Appendix C — 2 J. Corbato'

GENERAL ELECTRIC

Computer Department
Phoenix, Arizona

SUBJECT

COPIES:

March 11, 1965

DISTRIBUTION

E. Glaser, MIT (3)
A. Evans, Jr., Carnegie Tech.
B. Galler, University of Michigan
J. Ossanna, Bell Telephone Lab. (3)
R. Reeves, Ohio State University
R. Boennighausen, GE
R. Claussen, GE (2) for E. Jacks at GMTC
G. Cowin, GE for SDC
M. Dustin, GE
I. Epstein, GE
W. Estfan, GE (6)
W. Gutman, GE
W. Heffner, GE
J. Merner, GE
G. Oliver, GE
F. Raunikar, GE (2) for Dr. Laird, Penn Stat
R. Ruth, GE
H. Sassenfeld, GE (8)
G. Scott (3)
W. Shelly, GE
E. Vance, GE
J. Weil, GE
R. Turner, NASA Lewis Res. Center

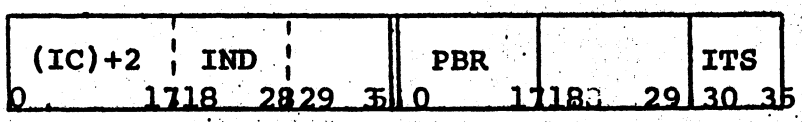
March 9, 1965

TO: DISTRIBUTION

Subject: MAC MEMO, REVISION V, ERRATA

Please note the following corrections to Revision V of February 26, 1965.

The format for the Store Control Double command page 22, should be changed to show that it is an ITS pair, i.e.,



The description of interrupt inhibit, page 25, should be changed to read "....permit inhibiting only timer interrupt....".

John F. Couleur

John F. Couleur, Manager
Advanced 600-Line Projects
Systems and Processors Operation

JFC/s

J. Poduska + J. Saltzer

GENERAL ELECTRIC

Computer Department
Phoenix, Arizona

SUBJECT

COPIES:

February 26, 1965

DISTRIBUTION

E. Glaser, MIT (3)
A. Evans, Jr., Carnegie Tech.
B. Galler, University of Michigan
J. Ossanna, Bell Telephone Lab. (3)
R. Reeves, Ohio State University
R. Boennighausen, GE
R. Claussen, GE (2) for E. Jacks at GMTC
G. Cowin, GE for SDC
M. Dustin, GE
I. Epstein, GE
W. Estfan, GE (6)
W. Gutman, GE
W. Heffner, GE
J. Merner, GE
G. Oliver, GE
F. Raunikar, GE (2) for Dr. Laird at Penn State
R. Ruth, GE
H. Sassenfeld, GE (8)
G. Scott (3)
W. Shelly, GE
E. Vance, GE
J. Weil, GE
R. Turner, NASA Lewis Res. Center

GENERAL ELECTRIC

Computer Department
Phoenix, Arizona

SUBJECT

COPIES:

February 26, 1965

To Attached Distribution:

We again request that you observe the proprietary nature of these memos, except that you may feel free to discuss the 636 (we have given it a new working name to avoid confusion) with any other receipients of the memo.

This is revision V and represents the final version of the 636. Several instructions have been added as a result of Bob Graham's coding of the Call and Entry Macros, thereby simplifying the housekeeping.

Since this is the final version, all future changes will be for purposes of clarification, defintion of portions not covered, or rarely we hope, plugging holes uncovered by programming.

Yet to be covered is the system interconnection, manipulation of the fault and interrupt vectors, and the method of automating "Fail Soft".

John F. Couleur, Manager
Advanced 600-Line Projects
Systems and Processors Operation

JFC/ss

Segmentation in the 636

The subject of segmentation and its benefits have been discussed elsewhere. The purpose of this memo is to describe the modifications to be made to the standard 635 processor to facilitate manipulation of segments.

To start, some definitions are in order. These defintions may or may not carry the full connotations as when used elsewhere, but will at least minimize semantic difficulties encountered when reading this memo.

