IPL-DDB - A.α.1

MEMO

ADVANCED SYSTEMS LABORATORY

DATE:	14 January 1974		
TO:	Distribution	LOCATION:	
FROM:	J. A. Wilson	LOCATION: PGAANX	EXT: 6428
SUBJECT:	IPL Architectural Definition		

1. Attached is the IPL Architectural Definition which is submitted for your approval. A meeting will be scheduled on Tuesday, January 22, following PRAM in Escondido, to solicit your comments.

[illegible]

J. A. Wilson

/mlt

Distribution:	T. H. Elrod	ASL/W
	R. O. Gunderson	
	M. F. Harris	ASL/W
	G. M. Schumacher	
	D. L. Slais	

IPL ARCHITECTURAL DEFINITION

1/14/74

TABLE OF CONTENTS

1. INTRODUCTION

1.1 The Architectural Definition is the first of three levels of definition of IPL as described in the IPL System Design Plan document. Each level will define IPL in more detail. The Architectural Definition defines:

- Inter-processor connection
- Virtual memory mechanism
- Data formats
- Direction to be followed in defining the instruction set.

1.2 The objective of the Advanced Systems Laboratory is to define a computer systems product line which spans the lease price range of $5K-300K/month and performance range of from one tenth of a 6600 to eleven times a 6600.

1.3 The definition of compatibility to be achieved in IPL was defined by the joint NCR/CDC task force in May, 1973. The required level of compatibility is level IV with level V desired. Level V is to be pursued only if the result does not compromise too severely the cost/performance of the line.

Level IV - This level established the minimum requirements which must be satisfied to have an integrated product line. These requirements are stated as:

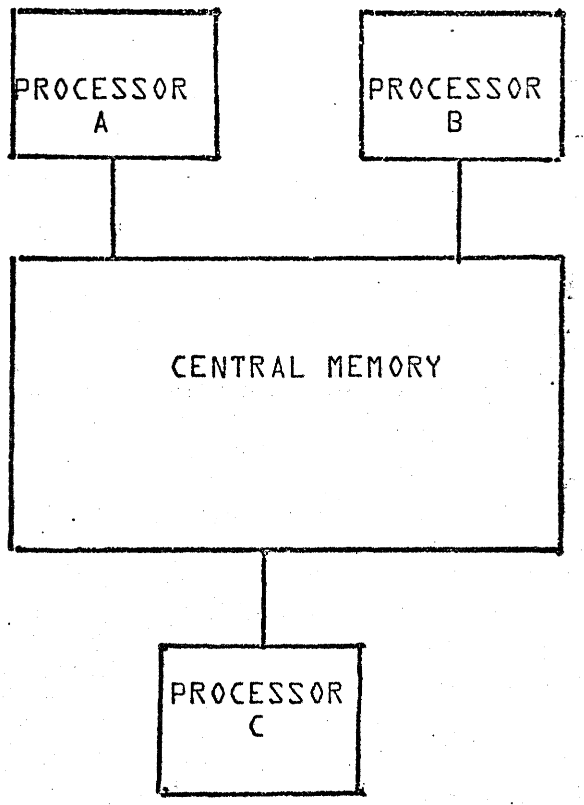
- Industry standard data representation on cards and tape.
- Complete portability of higher level language source code, including software writer's language. {Note, that users who write model dependent code cannot be fully masked.}
- Common data formats {internal and external}.
- Common disk recording formats.
- Common data communications protocol.
- Common operating system at the source level, providing JCL, file organizations, access methods, labeling conventions, etc.
- Common I/O channel.
- Common system console design.
- Consistent virtual storage definition.

Level V - This level includes Level IV plus a bit-compatible instruction set.

IPL CONFIGURATION

2.1 The primary aspect of configuration that will be addressed here is the relationship between the multiple processors typical of the IPL, and central memory. Further details relating to channel configuration and peripheral equipment support will be supplied at a later stage.

2.2 The basic configuration of the IPL is one where a central memory is accessed via a common addressing scheme by multiple processors:

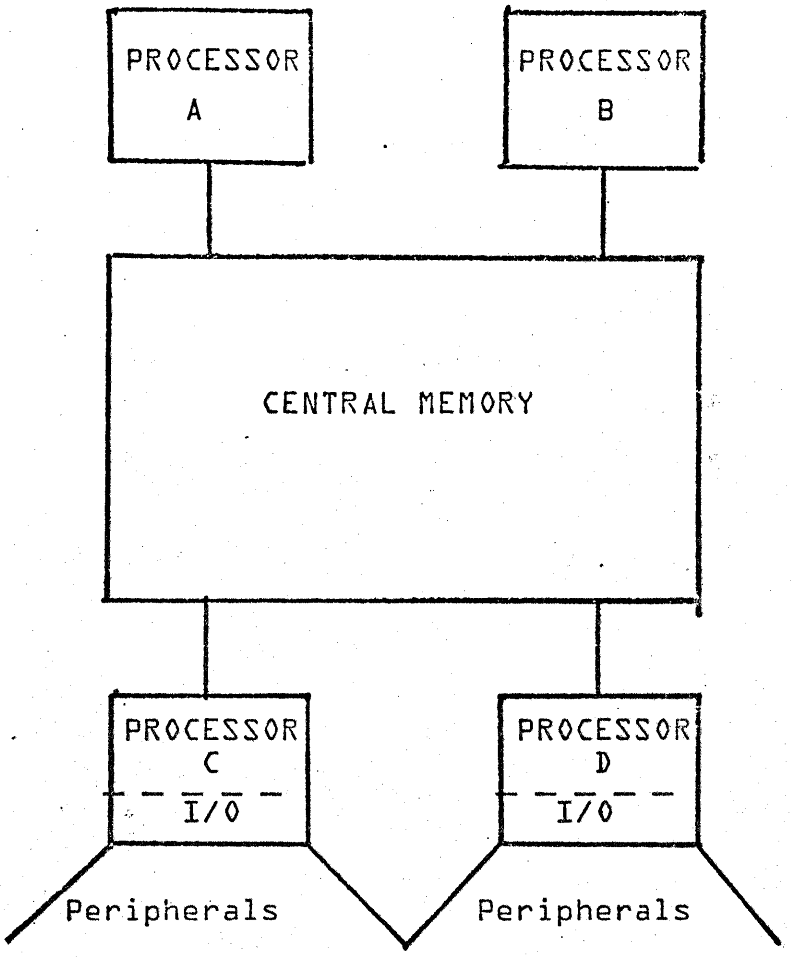


Central memory will not be the only communication path for the individual processors. However, the precise connections are still being developed and will be included in this specification at a later date.

