IBM

International Technical Support Centers

IBM 3090 VECTOR FACILITY TECHNICAL REFERENCE

GG24-3058-0

IBM 3090 Vector Facility
Technical Reference

Document Number GG24-3058

July 30th, 1986

International Technical Support Center
Poughkeepsie, N.Y. 12603

First Edition (July 1986)

This edition applies the IBM 3090 Vector Facility and its operating system support in MVS/SP Version 2 Release 1.3 Vector Facility Enhancement.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM program product in this document is not intended to state or imply that only IBM's program product may be used. Any functionally equivalent program may be used instead.

The information contained in this document has not been submitted to any formal IBM test and is distributed on an 'As Is' basis without any warranty either express or implied. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Publications are not stocked at the address given below. Requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM World Trade Corporation, International Technical Support Center Department 466/H52 Building 930, P.O.Box 390, Poughkeepsie, N.Y. 12602 U.S.A. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

c Copyright International Business Machines Corporation 1986

ABSTRACT

This document describes the IBM 3090 Vector Facility and the software support provided by the MVS/SP Version 2 Release 1.3 Vector Facility Enhancement and related software products.

This document provides SEs and IBM personnel involved in installing VF with an overview of the principles of the Facility, and how it fits into a large systems installation.

The information presented is based mainly on early hardware and programming specifications. Other sources of information are the installation of VF at ITSC Poughkeepsie in September 1985, and the center's experience using the facility.

LSYS ESSYS (68 pages)

PREFACE

This Bulletin provides information on installing the Vector Facility (VF) and discusses the items that have to be taken into consideration to ensure that the engineering and scientific workloads on a large systems processor can use VF.

HOW THIS BULLETIN IS ORGANIZED

This bulletin has three parts:

1. Vector Facility Architecture -
   A technical description of the Vector Facility and its functions, including instructions, registers, and operation principles. This part also highlights the physical installation requirements and hardware and software dependencies.

2. MVS/XA Support -
   A discussion of enhancements made in the MVS/XA system in order to make VF an integral part of a large system computing facility.

3. Implementation and Migration -
   How to actually install VF, and what to look out for in this process.

HOW TO USE THIS BULLETIN

This bulletin is a reference text for planning a VF installation, and provides some detailed information on the impact VF would have on a running installation.

It also provides reference material for people who are involved in application migration, and gives some understanding of the vector code generated in a FORTRAN compile, so that it can be modified to enhance performance.

RELATED PUBLICATIONS

The following documents provide additional information:

- IBM 3090 Processor Complex - A New Dimension in Versatility, G520-6063
- IBM System Summary: Processors, GA22-7001

- IBM System/370 3090 Processor Complex Installation Manual - Physical Planning, GC22-7074 with TNL GN22-2329
- IBM 3090 Model 200 Processor Complex Physical Planning Templates, GX22-7107
- IBM 3090 Model 200 and Model 400 Functional Characteristics, SA22-7121
- IBM 3090 Model 200 and Model 400 Channel Characteristics and Channel Configuration Guide, SA22-7120
- IBM System/370 Vector Operations, SA22-7125

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

The IBM 3090 Vector Facility (VF) was introduced to enhance the Engineering/Scientific (E/S) computation capabilities of the 3090 processors, and at the same time maintain the single system image that characterizes large system installations.

Along with the VF hardware facility, software was provided to enable E/S applications to utilize the new instructions. The software consists of:

- Enhancements to MVS/XA to allow it to manage the VF tasks along with current workloads.
- Enhancements to VM/SP HPO to enable CMS virtual machines to use VF.
- VS FORTRAN Version 2 to generate vector code from FORTRAN source statements.
- Other programs with E/S application oriented subroutines, and basic mathematical functions for vector processing.
- VPSS/VF to emulate 3838 Array Processor functions on VF.

The following discussion concentrates on the basic principles of VF itself and the MVS/XA operating system support, and addresses the implementation process from this viewpoint.

The Vector Facility (VF) is an extension to the execution facilities of a 3090 Central Processor. This extension enables the 3090 to execute vector instructions, which operate on strings of arithmetic data, called vectors.

This extension of the execution capabilities includes:

- 16 vector registers
- three new control registers
- 171 new vector instructions

The VF hardware is an option that may be attached to one or more of the CPs in a 3090 complex. The 3090 model 200 can have up to two VFs and the model 400 up to four.

The following chapters discuss in some detail the architecture of VF and the new functions introduced, along with examples of how the new instructions can be utilized. They also discuss installation considerations and requirements, configurability, and give an overview of the available software support.

The Vector Facility introduces the following functional elements as an extension to the 370 architecture:

- 16 vector registers

  These registers have a length of 32 bits, and a depth of 128, which is the 3090 VF section size. (See Figure 1) Therefore a vector register can hold an array (vector) of 32-bit numbers with a maximum length of 128. It can be used for a vector of fixed point numbers or a vector of single precision (short) floating point numbers.

  Vector registers may also be combined into pairs (even/odd pairs) and then carry 128-element vectors of long floating point numbers.

  Most vector instructions have one or more vector registers or vector register pairs as operands, and the format of the particular instruction determines the type of data contained in the vector register.

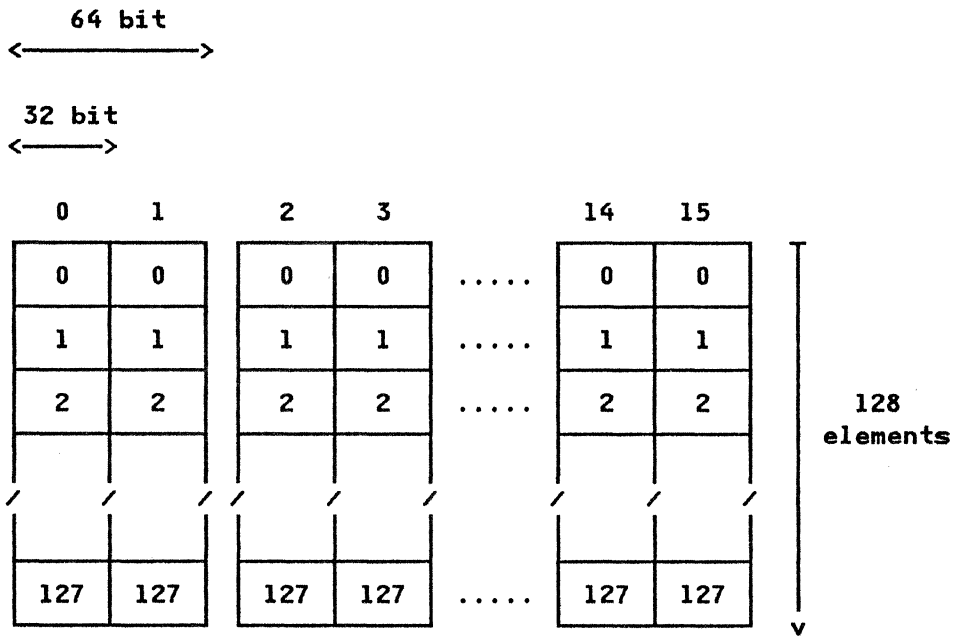


Figure 1. Vector Registers

- Three vector control registers

  These registers are provided to control the execution of vector operations and to record usage of the vector facility. These registers are:

  - Vector Status Register (VSR)
  - Vector Mask Register (VMR)
  - Vector Activity Count (VAC) Register

  These registers are discussed in Chapter 2.

- 171 new instructions

  Most of these instructions operate on vectors in storage or in vector registers. These instructions are discussed in Chapter 3.

In addition, the VF makes use of a new bit in control register 0, the Vector Control Bit (VCB). The VCB must be on for the VF to execute vector instructions.

Also, there are two new program exception codes:

1. Vector Operation Exception (VOP)

   This exception is encountered when a vector instruction is executed on a CP that has the VF installed but the VCB is not set on.

   It is also taken when the VF is not installed on that CP, but it is installed on another CP in the complex, and also if the VF is configured offline on the CP.

   This mechanism is utilized, as described later, by the operating system to create an environment for execution of vector tasks, such as save areas and affinity to a CP, if needed.

2. Unnormalized Operand Exception

   Unlike the S/370 floating point arithmetic, some vector instructions (for example MULTIPLY and DIVIDE) require source operands that are normalized floating point numbers, and this exception is introduced to notify software so that appropriate action may be taken.

For recovery purposes, two new bits are provided in the Machine Check Interruption Code:

Bit 6 — Vector Facility Failure

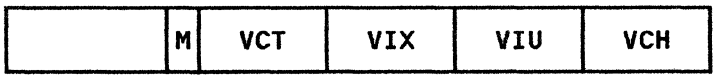
Bit 13 — Vector Facility Source

Bit 13 always accompanies the Instruction Processor (IP) Damage Bit, and, when set, indicates that the VF is the probable cause of IP damage.

Control of execution of vector instructions is provided through the bits of the Vector Status Register (VSR) and the Vector Mask Register (VMR). The Vector Activity Count (VAC) provides a timing facility that can be used to monitor the VF activity.

2.1 VECTOR STATUS REGISTER

The VSR consists of 64 bits, and the format is shown in Figure 2.



M — Vector Mask Mode Bit
VCT — Vector Count
VIX — Vector Interruption Index
VIU — Vector In Use Bits
VCH — Vector Change Bits

Figure 2. Vector Status Register

The Vector Mask Mode Bit, when set on, indicates that the operation on vectors by certain vector instructions is controlled by the Vector Mask Register. The VF is then running in Vector Mask Mode. How Vector Mask Mode is used is discussed in the section "Conditional Execution" on page 22.

The Vector Count indicates to vector instruction the length of the vector participating in an operation.

The Vector Interruption Index (VIX) is incremented during execution of certain vector instructions so as to serve as an index to the starting point when control is returned after an interrupt. The VIX is further discussed in the section "Interruptability" on page 15.

The VIU and VCH bits are sets of bits that indicate which vector register pair is in use (VIU), or have been changed (VCH). These bits, which are used to minimize saving and restoring of vector registers, are discussed in the section "Save/Restore Vector Registers" on page 25.

2.2 VECTOR MASK REGISTER

The Vector Mask Register (VMR) has one bit position per vector element, and the number of bits is equal to the section size (see Figure 3 on page 8).

The VMR has a dual role: It serves a target of vector compare operations, in much the same way as the PSW condition code is the target of scalar compare operations; and it controls which vector elements are to be part of vector operations when running under mask control.