A segment is a collection of information of sufficient size to warrant giving it a name. In current terminology it might be a subroutine, a collection of subroutines, a program or a collection of programs. Addressing within a segment is consecutive, starting with zero. A segment has a defined size which may be altered during execution.

A procedure segment is a segment which consists primarily of instructions. A procedure segment is considered "pure" if it is unaltered by execution.

A process is a sequence of procedures. The sequence may or may not be altered during execution.

A data segment is a segment which consists primarily of data.

A page is a fixed subdivision of addresses within a segment. For convenience, the page size will be an integer power of 2 (the addressing number base in the 635 and), a page always starts at an address which is an integer multiple of the page size.

A block is a subdivision of memory locations. For convenience, blocks in core memory will be the same sizes as pages.

A base register is a register used to point to a descriptor word which defines the location of a segment (or its page table) in core memory. The descriptor word also defines the segment size and any other restrictions on access to that segment. A base used to point to a descriptor word in this manner is an "external"base. Base registers also may be used to define locations within segments in which case they will be referred to as "internal" bases.

For convenience in discussion, the term "appending" a base register to an address will be used to infer the derivation of a core memory location from the contents of the specified base register. The mechanism of appending may then be defined separately. The term appending also infers a simultaneous verification that the address is within limits and that other access criteria have been met.

The concepts are complex. Suffice it to say that the complexity is required to eliminate the means and necessity for a segment to embed an absolute memory address within the segment. In any multiprogramming or time sharing environment it becomes necessary to move or remove segments and it is not always possible to replace the segment in the same location. Accordingly, the only basis for reference between segments is the use of invariant names, and the association and re-association, after moves, of names and addresses becomes a function of the operating system.

It is highly desirable to disassociate the variant and invariant portions of programs, or more specifically, the procedure (instruction) and constants from the data. So doing allows the simultaneous execution by several users of a common procedure segment, greatly improving use of memory in a multiprogramming environment. (Simultaneous compilation by several users can give an example of the possible memory space savings). In addition, the separation of variant and invariant information can greatly improve the efficiency of time sharing since invariant data need never be rewritten in bulk storage.

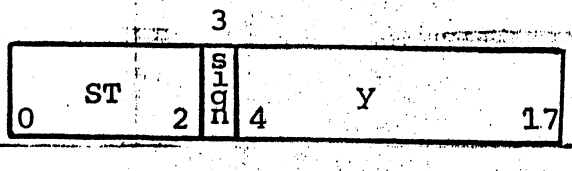
Meeting the requirements of relocation of programs requires at least one base register, the contents of which modify all memory accesses. It is desirable to contain in the base register an upper boundary to inhibit access to unused portions of the memory, thereby protecting other users from accidental or deliberate interference. By removing (and protecting) the base from the users direct influence, his program can be moved and isolated during execution without his knowledge or concern and without alteration of any portions of his information.

Separation of procedure and data requires at least one additional base, the contents of which are appended to all references to variable data. A different base setting is established for each user of the common procedure, thereby isolating his data from that of other users. Again, the second base must be isolated from the user and contain an upper boundary. It should be noted that the dual base also permits

separate procedures to use common data with complete protection. The advantages of segregation of variable data classes makes additional bases desirable, but the exact number is subject for debate. The 636 contains 11, eight of which are directly addressable.

The Instruction Cycle

To fetch the pair of instructions, the procedure base register is appended to the address from the instruction counter and the two instructions brought from memory. If bit #29 of the instruction is zero (mandatory on 625/635 programs) the effective address, i.e., after indexing, is appended to the procedure base and the operand fetched. If bit 29 is one, the address field of the instruction is interpreted as follows:



The three most significant bits constitute the segment tag specifying the address base register to be used for address preparation. The remaining 15 bits are interpreted as a signed 14 bit quantity to be additively combined with the selected index and internal base values. The signed address field provides, in effect, a positive and negative offset of 16K about the index value. The 262K max effective address is appended to the base register specified by the segment tag and the operand brought from memory.

Instructions

To fetch th[illegible] instructions, the procedure base register is appended to the address from the instruction counter and the two instructions brought from memory. If bit #29 of the instruction is zero (mandatory on 625/635 programs) the effective address, i.e., after indexing, is

Should indirection be specified, the indirect word will be fetched in the same manner as the operand. Normally, the full 18 bit address will be used, and the base register which was used to find the indirect word will be appended to the effective address. However, the modifier (tag) field may specify indirection to other segments in a manner to be described later.