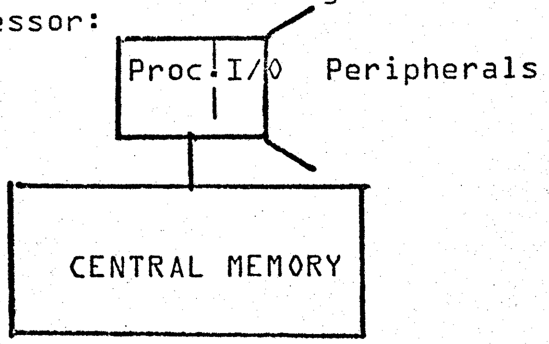
The addressing scheme employed is based on a virtual memory mechanism which is described fully in the next section. That mechanism forms the basis for the protection and security schemes devised for the IPL, and by ensuring that all memory references are via this mechanism, system- wide protection and security are guaranteed.

2.3 Implied in the configuration is a single operating system and a single instruction set. The individual processors will not necessarily be identical, but with the exception of some special I/O commands they will all be capable of executing the same processes.

Some processors will have the ability to communicate with peripheral devices. The operating system will recognize this unique feature and will assign work accordingly. The general configuration is shown below:



2.4 The IPL will embrace a range of processors of differing power. Total system power may be increased by utilizing a more powerful processor, or by adding processors of like power. Since all IPL models require an I/O capability it is the less powerful processors representing the low-end of the line that will satisfy this need. A minimum low-end configuration will typically contain a single processor:



More powerful systems will typically have two processors, one of which will have the I/O capability.

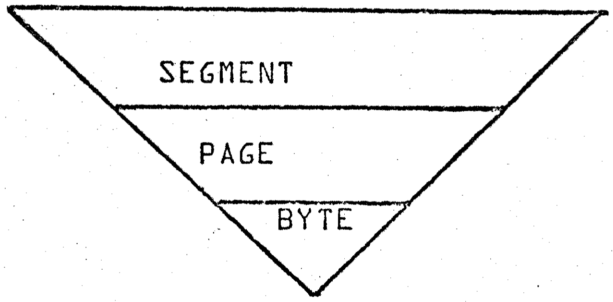
2.5 Depending on the power of a particular processor, certain operations may be optimized. For example, at the low-end BDP instructions may be executed more efficiently than their counterparts at the high-end. Nevertheless, basic I/O commands excepted, all processors can execute all code. This fact, which enables the configurations outlined here, also provides for parallel redundancy and all the benefits which are derived from it.

2.6 The instruction set to be used for the IPL is discussed in Section 5, and reliability issues will be the subject of future sections or appendices.

3. VIRTUAL MEMORY

3.1 Overview

3.1.1 In order to simplify programs all awareness of the actual size of physical memory has been removed from the user. Instead, the user works in a virtual memory space which has a finite size of 2^{43} bytes for any given user. To permit users to share data and code in a controlled manner, the entire information store is divided up into segments. Each segment has associated with it a set of attributes which control the access to that segment. A given user may address up to 2^{12} {4096} segments in a single process. Each segment has a maximum permissible length of 2^{31} bytes. To facilitate mapping segments into real memory, and to enable management of the very large memories envisaged for the IPL, segments are subdivided into pages. Page sizes may vary between a minimum of 256 bytes and a maximum of 64K bytes. In any given machine the page size will be fixed. The minimum page size permissible is termed a paragraph. Within this memory space addressing will be to the byte. The total hierarchy then is:



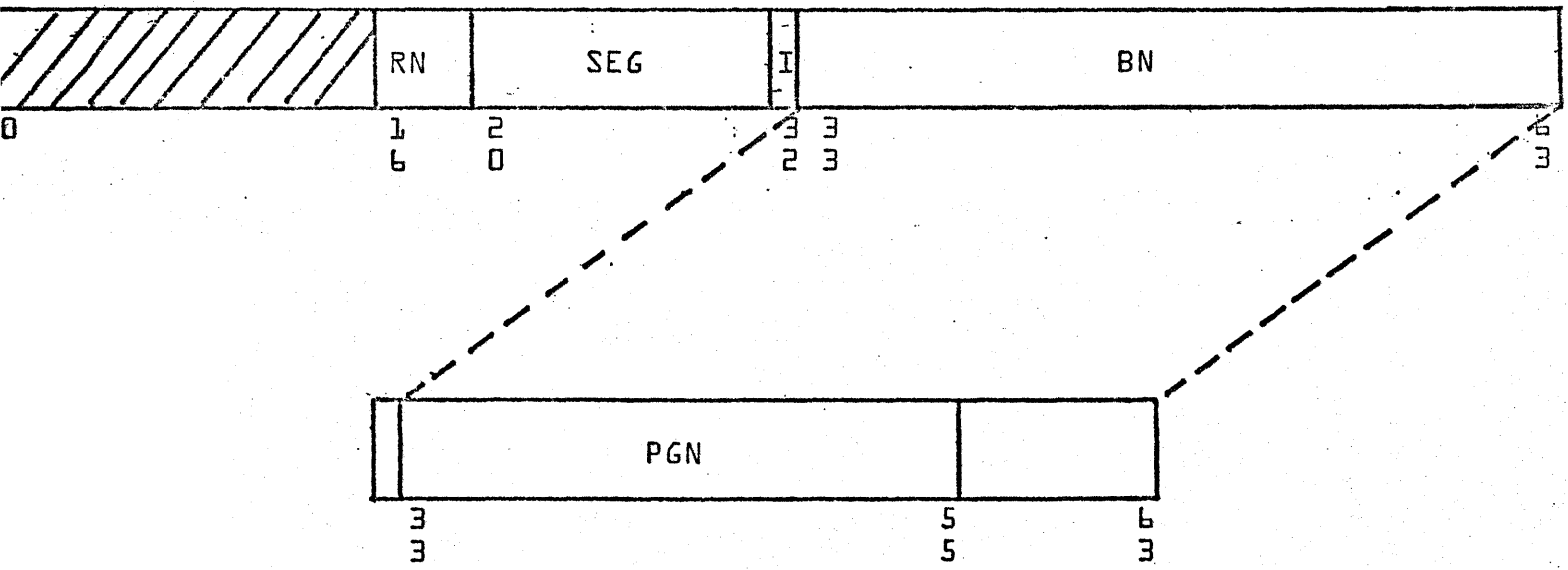
In general, users refer to a segment and a byte offset within a segment. Pages are transparent to the user in much the same way that banks are transparent to users in real memory.

3.1.2 Having established an environment in which many users may share code and data it is necessary to provide suitable protection mechanisms to insulate the individual users from each other. Two techniques are used to guarantee interprocess and intraprocess protection. The first is achieved via the segment attributes which have already been mentioned, the second is achieved by logically organizing the entire information store within a series of concentric rings corresponding to different states the machine operates in. Ring zero is the most priveleged ring. In general, a procedure executing in a particular ring has access to code and data in that ring and in any ring outside {greater ring number} its own. Access to inner rings can only be made through carefully controlled entry points or gates.

3.2 Memory Address Formation

3.2.1 This section specifies the logical algorithms used for translating the IPL process virtual address {PVA} into a real address. The formation of the PVA is a function of the instruction repertoire and how the various fields of the instruction are used to form an effective address.

