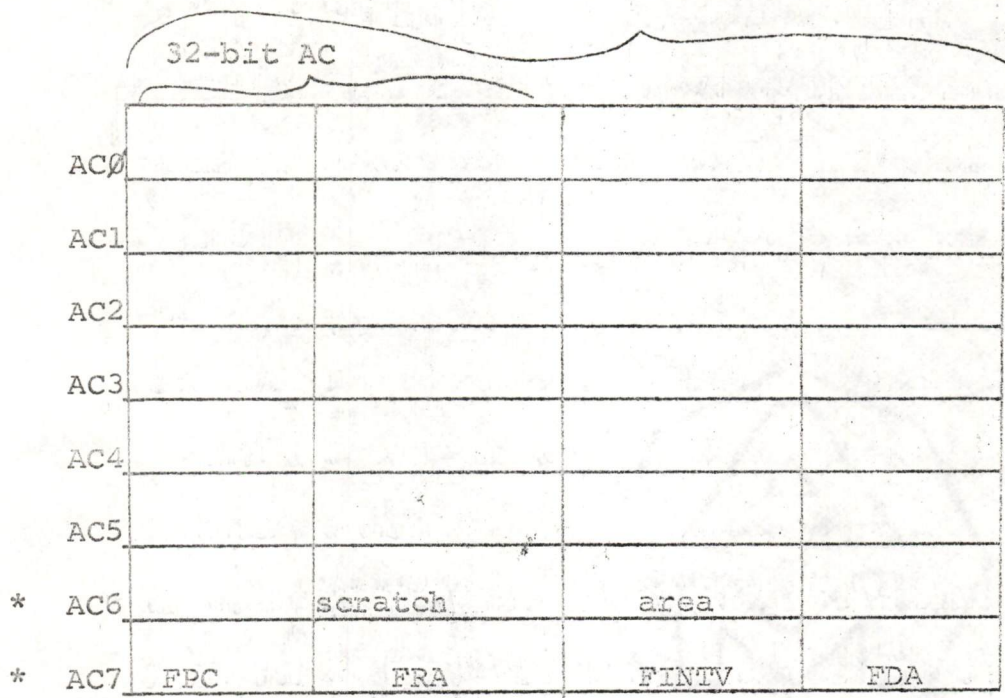
The required address computation will be done by the 11/40 hardware. The need to have the FPU supply the 11/40 with the instructions is thereby eliminated. The 11/40 condition codes will not be affected by the FPU unless the instruction is a CFCC.

When the 11/40 fetches an FPU instruction, it tests if the FPU is busy while it allows for higher priority bus requests. Once the FPU is free, the 11/40 will do the required address computation and notify the FPU. The FPU will then strobe in the required data from the 11/40's internal registers (like the PC, IR, etc.), do the required data fetches (stores from) into memory and allows the 11/40 CPU to proceed while it executes the FPU instruction.

## 3.0

THE FPU'S INSTRUCTION & DATA FORMATS

The FPU has, except for its scratch, address and status registers, 6 general purpose data registers called Accumulators "AC's". They are named AC0 through AC5 and are interpreted to be 32 or 64-bits long depending on the instruction. In case of a 32-bit instruction, only the top (i.e. left most) 32-bits are used, while the remaining (i.e. right 32-bits) of the AC remain unaffected. See Figure 3-1.



\* AC6 and AC7 are reserved for internal use.

AC7 is used to contain the following status registers:

- 1) FPC "Floating PC" - points to the word following the first word of the FPU instruction.
- 2) FRA "Floating Return Address" - points to the next instruction to be executed.
- 3) FINTV "Floating Interrupt Vector" - a 16 bit interrupt vector used by the FPU upon completion of the address computation part of an instruction when (FINTV)≠0. This is only used on the 11/05 and 11/20.
- 4) FEC "Floating Exception Code" - A number which identifies the cause of the interrupt.

FIGURE 3-1 Accumulator Layout



The FPU instruction set is divided in five formats as shown in Figure 3-2. Format F1 is used by the binary floating instructions. Format F2 is used by the unary floating instructions. Format F3 is used by the load and store convert to and from Integer instructions. Format F5 is used by some special instructions like Copy Floating Condition Code.

The fields of the formats of Figure 3-2 are interpreted in the following way.

- OC "Operation Code"  
The OC field of all FPU instructions is 4 bits long and contains a "17".
- FOC "Floating Operation Code"  
This field of the format specifies the specific floating point operation.
- FSRC "Floating Source"  
The floating source specifies the source operand of the instruction. The interpretation of the addressing modes is as shown below:
- MODE INTERPRETATION
- 0 AC0-AC5 contain the data. The "data" is considered 32 or 64 bits depending on the mode of the FPU (i.e. Floating or Extended).  
  
When AC6 or AC7 are specified, an OP code error will be given unless the instruction is a STX instruction.
- 1 R0-R7 contain the address of the data. When R=R7 the data is considered to be only 1 word long (i.e. 16 bits).
- 2 R0-R7 contain the address of the data. After the data has been fetched R0-R6 are incremented with 4 or 8 depending on the mode of the FPU. When R=R7, the data is considered to be 1 word long and therefore, R7 will be incremented with 2.
- 3 R0-R7 contains the address of the address of the data. R0-R7 are incremented by 2.
- 4 R0-R6 are decremented by 4 or 8, depending on the FPU mode. After that they contain the address of the data. When R=R7, R7 is decremented by 2 and contains the address of a 1 word data item.

