To: Ron Gruner From: Steve Wallach Topic: FHP Assessment

1 Introduction

This memo presents the conclusions and recommendations based on two trips to the RTP facility. Certain assumptions are made about the near and long term goals of FHP. Where appropriate these assumptions will be stated. For purposes of rationally presenting my thoughts, the following sections are developed: Name\_Space, JID/40N, re-microprogramming, KDS (JS Kernel), microFHP, and conclusion.

2 Name\_Space

There are two primary forces driving the redesign of Name\_Space: Performance of SPL programs and code density. The latter with respect to VAX comparisons.

Taking the latter first. Rather than attempting to explain why VAX has better code density, or the validity of the small sample, it is worthwhile noting that with the present definition of Name\_Space some very interesting side effects results. First of all, names must be 16 bits in length. Why?

- 1) A Name in Name\_Space is not really a user defined name or variable. Consider the existance of the array variable A and the integer scalars I and J. References to A, I, J, A[I], and A[J] requires the compiler use 5 names and not 3. Thus there is a multiplicative effect that generally results when array references occur. Of course, array references use the longest NIE (128 bits). The present effort to produce new 32 and 64 bit NTE's will solve al major part of this problem.
- 2) All the names in independently compiled subroutines when bound into a procedure object, must be unique in that object. That is there can not be multiple uses of the same name. In reality, this is somewhat of a lie. This bind

17:12:33 8/Sep/80

strategy must be adopted due to the extensive overhead of entering subroutine using the procedure object environments data in the object's root. If this strategy were adopted, an 8 bit name would suffice for many subroutines, but: of course subroutine call overhead becomes excessive and there goes performance.

Conclusion, proceed with the following redefinition, obtain better code density than VAX (for whatever that's worth).

These issues I consider more academic in nature, when compared to some other issues that either result in more substantive metrics and perceived marketing advantages. Let's enumerate some of these points.

- 1) Some of the SPL benchmarks run indicate a severe performance penalty as contrasted to the MV/8000. Since in a "TYPICAL" system it is not unusual to spend 50% or more in the system, this must be corrected. The correction being identification of the common addressing modes used in NTL's and accelerating them. However what thought has been given to NTE reference patterns for languages other that SPL and Fortran. Presumably COBJL performance will become an issue Does COBOL have a sufficiently similar or some day. different address mode pattern than SP\_L and Fortran? If so the present effort also accelerates CUBOL. If it does not, this should be considered. I do not know the answer to: this question, but someone should provide one. Pascal should be considered as part of this effort.
- 2) An opinion already publically discussed is the extent that your architecture is superior to a competitor is a function of the user perceived benefits. The notion of dial a pecision intergers and float in Fortran previously was noted. While not as obvious, but mentioned by some scientific and technology bigots is the notion of mixed mode arithmetic. This was discussed briefly with some people. From a performance and cope generator viewooint, эnу However, in the never ending advantages are minimal. search for all the hype and impact, direct support of mixed mode can be a product differentiator. The basic Name\_Space structure effectively minics a data tagged architecture. By directly supporting data types in a NTE and thus having a generic ADD and not Add Integer and ADD Float, some product differentiation can be obtained (the System/38 supports this type of "GENERIC" instruction structure). This feature will turn on certain customers. The most frequen-

17:12:33 8/Sep/80 tly mentioned drawbacks to this feature are the cost of implementation and what does it buy since mixed mode arithmetics do not occur frequently.

I believe that the latter objection has been answered. The first objection is simply a reaction to a change. From first nand experience on the Burroughs 6700 class of machines and the Raytheon Data Tagged AADC (I mentioned this only to prove the point), data tagging and mixed mode can be supported with NU loss of performance for the general case of constant arithmetics.

5) The entire issue of S\_languages, binding subroutines of different language together, and the development of support for newer high level languages as they come along.

Examining the latter two points and reaching conclusions based on the presented facts reveals as follows. Though not quite clear from the documentation (if it is not true , it could be made to work), multiple Procedure Environment Descriptor's can be supported in the same procedure object. Assume the following simplifications can be made: lhe static data pointer remains unchanged, the name table pointer remains unchanged (there is no reason that names across S-interpreter can not be supported), a common subroutine call and return mechanism across all S-languages exists (I believe this is the case), and the S-language identity can be incorporated in the NTE used to name the called subroutine. One obvious question, is the mechanism used to invoke the original S-language of the caller. Again, the arch. document is unclear about the macrostate stored in the frame pushed on the current stack. l am assuming that a reasonably sized bit field (4-8) can be placed on the stack and used to identify the S-language of the caller. In effect what has been described in a flat inter-language call.