3.2.2 The PVA is mapped into a 64-bit container. Three fields are used during address translation. These are the ring number {RN}, the segment number {SEG} and the byte number {BN}. The format of the PVA is shown below:



I is the invalid flag and, when set {I=1} denotes an invalid pointer.

3.2.2.1 The ring number is a four bit field used in access validation and is discussed in the next section.

3.2.2.2 The segment number is a 12-bit field that is used to access the segment descriptor. In effect this field is an index into the segment descriptor table. Segment numbers are assigned as needed by the operating system. Each process in the system has its own virtual address space and can have up to 4096 segments described in that address space. Some of the segments will be pre-assigned to system code and tables that are in a privileged machine state {typically ring zero}. Other segments will contain the code and data of the users application.

3.2.2.3 The byte number specifies the location to be accessed within a segment and is made up of three parts; the page number {PN}, the page offset {PO}, and the paragraph number.

3.2.2.3.1 The page number field is variable in size and ranges from 15 to 23 bits. The size is fixed on a per installation basis and will not vary while the system is running. The actual size of the page number field is contained as a mask in the page size mask register.

3.2.2.3.1.1 The page size mask register is set so that it can be used against bits 48 through 55 of the PVA to separate out the page number and the page offset. Bit positions 33 through 47 of the PVA are automatically included in the page number, and bits 56 through 63 are automatically included in the page offset.

3.2.2.3.1.2 The page size mask is 8-bits long and is always a logical prefix vector with {8-U} ones followed by U zeros where the page size is 2^U x paragraph size

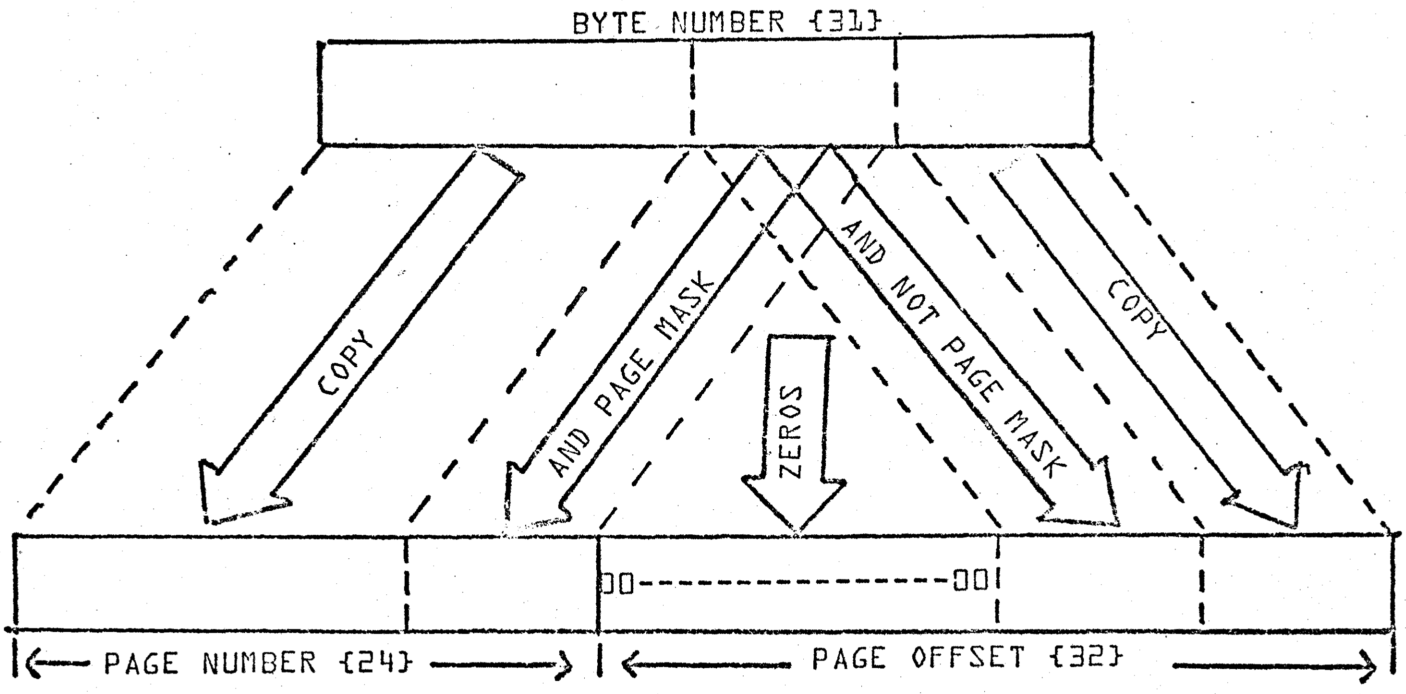
{$=2^U$ x 256 or $2^{(8+U)}$}. For example, U=2 yields a page size of $2^{(J+1)}$ = 1024 bytes. The corresponding page size mask would be set to:

"11111100".

3.2.2.3.2 The page offset is the displacement of the location to be accessed relative to the page boundary. This field varies with the page size and ranges from 8 to 16 bits.

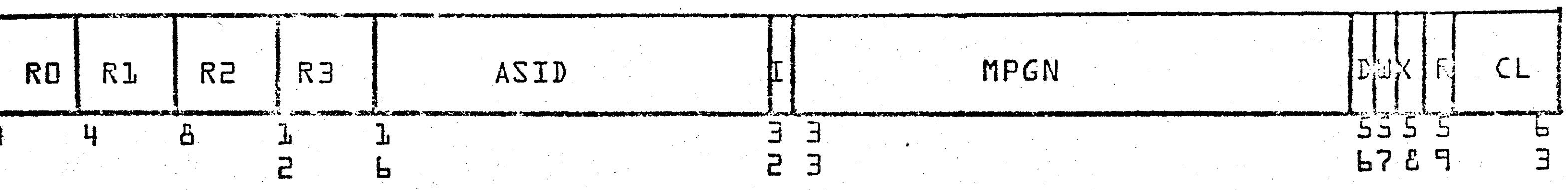
3.2.2.3.3 The paragraph number is specified as the 23-bit value contained in bits 33 through 55 of the PVA. It is used both to validate against trying to access beyond the defined length of a segment, and to allow a segment allocation unit that is smaller than a page.