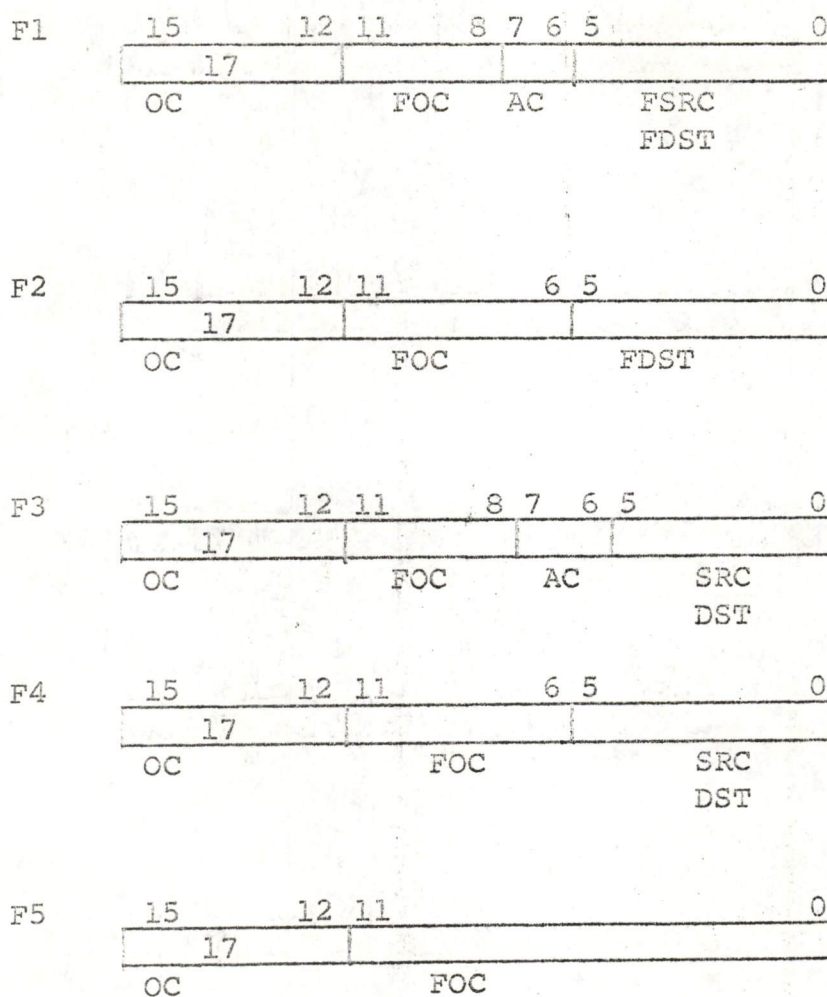


FIGURE 3-2. FPU INSTRUCTION FORMATS

## MODE

- 5 R0-R7 are decremented by 2. After that they contain the address of the address of the data.
- 6 The address of the data is determined by the regular index computation.
- 7 The address of the data is determined by the regular deferred index computation.

FDST "Floating Destination"  
The interpretation of this field is identical to that of the source.

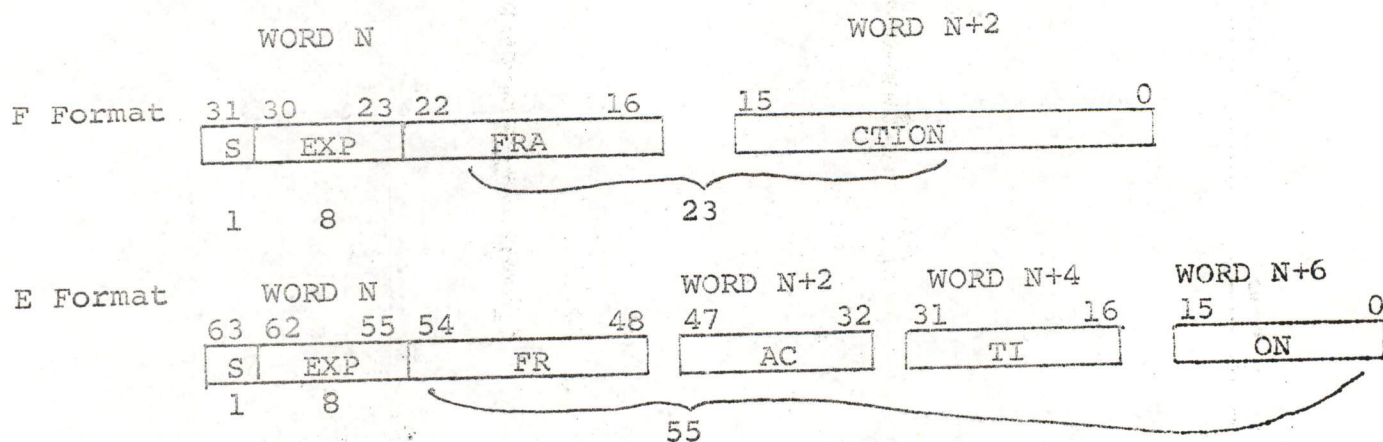
AC "Accumulator"  
This is a 2 bit field specifying AC0-AC3.