The active bits are the bit positions lower than the current VCT setting in the VSR. For example, if VCT=50, bit positions 0 through 49 are the active bits. Only the active bits contribute to mask control.

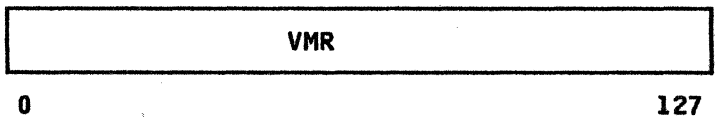


Figure 3. Vector Mask Register

2.3 VECTOR ACTIVITY COUNT

The Vector Activity Count (VAC) register (Figure 4) works much the same way as the Time of Day clock; it is an unsigned binary integer that is incremented by one every microsecond when vector instructions are being executed, and therefore accumulates time when the VF is in use.

As part of the new instruction set, privileged instructions are provided so that the software can modify and store the contents of the VAC register, and use it to record usage time.

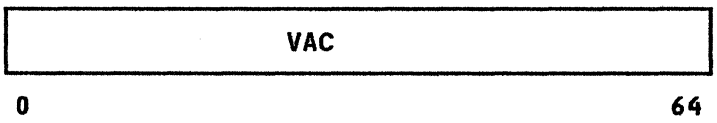


Figure 4. Vector Activity Count Register

2.4 VECTOR PARAMETERS

There are two vector parameters

- Section Size
- Partial Sum Number

Since these are constants that are not fixed by the architecture, there is an instruction provided that will store them in a fullword.

All the 171 new instructions have mnemonics that begin with a V, and since no other instructions begin with this letter, it uniquely identifies the vector instructions. Most of them are proper vector instructions; that is instructions that operate on entire vectors. The others work on the control fields in the VSR, the VMR, and the VAC, or on single vector elements.

The 144 proper instructions can be classified as follows:

- Simple Arithmetic Instructions; namely, ADD, SUBTRACT, MULTIPLY, and DIVIDE

  These instructions always have three operands, two input and one output. The output operand is always a vector register (or register pair for long floating point), and the input operand values are not destroyed by the operation. The input operands may be one of the following:

  - A scalar in a floating point register or a general purpose register
  - A vector in a vector register
  - A vector in storage.

  DIVIDE operates only on floating point data. MULTIPLY and DIVIDE require normalized input data.

- Logical Instructions; AND, OR, and Exclusive OR.

  Operands for these instructions are the same as for the simple arithmetic instructions, except that only 32-bit vector elements are possible and the instructions do not distinguish between fixed and floating point data.

- Compare Instructions

  Operand properties are the same as for simple arithmetic instructions, except that the output vector register operand is replaced by a mask. The mask is used to specify what the two input operands are compared for; for example, equal, high, low, not high, and so on. Unlike the scalar compare instructions, the vector compare instructions do not set a condition code, rather they set bits in the Vector Mask Register (VMR) corresponding to the elements where the compare is successful.

- Load and Store Instructions

  These instructions transfer vectors from storage to vector registers, or vice versa, or from one vector register to another.

- Shift Left/Right Instructions

  In these instructions, one scalar operand specifies the number of bits all elements in a vector register are to be shifted left or right, and then placed in another vector register.

- Maximum/Minimum Absolute/Signed Instructions

  These instructions compare a scalar value with elements of a vector to determine minimum or maximum, and replace the scalar operand with the result.

- Save/Restore Vector Registers

  These instructions act on vector register pairs, and are controlled by the VIU and VCH bits in the VSR.

- Compound Instructions; MULTIPLY AND ADD, MULTIPLY AND SUBTRACT

  These instructions perform two arithmetic operations, with input operands having the same characteristics as for simple MULTIPLY, except that fixed point operation is not permitted. The product of the two input operand values is then simply added to or subtracted from the contents of the output vector register.

- Partial Sum Instructions; ACCUMULATE, and MULTIPLY AND ACCUMULATE

  The ACCUMULATE instruction calculates the sum of all elements in a vector, and the MULTIPLY AND ACCUMULATE multiplies all corresponding elements in two vectors and takes the sum of the products (inner product).

  In both instructions, the sum is calculated by partial sums. The target operand is a vector register that receives a vector of partial sums. The number of elements in the partial sum is a hardware dependant constant and is the length of this vector. A special instruction, SUM PARTIAL SUMS, converts the partial sum vector into a scalar sum.

  In Figure 5, the values in Vector Register 1 (VR1) is accumulated into partial sums in Vector Register 0 (VR0).

  The partial sum instructions can also operate on vectors in storage.

Instructions operating on vectors in storage requires operands to be on an integral boundary. This means that operations on fixed point or short floating point vectors require the storage vector to start on a word boundary. For long floating point vectors, the starting point must be a double word boundary. Violation of these requirements results in a specification exception.

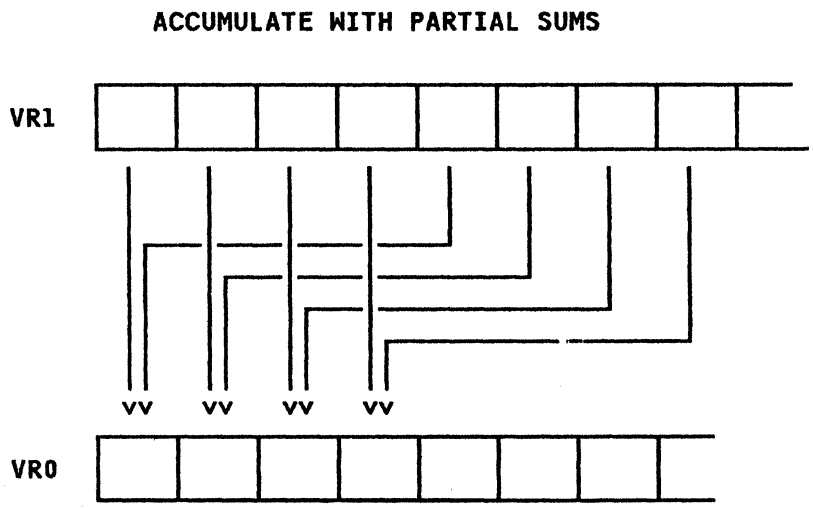


Figure 5. Partial Sum Accumulation

3.1 INSTRUCTION FORMATS

The most common proper vector instructions have certain operand properties. Among these are:

- There are three operands, two source operands and one target operand.
- One of the source operands is either a vector register or a scalar register.
- The other source operand is either a vector register or a vector in storage.
- The target operand is a vector register.
- Source operands are never destroyed by the operation.

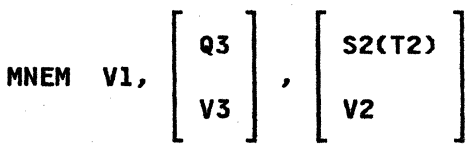
Figure 6
shows the format of this type of instructions. The operands can be considered in the following classes:

QST - Scalar register - storage

QV - Scalar register - vector register

VST - Vector register - storage

VV - Vector register - vector register



V - Vector Register
Q - Scalar Register
ST- Storage
S - Storage Address (General Purpose Register)
T - Stride (General Purpose Register)

Operand 1	Operand 2	Operand3	Format
V	ST	Q	QST
V	V	Q	QV
V	ST	V	VST
V	V	V	VV

Figure 6. Format of the Most Common Instructions

The QST and VST instructions refer to storage vectors, and can have stride specification. The stride is a 32-bit signed integer. It determines the number of element locations by which the operation advances when procedding from one element to the next.

Both the storage address of the vector and the stride are contained in general purpose registers (31-bit addressing).

At the completion of the operation, the address register is updated to point beyond the storage vector. The length of the vector, as counted by the number of elements, is determined by the value of the VCT.

The stride determines which storage positions should be taken as vector elements. For example, stride=1 with a long floating point operation means that every double word should be taken; for stride=2 every other; and so on. The stride may be negative or zero. If negative, the address of the next element will be decremented, and if zero, the same element will be taken every time.

If the stride is not specified, or if 0 is specified in the T2 operand, the default of stride=1 is the assumed.

The instructions belonging to this class are:

- The Simple Arithmetic and the Compound instructions
- The Logical instructions
- The Partial Sum instructions

- The Load/Store instructions, except LOAD INDIRECT and STORE INDIRECT

  This case differs in that Loads and Stores have only two operands, and therefore the Q3/V3 operand does not occur.

- The Compare instructions, except that the V1 operand is replaced by a mask.

Examples of this instruction format are shown in Figure 7.

Note that multiplication of floating point data always results in long floating point products.

```
QST: VMES V0,F0,R2(R3)

     Multiply the contents of floating point reg 0 by every
     element of the vector starting at the address in R2 and
     with the stride in R3, and place the product in VR0 and VR1

QV:  VMQ V0,R3,V2

     Multiply the contents of general register 3 by every element
     of the vector in VR2 and place the product in VR0.

VST: VME  V0,V3,R2(R3)

     Multiply every element of V3 by every element
     of the vector starting at address in R2 and with
     the stride in R3, and place product in VR0 and VR1.

VV:  VMER V0,V3,V2

     Multiply every element of V3 by every element
     of V2, and place the product in VR 0 and VR1.
```

Figure 7. Examples of Instructions

3.2 INTERRUPTABILITY

Vector instructions operating on multiple vector elements, earlier referred to as proper vector instructions, are all interruptible. This means that the operation may be interrupted between elements, and there is a mechanism allowing execution to continue from that point when the task regains control.

For most of these instructions, the VIX field in the VSR is used to monitor the operation. When the instruction starts executing, the VIX has

the value zero, and is incremented by one for every element done. As the VCT determines the number of elements in the vector, the operation is finished when VIX=VCT. At this point, the PSW instruction address is updated to point to the next instruction.

Whenever execution is resumed following an interrupt, the VIX will have the current element number, and after restart, the instruction will continue from that point.

For QST and VST format instructions, the storage address register, RS2, is also incremented so as to point to the current element at any given time.

This interrupt mechanism is illustrated by an example in Figure 8 where two vector registers, VR2 and VR3, are added to VR1.