3.2.2.4 The formation of the page number and the page offset from the byte number and the page size mask is illustrated below:

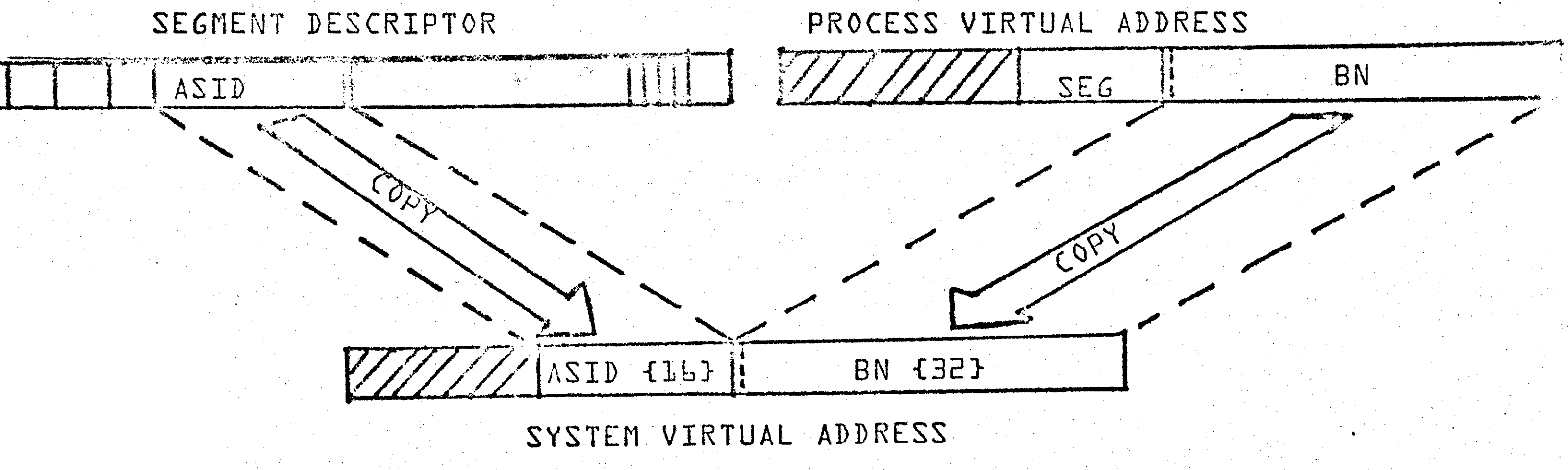


3.2.3 Memory Tables - Two memory contained tables are used to translate the PVA into a real address. These are the process segment table and the system page table. They are specified with real addresses in special programmable registers. The registers can only be manipulated by privileged routines of the operating system.

3.2.3.1 The process segment table is specified by two values: the segment table address {STA} and the segment table length {STL}. The STA is the first real address of the first entry of the process segment table. Each entry is 64-bits long and is accessed by indexing the STA with the appropriate segment number. The segment table length indicates the number of usable entries in the segment table. The segment number to be used as an index must be less than or equal to the value of the STL. The format of the segment table entries {segment descriptors} is shown below:



The process segment table entries are used primarily to validate access. They are also used to convert the PVA to a system virtual address {SVA}, by substituting a 16-bit active segment identifier for the 12-bit process segment number. The segment table entry is known as a segment descriptor. The formation of the SVA is illustrated:



3.2.3.1.1 The active segment identifier {ASID} is a software supplied value that relates the process' segment number to one of a global set of segments active in the system. Two processes which are sharing a segment may use different segment numbers to address the segment, but will have the same ASID. The ASID is substituted for the segment number in the PVA before the system page table is accessed.

3.2.3.1.2 The W, R, and X flags indicate the type of access that is permitted to the segment. These quantities, the ring numbers R0-R3 and the call limit {CL} are discussed more fully in the next section.

3.2.3.1.3 The maximum paragraph number {MPGN} is used to ensure that the byte number from the PVA does not reference beyond the end of the segment. The PGN must not be greater than the maximum paragraph number of the segment as specified in the descriptor.

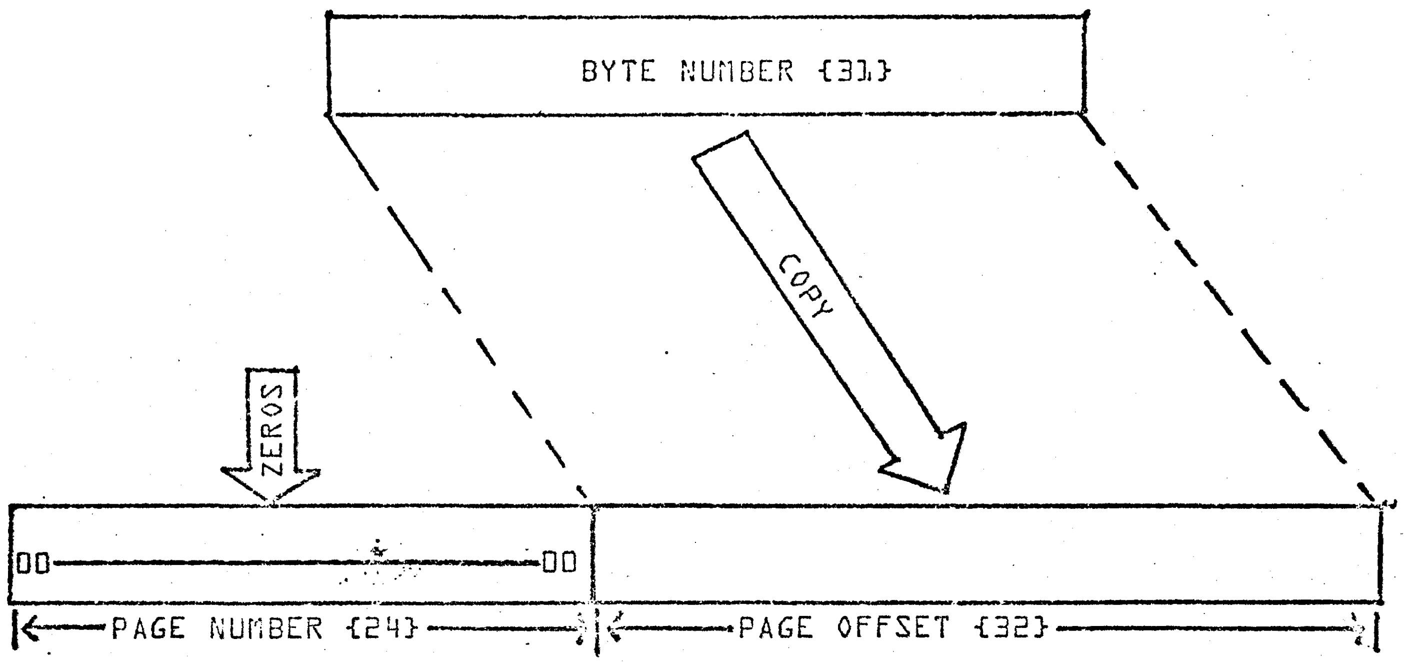
3.2.3.1.4 The invalid flag {I} indicates whether the segment descriptor contains valid information. If a process is removed from memory and placed on secondary storage, its segments are considered to be no longer active and the ASID is released. Hence, when the process returns to memory the entries in the segment table are no longer correct and are marked invalid. As each segment is used a new value for the ASID is supplied. Attempting to use a segment descriptor with an invalid bit set causes a trap so that the operating system can make the segment descriptor valid.

3.2.3.1.5 The direct flag {D} is used to indicate direct addressing of the segment. This is a special mode of operation that reduces fragmentation of real memory when several segments of less than one page in length can be grouped together. The address translation mechanism for the direct address mode proceeds as follows:

{i} Zeros are placed in the paragraph portion of the segment/page identifier. That is, the page number is forced to zero.