If the instruction is a transfer, the next instruction will be fetched in the same manner as an operand. In addition, if bit 29 is a one, the Procedure Base register will be set from the base indicated by the segment tag in the address field of the transfer instruction, or by an indirection which locates the final address. Indirection transfer to another segment may be accomplished by the Special indirect modifier.

Address Appending

Two modes of address appending are provided and controlled by the segment descriptor.

but relocated protected; normal GECOS relocation

The non page mode is used when an entire segment is stored in contiguous memory locations. The function of this mode is to eliminate possible paging overhead, at the expense of moving other segments to make space available. An example where paging overhead might be prohibitive is in list processing programs. Non paged segments are located modulo 64 or 1024 in memory and tested for exceeding boundary limitations.

The paged mode causes segments to be subdivided into pages, and memory to be subdivided into equivalent blocks. Application of a stored transform (page table) then permits the contiguously addressed segments to be located in non contiguous blocks of memory, making possible the expansion and contraction of segments without moving the remaining contents of memory.

To implement paging, a segment page table will be referenced which will contain the block location for each page of the segment. Page sizes will be 64 or 1024 words, and for practical reasons a segment must be entirely of one or the other size.

Can different segments be of different size? Assume so

Description

Eleven base registers, and an associative memory of 16 words minimum will be added to the Standard 635: The base registers are grouped as follows:

1 - Descriptor base register divided into three fields. Bits 0-17, the address modulo 64; bits 19-26, the boundary; and bits 27,28, the descriptor field.

8 - Address Base Registers, numbered 0-7, containing two fields. Bits 0-17 the address field; bits 18-23 the control field.

1 - Procedure Base register consisting of one 18 bit address field.

1 - Temporary Base register containing an 18 bit effective base pointer.

The Descriptor Base Register locates the base of the Descriptor segment, if non paged, or the descriptor segment

page table, if paged, and defines the size of the descriptor segment. The base may not be altered by an object program, nor may the contents of the descriptor segment be altered by an object program.

The Descriptor Segment contains 36 bit descriptor words which locate, in core memory, a segment to be addressed during the process, or its page table. The descriptor word also defines ~~its size,~~ and designates whether it may be altered, executed as a process, and whether it is presently in core. Adjacent to the Descriptor segment, maintained by the control program, are stored segment names (in any form) to be used during execution of the process (sequence of segments). No fixed addressing relationship between names and descriptor words is maintained by the hardware, but the descriptor segment maintenance routine must be able to accept an address from an object program, correlate it with a name, and place in the address in the descriptor segment a related descriptor word.

The address base registers contain address information in one of two forms. If a base references an address within a segment, this address becomes part of the effective address and the base is called internal. If the address base register references a descriptor word within a descriptor segment, it is called external. The control field designates a base as internal or external, and if internal, specifies another base as external.

A referenced external address base locates a segment by pointing to a descriptor word in the descriptor segment that contains either the base address of the segment desired, or the base address of the page table which in turn locates the appropriate page of the segment.

The procedure base register contains the pointer to the procedure segment descriptor and the temporary base register contains the descriptor pointer associated with the immediate effective address.

Since object procedures do not store descriptor words but instead manipulate pointers (as the contents of external bases) to these words, the object program cannot alter descriptor words which are located in the write protected descriptor segment, thereby preserving the integrity of the protection provided by the descriptor word. In addition, the only region where segment locating absolute addresses are contained is within the descriptor segment where they may be retained as long as useful, may be readily removed when the need arises to remove the user, and may be re-associated with segment locations when the user returns to the machine. Pointers to descriptor words may also be held in indirect word pairs, permitting indirection to segments.

In summary, a complete address in the 636 consists of an effective address and a pointer to a descriptor word in the descriptor segment which provides (directly or indirectly) a base address. The pointer may be obtained either from an external base or from a specially tagged indirect word.