Mnemonics VAER VR1,VR3,VR2

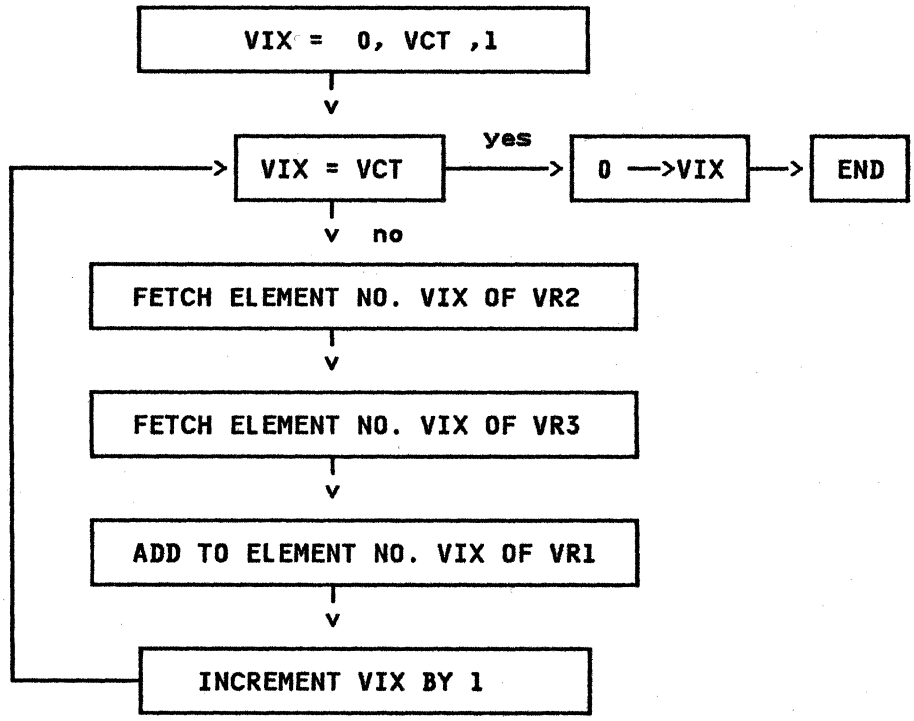


Figure 8. Example of Interrupt Control

The interruptible instructions that use this type of control are called class IM and IC.

3.3 SECTIONING

The section size limits the size of vectors that can be held in vector registers or operated on by vector instructions. If a vector has a length of more than 128 elements, it must be divided into sections of 128 elements or less. For example, a vector of length 300 would be divided into two sections of length 128, and one of length 44.

This process is referred to as sectioning, and is done by software loops called sectioning loops. In a section loop it is necessary to update storage addresses, although storage vector operands usually are updated by the instruction execution itself.

For loop control purposes, the VF has a special instruction:

LOAD VCT AND UPDATE (VLVCU)

The VLVCU instruction has only one operand, a general purpose register (GR) that initially contains the vector length. The instruction operates as follows:

1. Inserts the smaller of the section size (SS) or the content of the GR into the VCT in the VSR.

2. Subtracts the new value of the VCT from the contents of the GR.

3. Sets condition codes corresponding to the result of the two updates.

The condition codes set by the VLVCU may then be used to control the sectioning loop. Four different condition codes can be set:

CC	VCT	GR
0	0	0
1	0	negative
2	SS	positive
3	positive	0

Condition codes 2 and 3 are the ones usually utilized in sectioning loops.

Figure 9 shows an example of events in a loop sectioning a vector of length 330 with a VLVCU instruction, and Figure 10 gives a possible assembler coded sectioning loop for handling the sum of two long vectors.

With vector length equal to 330,
the VLVCU executes four times
to give the following output:

```
1. Load VCT    MIN(128,330) = 128  ----> VCT
   update GR   330 -128     = 202  ----> GR0    CC=2

2. Load VCT    MIN(128,202) = 128  ----> VCT
   update GR   202 - 128    =  74  ----> GR0    CC=2

3. Load VCT    MIN(128,74)  =  74  ----> VCT
   update GR    74 - 74     =   0  ----> GR0    CC=3

4. Load VCT    MIN(128,0)   =   0  ----> VCT
   update GR     0 -  0     =   0  ----> GR0    CC=0
```

Figure 9. Role of the VLVCU Instruction

To do the sum C = A + B,
where A,B, and C are short
floating point vectors:

```
      L R0,N           Load Vector Length to R0
      LA R1,A          Address of A to R1
      LA R2,B          Address of B to R2
      LA R3,C          Address of C to R3
      VLVCU R0         Load And Update R0
      BNP   ERROR      Vector Length must be positive
LP    VLE V1,R1        Load Section of A to V1
      VAE V1,V1,R2     Add Section of B
      VSTE V1,R3       Store Section into C
      VLVCU R0         Load VCT and Update R0
      BNZ LP           Branch on CC not zero
```

Figure 10. Sectioning Loop Example

In Figure 10, storage addresses R1, R2, and R3 in the VST instructions VLE, VAE, and VSTE, respectively, are updated to point to the next section by execution of the instruction.

The initial VLVCU sets the VCR and updates R0 to 128 less than initial length, and the CC here is not used for loop test. The second VLVCU causes a CC=0 after the last section is handled, and the branch is not taken.

3.4 STORAGE ACCESS MODES

Data in storage can be accessed by vector instructions in two ways:

1. Sequential Access

   Most vector instructions access storage sequentially. Vector elements are full words or double words adjacent to each other or uniformly spaced throughout a storage area. Storage vectors are said to have a stride of more than one if the elements are not adjacent. A stride of more than one can be specified in the instruction in terms of the number of words or double words between every element.

   All the arithmetic instructions (including the compound instructions that address storage vectors), loads and stores, compares, and logical instructions access storage as described above, with or without a stride.

2. Indirect Access

   Indirect access refers to the access of vectors in storage when the elements are not equally spaced. There are two ways of accessing storage indirectly: by the Load and Store Indirect instructions; or by using masked addressing to vector elements.

   The Load and Store Indirect instructions are available for long floating point, fixed point, or short floating point operations. They all have a source operand that is a vector register holding a vector of element numbers providing the offset into the storage area.

   The use of the LOAD INDIRECT instruction is shown in Figure 11. Masked accessing is controlled by the Vector Mask Register. Only those elements corresponding to one-bits in the VMR are affected.

   There are two ways of loading and storing vectors under the control of VMR:

   a. Using LOAD MATCHED and STORE MATCHED Instructions

      These instructions work the same way as a load and store with sequential access, including a possible stride, with the exception that only elements corresponding to VMR one-bits are accessed. With a LOAD MATCHED instruction, elements corresponding to the one-bits are brought into the Vector Register in corresponding element locations. (Figure 12). Those locations in the Vector Register corresponding to zero-bits in the VMR are left unchanged.

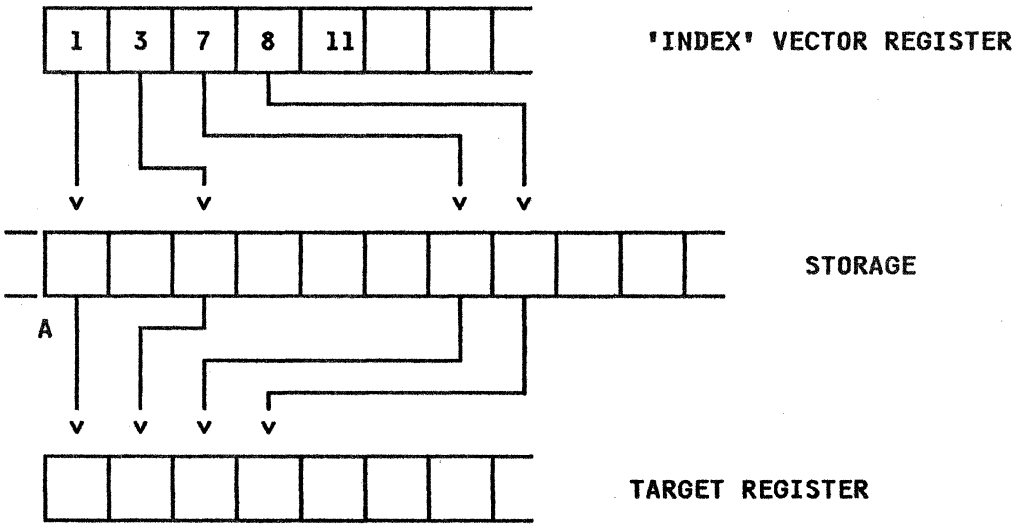


Figure 11. Indirect Access using the Load Indirect Instruction

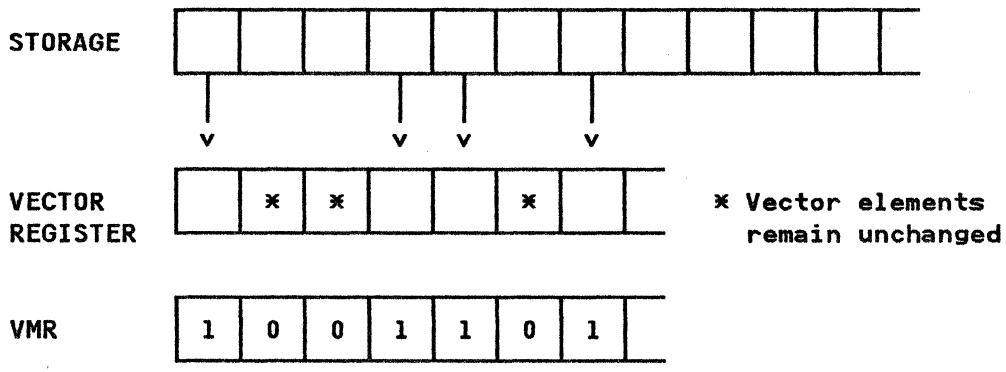


Figure 12. Matched Access using the LOAD MATCHED Instruction

b. Using LOAD EXPANDED and STORE COMPRESSED Instructions

These instructions work similarly; however, as the names indicate, a LOAD EXPANDED takes elements from storage by sequential access, possibly with a stride, and bring them into locations in

the VR according to bit settings in VMR (Figure 13). STORE COMPRESSED works in the opposite way.

Unevenly spaced storage vectors have to be loaded into vector registers in order to perform arithmetic and other operations on them.