SRC "Source"  
Regular PDP-11 source field.

DST "Destination"  
Regular PDP-11 destination field.

3.1 THE FPU'S DATA FORMATS

The FPU handles two types of floating point data: Floating "F" which is 32 bits long, and Extended "E" which is 64 bits long. Both formats assume normalized numbers only. The fraction is represented in sign-magnitude notation with the binary radix point to the left. The most significant bit of the fraction is not stored because it is redundant. This bit is always a 1 except when the exponent is 0, then the number is declared to be zero. The F and E format are shown in Figure 3-3 below.



S=Sign of fraction

EXP=8 bit exponent, in excess  $200_8$  notation, radix =2

FRACTION=23 or 55 bit fraction in sign-magnitude notation, radix point to the left

FIGURE 3-3 Floating Point Data Format

#### 4 THE FPU'S INSTRUCTION SET

Appendix A lists the complete FPU instruction set, a description of which is given below. Appendix B lists some maximum and minimum execution times.

##### 4.1 THE FPU PROGRAM STATUS REGISTER

The FPU's program status register is shown in Figure 4-1. It has four mode bits:

- 1) FT, the FPU's Truncate Mode Bit. This bit, when set, causes the result of any floating point operation to be truncated rather than rounded.
- 2) FD, the FPU's Double Precision Integer Mode Bit. This bit is active in conversion between integer and floating point format. When on, the integer format assumed is double precision 2's complement (i.e. 32 bits). When off, the integer format that is assumed is single precision 2' complement (i.e. 16 bits).
- 3) FE, the FPU's Extended Precision Mode Bit. This bit determines the precision that is used for floating point calculations. When set, extended precision is assumed - when reset, normal precision is used.
- 4) FMM, the FPU's Maintenance Mode Bit. The FMM enables special maintenance logic. The exact nature of this logic will be detailed in a later memo.

Along with the four mode bits, the status register contains four condition codes, FC, FV, FZ and FN. These are loaded into the CPU's C, V, Z, and N condition codes by the Copy Floating Condition Codes instruction.

The way in which each instruction affects the floating condition codes is detailed in the instruction definitions. The FC condition code bit has two meanings:

- 1) For the STCXJ instruction, which converts a floating point number to an integer, the FC bit is set if the resulting integer is too large to be stored in the specified register.
- 2) In all other cases, the FC bit indicates that the absolute value of the floating point result was larger than the largest integer that can be represented in M bits, where M is the width of the fraction. In the

THE FPU PROGRAM STATUS REGISTER (continued)

extended mode,  $M = 56$  bits and in floating mode  $M = 24$ . This allows sign-magnitude integer arithmetic with 24 and 56 bits of precision, not including the sign bit, to be performed with the FPU.

The FPU's Program Status Register also contains six interrupt enable bits. The FPU interrupt vector is at core location  $240_8$ .

1) FIC FLOATING INTERRUPT ON INTEGER CONVERSION ERROR

When FIC is set, and the STCXJ instruction causes FC to be set, a trap will occur. If the interrupt occurs, the instruction is aborted leaving the contents of all the registers untouched.

2) FIV FLOATING INTERRUPT ON OVERFLOW

When this bit is set, floating overflows will cause an interrupt. The result of the operation causing the interrupt will be correct except for the exponent which will be off by 400 (octal). If the bit is off, the result of the operation will be the same as detailed above and no interrupt will occur.

3) FIU FLOATING INTERRUPT ON UNDERFLOW

When this bit is on, floating underflow will cause an interrupt. The result of the operation, causing the interrupt, will be correct except for the exponent which will be off by 400 (octal). If the bit is off and underflow occurs, the result will be set to zero.

4) FIOR FLOATING INTERRUPT ON OUT OF RANGE

When this bit is on, and the FC bit is set because the result is out of integer range, an interrupt occurs. Out of integer range means that the absolute value of the result is greater than or equal to  $2^{XL}$  where  $XL=24$  if floating mode, or 56 if extended mode.

5) FIUV FLOATING INTERRUPT ON UNDEFINED

When this bit is on and a  $-\emptyset$  is obtained from memory, an interrupt will occur. When this bit is off  $-\emptyset$  can be loaded and used in any arithmetic operation. The result of such operation is undefined.

6) FICD FLOATING INTERRUPT ON CPU DISMISSED

The FICD bit, when on, will cause an interrupt to occur when the address computation performed by the 11/20 and 11/05 is done. On the 11/40 this bit will be ignored. For a complete description of the use of this enable, see Section 2.1.4.

7) FIE FLOATING INTERRUPT ENABLE

All interrupts by the FPU are disabled when this bit is off.