To avoid accessing the descriptor segment and then the page table during an instruction or operand fetch, an associative memory is provided which accepts as a reference either the pointer, and produces the page table location (i.e., the descriptor word), or the pointer and page number, and produces the block location of the page. A usage algorithm assures that the preceding N descriptor word and/or page references are retained in the associative memory for fast reference.

Detailed Description

This Section outlines the features to be embodied in the 636 to implement the functions described in the preceding section. In conjunction with the addition of the eleven bases and the associative memory, the present 600 repertoire will be modified to include: Sixteen new faults; thirty-six new instructions to manipulate bases; two new IT address modifier tags; and additional instructions to facilitate handling interrupts.

Base Registers and Descriptors

The format for each base register type is given in the accompanying figure. Descriptors appear both explicitly and implicitly in the Descriptor and Procedure Base registers. Descriptors held in the base and page table memory entries are combined logically (logical AND) to produce the actual descriptor information that applies to the page being referenced.

The Descriptor Bits and their locations are as follows:

A. Descriptor (segment/page)

27. Page (block) size. If 0, the page (block) size is 1024 words. If 1, the page (block) size is 64 words.

28. Paging. If 0, the segment is paged. If 1, the segment is not paged. If the segment is non paged, the segment is tested for boundary conditions as if it were paged.

29. Spare

30. Write permit. If 1, writing is permitted in this segment (page). If 0, writing permitted only if in Master mode.

31. Master Access. If 1, access is permitted to this segment (page) in Slave mode.

33,34,35. Class

A. Directed Fault, bits 30,31,32 specify fault type.

B. Data (may not be executed).

C. Procedure, Execute Only (may be executed only and not accessed as data by other procedures. Self access is permitted).

D. Procedure, Slave (may contain data).

E. Procedure, Master (may contain data).

If Segment Type is directed fault, bits 30,31,32 are interpreted as follows: Directed fault 0-7. The like numbered directed fault will be generated. Two directed faults will be reserved to denote the absence of a page or descriptor.

In a page table word, bits 25 and 26 are interpreted as follows:

25 Used.	If 1, this page has been referenced.
26 Written.	If 1, information has been entered in this page.
27-28	Not used.
29-35	Identical with segment descriptor.

B. Address Base Register Control Field

18, 19, 20. Specify the external base address if the base is internal.

21. Internal/External. If 0, the base is an internal base and an external base must be specified.

22. Lock base. If 0, the base may be reloaded in Slave mode. If 1, the base is locked against change in Slave mode.

23. Unassigned.

C. Descriptor Segment Base Register

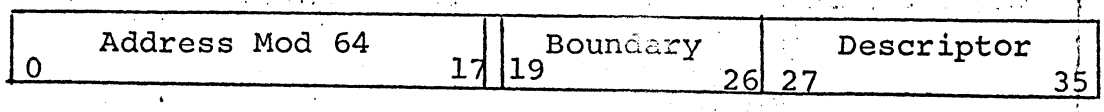
27. Page size. 64/1024

28. Paging. ON/OFF

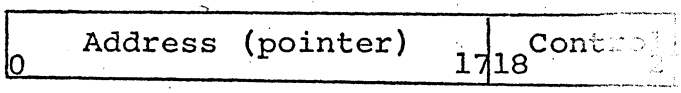
D. Procedure Base Register

External/Internal and Lock Base bits are implied. The base is always External and Alterable.