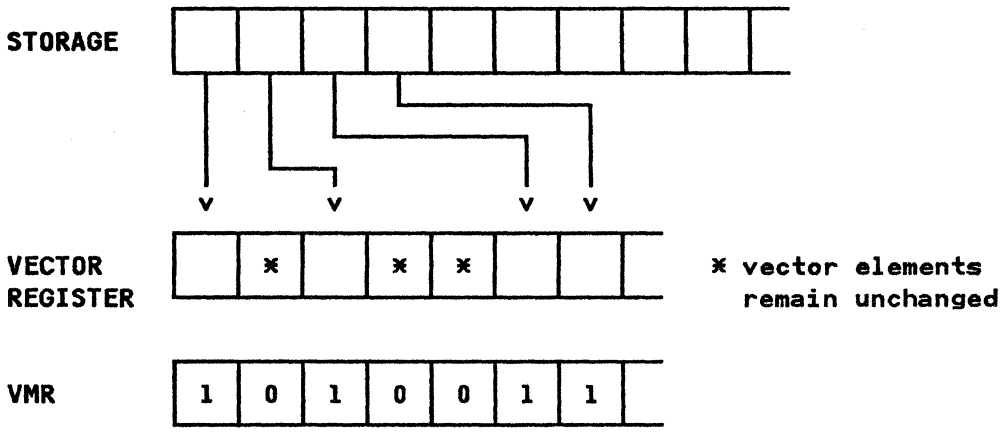


Figure 13. Matched Access using the LOAD EXPANDED Instruction

3.5 SPARSE VECTORS

Sparse vectors are vectors in which most elements are zero, and it is useful to have a representation of such items so that the zero elements do not take up any storage.

A sparse vector can be represented by a dense vector and a bit mask where one-bits represent non-zero elements.

To convert to this scheme, we might use the following steps:

1. Do a VECTOR COMPARE (VCDQ) with scalar zero to create a mask in VMR.

2. Store VMR as a bit mask.

3. Do a STORE COMPRESSED into a dense vector in storage.

Alternatively, if the sparse vector is in storage, the bit mask created by a VECTOR COMPARE can be used by the LOAD BIT INDEX (VLBIX) to create an Index Vector that can be used in a LOAD INDIRECT to place the dense vector in a vector register.

3.6 COMPOUND INSTRUCTIONS

Compound instructions are the most efficient of the arithmetic instructions provided by the VF. They combine two arithmetic operations in one instruction. There are two sets of compound instructions which act only on floating point data.

- MULTIPLY AND ADD instructions
- MULTIPLY AND SUBTRACT instructions

These instructions have the QST, QV, VST, or VV format and operate on short or long floating point data (Figure 6 on page 14). Since three source operands are required, the target operand is also a source operand. Since these two instructions produce a long floating point result, the target operand must be a vector register pair. The MULTIPLY AND ADD instruction multiplies the second operand vector by the third operand vector or scalar and adds the product to the first operand target vector register pair. Similarly, the MULTIPLY AND SUBTRACT instruction subtracts the product from the target.

Thus, the compound instructions, when in the pipeline, can produce two floating point operations per CPU cycle, whereas the other arithmetic instructions produce only one.

3.7 CONDITIONAL EXECUTION

Conditional execution (also called vector mask mode) is execution under control of the active bits in the VMR. The following arithmetic or logical instructions may execute under vector mask mode:

- ACCUMULATE
- ADD
- AND
- DIVIDE
- EXCLUSIVE OR
- LOAD COMPLEMENT
- LOAD NEGATIVE
- LOAD POSITIVE
- MAXIMUM ABSOLUTE
- MAXIMUM SIGNED

- MINIMUM SIGNED
- MULTIPLY
- MULTIPLY AND ACCUMULATE
- MULTIPLY AND ADD
- MULTIPLY AND SUBTRACT
- OR
- SHIFT LEFT SINGLE LOGICAL
- SHIFT RIGHT SINGLE LOGICAL
- SUBTRACT

The execution is under mask mode when the vector mask mode bit in the VSR is set to one. This bit is set on or off using the SET VECTOR MASK MODE (VSVMM) instruction.

When executing in mask mode, operation takes place on all vector elements involved, but only the target operand is updated for those elements that have corresponding bits set to one in the VMR.

The purpose of mask mode is to have operations on individual vector elements depend on results of compare operations on vectors. A vector compare sets VMR bits and the subsequent execution takes place accordingly.

For example, the following FORTRAN construct would safeguard against division by zero by presenting the quotient only for those elements where B(I) is nonzero:

```
    DO 1 I=1,N

    IF(B(I) .NE. 0.0) C(I)=A(I)/B(I)

    1 CONTINUE
```

This may be done with a VECTOR COMPARE, a VECTOR LOAD, a VECTOR DIVIDE and a VECTOR STORE, all under mask mode.

The VECTOR COMPARE instruction is used to set up the vector mask mode environment. The instruction appears in the same four formats as the arithmetic instructions: QST, QV, VST, and VV; and can compare vectors and scalars in the three data formats; fixed point, long floating point, and short floating point. Operand 3 is compared algebraically element-by-element with operand 2.

VECTOR COMPARE differs from these formats only because the first operand vector register is replaced by a 4-bit mask, the modifier. This mask indicates the intention of the compare: equal, second operand high, or some other relationship. The compare does not set a condition code, but

sets VMR bits to one corresponding to those elements of the vector for which the compare is successful, based on what is set in the mask. The layout of the VECTOR COMPARE is shown in Figure 14.

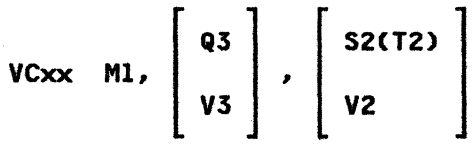


Figure 14. VECTOR COMPARE Instruction

The first three bits in M1 are used as modifiers, and, as shown in Figure 15, they are the same as the condition codes resulting from ordinary (scalar) compare operations. The fourth bit is set to zero.

MASK				OPERATION
Modifier Bits				VMR bit set to one if the element of op3 or scalar in op3, as compared to the element of op2, is:
M0	M1	M2	M3	
0	0	1	0	high
0	1	0	0	low
0	1	1	0	not equal
1	0	0	0	equal
1	0	1	0	not low
1	1	0	0	not high
1	1	1	0	any

Figure 15. Modifier Bit Settings

For example, to compare two long floating point vectors residing in vector register pairs V2 and V4 for equal elements, we could use the instruction:

```
VCDR  8,V2,V4
```

3.8 SAVE/RESTORE VECTOR REGISTERS

To save and restore a vector register involves transfer of considerably more data than with scalar registers - 512 bytes per register or 8K bytes if all registers are saved or restored.

In order to minimize the overhead involved in status saving (when vector tasks are interrupted) and status restoring (when they regain control), the VF includes special instructions and the Vector in Use (VIU) and Vector Change (VCH) bits in the VSR.

Saving and restoring vector registers is also necessary when switching between programs.

3.8.1 Save/Restore Instructions

Nine instructions are used for saving and restoring vector registers and the control registers; among them are the only three privileged instructions in the entire set of vector instructions. The nine instructions are:

1. VRRS - Restore Vector Register
2. VRSV - Save Vector Register
3. VRSVC - Save Changed Vector Register (privileged)
4. VSRRS - Restore Vector Status Register
5. VSRSV - Save Vector Status Register
6. VMRRS - Restore Vector Mask Register
7. VMRSV - Save Vector Mask Register
8. VACRS - Restore Vector Activity Count (privileged)
9. VACSV - Save Vector Activity Count (privileged)

The first three of these instructions work on a vector register pair, and execute under control of the VIU and VCH bits.

3.8.2 VIU and VCH Bits

There are eight VIU bits, one for each vector register pair. A VIU bit on means the VR is in use, so it should be saved. It indicates which VR

pairs should be saved by VRSV and restored by VRRS. The VIU bit is set to one when:

- One of the VRs in the VR pair has been modified, either as a target operand or any other way.

- The corresponding VCH bit is set on.

A VIU bit is set off by a Clear VR (VRCL).

The VIU bits ensure that only VR pairs that are used are saved by the VRSV instruction and restored by the VRRS instruction. The bits may be used by programs running in problem program mode branching to other programs, such as subroutines.

There are also eight VCH bits, one for each vector register pair. A VCH bit on means the VR has been changed. It indicates that the pair should be saved by the VRSVC instruction. The VCH bit is set to one when a VR in the pair is modified by any means, and is set to zero when:

- A VRSVC instruction is issued

- The corresponding VIU bit is set to zero.

The VCH bits are provided for a control program running in supervisor state and using the same save area repeatedly, when switching takes place between tasks using the VF. The necessary processing time is further minimized by avoiding subsequent saves of VRs that have not been changed since the last save. The following scenario should illustrate the use of these bits:

1. A task switch occurs, and a VRSVC saves the VR pairs with VCH bits on.

2. The VRSVC sets the VCH to zero.

3. The task regains control, and the VR with VIU bits on is restored by VRRS.

4. At the next status save, only VCH indicated VR pairs are saved which avoids saving VRs that were not changed since the last save.

VSR, VMR, and VAC will also be saved and restored.

4.0 PROGRAM EXCEPTIONS

Program interruptions or program exceptions for vector instructions generally follow the same rules as for scalar instructions. There are, however, a few differences. Unnormalized Operand Exception is introduced with the VF. The arithmetic exceptions that can be caused by the interruptible instructions are

- Exponent Overflow
- Exponent Underflow
- Fixed Point Overflow
- Floating Point Divide
- Significance
- Unnormalized Operand

Also, a few new circumstances cause specification exceptions, as discussed below.

When a compound instruction results in an exponent overflow, only the multiplication part of the operation is completed. The overflowed product, as in the scalar case, is then placed in the result vector register location.

When a compound operation results in an exponent underflow, no interruption occurs, regardless of the PSW mask, and a true zero takes the place of the product.

Specification exceptions can be encountered with vector instructions in the following cases:

- An invalid VR number is specified when a pair is implied.
- Storage vectors not on integral boundaries.
- Stride is in the same GPR as the address in a VST or QST instruction.
- The third operand is the same GPR as the address in a QST instruction.
- The VSRRS attempts to load into VSR values that have:
    1. Other than all zeroes in bits lower than the Mask Mode Bit
    2. The VCT part exceeds the section size
    3. The VIX part exceeds the section size

- An odd register number is specified with a VRRS, VRSV, or a VRSVC instruction.
- A LOAD ELEMENT (VLEL) or EXTRACT ELEMENT (VXEL) instruction refers to an element number higher than the section size.