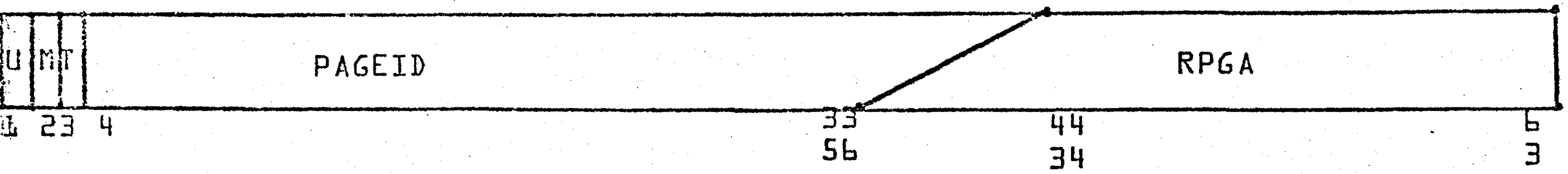
{ii} The"physical page address" is recognized as the segment relocation address and is added to the 31-bit physical memory address.

This process is illustrated as follows:

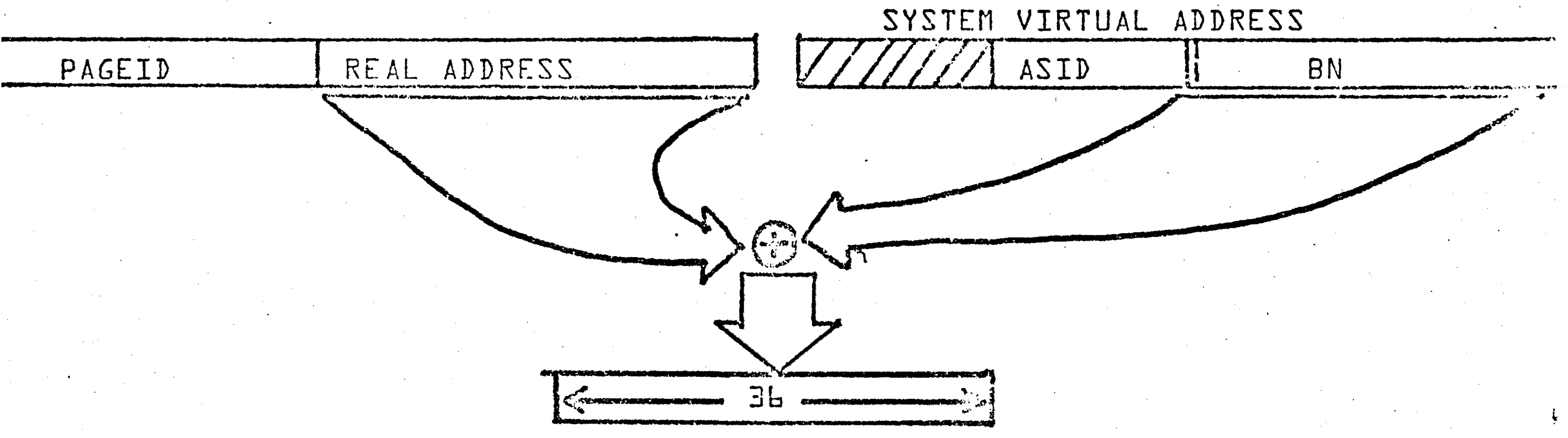


3.2.3.2 The <u>system page table</u> is specified by two values: the page table address {PTA} and the page table length {PTL}. The page table address is the real address of the first entry of the system page table. Each entry is 64-bits long. The desired entry in the table is located with a combination of indexing and linear searching. The page table length is a mask that is used to force the index used to access the page table to be modulo the size of the table. The table size is a function of real memory size and the page size, and is a multiple of the number of page frames in real memory - usually 2-4 times the number of available page frames.

3.2.3.2.1 The system page table entries are used to locate the proper page frame to be accessed and record usage of the page frame. Their format is illustrated below:

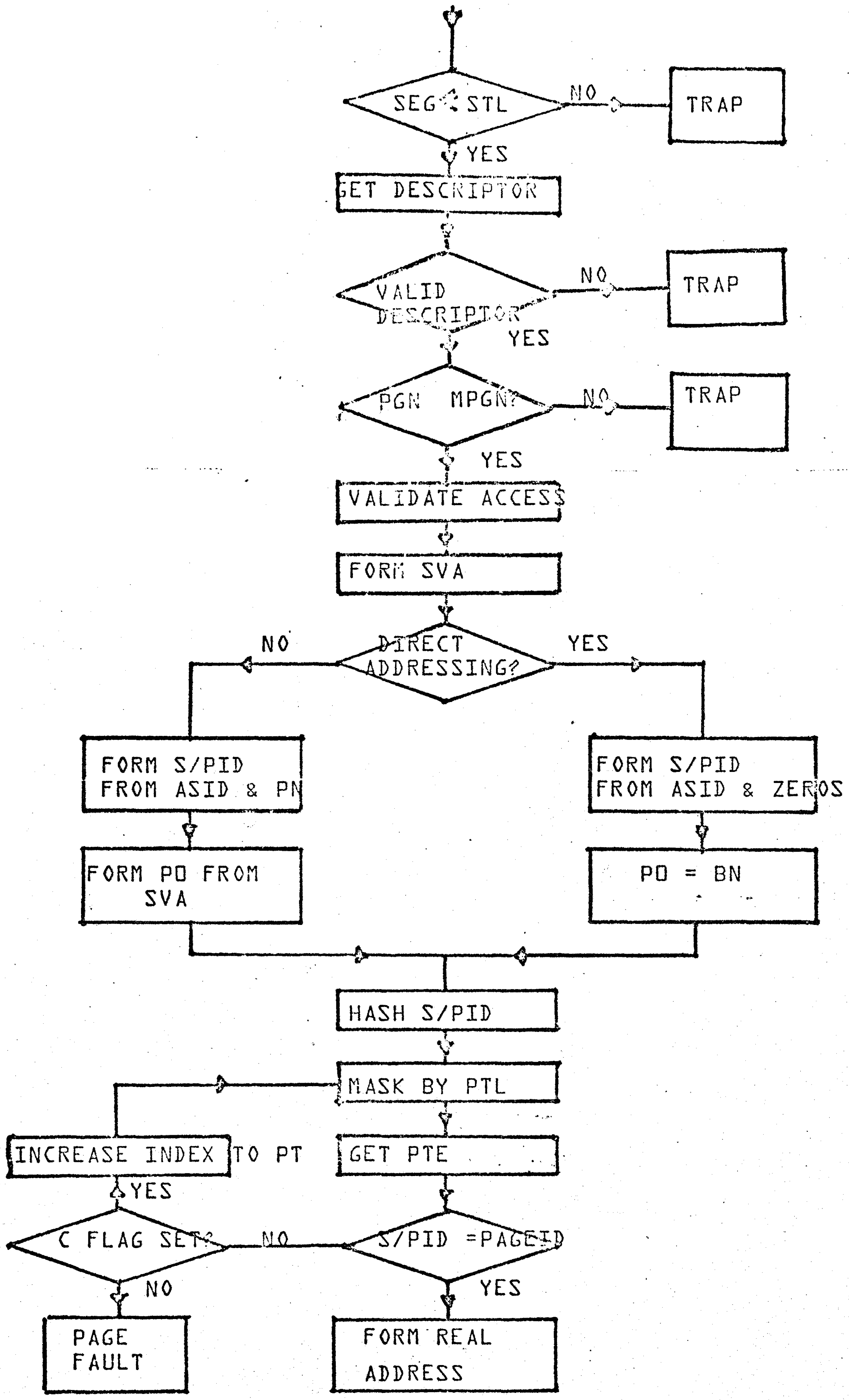


3.2.3.3 The <u>real address</u> is formed by adding the real paragraph address and the SVA byte offset.



TARGET REAL ADDRESS

3.2.3.4 The entire address formation {excluding access validation} is described by the following flow chart:



NCR/CDC PRIVATE

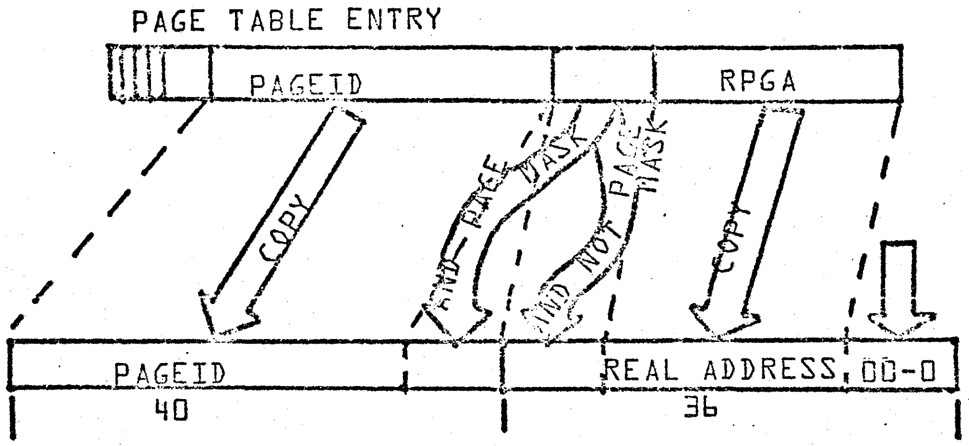
ADDRESS FORMATION

DRAFT

The page table contains one entry for each frame of real memory. The entries are placed in the table according to a hash index that is generated from the SVA. Since many SVA's will hash to the same index it is necessary to specify the algorithm to be used to continue searching the table. This is a straight linear search.

3.2.3.2.2 The <u>page identification {PAGEID}</u> consists of the ASID and the page number derived from the PVA. It is used to identify the SVA to be translated by the particular entry.

3.2.3.2.3 The <u>real paragraph address {RPGA}</u> specifies the 256 byte boundary in real memory at which this section of the SVA is mapped. Because of the paragraph size allocation unit, the final real address should be formed by addition of the real paragraph address and the SVA byte offset. The formation of these quantities is diagrammed.



3.2.3.2.4 The <u>used{U}</u> and <u>modified {M}</u> flags indicate whether the page table entry has been used for address translation, and when used, if the real memory location was modified.

3.2.3.2.5 The <u>T-flag</u> is used as a lock-out. When set this flag indicates that the page table entry cannot be used by the CPU for address translation because the block is being modified by I/O.

3.2.3.2.6 The <u>control {C} flag</u> controls the search of the page table for the proper SVA. If C is not set, then the block of SVA space is not in real memory and a page fault is generated.

3.3 PROTECTION MECHANISM

3.3.1 Two mechanisms are used in the IPL for controlling access to a segment. First, when a user creates a segment he indicates the type of access other users may have to that segment. The options of read, write and execute are denoted by individual flags in the segment descriptor. The W-flag must be set if the segment is to be modified. The R-flag must be set if data is to be fetched from the segment. The X-flag must be set if the segment contains executable code and constants.

3.3.2 The ability to grant access rights to a particular segment is not sufficient control, and that mechanism is augmented by a technique governing intra-process control. This technique is an extension of the common two state {system state and user state} machines. The IPL may operate in any of sixteen states. These states are rings of protection. In general, segments in the same ring have access to each other limited only by their prescribed access modes. In addition, segments in lower-numbered rings have unlimited access to segments in high-numbered rings, subject to the access modes of those segments.

3.3.3 By definition, passing control outwards {to a greater ring number} from a segment is legal. However, passing control inwards {to a smaller ring number} is carefully controlled, and is achieved by providing the callee with a gate through which the caller must pass. The most common example of this process occurs when a user calls on the operating system to perform a task.

3.3.4 It is frequently convenient to allow a segment to execute in several rings. This is accomplished by giving the segment an execute bracket. This bracket delimits the rings in which the segment may be executed - always provided that the segment has execute access granted via the X-flag. The R0-R3 fields in the segment descriptor are used to denote the rings of which a segment may be a member. If a process is executing in a ring contained in the execute bracket of a segment, and control is transferred to that segment, then the ring of execution is unchanged. If the current ring of execution is less than the ring bracket, then when control is transferred to that segment the ring of execution is set equal to the smallest ring number in the bracket. In a similar way, if the current ring of execution was greater than the ring bracket it would be set equal to the greater ring number in the bracket, assuming the segment had a gate. In this context it is also useful to specify a gate bracket. An attempt to execute a segment from a ring greater than the gate bracket is prohibited. The fields R1, R2 and R3 are used to denote the execute bracket {R1, R2} and gate bracket.

3.3.5 The concept of ring brackets is extended to read, and write protection. A process must be executing within the read or write bracket of a segment, and appropriate access must have been granted for their operations to be executed. The complete set of conditions for reading, writing and executing a segment are given below.

3.3.5.1 <u>Write Access</u>

$W = 1$ P.RN is the current ring of execution.