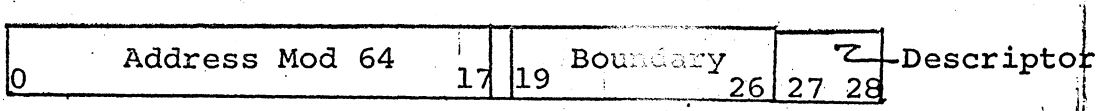
A. Segment Descriptor Word



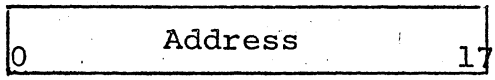
B. Address Base Register



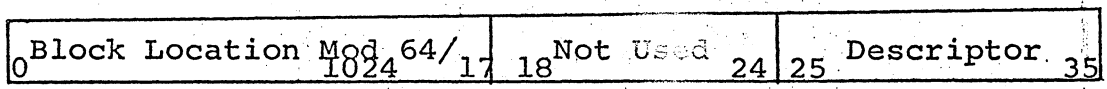
C. Descriptor Segment Base Register



D. Procedure Base Register - Temporary Base Register



E. Page Table Word



Base Register Formats

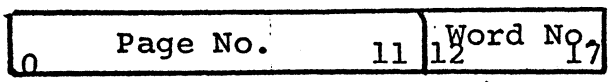
Address Formation

The effective address Y' and the offset address from a descriptor word or page table word are the principal constituents of the 24 bit memory address. The addressing mode, non-paged, paged, or absolute, establishes the manner in which they combine.

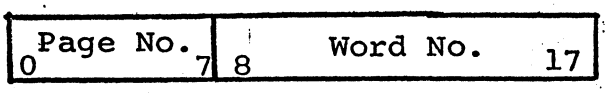
To describe the address manipulation required, a symbology will be established.

Y'	the effective address (y+i+b)
P^{64}	the page number of an address paged to 64 word pages, i.e., bits 4-11 of the address
P^{1024}	the page number of an address paged to 1024 words
W^{64}	the word number of an address paged to 64 word pages
W^{1024}	the word number of an address paged to 1024 word pages
A_B	the 18 bit address contained in the descriptor
N_B	the number of pages/boundary
Y_P	the address generated to enter the page table, i.e., the location where the block number will be found.
A_P	the address of the page word
$\xrightarrow{N}$ ($\xleftarrow[N]{}$)	the address to the left of the arrow is shifted N places to the right (left).
J@K	Field J is left concatenated with field K.
B(N(M))	base pair. Internal base N and external base M specified by the control field of N.

The effective address Y' is formed by adding the final address (y+i) and the contents (b) of the internal base if specified. In the paged mode this 18 bit-address is interpreted to be a page number and word number and as a normal address field otherwise.



64 word pages



1024 word pages

In the non-paged mode the 24 bit core location address is $(Y' + A_B \overleftarrow{\ }_6)$ i.e., the descriptor address field, left justified with the most significant bit of the 24 bit address field, is added to the effective address. The Boundary test forms the difference of the number of blocks allowed (N_B) as specified in the boundary field of the descriptor and the block address (effective address bits 0-7 for 1024 word blocks). The test is satisfied by a positive result. Specifically, $N_B - P^{1024}\overrightarrow{\ }_{19} \geq 1$ or $N_B - P^{64}\overrightarrow{\ }_{15} \geq 1$, where N_B is assumed right justified.

In the paged mode, the page number field and the offset address from the descriptor combine to form the address of a word in the page table, the contents of which provide the page core location. This page word address is concatenated with the word number field of the effective address to produce a core location.

If paging mode is selected,

$Y_P^{64} = (A_B \overleftarrow{6} + P^{64} \overrightarrow{6})$

Boundary Test $N_B - P^{64} \overrightarrow{15} \geq 1$

The final address = $(A_P \overleftarrow{6} \odot W^{64})$

$Y_B^{1024} = (A_B \overleftarrow{6} + P^{1024} \overrightarrow{10})$

Boundary Test $N_B - P^{1024} \overrightarrow{19} \geq 1$

Final address = $(A_P \overleftarrow{6} \odot W^{1024})$

Transferring

The Temporary Base register contains the immediate effective base pointer. Its contents always coincide with those of the external address base register dictated by the current instruction. The TBR is therefore the source of the base information necessary for memory addressing. Conversely TBR is the register from which all address base registers (including PBR) are set. For example, during a procedure instruction fetch, TBR is set from PBR if bit 29 is set to 0. If bit 29 is set to 1, TBR is set from the external address base registers specified in the selected internal base. A transfer causes the Procedure base register to be set from TBR. Consequently, a transfer with bit 29 set to 1 loads PBR from the selected external base via TBR.

Modifications to the present 600 Repertoire

Sixteen new faults will be added. They are:

MME's 2,3,4 (To preserve 635 compatability, the present 600 MME is not to be used for 636 Software)

Directed Faults 0-7

Trouble

Spares 1-6

Sixty-one new instructions to manipulate the bases will be added.