The following four programming examples illustrate how a few elementary operations involving vectors can be coded in Assembler Language. FORTRAN notation is used to outline the problem to be solved.

In the example shown in Figure 16, we multiply two vectors, A and B, element-by-element, to form vector C.

The addresses of the storage vectors A, B, and C are initially loaded into registers R1, R2, and R3, respectively. The vector length is loaded into R0 and the stride into R4. Inside the sectioning loop, the VLE loads A into VR1, the VME multiplies that by B, and VSTE stores the result into C. These three instructions each update the storage address register involved, so that for every section done, R1, R2, and R3 are updated to point to the next section.

The VLVCU instruction can be placed on the top of the sectioning loop, since none of the following instructions modify the condition code, and the branch (BC 2) is taken as long as the VCT reflects a full section.

Fortran code:

```
      DO 1 J=1,N
    1 C(J)=A(J)*B(J)
```

Assembler Code:

```
      L      R0,N          Load Vector Length in GR0
      LA     R1,A          Address of A to GR1
      LA     R2,B          Address of B to GR2
      LA     R3,C          Address of C to GR3
      L      R4,T          Stride of A,B,C to GR4
LP    VLVCU  R0            Load and update GR0
      VLE    V1,R1(R4)     Load section of A in V1
      VME    V2,V1,R2(R4)  Multiply section of A by B
      VSTE   V2,R3(R4)     Store section in C
      BC     2,LP          Branch on CC if not last section
```

Figure 16. Multiplication of Two Vectors

In the next example, shown in Figure 17, the MULTIPLY AND ADD instruction is employed to compute the sum:

D=A*B+C

where A, B, C, and D are long floating point vectors, so they all have to be aligned on double-word boundaries.

The addresses of the vectors A, B, C, and D are loaded into registers R1, R2, R3, and R4, respectively. The vector length is loaded into R0. The stride here is assumed to be 1, so (RT) is left out. Inside the sectioning loop, the VLVCU updates VCT and sets a condition code, the VLD loads A into VR2 and VR3, the other VLD loads VR4 and VR5 with C. The VMAD multiplies A (in VR2 and VR3) by B and adds the product to VR4 and VR5, which previously contained C. The resulting VR4 and VR5 is then stored into D by the VSTD. The four storage operands are updated to address the next section, and the branch is taken if there are more sections.

The same computation could have been done with a separate VMD and VAD instruction, and with a similar performance since the second VLD would not have been necessary, so the vector instruction count would be the same.

Fortran code:

```
      REAL*8 A,B,C,D
      DO 1 I=1,N
    1 D(I) = A(I)*B(I) + C(I)
```

Assembler Code:

```
      L     R0,N          Load Vector Length in GR0
      LA    R1,A          Address of A to GR1
      LA    R2,B          Address of B to GR2
      LA    R3,C          Address of C to GR3
      LA    R4,D          Address of D to GR4
LP    VLVCU R0            Load and update GR0
      VLD   V2,R1         Load section of A in V2 and V3
      VLD   V4,R3         Load section of C in V4 and V5
      VMAD  V4,V2,R2      Multiply section A*B and
*                         add to V4 and V5 that initially was C
      VSTD  V4,R4         Store D section from V4 and V5
      BC    2,LP          Branch on CC if not last section
```

Figure 17. Using MULTIPLY AND ADD

The example in Figure 18 shows a computation of the inner product S of two vectors A and B using the partial sum method.

In the assembler code, after initializing the address and length registers, a VZPSD is issued to zero V0 for the partial sum. In the sectioning loop, after issuing the VLVCU, a vector A is loaded into the V8,V9 pair, and the MULTIPLY AND ACCUMULATE (VMCD) with B is issued to cause partial sums to be placed in V0 and V1. When all sections are finished, a SUM PARTIAL SUMS instruction (VSPSD) places the scalar product S in a floating point register, to be subsequently stored into S.

Fortran code:

```
      REAL*8 A,B,S
      S = 0.
      DO 1 I=1,N
    1 S = S + A(I)*B(I)
```

Assembler Code:

```
     L      R0,N          Load Vector Length in GR0
     LA     R1,A          Address of A to GR1
     LA     R2,B          Address of B to GR2
     VZPSD  V0            Zero partial sums
LP   VLVCU  R0            Load And Update GR0
     VLD    V8,R1         Load Section of A in VR8,VR9
     VMCD   V0,V8,R2      Multiply section of B, and place
*                         partial sums in VR0 and VR1
     BC     2,LP          Branch on CC if not last section
     SDR    F0,F0         Clear FPR0 to zero
     VSPSD  V0,F0         Sum partial sums to scalar FPR0
     STD    F0,S          Store scalar sum
```

Figure 18. Partial Sum Method

The last example, shown in Figure 19
describes a conditional execution based on a compare. The computation is to sum two vectors (C = A + B) only for the elements of A that are positive; otherwise, the value of element of C remains unchanged. After the initialization of length and address registers, a floating point register is set to zero for use by the compare instruction and the vector mask mode is set on.

In the loop, A is loaded into the V0 pair, and the VECTOR COMPARE (VLCQ) is specified so as to compare every element of V0 to scalar zero for high; or, in other words to set VMR bits for elements of V0 that are positive. The VAD adds A to B in the V2,V3 pair. Since the program is now running under mask mode, this takes place only for those elements corresponding to VMR bits set to one. In order not to modify other elements of C, we now do a STORE MATCHED (VSTMD) of the V2 pair. When all sections are processed, mask mode is suspended by the VSVMM.

Fortran code:

```
      REAL*8 A,B,C
      DO 1 I=1,N
    1 IF( A(I) .GT. 0.0 ) C(I) = A(I) + B(I)
```

Assembler Code:

```
      L      R0,N          Load Vector Length in GR0
      LA     R1,A          Address of A to GR1
      LA     R2,B          Address of B to GR2
      LA     R3,C          Address of C to GR3
      SDR    F0,F0         Clear FPR0 to Zero
      VSVMM  1             Vector-mask mode on
LP    VLVCU  R0            Load And Update GR0
      VLD    V0,R1         Load Section of A to VR0 and VR1
      VCDQ   4,F0,V0       Compare section of A
*                          greater than content of
*                          FPR0, which is zero
      VAD    V2,V0,R2      Conditionally add section of A
*                          to B into VR2 and VR3
      VSTMD  V2,R3         Store Matched Section in C
      BC     2,LP          Branch on CC if not last section
      VSVMM  0             Vector-mask mode off
```

Figure 19. Vector Compare and Conditional Execution

6.0 CONFIGURABILITY

When a VF is installed, it can be configured online or offline to the CP by the service processor.

Changes to the service processor to allow VF configurability include three new Service Processor Commands:

- READ SCP INFO
- CONNECT VECTOR
- DISCONNECT VECTOR

The operating system uses these commands when physically configuring the VFs on or off.

This section discusses the physical installation of the Vector Facility, highlights the schedules, and outlines the software packages required to utilize the VF.

7.1 PHYSICAL INSTALLATION REQUIREMENTS

The VF represents an extension to the computing facilities of the CP, and it also represents a physical add-on to the central processor. Therefore VF will add to power, floor space, and cooling capacity requirements.

The VF is optional in the sense that it can be installed on only one CP of the 3090-200, or on both. On the 3090-400, it may be installed on one, two, three, or all four of the CPs. However, only one VF can be installed on a CP.

The first VF may be installed only on:

- CP1 on a model 200
- CP1 on either side of a Model 400.

Vector facilities can be installed in two ways:

1. VF(s) can be shipped already installed on the 3090. In this case, no additional install time is required.
2. VF(s) can be installed in the field (MES) on an already installed 3090. In this case, the following system hours apply for Model 200:

   7 hours for installing the first VF (or both at the same time).

   5.5 hours to install the second VF (if the first one is already installed).

The VF requires additional floor space, and at least one additional frame. The 3090 Model 200 requires one additional frame, which can house up to two VFs. This frame represents 20% additional floor space.

The 3090 Model 400 needs only one additional frame if VFs are installed on only one side. If VFs are to be installed on both sides, two additional frames are required.

7.1.1 Power

Each VF adds to the power load:

- 0.1 kVA to the 50/60 Hz circuits, and
- 6.82 kVA to the 400 Hz circuits.

Two VFs on the Model 200 are configured on a single power boundary. The same applies to either side of the Model 400.

7.1.2 Cooling Capacity

Each VF represents the following additional requirement to the cooling capacity:

- 1330 BTU/Hour for air conditioning
- 15390 BTU/Hour for chilled water

7.1.3 Microcode

As the 3090 can run either in 370 or XA mode, different microcode levels are required for supporting the VF in these two modes.

7.2 SOFTWARE SUPPORT

The software support consists of operating system support as well as compilers and application oriented packages.

The VF operates in the following modes:

1. In XA mode under MVS/XA
2. In 370 mode under VM/HPO

The support provided by these two operating systems does not remove any existing capability, so downward compatibility is maintained.

7.2.1 MVS/XA Support

The MVS support is an upgrade to MVS/SP 2.1.3, referred to as Vector Facility Enhancement (VFE), and service updates (PTFs) to related products, as listed below:

- MVS/SP 2.1.3 VFE (JES2 or JES3)
- Either of the following:
  - DFP/XA 1.2 with PTF UZ40733
  - DFP/XA 2.1 with PTF UZ40795
- Either of the following:
  - EREP 3.2 with PTF UR90065
- Either of the following:
  - RMF 3.3 with PTFs UZ90402, UZ44925, and UZ44554
  - RMF 3.4. (VF Support is standard)

These PTFs are applied to PUT level 8505 or later, and should be installed prior to VF installation. The chapter "MVS/XA Support" on page 39 discusses the MVS support in more detail.

7.2.2 VM/HPO Support

This support is provided as an upgrade to VM/HPO 4.2 and enables CMS virtual machines to execute programs with vector instructions.

VM supports the VF in 370 mode, and therefore, the 3090 Model 400 will run only in partitioned mode under VM.

7.2.3 Other Software Support

In order for applications to exploit the VF, some additional software is enhanced or provided. These items fall in two categories:

1. Programs used for preparation of code running on VF. These programs include:
   - Assembler H 2.1.0, which needs PTF UP90225 to assemble the new vector mnemonics.