$R0 \le P.RN$

$PVA.RN \le R1$

3.3.5.2 <u>Read Access</u>

$R = 1$

$R0 \le P.RN$

$PVA.RN \le R2$

3.3.5.3 <u>Execute Access</u>

$X = 1$

$R1 \le PVA.RN \le R2$

3.3.6 When a procedure makes a call on another procedure executing in an inner ring, the right to make the call must first be validated, and the proper use of the gate must be checked. The authority to make the call has been given to the caller if:

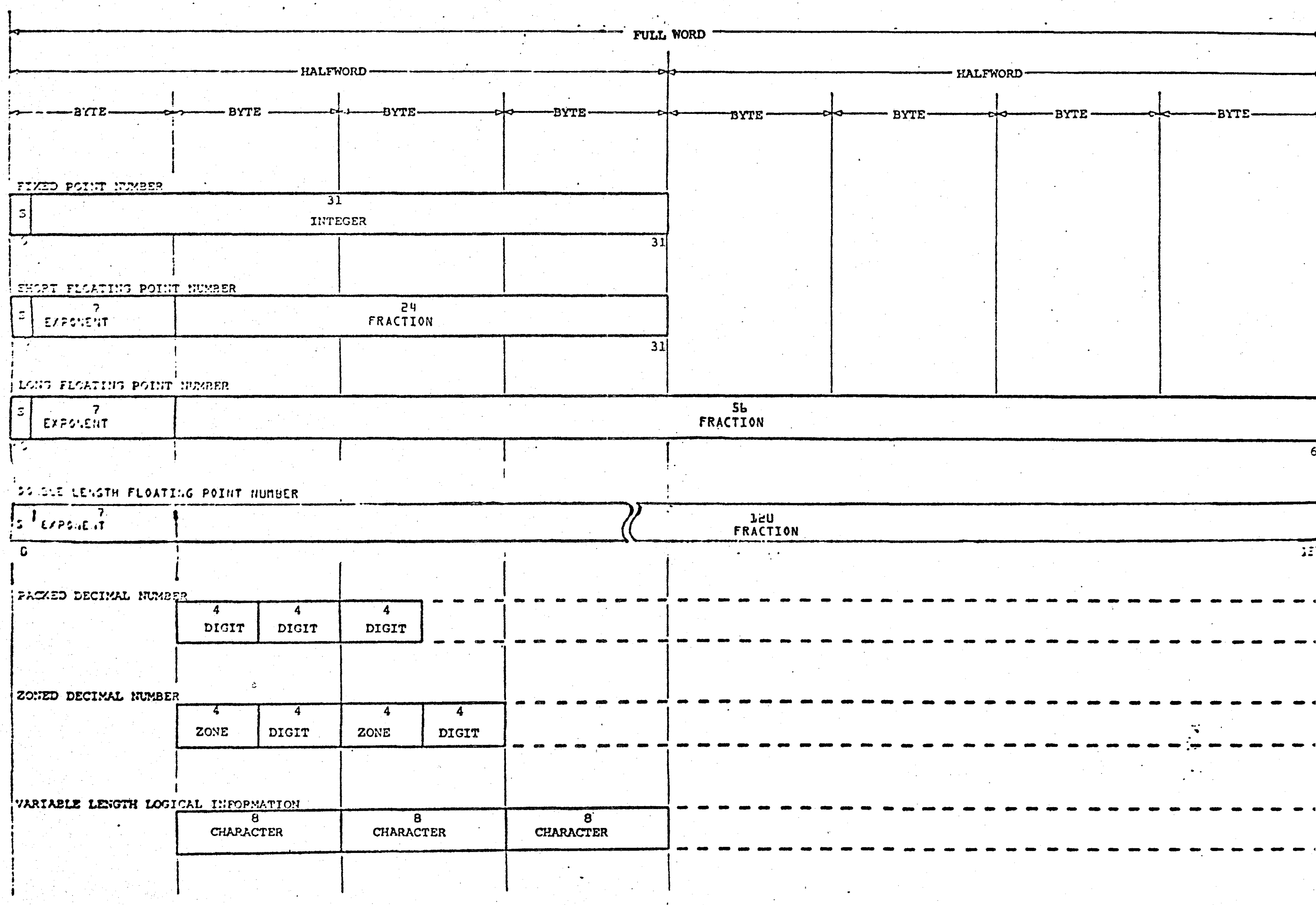
$PVA.RN \le R3$

Having validated the right of the caller to make the call, the entry address must be verified. This is done by comparing the CL field of the descriptor with the PVA.BN, to ensure that the entry is via the appropriate transfer vector. If the address is within range of the transfer vector, the gate is allowed. In this case the current ring of execution is set to R2. Execution now proceeds as normal.

3.3.7 To ensure protection when returning from an outward call, outward calls are trapped by the operating system which then simulates the CALL operation. In this case the current ring of execution is set to R1.

4. D A FORMATS

4.1 The data formats supported by the IPL are diagrammed below:



DATA FORMATS

4.2 An 8-bit unit of information is fundamental to most of these data formats. The location of a stored field is specified by the address of the leftmost byte of the field. Variable-length fields may start on any byte location, but a fixed-length field of 4- or 8-bytes must have an address that is a multiple of 4 or 8, respectively.

4.3 Alpha-numeric data is carried either in ASCII or in EBCDIC, which codes are shown on the following page:

BI[illegible] POSITIONS → 01
→ 23
→ 4567

	00				01			
4567	00	01	10	11	00	01	10	11
0000	NUL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

NUL	Null/Idle	DC1	Device control
SOH	Start of heading	DC2	Device control
STX	Start of text	DC3	Device control
ETX	End of text	DC4	Device control (stop)
EDT	End of transmission	NAK	Negative acknowledge
ENQ	Enquiry	SYN	Synchronous idle
ACK	Acknowledge	ETB	End of transmission block
BEL	Audible or attention signal	CAN	Cancel
BS	Backspace	EM	End of medium
HT	Horizontal tab	SUB	Start of special sequence
LF	Line feed	ESC	Escape
VT	Vertical tab	FS	File separator
FF	Form feed	GS	Group separator
CR	Carriage return	RS	Record separator
SO	Shift out	US	Unit separator
SI	Shift in	SP	Space
DLE	Data link escape	DEL	Delete

EIGHT-BIT REPRESENTATION FOR CODED INFORMATION (ASCII)

BIT POSITIONS - 01 / 23

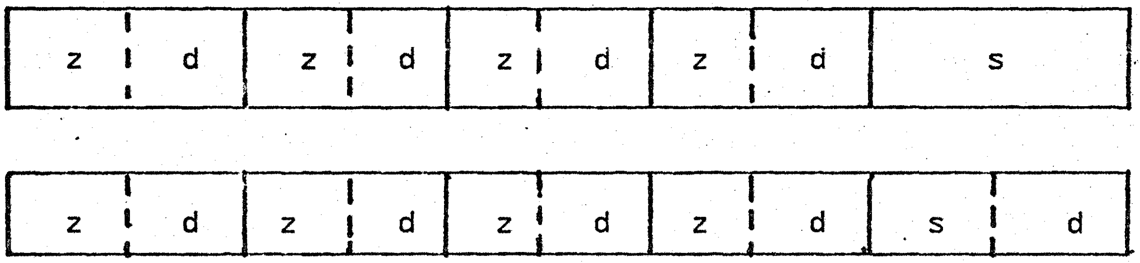
4567	00				01				10				11			
	00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11
0000	NUL	DLE	DS		SP	&										0
0001	SOH	DC1	SOS						a	j			A	J		1
0010	STX	DC2	FS	SYN					b	k	s		B	K	S	2
0011	ETX	TM							c	l	t		C	L	T	3
0100	PF	RES	BYP	PN					d	m	u		D	M	U	4
0101	HT	NL	LF	RS					e	n	v		E	N	V	5
0110	LC	BS	EOB	UC					f	o	w		F	O	W	6
0111	DEL	IL	PRE	EOT					g	p	x		G	P	X	7
1000		CAN							h	q	y		H	Q	Y	8
1001		EM							i	r	z		I	R	Z	9
1010	SMM	CC	SM		¢	!		:								
1011	VT	CU1	CU2	CU3	.	$	,	#								
1100	FF	IFS		DC4	<	*	%	@								
1101	CR	IGS	ENQ	NAK	{	}	_	'								
1110	SO	IRS	ACK		+	;		=								
1111	SI	IUS	BEL	SUB	|	¬	?	"								