If this is done and the S-language interpreter is present (which it is on sprint), the overhead of S\_language switch is minimal (I believe 1 or 2 microcycles as worst). More will be said about speed versus architecure after the next point.

Are multiple S\_languages a boom or bane ? Borrowing from a

17:12:33 8/Sep/80

past leader, it is neither, they are a canard. 1 believe the following is true about S\_languages especially with respect to FHP. The intersection of the Fortran, SPL, and Cobol 3\_languages, the instructions in common (forgetting opcode assignment), and the disjoint set correlates with the Eclipse M/600 or the MV/8000. The word addressed floating point is certainly the Eclipse Fortran S\_language, the pyte granular Commercial set is the Copol S\_language set, and the character instruction and the various privileged instructions are the SPL set. In reality all S= language design has accomplished is permitted a degree of freedon to the compiler designers in determining the desired object code to generated. Unlike the B-1700 where the compiler writer in addition to instruction semantics could choose descriptor format (read NTE format), in FHP only instruction semantics are permitted. Already existing in the structure of the NTE is the superset notion of all the addressing and naming conventions required for the anticipated high level languages (and not augmentable by a S-language).

What does this mean? Other than eliminating multiple opcode decoders in the  $I^{2}$  (so I'm told), very little else would be gained in the some total of all microcode develobed for Sprint and I suspect future FHP's. lhe same functionality will always exist. Of course the real downside (as you correctly perceived), is that software development may want an S\_language for each additional compiler supported. This could then potentially translate into the massive microprogramming effort SO feared. Classically additional languages required additional: run-time support. And the run-time support was transportable from one machine to another. I think this issue is more a question of management control. By simply dictating that until further notice, no additional S-languages, and that all future compilers must choose one of the available languages, you still maintaing the user perception of al benefit of the architecture and you leave open the opportunity for augmentation of the architecture in an orderly way in the future. This approach only makes sense if one of the available S\_languages can be used for the more immedia ate compiler (i.e, Pascal, PL/1, C, RPG, TBasic , TAPL) development efforts

What this means is that all the technical and performance objections to S\_languages can be solved with the same level.

> 17:12:33 8/Sep/80

of thought as now going in to Name \_Space or recoding the Kernel microcode. All other objections can be handled with a firm management committment.

Une last observation. Based on the  $S_{ops}$  listed in the FHP arch. document dated Nov/5/79, the intersection of the S-languages supports my previous analogy with the MV/8000. There are approximately 73 SPL S-oos. The intersection of SPL and Fortran results in just 30 additional S-ops being defined. As you can imagined, these 30 deal with the floating point and character string data types of Fortran. Again intersecting this set with Cbboly results in 48 additional S-ops. You guessed it, the additional S-ops deal with decimal data types and the editing and searching semantics of Cobol. In total 141 unique S\_ops exists. what does this analysis mean. One opvious conclusion, is to combine all S-ops into one instruction set. This eliminates all context switches and program bind proplems among different 5-ops modules. What we see, is a scientific and conmercial instruction set built on a pase (or SPL) instruction set. Given the 8 bit S-op encoding, sufficient space is left for expansion. Also all three base languages have the complete instruction set available. Ihus, Fortran COULD have some of the commercial capabilities of Cobol: (1 am not advocating this, but only what can be done. Ôf course P\_V1 DDES have both commercial, character, and scientific data types). <u>This should be given serious</u> consideration.

The next step is to then realize that most of the instruction set has the same semantics applied to different data types. This naturally leads to binding the data type in the NIE. Other than the reasons previously given for this abstraction, some secondary benefits are: for languages with run-time coercion of data types (like APL), a natural, way exists to support such an interpreter, and lastly a convenient way is now defined to MATCH the data types of input actual argument against the data type expected. The Fortran standard says passing the wrong argument type is an Undefined results occur. There has been many a error. paper that mentions this error as an area that should be given aid by the compiler. Software reliability is a big selling feature. In this case, the data types (optionally) of the passed arguments are matched against a template at the calilee'ss site.

> 17:12:33 8/Sep/80

4) Presently intra-object calls that stay within the current domain are expensive. This expense results in multiple copies of the run-time environment existing in memory. A copy is bound to every procedure object and not shared. Though the architecture supports shared procedure objects, the call time and name cache fill/flush situation reduces performance. It seems inconceivable that in an architecture, with its object addressing, that the runtimer are not shared as a matter of policy.