- VS FORTRAN V2 Compiler, Library and Interactive Debug is capable of vectorization; that is, to create object code containing vector instructions from FORTRAN source code. Vectorization is optional, and can be selected at various levels through parameters.

- FORTRAN Language Conversion program is provided for conversion of IBM FORTRAN IV source code to VS FORTRAN level 77 syntax.

These programs run on any S/370 supported by MVS or VM, and on any level of these operating systems, but vector code generated by Assembler or FORTRAN requires a VF to execute.

2. Subroutines and packages executing vector instructions.

   The Engineering and Scientific Subroutine Library (ESSL) consists of very efficient programs for solving numerical problems by linkage from routines written in FORTRAN or other languages.

   VPSS/VS simulates 3838 operation on the VF.

IBM Vector Facility for the IBM 3090 Central Processing Complex is supported by MVS/System Product Version 2 Release 1.3 Vector Facility Enhancement. Modifications have been made to the MVS/SP 2.1.3 Base Control Program to effectively manage the jobs using vector instructions by dynamically recognizing a vector user and by setting up the proper vector environment for the task.

The following discussion summarizes some of the typical characteristics of the Vector Facility use and the jobs that make use of the facility. Vector jobs typically have the following characteristics:

- High Problem Program State (> 90 %)
- High CPU Utilization (> 80 %)
- Long Running

The Vector Facility architecture introduces:

- Large vector registers (8K bytes)
- Several new vector instructions
- Vector control and status registers
- New program check interruptions
- New machine check interruption codes

When managing vector tasks, the following two factors are the most important ones in the multi-tasking environment:

1. **Vector Status Saving/Restoring**

   The vector tasks use vector registers that contain a large amount of data. Saving/restoring the status for those tasks involves a significant amount of data transfer to/from the main storage, which could result in a system performance degradation in multi-tasking environment.

2. **Vector Affinity**

   The vector tasks have to run on VF-capable processors when they issue vector instructions. When they do not issue vector instructions, they can run on any processors in the complex.

The Dispatcher has been modified to manage these characteristics of vector tasks, along with other related components of MVS. The primary goals are:

- Single system image for a mixed vector and non-vector environment
- No measureable overhead increase for non-vector jobs
- Minimize MVS interaction with vector jobs

Note that the support provided by MVS/SP 2.1.3 Vector Facility Enhancement is limited to programs running in task mode. No support has been provided for non-task mode code. For example, if SRB routines were to issue vector instructions, they would have to manage their own vector environment entirely.

Since the Vector Facility can be online or offline separately from its associated CPU, some extensions have been made to the CONFIG and DISPLAY commands, as discussed below.

8.1 CONFIG COMMAND

CONFIG CPU command has been extended with the parameters VFON and VFOFF. These parameters are used with CONFIG CPU ONLINE. VFON is used to cause the VF to go online with the CPU, and VFOFF to cause it to be left offline.

The syntax is as follows:

```
                                  [ , ONLINE  [ , VFON  ] ]
CONFIG CPU(n) [                   [           [ , VFOFF ] ] ]
                                  [ , OFFLINE             ]
```

The CONFIG command also accepts a new operand, VF(n), to independently reconfigure the Vector Facility of the online CPU to which it is attached:

```
                              [ , ONLINE  ]
CONFIG VF(n) [                [           ]
                              [ , OFFLINE ]
```

Note that the CONFIGxx PARMLIB member allows the same syntax commands without 'CONFIG' or 'CF'.

The reconfiguration process due to these commands varies depending on the current status of the CPU and the VF. Figure 20 shows how it works in each case.

CONFIG CPU(n),	Status before CF CPU(n)							
	CP 0	VF 0	CP 0	VF X	CP X	VF X1	CP X	VF X2
ONLINE	0	0	0	X	0	0	0	X
ONLINE,VFOFF	0	X	0	X	0	X	0	X
ONLINE,VFON	0	0	0	0	0	0	0	0
OFFLINE	X	X	X	X	X	X	X	X

0: Online X: Offline

Note: 1 - VF went offline with CP
2 - VF was offline prior to CP

Figure 20. Reconfiguration by CF CPU,ONLINE Command

The CONFIG ONLINE/OFFLINE command has also been enhanced to display information about the Vector Facilities.

8.2 DISPLAY COMMAND

The following commands now include VF related information on the console display:

- D A,A

  '*VF*' is displayed in the AFF field on message IEE105I if VF affinity is required but no VF processor is available. This information can be used to check any users who require the VF but cannot be dispatched due to VF unavailability.

- D M=CPU

  Message IEE490I now includes the VF status as follows:

```
IEE490I hh.mm.ss MATRIX DISPLAY id
         ID        STATUS         SERIAL
        CPU 1   ONLINE   VFON    170999
        CPU 2   ONLINE   VFOFF   270999
        CPU 3   OFFLINE  VFOFF   370999
        CPU 4   ONLINE           470999   (No VF installed)
```

- D M=CONFIG(xx)

  The deviation information displayed by this command now includes VF status as follows:

```
IEE097I hh.mm.ss DEVIATION STATUS id
        CPU          DESIRED    ACTUAL
        1            ONLINE     OFFLINE
        VF           DESIRED    ACTUAL
        1            ONLINE     OFFLINE
        VF           DESIRED    ACTUAL
        2            OFFLINE    ONLINE
```

9.0 PHYSICAL AND LOGICAL RECONFIGURATION

As discussed earlier, the Vector Facility can be separately reconfigured from its related CPU. This reconfiguration may be either physical or logical.

When the physical reconfiguration of the Vector Facility is implied by the CONFIG command, the CONNECT VECTOR or DISCONNECT VECTOR command is issued to the service processor.

1. The VF is logically and physically reconfigured by one of the following commands:

   - CONFIG VF(n)
   - CONFIG CPU(n),ONLINE,VFON
   - CONFIG CPU(n),ONLINE,VFOFF

2. The VF is logically reconfigured by one of the following commands:

   - CONFIG CPU(n),ONLINE
   - CONFIG CPU(n),OFFLINE

   These commands do not change the physical status of the associated VF. However, the VF is made logically offline if the CPU is brought logically offline, since the VF cannot be used without its associated CPU.

   Therefore, CONFIG CPU(n),ONLINE results in the associated VF having the same online or offline status to MVS as it had prior to the preceding CONFIG CPU(n),OFFLINE processing.

   Figure 21 shows two examples of the command processing.

	Logical Status		Physical Status	
Operating steps	CPU	VF	CPU	VF
1. Initial	ONLINE	ONLINE	ONLINE	ONLINE
2. CF CPU,OFFLINE	OFFLINE	OFFLINE	OFFLINE	ONLINE
3. CF CPU,ONLINE	ONLINE	ONLINE	ONLINE	ONLINE
1. Initial	ONLINE	ONLINE	ONLINE	ONLINE
2. CF VF,OFFLINE	ONLINE	OFFLINE	ONLINE	OFFLINE
3. CF CPU,OFFLINE	OFFLINE	OFFLINE	OFFLINE	OFFLINE
4. CF CPU,ONLINE	ONLINE	OFFLINE	ONLINE	OFFLINE

Figure 21. Examples of CF CPU,ONLINE processing

10.0 PROGRAM CHECK INTERRUPT HANDLING

Two new program check interruptions have been introduced by the Vector Architecture:

- Unnormalized Operand Exception (UOE)
- Vector Operation Exception (VOP)

The UOE is treated as a regular program check, and no special processing has been added to MVS.

10.1 VECTOR OPERATION EXCEPTION

The VOP interrupt is used to recognize a vector task, in conjunction with the Vector Control Bit in one of the control registers. If a vector instruction is issued while the Vector Control Bit is off, a VOP program check is generated and the Program Check First Level Interrupt Handler (FLIH) gets control. It then passes control to the new Vector Second Level Interrupt Handler (SLIH), which does the necessary environment-setup for the interrupted vector task.

This setup consists of:

- Creating a save area for the vector registers
- Establishing vector affinity

10.2 VECTOR SLIH

The general functions of the vector SLIH are to:

- Validate the environment at the time of the VOP.
- Do a GETMAIN for the required save areas.
- Set indication that the task has vector affinity.
- Set the Vector Control Bit on.
- Return control to program FLIH, which passes control to the task.

Note that there is no difference between a vector task and non-vector tasks when a task is first created. It is when a task issues its first vector instruction that MVS recognizes the task as a vector task. MVS

then assigns the task a dynamic vector affinity, which directs the dispatcher to run the task only on a VF capable processor.

However, a vector task loses its vector affinity if it does not use vector instructions for a certain period of time. The task is then returned to its original CPU affinity (if specified), and is no longer restricted to running on a vector capable processor. This dynamic vector affinity allows a vector task to have a greater access to system resources, and also reduces the dispatching overhead associated with affinity dispatching. The detection of a vector task not using the VF for a certain time is performed by the Job Step Timing routine.

When the task that has lost its vector affinity due to 'non-use' issues a vector instruction again, the VOP is generated again. The task is again recognized as a vector task and is assigned a vector affinity. The previous vector status is restored from its save area, and is eventually dispatched on a vector capable processor.

The Dispatcher is responsible for saving and restoring VF status and also for affinity dispatching the VF task. The vector affinity is dynamically assigned by the Vector SLIH, and dynamically removed by the Job Step Timing routine.

11.1 VECTOR AFFINITY MANAGEMENT

The Vector affinity management has two primary functions:

1. Dynamically assign/unassign vector affinity to a task

2. Affinity dispatch a vector task on a VF capable processor

The first function is performed by the Vector SLIH and the Job Step Timing routine. When a task issues a vector instruction for the first time or after a period of vector non-use, the Vector SLIH is invoked, and it assigns a vector affinity to the task. When the vector task has not used the Vector Facility for a certain period of time, the Job Step Timing routine removes the vector affinity from the task, and turns off the Vector Control Bit. This allows the task to run on a non-VF capable processor. When this task issues a vector instruction again, the vector affinity is re-assigned by the Vector SLIH.

The second function is performed by the Dispatcher, in a manner similar to the existing processor affinity dispatch. At each dispatch, the task is eligible to run on any processor that meets its affinity requirements. The affinity of a VF task is computed at each dispatch to take advantage of a possible change in the status of available VF hardware features in the complex. That is, a VF capable processor may have come online after the last task preemption.