BIT

0	FC	;Floating Carry
1	FV	;Floating Overflow
2	FZ	;Floating Zero
3	FN	;Floating Negative
4	FMM	;Floating Maintenance Mode
5	FT	;Floating Truncate Mode
6	FD	;Floating Double Precision Mode
7	FE	;Floating Extended Mode
8	FIC	;Floating Interrupt on Conversion Error
9	FIV	;Floating Interrupt on Overflow Error
10	FIU	;Floating Interrupt on Underflow Error
11	FIOR	;Floating Interrupt on Out of Range Error
12	FIUV	;Floating Interrupt on Undefined Variable
13	FICD	;Floating Interrupt on CPU Dismissed
14	FIE	;Floating Interrupt Enable
15	RUN	;FPU's Run Status

FIGURE 4-1. Layout of FPU Program Status Register

INSTRUCTION: Set Floating Mode  
 MNEMONIC: SET F  
 OPERATION:  $FE \leftarrow \emptyset$   
 FORMAT:

1	7	0	0	0	1
---	---	---	---	---	---

INSTRUCTION: Set Extended Mode  
 MNEMONIC: SETE  
 OPERATION:  $FE \leftarrow 1$   
 FORMAT:

1	7	0	0	0	2
---	---	---	---	---	---

INSTRUCTION: Integerize Floating/Extended  
 MNEMONIC: INTX FSRC  
 OPERATION:  $AC4 \leftarrow \mathcal{I}(FSRC)$ ;  $AC5 \leftarrow (FSRC) - \mathcal{I}(FSRC)$   
 $FC \leftarrow 1$  if  $|FSRC| \geq 2^{XL}$  else  $FC \leftarrow \emptyset$ \*  
 $FV \leftarrow \emptyset$   
 $FZ \leftarrow 1$  if  $(FSRC) = \emptyset$  else  $FZ \leftarrow \emptyset$   
 $FN \leftarrow 1$  if  $(FSRC) < \emptyset$  else  $FN \leftarrow \emptyset$

FORMAT:

1	7	0	3	FSRC
---	---	---	---	------

$\mathcal{I}(FSRC)$  is the integer part of  $(FSRC)$  i.e.  $(FSRC)$  is fixed and then floated. Note that the integer is obtained by truncation i.e. 5.9 becomes 5. If  $|FSRC| \geq 2^{XL}$ ,  $\mathcal{I}(FSRC) = (FSRC)$ . Note that the fractional part of  $(FSRC)$  is stored in AC5.

INSTRUCTION: Clear Floating/Extended  
 MNEMONIC: CLRX FDST  
 OPERATION:  $FDST \leftarrow \emptyset$   
 $FC \leftarrow 0$   
 $FV \leftarrow 0$   
 $FZ \leftarrow 1$   
 $FN \leftarrow 0$

FORMAT:

1	7	0	4	FDST
---	---	---	---	------

\* XL = 24 if FE mode =  $\emptyset$   
 = 56 if FE mode = 1

INSTRUCTION: Negate Floating/Extended  
 MNEMONIC: NEGX FDST  
 OPERATION:  $FDST \leftarrow -(FDST)$   
 $FC \leftarrow 1$  if  $|(FDST)| \geq 2^{XL}$  else  $FC \leftarrow 0^*$   
 $FV \leftarrow \emptyset$   
 $FZ \leftarrow 1$  if  $(FDST) = \emptyset$  else  $FZ \leftarrow \emptyset$   
 $FN \leftarrow 1$  if  $(FDST) < \emptyset$  else  $FN \leftarrow \emptyset$

FORMAT:

1	7	0	5	FDST
---	---	---	---	------

INSTRUCTION: Make Absolute Floating/Extended  
 MNEMONIC: ABSX FDST  
 OPERATION:  $FDST \leftarrow -(FDST)$  if  $(FDST) < 0$  else  $FDST \leftarrow (FDST)$   
 $FC \leftarrow 1$  if  $|(FDST)| \geq 2^{XL}$  else  $FC \leftarrow \emptyset^*$   
 $FV \leftarrow 0$   
 $FZ \leftarrow 1$  if  $(FDST) = 0$  else  $FZ \leftarrow \emptyset$   
 $FN \leftarrow \emptyset$

FORMAT:

1	7	0	6	FDST
---	---	---	---	------

INSTRUCTION: Test Floating/Extended  
 MNEMONIC: TSTX FDST  
 OPERATION:  $FDST \leftarrow (FDST)$   
 $FC \leftarrow 1$  if  $|(FDST)| \geq 2^{XL}$  else  $FC \leftarrow 0^*$   
 $FV \leftarrow 0$   
 $FZ \leftarrow 1$  if  $(FDST) = \emptyset$  else  $FZ \leftarrow \emptyset$   
 $FN \leftarrow 1$  if  $(FDST) < 0$  else  $FN \leftarrow 0$

FORMAT:

1	7	0	7	FDST
---	---	---	---	------

INSTRUCTION: Load Floating/Extended  
 MNEMONIC: LDX FSRC, AC  
 OPERATION:  $AC \leftarrow (FSRC)$   
 $FC \leftarrow 1$  if  $|(FSRC)| \geq 2^{XL}$  else  $FC \leftarrow 0^*$   
 $FV \leftarrow 0$   
 $FZ \leftarrow 1$  if  $(FSRC) = \emptyset$  else  $FZ \leftarrow \emptyset$   
 $FN \leftarrow 1$  if  $(FSRC) < 0$  else  $FN \leftarrow \emptyset$