Load Base Register (n) LBR(n) (eight instructions)

$C(Y)_{0-23} \Rightarrow C(BR(n))$ only if the procedure is ~~execute only~~ or master or the block base control bit (in the Base Address Register) is 0, otherwise a fault will occur. The lock base bit will not be altered in slave mode.

Store Base Register (n) SBR(n) (eight instructions)

$C(BR(n)) \Rightarrow C(Y)_{0-23}$ Either an internal base or an external base may be specified.

$0 \Rightarrow C(Y)_{24-35}$

Load Descriptor Base Register LDBR

$C(Y)_{0-35} \Rightarrow DBR$ This instruction may be executed only in the Master mode. The op code will not be identical with the 635 LBAR instruction.

Store Descriptor Base Register SDBR

$C(DBR) \Rightarrow C(Y)_{0-35}$ This instruction may be executed only in the Master Mode. The op code will not be identical with the 635 SBAR instruction.

Store Bases STB

The eight address bases are stored in sequence in double word stores as in SBR(n). The address of this instruction must be modulo 8. The lock base bit will not be altered in Slave mode.

Load Bases LDB

The eight bases are loaded in sequence in double word loads as in LBR(n). If the requirements on the lock base bit are not met, the base contents will not change, and the sequence will go on to the next base without generating a fault.

Effective Address to Pair EAP(n) (eight instructions)

$Y' \Rightarrow C(n)_{0-17}$; $P' \Rightarrow C(m)_{0-17}$. The effective address Y' replaces the contents of the specified internal base and the effective pointer replaces the contents of the coupled external base. If the base specified by the instruction is external, the effective pointer replaces the contents of that base.

Effective Address to Base EAB(n) (eight instructions)

$Y' \Rightarrow C(n)_{0-17}$ The effective address Y' (y+mod+int.base) is put into $(n)_{0-17}$. An associated external base is not changed.

Add to Base ADB(n) (eight instructions)

$C(Y')_{0-17} + C(Bn)_{0-17} \Rightarrow C(Bn)_{0-17}$ The base (n) specified may be either external or internal. The usual lock base conventions apply.

Transfer and Set Base TSB(N(m)) (eight instructions)

$C(IC)+1 \Rightarrow C(B_N)_{0-17}$,

$C(PBR) \Rightarrow C(B_m)_{0-17}$,

$Y' \Rightarrow C(IC)$,

$P' \Rightarrow C(PBR)$

where N is an internal base and M is the external base specified by N. If the specified base is external, the transfer of C(IC)+1 is inhibited.

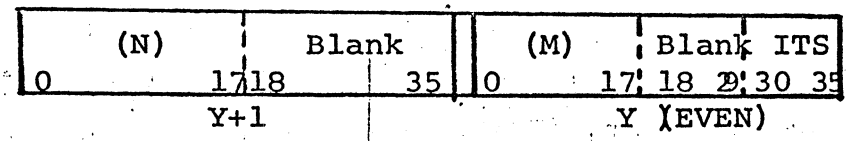
Load Control Field LDCF

This instruction permits alteration of the linkage between an internal base and the associated external base. The address field of the word specified by the effective address is interpreted as three 6 bit characters. Two 3 bit fields in character 0 designate the bases to be affected. Characters 1 and 2 contain the new control field information. In slave mode, the lock base bits in the designated base address registers will be unaltered. The format is

Ch 0		Ch 1	Ch 2	
A	B	Control Field Reg. A	Control Field Reg. B	Blank
0 2	3 5	6 11	12 17	18 35

Store Pair STP(N) (eight instructions)

This instruction stores the contents of an internal base (N) and the specified external base (M) as an ITS modified word pair. The format is