There are two suggestions in this area. Provide a FLAT intra-object procedure call and return, or sufficiently expediate architecturally CALL for this case, and secondly redisgn the name cache such that its asociation is on AON!!Name not just Name. This eliminates the need for name cache flush/fill on a return.

## 3 UID/AUN and machine state issues

Many of the performance issues, especially during cross domain call and fault processing is the conversions between ADN/UID and JID/AUN, and the potential for an excessive amount of machine saving and restoring.

The ADN/UID issue is the more peculiar of the two. Promoted as the vechicle to avoid ambiquous names, make software development assilier and more reliable (the ability to encassulate, at will, data or other things in name objects) will probably achieve many of these objectives. However, since that cost of hardware is not yet free, the exact construct that software wants to eliminate has become a burden on the hardware. ADN's were created so that software could easily index into sparse (relative to the length of a UID) tables. Allso to eliminate the ourden on the hardware to maintain 80 bits of object ID. The goal is noble, but the cure may be worse that the cause. Extensive time is spent converting ADN to persistant UID during a context swap. This is due to the ADN not be persistant. The ATJ must be purged upon context swap, since it associates on ADN and not UID.

There are proposals to fix this problem. All of which involve additional hardware accelerators for UID/ADN conversions. Before any consideration be given to applying hardware solutions, ananalysis must be made of the design of the ADN/UID abstraction. Afterall, this is the root of the problem. While I am not as yet finished with my analysis (in reality in cooperation with Steve

> 17:12:33 8/Sep/80

> > ····· ··· ··· ··· ···

Data General Corporation Company Confidential

2

Schleimer and Doug Wells), some simplifications are worth noting. Among them are:

- 1) Contracting the ULD from 80 bits to 64 bits.
- 2) If AJN's remain, allocate the lower: half (the first 8 KD to: system maintained. Thus this AUN need never be converted to UID and the construction of a split AIU becomes feasible. The system ATU need not be flushed on context switches, since its ADN association are system-wide. (Already implemented, conceptually or in reality on the Prime 750, VAX, and the MV/8000.)
- 3) Investigate the possibility for defining for FHP\_1 (using the nomenclature in one of your memos) to a 12B pointer for which only 64 bits are "ACTIVELy" interpreted. Actively is presently undefined. Or simply, for FHP\_1 only supporting a 64 bit pointer, but allocating 128 bits in main memory. Thus, when it becomes judicious, all 128 of the pointer: have meaning. It is my understanding that in the first releases of the JS, UID's (for the user) are not supported. The System/38 employs a similar approach. Their architecture provides for a 40 bit segnent number, only 24 of which is supported in the first incarnation. Ihis is only worth considering if the AIU is widehed to accept a 64 bit address.
- 4) Since only four domains are supported, incorporate the protection access bits with physical address generation. Thus the ATU serves two purposes (protection validity and physical address generation) and the protection cache is eliminated.

Machine state issues involve the excessive machine state that is either created across machine instructions or saved as a result of page faults. The excessive machine state preated comes aboutdue to the definition and implementation of the CALL instruction. Needless to say this is not a surprise and an extensive effort is underway to correct this situation.

The state save issue with respect to page faults may not be so: easy to fix. Experience with the MV/8000 indicates that in the vast majority of time a short context block (only useable state need be saved). In effect most instructions are restartable. This is not due to the implementation nor the architecture but as a result of the fact that most instructions perform very simple

> 17:12:33 8/Sep/80

operations (the same is true in Sprint - examine the s-op distribution for SPL programs). The response to why Sprint can not do this (other than the present design) generally involves responses like: For worse case we have to be capable of saving all state, the page fault handler in microcode using additional frames on the nicrostack, thereby creating state that must be saved, and the page fault handler resides on its own unique virtual processor thus a process switch must be performed. Again the problem manifests itself due to the high level design and not the implementation.

This design must be re-evaluated. There are two many cases that the microcode is structured solely to mimic a high level software construct without the appropriate reduction to a hardware control mechanism. The advantages of this high level abstraction are clear (they happen to be Huber's  $\Xi_{1}\Xi_{2}$  thesis at M.1.1). However the same result can be achieved by a redesign that puts some additional burden on the kernel software. This burden would relieved of the microcode and make it feasible to simply restart instructions. Ultimately, state saving and restoring is more efficient due to the elimination of additional microstate.

4 KUS - Kernel

This discussion pertains to the certain capabilities KOS does not support that it should. ADS and ADS/V3: permit three process types to exist:swappable, pre-emptible, and resident. As near as I can teld, KOS only permits swappable (or general ourpose interactive user). If the first product offerring only contains Fortran and/or Sprint is to sell into the real-time marketplace, some additional capabilities over and above that which is presently supported must be provided.