FORMAT:

1	7	1	AC	FSRC
---	---	---	----	------

\*XL=24 if FE mode =1  
 56 if FE mode=1



INSTRUCTION: Store Floating/Extended  
MNEMONIC: STX AC, FDST  
OPERATION: FDST ← (AC)  
FC ← FC  
FV ← FV  
FZ ← FZ  
FN ← FN

FORMAT:

1	7	1	4+AC	FDST
---	---	---	------	------

INSTRUCTION: Add Floating/Extended

MNEMONIC: ADDX FSRC, AC

OPERATION:  $AC \leftarrow (AC) + (FSRC)$  if  $|(AC) + (FSRC)| \geq XLL$  OR FIU=1  
 else  $AC \leftarrow \emptyset^{**}$   
 $FC \leftarrow 1$  if  $|(AC)| \geq 2^{XL}$  else  $FC \leftarrow \emptyset^*$   
 $FV \leftarrow 1$  if  $|(AC)| \geq XUL$  else  $FV \leftarrow \emptyset^{***}$   
 $FZ \leftarrow 1$  if  $(AC) = \emptyset$  else  $FZ \leftarrow \emptyset$   
 $FN \leftarrow 1$  if  $(AC) < \emptyset$  else  $FN \leftarrow 1$

FORMAT:

1	7	2	AC	FSRC
---	---	---	----	------

INSTRUCTION: Subtract Floating/Extended

MNEMONIC: SUBX FSRC, AC

OPERATION:  $AC \leftarrow (AC) - (FSRC)$  if  $|(AC) - (FSRC)| \geq XLL$  OR FIU=1  
 else  $AC \leftarrow \emptyset^{**}$   
 $FC \leftarrow 1$  if  $|(AC)| \geq 2^{XL}$  else  $FC \leftarrow \emptyset^*$   
 $FV \leftarrow 1$  if  $|(AC)| \geq XUL$  else  $FV \leftarrow 0^{***}$   
 $FZ \leftarrow 1$  if  $(AC) = \emptyset$  else  $FZ \leftarrow \emptyset$   
 $FN \leftarrow 1$  if  $(AC) < \emptyset$  else  $FN \leftarrow 1$

FORMAT:

1	7	2	4+AC	FSRC
---	---	---	------	------

\* XL = 24 if FE=0  
 = 56 if FE=1

\*\*XLL = smallest number that is not identically zero  
 =  $2^{-128}$

\*\*\*XUL = largest number that can be represented  
 =  $2^{127} * (1 - 2^{-XL-1})$

INSTRUCTION: Multiply Floating/Extended

MNEMONIC: MULX FSRC, AC

OPERATION:  $AC \leftarrow (AC) * (FSRC)$  if  $| (AC) * (FSRC) | \not\geq XLL$  OR  $FIU=1$   
 else  $AC \leftarrow \emptyset^{**}$   
 $FC \leftarrow 1$  if  $| (AC) | \geq 2^{XL}$  else  $FC \leftarrow \emptyset^*$   
 $FV \leftarrow 1$  if  $| (AC) | \geq XUL$  else  $FV \leftarrow \emptyset^{***}$   
 $FZ \leftarrow 1$  if  $(AC) = \emptyset$  else  $FZ \leftarrow \emptyset$   
 $FN \leftarrow 1$  if  $(AC) < \emptyset$  else  $FN \leftarrow \emptyset$

FORMAT:

1	7	3	AC	FSRC
---	---	---	----	------

INSTRUCTION: Divide Floating/Extended

MNEMONIC: DIVX FSRC, AC

OPERATION: Case 1  $(FSRC) \neq \emptyset$   
 $AC \leftarrow (AC) / (FSRC)$  if  $| (AC) / (FSRC) | \not\geq XLL$  OR  $FIU=1$   
 else  $AC \leftarrow \emptyset^{**}$   
 $FC \leftarrow 1$  if  $| (AC) | \geq 2^{XL}$  else  $FC \leftarrow \emptyset^*$   
 $FV \leftarrow 1$  if  $| (AC) | \geq XUL$  else  $FV \leftarrow \emptyset^{***}$   
 $FZ \leftarrow 1$  if  $(AC) = \emptyset$  else  $FZ \leftarrow \emptyset$   
 $FN \leftarrow 1$  if  $(AC) < \emptyset$  else  $FN \leftarrow \emptyset$   
Case 2  $(FSRC) = \emptyset$   
 $AC \leftarrow (AC)$   
 $FC \leftarrow (FC)$   
 $FV \leftarrow (FV)$   
 $FZ \leftarrow (FZ)$   
 $FN \leftarrow (FN)$

FORMAT:

1	7	3	4+AC	FSRC
---	---	---	------	------

INSTRUCTION: Reverse Subtract Floating/Extended

MNEMONIC: RSUBX FSRC, AC

OPERATION:  $AC \leftarrow (FSRC) - (AC)$  if  $|(FSRC) - (AC)| \geq XLL$  OR  
 $FIU=1$  else  $AC \leftarrow \emptyset^{**}$   
 $FC \leftarrow 1$  if  $|(AC)| \geq 2^{XL}$  else  $FC \leftarrow \emptyset^*$   
 $FV \leftarrow 1$  if  $|(AC)| > XUL$  else  $FV \leftarrow \emptyset^{***}$   
 $FZ \leftarrow 1$  if  $(AC) = \emptyset$  else  $FZ \leftarrow \emptyset$

FORMAT:

1	7	4	AC	FSRC
---	---	---	----	------

INSTRUCTION: Compare Floating/Extended

MNEMONIC: CMPX AC, FDST

OPERATION:  $AC \leftarrow (AC)$   
 $FC \leftarrow 1$  if  $|(AC)| \geq 2^{XL}$  else  $FC \leftarrow \emptyset^*$   
 $FV \leftarrow 1$  if  $|(AC)| > XUL$  else  $FV \leftarrow \emptyset^{***}$   
 $FZ \leftarrow 1$  if  $(AC) = \emptyset$  else  $FZ \leftarrow \emptyset$   
 $FN \leftarrow 1$  if  $(AC) < \emptyset$  else  $FN \leftarrow \emptyset$

FORMAT:

1	7	4	4+AC	FDST
---	---	---	------	------

\* XL = 24 if FE=0  
 = 56 if FE=1

\*\*XLL = smallest number that is not identically zero  
 =  $2^{-128}$

\*\*\*XUL = largest number that can be represented  
 =  $2^{127} (1 - 2^{-XL-1})$

INSTRUCTION: Reverse Divide Floating/Extended

MNEMONIC: RDIVX FSRC,AC

OPERATION: Case 1 (AC) =  $\emptyset$   
 $AC \leftarrow (FSRC)/(AC)$  if  $|(FSRC)/(AC)| \geq XLL$  OR  
 FIV=1 else  $AC \leftarrow \emptyset^{**}$   
 $FC \leftarrow 1$  if  $|(AC)| \geq 2^{XL}$  else  $FC \leftarrow \emptyset^*$   
 $FV \leftarrow 1$  if  $|(AC)| \geq XUL$  else  $FV \leftarrow \emptyset^{***}$   
 $FZ \leftarrow 1$  if (AC) =  $\emptyset$  else  $FZ \leftarrow \emptyset$   
 $FN \leftarrow 1$  if (AC) <  $\emptyset$  else  $FN \leftarrow \emptyset$

Case 2 (AC) = 0

$AC \leftarrow (AC)$   
 $FC \leftarrow (FC)$   
 $FV \leftarrow (FV)$   
 $FZ \leftarrow (FZ)$   
 $FN \leftarrow (FN)$

FORMAT:

1	7	5	AC	FSRC
---	---	---	----	------

INSTRUCTION: Load & convert from Extended Floating to Floating/Extended

MNEMONIC: LDCYX FSRC,AC

OPERATION:  $AC \leftarrow C_{YX}(FSRC)$  if  $|(FSRC)| \geq XLL$  or FIU=1 else  $AC \leftarrow \emptyset^{**}$   
 $FC \leftarrow 1$  if  $|(AC)| \geq 2^{XL}$  else  $FC \leftarrow \emptyset^*$   
 $FV \leftarrow 1$  if  $|(AC)| \geq XUL$  else  $FV \leftarrow \emptyset^{***}$   
 $FZ \leftarrow 1$  if (AC) =  $\emptyset$  else  $FZ \leftarrow \emptyset$   
 $FN \leftarrow 1$  if (AC) <  $\emptyset$  else  $FN \leftarrow \emptyset$

FORMAT:

1	7	5	4+AC	FSRC
---	---	---	------	------

\* XL = 24 if FE =  $\emptyset$   
 = 56 if FE = 1

\*\*XLL = smallest number that is not identically zero  
 =  $2^{-128}$

\*\*\*XUL = largest number that can be represented  
 =  $2^{127} (1 - 2^{-XL-1})$

$C_{YX}(FSRC)$  is defined as (FSRC) converted from Y mode ( $Y=-X$ ) to the current mode, i.e. Floating or Extended. The source is assumed to be opposite to the current mode. Specifically, if the current mode is F and the FT bit is set, then FSRC  $\langle 63:32 \rangle$  are loaded into AC  $\langle 31:0 \rangle$ . If the FT bit is zero, the result is rounded using FSRC  $\langle 31 \rangle$ . Note that FSRC  $\langle 31:0 \rangle$  is unaffected. If the current mode is E, AC  $\langle 63:32 \rangle$  are loaded from FSRC  $\langle 31:0 \rangle$  and AC  $\langle 31:\emptyset \rangle$  are cleared. Similarly,  $C_{XY}(FSRC)$  converts (FSRC) from X to  $-X$  mode by truncating or rounding ( $FT=1$  or  $\emptyset$  when  $X=E$  or loading trailing zeros if  $X=F$ ).