11.2 VECTOR STATUS SAVING

If a task is switched, the status of the current task has to be saved first. The vector status should be saved only when the last task was using the Vector Facility. The task's Vector status is saved as follows:

- Use SAVE CHANGED VECTOR REGISTER instruction to save status

- Record the Vector Activity Count.

  This value is later used by the Job Step Timing routine for utilization analysis.

The dispatcher then calls the Job Step Timing routine for utilization analysis, which removes the vector affinity if the task has not used the Vector Facility for more than a pre-set period of time.

11.3 VECTOR STATUS RESTORATION

When dispatching a vector task, the previous contents of the vector registers have to be restored.

- Non-VF task dispatching

  Since the task was not using the Vector Facility, the vector status is not restored. Additionally, the Vector Control Bit is unconditionally reset to zero when a non-VF task is dispatched.

- VF task dispatching

  The vector status is restored as follows:

  1. Restore Vector Status Register.
  2. Restore Vector Mask Register.
  3. Restore Vector Registers.
  4. Record the Vector Activity Count.

     This value is later used by the Job Step Timing routine for utilization analysis.

  The task is then ready to be dispatched.

11.4 PROGRAM LINKAGE

Program linkages are handled as follows:

- ATTACH/DETACH

  Since the vector environment is controlled a on task basis, the status saving/restoring of a vector environment is taken care of by MVS if the linkage utilizes an ATTACH/DETACH macro.

- Other Supervisor Assisted Linkages

  The vector environment is not saved and restored across supervisor assisted linkages. That is, if program A links to program B and program B wishes to modify vector registers, it must first save its

caller's registers and restore them on exit. The vector environment is managed in the same manner as floating point registers.

12.0 RESOURCE MANAGEMENT AND MEASUREMENT

This chapter describes the Vector Facility support provided by the System Resource Manager (SRM), System Management Facility (SMF), and Resource Measurement Facility (RMF).

12.1 SRM SUPPORT

The SRM control over a unit of work applies to vector tasks. However, the following modifications have been made to assist the basic characteristics of vector tasks:

1. I/O enablement order

2. Vector Wait

12.1.1 I/O Enablement Order

The vector tasks place a high demand on the processor cache when the vector elements
are placed close together in storage; that is, when the stride value is small.

In order to increase the cache hit ratio, a mechanism called 'Selective Enablement' has been introduced by MVS/XA. Since the I/O interruptions are the primary cause of the cache discard, this mechanism keeps the number of processors accepting I/O interrupts to a minimum, thus allowing the other processors to keep running without I/O interruptions. Refer to Figure 22.

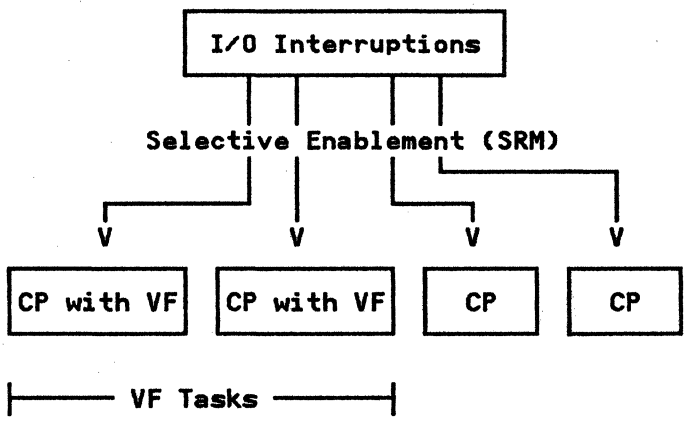


Figure 22. Selective Enablement by SRM

If the number of interruptions detected by TPI exceeds its high threshold value, SRM tries to increase the number of processors accepting I/O interruptions. When selecting the processor to be enabled for I/O interruptions, it should avoid the processor with VF. Therefore, in order to sustain the cache hit ratio of the VF capable processors, the selection order has been changed to:

1. CP without the VF

2. CP with the VF

Keeping the VF capable processors free from I/O interruptions has another advantage. When a vector task is interrupted and the dispatcher eventually gets control, instead of resuming the task, another task of higher priority (if any) could be dispatched. The dispatcher then has to save the status of the vector task. If the next task is a vector task, the status has to be restored. Because the vector registers have large amounts of data, this results in significant overhead in the dispatcher path. The selective enablement priority of non-VF processors reduces the chances that the dispatcher will get control of the VF capable processor, and reduces the status save/restore overhead of the dispatcher.

12.1.2 Vector Wait

In order to avoid abending a vector task that cannot keep running because no VFs are currently online, another swapout, called 'Vector Wait', has been introduced. Refer to Figure 23.

SRM has been changed to detect a vector wait situation in an address space and to place the address space in vector wait by:

- Initiating a swapout.
- Issuing a new message (IRA700I) informing the operator that the address space is in vector wait.

  The operator can then cancel the job if the VF will not be available within reasonable time.

When a VF is brought back online, SRM makes any address spaces that were in vector wait eligible to run again.

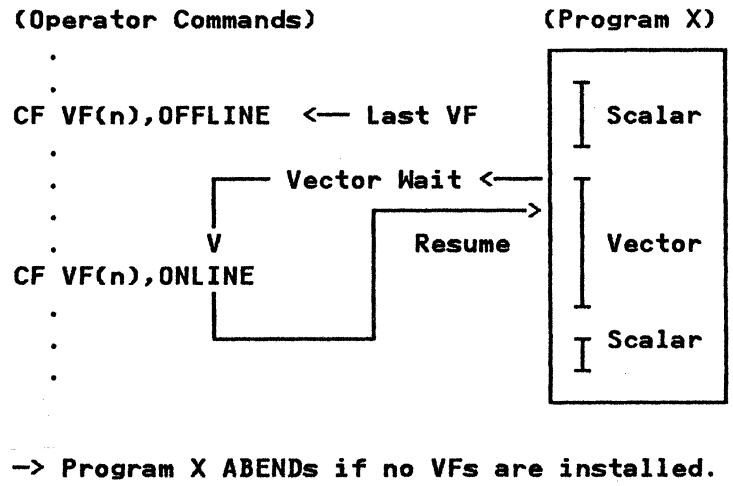


Figure 23. Vector Wait

12.2 CHANGES TO SMF RECORD TYPES

Some new fields have been defined in Type 30 and Type 22 SMF records to provide information on a vector job:

1. Type 30 SMF record

The SMF type 30 record contains six subtypes. All the subtypes except subtype 1 (job start) contain VF usage and VF affinity time.

The added fields are shown in Figure 24.

Name	Offset(Hex)	Type	Len	Description
SMF30JVU	20 (14)	Signed	4	Job/Step VF usage time
SMF30IVU	24 (18)	Signed	4	Initiator VF usage time
SMF30JVA	28 (1C)	Signed	4	Job/Step VF affinity time
SMF30IVA	32 (20)	Signed	4	Initiator VF affinity time

Figure 24. New Fields in SMF Record Type 30

VF Usage Time The TCB time spent for the VF instructions issued by the job.

VF Affinity Time The TCB time spent for the job while it has a vector affinity. Remember that a task loses its vector affinity because of 'non-use'.

Figure 25 illustrates the relationship among these values.

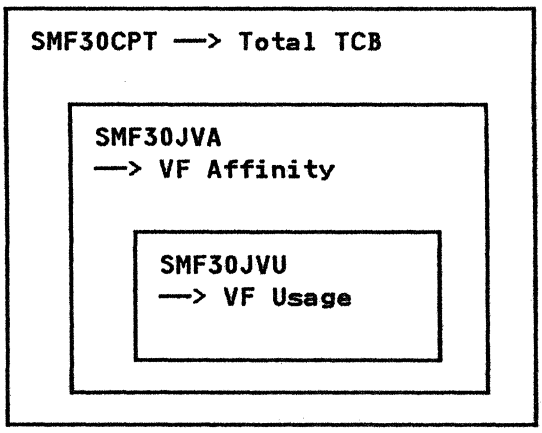


Figure 25. Relationship Among Vector Times

Although not directly related to the Vector Facility, two additional fields have been added to subtypes 2 and 3 of the Type 30 SMF record, as described in Figure 26.

Name	Offset(Hex)	Type	Len	Description
SMF30IST	36 (24)	Char.	4	Interval start time
SMF30IDT	40 (28)	Char.	4	Interval start date

Figure 26. Interval Start Fields in SMF Record Type 30

The post-processing program can now extract the interval start values from the subtype 2 and 3 records, and can produce statistical data with greater accuracy. Without this information in the records, post-processing would have to rely upon the specified amount on the SMF interval option. The actual SMF intervals can vary somewhat from this specified value; for example, records expected at 10-minute intervals are sometimes produced at 11- or 12-minute intervals.

2. Type 22 SMF record

   The online/offline status of the VF hardware is reported in the Type 22 SMF record. SMF22VFI bit (offset 0, bit 0) indicates that the VF was online if set to 1.

12.3 RMF SUPPORT

VF affinity time has been added to the RMF Monitor I CPU Activity Report. This data is obtained by RMF by sampling the Vector Control Bit on each processor. The general format of the CPU Activity Report is changed as shown in Figure 27.

CPU NUMBER	VFON	VF AFFINITY HH.MM.SS	WAIT TIME HH.MM.SS.TTT	WAIT TIME PERCENTAGE	CPU SERIAL NUMBER
1	VF	00.01.41	00.00.11.100	80.40	170115
2	--	---------	00.01.31.999	9.70	270115
TOTAL/AVERAGE				45.05	

Figure 27. VF Affinity Time Reported by RMF

No change has been made to the Monitor II or Monitor III reports. Standard work flow analysis of Monitor III also applies to vector jobs.

The most important extentions to the Reliablilty, Availability, and Servicability features introduced with the VFE are

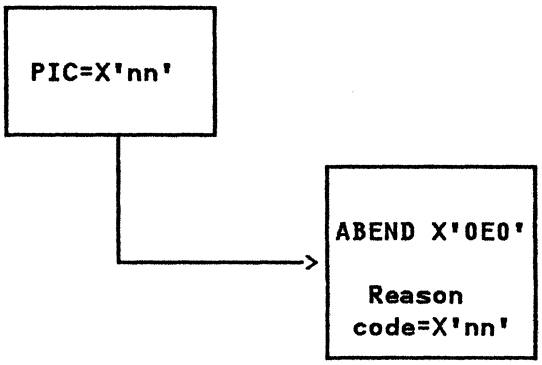
- A new abend code (0E0) for the new program interrupts.
- Presentation of vector related data in dumps and to IPCS/XA.
- Extension to Checkpoint/Restart in DFP/XA.