These capabilities must include: 1)the notion of a resident process. KDS already permits this by virtue of the page fault handler virtual processor. It seems appropriate that a user visible virtual processor type of resident is appropriate. Uther= wise there is no guaranteed interrupt response time 2)the ability to wire and unwire pages of a resident's processes working set.

Effectively, the resident process of AUS/VS does not mean that the entire process is resident, only that the ?WIRE and ?UNWIRE calls are supported. Pages that are not wired are faulted in and out.

Additionally there appears to be no notion of multi-tasking within JPOS. The comment in reaction to this statement was that multiple processes can be used. However, processes are expensive

> 17:12:33 8/Sep/80

Data General Corporation Company Confidential

3

to create, manage, and provide communciations between as compared to tasks. The question of multi-tasking on the MV/8000 and its support under AUS/VS and Fortran 777 was a very common question during customer presentations.

A question you can answer is to what extent does Sprint have to be perceived as a high end offerring of the current product line? Expanding one step further is Sprint's relationship with ELLIPSE hardware and software the same as BBS's relationship?

## 5 Re-microprogramming

Clearly identified as one of the critical redesigns necesarry for performance enchancement. The organization of the approach taken by L. Schiller should get acceptable results. If one were to read his workplan, the interesting notion of microcode generation from IBM's P\_VS is brought up. Independent of S=languages, the amount of microcode needed to be developed for FHP with its extensive support for US functions would dictate that the feasibility of generating microcode in this fashion at least be examined.

## 5 microfHP

Several suggestions were made relative to the block diagrams presented. The major one being incorporating a smaller name cache within the ALU chip. This eliminates the need for one of the chips and reduces the risk of the project. Uf course the performance consequences of this are not quantified. Additional analysis could not be done without a detailed determination of the chip arearequired for all the listed functions and blocks. Mitchell realizes that this is the next critical step in the project.

## 7 Conclusion

The above sections have enumerated suggestions for some changes to the architecture/implementation of Sprint. It has been assumed that the goals and objectives of sprint (as you said at your staff meeting that I was present at) was to make the computing world and Data General say that this is workth waiting for and/or this is the best thing since white bread.

The way the architecture contributes to this is directly proportional to the perceived user benefits. Clearly performance is one obvious user benefit. Whetstope at the 2000 level is vary good, though higher would be better with no incremental product

> 17:12:33 8/Sep/80

cost. Will the vectorizer do this? It would be highly desirable to have this ready by announcement.

A stronger statement concerning compatibility would help. Dempatibility comes in many flavors. The MV/8000 chose binary, Sprint has somewhat chosen high level language and AUS file level. This should be sufficient with the right amount of marketing hype and technical backup. Many present DG users though impressed with the binary compatibity of the MV/8000 would have been satisfied with a recompile. In fact Cobol users must recompile. I don't known what the situation is, but the Sprint Fortran '77' or some mechanical translator should be able to compile Eclipse Fortran 5 to Sprint. Another desirable feature of Fortran should be the capability to compile IBM's Fortran. Prime's Fortran does this, and they get their fair share of business via this route.

In fact a generic approach of compiling IBM fortran and Cobol should enhance Sprint.

A fact brought up by many DG customers is the lack of a 32-bit bus for the MV/8000. DECLuses their DR780 attachment to the S31 to their advantage in many marketing situations.

Since Sprint is inherently a tightly coupled multi-processor, provide the mechanisms that permit two JP's controlled by one IUP. Perhaps this is the mig-like kicker so frequently mentioned.

Of course the real win would be to announce any type of tandem, non-stop, or other ARM features. These features always help sell. It is not clear, within the time contraints of Sprint announcement goals what can be accomplished. However here are some ideas that may help Sprint's ARM story.

- 1) Permit the memory and I/D controllers boards to be electrically disconnected from the host processor without powering down the entire bay.
- Permit a form of graceful degradation by microcoding the E=30X functionality into the fetch unit. Thus if any off the 3 E=80X boards fail, processing can continue at slower performance.
- 3) Permit the processor to continue functioning with TBS replaced with a 6053 or equivalent. One of the often mentioned remarks concerning the MV/8000 is the sensitivity of the machine to the floppy, MBC, and 6053. Customers wanted to know if backup units could be made available. Nhile customers may have spare 6053's, 1 doubt a spare TBS

17:12:33 8/Sep/80

wild be available.

Graceful degradation as a form of high availability is as valid an approach as duplication (though not guaranteeing the same level of availability). In an uni-processor that may be the best one could hope for.