INSTRUCTION: Store & Convert from Floating/Extended to Extended/Floating

MNEMONIC: STCXY AC, FDST

OPERATION:  $FDST \leftarrow C_{XY}(AC)$  if  $|C_{XY}(AC)| \geq XLL$  or  $FIU=1$   
 else  $FDST \leftarrow \emptyset^{**}$   
 $FC \leftarrow 1$  if  $|AC| \geq 2^{XL}$  else  $FC \leftarrow \emptyset^*$   
 $FV \leftarrow 1$  if  $|AC| \geq XUL$  else  $FV \leftarrow \emptyset^{***}$   
 $FZ \leftarrow 1$  if  $(AC)=\emptyset$  else  $FZ \leftarrow \emptyset$   
 $FN \leftarrow 1$  if  $(AC) < \emptyset$  else  $FN \leftarrow \emptyset$

FORMAT:

1	7	6	AC	FDST
---	---	---	----	------

\*  $XL = 24$  if  $FE = \emptyset$   
 $= 56$  if  $FE = 1$

\*\*XLL = smallest number that is not identically zero  
 $= 2^{-128}$

\*\*\*XUL = largest number that can be represented  
 $= 2^{127} (1 - 2^{-XL-1})$

INSTRUCTION: Load & Convert Integer/Double to Floating/Extended

MNEMONIC: LDCJX SRC, AC

OPERATION:  $AC \leftarrow C_{JX}(SRC)$   
 $FC \leftarrow 1$  if  $|C_{JX}(SRC)| \geq 2^{XL}$  else  $FC \leftarrow \emptyset^{**}$   
 $FV \leftarrow \emptyset$   
 $FZ \leftarrow 1$  if  $(AC) = \emptyset$  else  $FZ \leftarrow \emptyset$   
 $FN \leftarrow 1$  if  $(AC) < \emptyset$  else  $FN \leftarrow \emptyset$

FORMAT:

1	7	6	4 +AC	SRC
---	---	---	-------	-----

$C_{JX}(SRC)$  specifies a conversion from an integer with precision specified by J to a floating point number with precision specified by X, i.e. if  $J=I$  and  $X=F$  the source is assumed to be a 16-bit 2's complement integer which is converted to a sign magnitude floating point number with a 24 bit fraction. In the case of  $C_{DF}(SRC)$ , the fraction is truncated, i.e., only the highest 24 significant digits are used.

INSTRUCTION: Store Converted from Floating/Extended to Integer/Double

MNEMONIC: STCXJ AC, DST

OPERATION:  $DST \leftarrow C_{XJ}(AC)$  if  $-2^{JL} \leq C_{XJ}(AC) \leq 2^{JL}-1$  else  $DST \leftarrow (DST) *$   
 $FC \leftarrow 1$  if  $-2^{JL} > C_{XJ}(AC) > 2^{JL}-1$  else  $FC \leftarrow \emptyset *$   
 $FV \leftarrow \emptyset$   
 $FZ \leftarrow 1$  if  $(DST) = \emptyset$  else  $FZ \leftarrow \emptyset$   
 $FN \leftarrow 1$  if  $(DST) < \emptyset$  else  $FN \leftarrow \emptyset$

FORMAT:

1	7	7	AC	DST
---	---	---	----	-----

\* $JL=15$  if FD mode =  $\emptyset$   
 $=31$  if FD mode = 1

\*\* $XL=24$  if FE= $\emptyset$   
 $=56$  if FE=1

INSTRUCTION: Load FPU's Program Status

MNEMONIC: LDFPS SRC

OPERATION:  $FPS \leftarrow (SRC)$

FORMAT: 

1	7	7	4	SRC
---	---	---	---	-----

INSTRUCTION: Store FPU's Program Status

MNEMONIC: STFPS DST

OPERATION:  $DST \leftarrow (FPS)$

FORMAT: 

1	7	7	5	DST
---	---	---	---	-----

INSTRUCTION: Store FPU's Exception Code

MNEMONIC: STFEC DST

OPERATION:  $DST \leftarrow (FEC) * 2$

FORMAT: 

1	7	7	6	DST
---	---	---	---	-----



INSTRUCTION:           LOAD Maintenance Counter

MNEMONIC:             LDMC

OPERATION:            MC  $\leftarrow$  (R $\emptyset$ )

FORMAT:

1	7	$\emptyset$	$\emptyset$	1	$\emptyset$
---	---	-------------	-------------	---	-------------

The ROM cycle counter (RCC) decrements each ROM cycle. In maintenance mode the next ROM word will not be fetched if the RCC= $\emptyset$ .