These items are discussed briefly in the following sections.

13.1 ABEND CODE 0E0

A new system abend code uniquely defines new program exceptions. The current program exceptions, code 0-F, are presented as they have been up to now; that is, as a 0Cx. With the 0E0 abend, the program exception code appears in the reason code (Figure 28).

The VF presents two new exceptions; The VOP, with code X'19' and the UOE, with code X'1D'. The VOP is a normal condition that is processed by the vector SLIH, whereas UOE results in an 0E0 abend with a reason code of X'1D'.



Example:

```
PIC=X'04' ===> ABEND X'0C4' RC=X'04'
PIC=X'1E' ===> ABEND X'0E0' RC=X'1E'
```

Figure 28. Presentation of New Program Exceptions

13.2 DUMPING SERVICES

Vector-related data is provided in dumps as follows:

1. SNAP/ABDUMP

   No new options have been added. The VF status is dumped as part of the current REG option for any task that uses the VF. Dumping of VF status involves 16 vector registers, the Vector Status Register, and the Vector Mask Register.

2. Stand-Alone Dump (SADMP)

   - Low-speed (formatted) SADMP

     The low-speed SADMP will not dump VF status. When the low-speed SADMP is used to dump virtual storage, the dump is of a single address space that is likely to be a system address space rather than a user address space. Thus, the VF status dumping capability has been omitted for low-speed SADMP.

   - High-speed (unformatted) SADMP

     The high-speed SADMP runs on each CPU in a multiprocessing configuration, and dumps the following VF status for each processor:

     - Vector Status Register
     - Vector Mask Register
     - Vector Activity Count
     - Vector Registers marked as in-use by the in-use bits in the vector status register.

     No console messages are provided to indicate the progress of VF status dumping.

     SADMP dumps VF status data, one set for each CPU, in a format similar to the one representing an MVS address space. In VF dump output records, SADMP sets the ASID and virtual address fields as though the record were mapping a page of virtual storage in an MVS address space.

     The fields are set as follows:

     ASID - X'FFF8'

     Virtual Address - CPU id followed by three bytes of zeros

     The dump can be formatted by one of the usual dump formatting utilities (IPCS or PRDMP). The dump access routines under PRDMP

and IPCS access the VF data as though it occupied a range of virtual addresses in an MVS address space.

3. IPCS/XA

   The VF data is included in Stand-Alone and SVC DUMP dumps in two new forms:

   - Vector Register Save Area
   - Actual VF content dump as a pseudo ASID produced by SADMP.

   IPCS/XA has been enhanced to provide:

   - Access to the VF data in these new formats.
   - Two-dimensional formatting of VF data.

   Additional output of VF data is now created whenever data for a TCB or CPU that is associated with a VF is requested. Also, user-written exit routines access VF data saved by SADMP by requesting storage from pseudo ASID X'FFF8'.

4. Print Dump (PRDMP)

   VF data is automatically printed in two circumstances:

   - When CPUDATA is requested. If there is VF data associated with that CPU, it will be printed.
   - When a TCB is requested. If there is VF data associated with that TCB, it will be printed.

   Note that printing the VF data is not optional, and no function such as 'VF-data-only printing' is provided.

   Also, user-written exit routines access the VF data saved by SADMP by requesting storage from pseudo ASID X'FFF8'.

5. SPZAP

   The DUMPT function has been extended to interpret the 171 new Vector Facility operation codes.

13.3 CHECKPOINT/RESTART

Checkpoint/Restart has been enhanced to handle vector tasks by including the vector status save areas with the checkpointed data. The necessary updates to DPF/XA are included in a PTF.

13.4 OPERATING ENVIRONMENT

The MVS/System Product 2.1.3 Vector Facility Enhancement requires that the processor be executing in IBM 370-XA mode.

The Vector Facility has to be installed in the complex in order to run the jobs that issue vector instructions. MVS/SP 2.1.3 Vector Facility Enhancement can operate without the Vector Facility hardware, but the vector jobs will be abended when they first issue a vector instruction.

IMPLEMENTATION AND MIGRATION

Implementation of the VF in a Large Systems Computing environment is a complex process that requires planning and strategy decisions in addition to the physical installation steps. This section concentrates mainly on the installation tasks and highlights some recommendations based on early experience.

14.0 IMPLEMENTATION

The introduction of the VF computing facilities into a system requires a series of installation steps, which should be taken in a certain sequence to ensure that hardware and software installation do not overlap.

A recommended sequence is:

1. Install the necessary software packages

2. Install the MVS/XA support

3. Install the VF

Software such as FORTRAN V2 and LCP runs on the current level of MVS, and can therefore be tested and put into production before the MVS/XA VFE is installed.

MVS/XA VFE can be installed, tested, and put into production prior to installation of the hardware. This is recommended in order to identify any problems caused by changes in software and not hardware.

By installing the VF last, the support is immediately in place for the verification tests.

In the following sections we will discuss the installation of MVS/SP 2.1.3 VFE and its related software upgrades.

MVS/System Product 2.1.3 Vector Facility Enhancement requires:

- A processor executing in 370-XA mode.

- The Vector Facility installed in the complex (required only to run the jobs that issue vector instructions).

MVS/SP 2.1.3 is a prerequisite for MVS/SP 2.1.3 Vector Facility Enhancement, and MVS/SP 2.1.3 VFE operates with the same level of the JES2 or JES3 components as the MVS/SP 2.1.3 Base Control Program.

MVS/SP 2.1.3 VFE is mutually exclusive with MVS/SP 2.1.3 Availability Enhancement (AE), and therefore cannot be installed with AE as a base.

The installation process use either:

- System Modification Program Extended (SMP/E) - 5668-949

- System Modification Program Release 4 with PTF UR03129

For the MVS/SP 2.1.3 base, the ACCEPT processing must be done before VFE RECEIVE processing.

An installation scenario based on PUT 8505 as a starting point is shown below. The steps of the SMP processing are:

- ACCEPT
- STAGE 1 Sysgen
- JCLIN
- APPLY
- ACCEPT REDO

Required PTFs for the 2.1.3 base were UZ80022, UZ80023, and UZ80024.

Additional installation steps upgrade related software products and consist of:

- Generate new Stand-Alone Dump program.

  This is necessary in order to have the SA Dump format vector registers and other VF related data in a dump.

- Install PTFs on:
  - DFP
  - EREP
  - RMF
- There is no need to re-generate IPL records and no clean-up jobs are required.

15.0 MIGRATION CONSIDERATIONS

When migrating a Large Systems Computing environment to utilize the vector facility for E/S applications, various rework is necessary in addition to the actual installation of the hardware and software. This includes:

- Changes to application programming practices
- Rework of applications in order to exploit the VF
- Retuning of the operating system
- Changes in backup strategies
- Changes in billing routines

We will restrict ourselves here to consider the system oriented changes, since application migration is beyond the scope of this document.

15.1 CHANGES TO THE OPERATING SYSTEM

The workload using the VF is similar to a CPU-bound floating point workload. The most important differences are:

- The VF enhances the performance of E/S programs so you can run applications that make greater demands on storage, and that will have more frequent storage references, such as larger matrix computations. Consequently, there is an increased demand on real storage.
- Task switches between VF tasks require saving and restoring vector registers, which means more work for the dispatcher.
- The MVS Vector Affinity function ensures that, in systems with only one VF, the vector workload is dispatched on that CP and therefore introduces a workload imbalance.

When migrating to VF, one should plan additional central storage or expanded storage. In a normal workload environment, the effect of VR save and restore appears to be minimal. First of all, MVS tries to minimize the rate of task switches by letting only one of the CPs handle I/O interrupts. With only one VF, I/O interrupts are taken by the non-VF CP. Secondly, the VR save/restore mechanism in the VF helps to minimize the data transfer to and from storage.

It does not seem to be necessary to re-evaluate performance or tuning parameters.

15.2 BILLING ROUTINES

Along with other resource usage data, SMF records now contain:

- Vector usage time
- Vector affinity time

This data is valuable for capacity planning purposes, and might also be used for charging for the use of the VF.

The current billing algorithms might have to be reviewed for the impact of VF on the current method, or to determine whether charges for VF usage should be distributed to the users.

It might also be appropriate to modify the current IEFACTRT to present this data to the users on output listings. This information might help in application migration projects.

15.3 JES3 INCOMPATIBILITY

Along with the changes made to MVS/SP 2.1.3 to support the VF, changes were made to the Type 6 SMF record layout. The Type 6 record is used by JES3 to report data on printer usage. (JES2 uses its own layout and is therefore not impacted by this change.)

JES3 installations that are using accounting routines that depend on the Type 6 record must plan to incorporate changes in these routines when migrating to a VF system.

15.4 BACKUP STRATEGY

The VF workload is incompatible across processors in the sense that it is locked to a system with VF installed. It can run only on a 3090 with VF. This may introduce a change to the current backup strategy.

If the current backup arrangement implies use of a non-VF backup processor, the following two alternatives exist for backing up the vector workload:

1. Arrange for a 3090 with VF as a backup processor.
2. Amend development and operation procedures so as to be able to use a non-VF processor.

The first alternative needs no further comment, whereas the second alternative needs parallel maintenance of vector and scalar versions of the workload. Whenever applications are changed, the applications would also have to be compiled with the non-vector option, and with possible linkage to non-vector versions of ESSL routines or other subroutines.

The additional maintenance introduced by this method is in addition to the performance impact that comes when applications have to be run in scalar mode.

READER'S COMMENTS

Title: IBM 3090 Vector Facility
Technical Bulletin GG24-3058

You may use this form to communicate your comments about this publication, its organization or subject matter with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Comments:

Reply requested

Yes / No

Name : _____

Job Title : _____

Address :

_____

Reader's Comment Form

Cut or Fold Along Line

Fold and tape | **Please Do Not Staple** | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

IBM International Technical Support Center
Department H52, Building 930
P.O. Box 390
Poughkeepsie, New York 12602
U.S.A.

Fold and tape | **Please Do Not Staple** | Fold and tape

IBM®

GG24-3058-0

IBM 3090 Vector Facility
Technical Reference

GG24-3058-0

PRINTED IN THE U.S.A.