ACK	Acknowledge
BEL	Bell
BS	Backspace
BYP	Bypass
CAN	Cancel
CC	Cursor Control
CR	Carriage Return
CU	Customer Use 1
CU2	Customer Use 2
CU3	Customer Use 3
DC1	Device Control 1
DC2	Device Control 2
DC4	Device Control 4
DEL	Delete
DLE	Data Link Escape
DS	Digit Select
EM	End of Medium
ENQ	Enquiry
EOT	End of Transmission
ESC	Escape
ETB	End of Transmission Block
ETX	End of Text
FF	Form Feed
FS	Field Separator
HT	Horizontal Tab
IFS	Interchange File Separator
IGS	Interchange Group Separator
IL	Idle
IRS	Interchange Record Separator
IUS	Interchange Unit Separator
LC	Lower Case
LF	Line Feed
NAK	Negative Acknowlege
NL	New Line
NUL	Null
PF	Punch Off
PN	Punch On
RES	Restore
RS	Reader Stop
SI	Shift In
SM	Set Mode
SMM	Start Manual Message
SO	Shift Out
SOH	Start of Heading
SOS	Start of Significance
SP	Space
STX	Start of Text
SUB	Substitute
SYN	Synchronous Idle
TM	Tape Mark
UC	Upper Case
VT	Vertical Tab

EXTENDED BINARY - CODED - DECIMAL INTERCHANGE CODE (EBCDIC)

4.4 Packed Decimal Numbers - In the packed format, two decimal digits normally are placed adjacent in a byte, except for the rightmost byte of the field. In the rightmost byte a sign is placed to the right of the decimal digit. The digits 0-9 have the binary encoding 0000-1001. The codes 1010-1111 are invalid as digits. This set of codes is interpreted as sign codes with 1010, 1011, 1100, 1110, 1111 recognized as plus and with 1101 recognized as minus. 1100 is the preferred code for plus. The codes 0000-1001 are invalid as sign codes.

4.5 Zoned Decimal Numbers - In the zoned format decimal digits are represented by their encoded form either in the ASCII or EBCDIC character set. In those forms the low-order four bits of a byte are normally occupied by a decimal digit, and the four high order bits are called the zone. Zone codes are 0011 for ASCII and 1111 for EBCDIC. Two forms of trailing sign are supported. In the first form the last byte of the number contains a sign consisting of either a plus or a minus in the appropriate binary encoded form. In the second form the last byte contains a decimal digit in its low-order four bits and a sign in the upper four bits. The sign uses the same convention as for the packed decimal format. These two formats are illustrated below:



Arithmetic is performed on operands in the packed format. Instructions will be provided to translate between the two formats and illegal binary encoded forms will be detected.

IPL INSTRUCTION REPERTOIRE

5.1 The cost effectiveness of the IPL is very sensitive to the instruction set chosen for it. Consequently, this item will take longest to define of all items, and will involve many measurements in an attempt to ensure that the optimum set is defined. To facilitate this effort the instruction set has been broken into three sub-sets comprising BDP instructions, scientific instructions and "general" instructions. It is anticipated that the general instructions will be heavily used in systems programming work, and quite heavily in scientific and commercial computations.

5.2 The low-end of the IPL is typical of the NCR market-place today, and will be used to dictate the requirement for BDP instructions. Memory compaction, with byte addressability characterizes the desire to minimize cost at this end of the spectrum. Complex, memory-to-memory descriptor driven operations have been defined at the present time for the low-end, while more conventional operations have been proposed at the high-end. An effort is in progress to merge these two approaches so that a single set of operations will result. The governing parameters are those of the low-end, and if necessary, an interpretive mode will be used at the high-end to achieve the level 5 compatibility goal.

5.3 At the other end of the spectrum the high-end attributes {performance} will decide the final format of the scientific instructions. These instructions will be simple and probably operate register to register to optimize performance. The instructions will form part of a virtual machine that will be emulated by a fast micro-processor at the low-end.

5.4 The general instruction set is the most difficult to derive in that the requirements have not been subject to the same analysis as the scientific and BDP fields. Since system code will use these instructions almost exclusively it is important that the problem be researched as thoroughly as possible. For this reason independent approaches from the low-end, and high-end viewpoints of the IPL are being made, with an aim to coalesce those divergent approaches into a single instruction set. At the same time statistical data is being gathered from existing operating systems and compilers. This data will be used to validate any instruction set that results.

5.5 The overall approach is to seek a fast, register-oriented machine for the high-end of the line which may use cache memories, instruction stack, parallel functional units, etc., in order to achieve the desired performance. The complex operations needed for the low-end machines will be trapped and interpretively executed if necessary. At the low-end a virtual machine capable of being emulated by a micro-processor will be evolved such that memory utilization {and consequently, overall cost} may be minimized.

5.6 Other virtual machines, such as COBOL virtual machines, will be permitted as long as they conform to the IPLOS interfaces.

6. OPERATING SYSTEM

6.1 Architectural considerations up to this point have concentrated on hardware characteristics, although the processor configurations and virtual memory mechanism are not without software implications. However, one important component of the system architecture is the operating system, and this section discusses some of the basic philosophies leading up to its definition.

6.2 The IPL Operating System {IPLOS} will be developed as a collection of subsystems all of which are administered as user work. In other words, the system will be organized as a collection of intercommunicating user and system processes, each having different levels of capability, protection and security. There will be a small amount of code, having maximum exposure to hardware characteristics, which will manage inter-process messages. For purposes of reliability, security and measurability it is desirable to organize system services as separate sub-systems. However, the resultant performance penalties appear to be unacceptable. Consequently, the IPLOS is planned to utilize both integrated and segregated system services. Interfaces will be defined such that future changes may be accomplished with a minimum effort. Such a measure will compromise reliability and security for performance.

6.3 A single operating system based on the general multiple processor configuration discussed in section 2 is planned for all IPL models.

6.4 The virtual memory organization described in section 3 is required by the IPLOS to enable code and data sharing in a controlled environment, maximizing security and protection for the system and user alike.

6.5 The IPLOS will organize all external I/O in an implicit fashion.

6.6 All IPLOS code will be written in a high level implementation language {SWL}. Any use of IPL assembly language will be done in the context of the SWL environment, and will be kept to an absolute minimum.