INSTRUCTION:           Store A register in AC $\emptyset$

MNEMONIC:             STA $\emptyset$

OPERATION:            AC $\emptyset$   $\leftarrow$  (AR)

FORMAT:

1	7	$\emptyset$	$\emptyset$	1	1
---	---	-------------	-------------	---	---

INSTRUCTION:           Store B register in AC $\emptyset$

MNEMONIC:             STB $\emptyset$

OPERATION:            AC $\emptyset$   $\leftarrow$  (BR)

FORMAT:

1	7	$\emptyset$	$\emptyset$	1	2
---	---	-------------	-------------	---	---

INSTRUCTION: Store Q register in AC $\emptyset$

MNEMONIC: STQ $\emptyset$

OPERATION: BR  $\leftarrow$  (QR)  
AC $\emptyset$   $\leftarrow$  (BR)

FORMAT:

1	7	$\emptyset$	$\emptyset$	1	3
---	---	-------------	-------------	---	---

## APPENDIX A

SUMMARY OF FPU INSTRUCTION SET

<u>INSTRUCTION</u>	<u>MNEMONIC</u>	<u>OP CODE</u>	<u>DESCRIPTION</u>
Copy Floating Condition Codes	CFCC	170000	CC ← FCC
Set Floating Mode	SETF	170001	FE ← 0
Set Extended Mode	SETE	170002	FE ← 1
Load Maintenance Counter	LDMC	170010	MC ← (RO)
Store AR Register in AC $\emptyset$	STA $\emptyset$	170011	AC $\emptyset$ ← (AR)
Store BR Register in AC $\emptyset$	STB $\emptyset$	170012	AC $\emptyset$ ← (BR)
Store QR Register AC $\emptyset$	STQ $\emptyset$	170013	BR ← (QR) AC $\emptyset$ ← (BR)
Integerize Floating/ Extended	INTX FSRC)	170300+FSRC	AC4 ← integer part of (FSRC); AC5 ← fractional part of (FSRC)
Clear Floating/ Extended	CLRX FDST	170400+FDST	FDST ← $\emptyset$

## APPENDIX A (continued)

<u>INSTRUCTION</u>	<u>MNEMONIC</u>	<u>OP CODE</u>	<u>DESCRIPTION</u>
Negate Floating/ Extended	NEGX FDST	170500+FDST	$FDST \leftarrow - (FDST)$
Make Absolute Floating/Extended	ABSX FDST	170600 +FDST	$FDST \leftarrow  (FDST) $
Test Floating/ Extended	TSTX FDST	170700+FDST	$FCC \leftarrow \text{condition of } (FDST)$
Load Floating/ Extended	LDX FSRC, AC	171000+AC*100+FSRC	$AC \leftarrow (FSRC)$
Store Floating/ Extended	STX AC, FDST	171400+AC*100+FDST	$FDST \leftarrow AC$
Add Floating/ Extended	ADDX FSRC, AC	172000+AC*100+FSRC	$AC \leftarrow (AC) + (FSRC)$
Subtract Floating/ Extended	SUBX FSRC, AC	172300+AC*100+FSRC	$AC \leftarrow (AC) - (FSRC)$
*Multiply Floating/ Extended	MULX FSRC, AC	173000+AC*100+FSRC	$AC \leftarrow (AC) * (FSRC)$
Divide Floating/ Extended	DIVX FSRC, AC	173400+AC*100+FSRC	$AC \leftarrow (AC) / (FSRC)$
Reverse Subtract Floating/Extended	RSUBX FSRC, AC	174000+AC*100+FSRC	$AC \leftarrow (FSRC) - (AC)$
Compare Floating/ Extended	CMPX AC, FDST	174400+AC*100+FSRC	$FCC \leftarrow \text{condition of } (FDST) - (AC)$

## APPENDIX A (continued)

<u>INSTRUCTION</u>	<u>MNEMONIC</u>	<u>OP CODE</u>	<u>DESCRIPTION</u>
Reverse Divide Floating/Extended	RDIVX FSRC,AC	175000+AC*100+FSRC	AC ← (FSRC) / (AC)
Load & Convert from Extended/Floating to Floating/Extended	LDCYX FSRC,AC	175400+AC*100+FSRC	AC ← converted (FSRC)
Store & Convert from Floating/Extended to Extended/Floating	STCYX AC,FDST	176000+AC*100+FDST	FDST ← converted (AC)
Load & Convert Integer/ Double to Floating/ Extended	LDCJX SRC,AC	176400+AC*100+SRC	AC ← converted (SRC)
Store & Convert Floating/ Extended to Integer/ Double	STCXJ AC,DST	177000+AC*100+SRC	DST ← converted (AC)
Load FPU's Program Status	LDFPS SRC	177400+SRC	FPS ← (SRC)
Store FPU's Program Status	STFPS DST	177500+DST	DST ← FPS
Store FPU's Inception Code	STFEC DST	177600+DST	DST ← (FEC) * 2

APPENDIX B  
FLOATING POINT EXECUTION TIMES

An initial analysis of our floating point algorithms resulted in the execution times of the table below. These times apply to AC to AC operations.

The following approximation can be used to find the execution time for memory referencing operations: Take time of table below and add to it (0.25  $\mu$ sec + memory access/cycle time) \* number of memory references. This time has to be corrected further for possible memory cycles for the address computation (e.g. add 1 memory access time for mode 6 "A(R)").

FLOATING POINT EXECUTION TIMES FOR AC-AC OPERATIONS

INSTRUCTION	EXECUTION TIME IN $\mu$ SEC			
	SINGLE PRECISION		EXTENDED PRECISION	
	MIN.	MAX.	MIN.	MAX.
ADDX	1.8	3.5	2.4	5.1
SUBX	1.8	3.5	2.4	5.1
MULX	2.7	5.5	4.8	10.0
DIVX	3.0	6.0	5.0	12.0