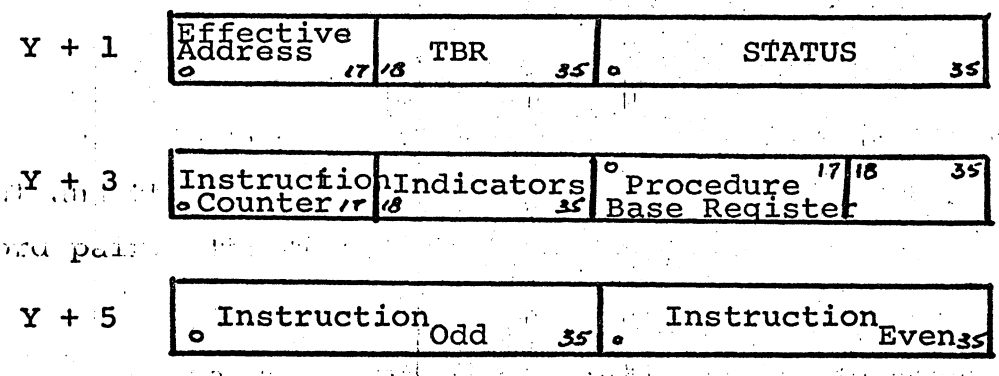


If an external base is designated by the instruction, a blank odd word will be stored.

The following new instructions will be added to facilitate handling interrupts.

Store Control Unit SCU

The instruction pair being held in the control unit, the instruction counter, the indicators, the Procedure Base register, effective address, the Temporary Base Register, the Segment Tag Registers, and the state flip-flops are stored in three double words (6 words). The address of this instruction must contain zeros in bits 15, 16 and 17. This instruction must be used when a fault or interrupt occurs on an operand fetch. The index register and the internal base, if any, is included in the effective address. The active (faulting) instruction may be determined from the stored status information. The format in storage will be



If Y_0=0 (status word sign bit) the fault occurred during the instruction address preparation (between instruction executions) and C(IC) is the address of the last successfully executed instruction. Otherwise, the fault occurred during instruction execution. The active instruction (ODD or EVEN) may be determined from the least significant bit of IC, $(Y+3)_{17}$.

Restore Control Unit RCU

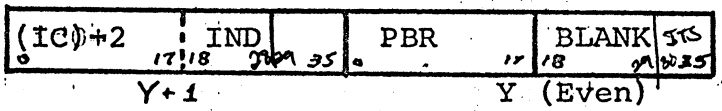
This instruction restores the control unit to reinitiate an interrupted instruction. If the instruction was interrupted on an instruction fetch (between instructions) it will reinitiate the instruction fetch. If interrupted on an operand (or indirect) fetch, it will reinitiate the operand fetch, using the stored faulting instruction. It is permissible to alter the faulting instruction word prior to executing RCU.

If SCU is the first instruction executed following a fault and RCU the last instruction prior to return to the breakpoint, the processor will restore to its former condition and continue the instruction as though the fault had never occurred.

Should a missing page fault be generated by the SCU instruction, the procedure (program) will be aborted by overriding all faults with the Trouble Fault.

Store Control Double STCD

Stores C(IC) + 2, the Indicator registers and the Procedure Base Register in an even-odd pair, with the following format:



Return Control Double RTD

Restores the Instruction Counter, Indicators and Procedure Base Register from an even-odd pair.

Sixteen new arithmetic instructions will be added.

Load Index From Lower (n) LXL(n)

$C(Y)_{18-35} \Rightarrow C(X_n)$

Store Index in Lower (n) SXL(n)

$C(X_n) \Rightarrow C(Y)_{18-35}$

$C(Y)_{0-17}$ are not altered.

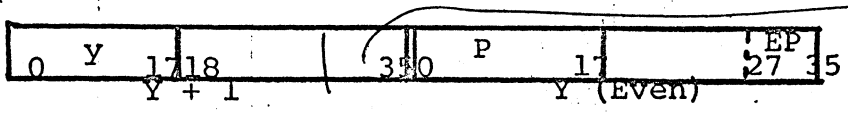
Four new IT address modifiers will be added.

The first, if recognized, whether an an instruction modifier or indirect word modifier, will cause an instruction execution instead of the operand fetch. The modifier is:

Execute Pair

The function of this modifier is to permit the use of an indirect word pair to cause a trap to some procedure (subroutine). Both indirect word and procedure are under control of the programmer (Compiler) and the operating system need not be called to establish the trap, remove the trap, or provide the linkage to the procedure.

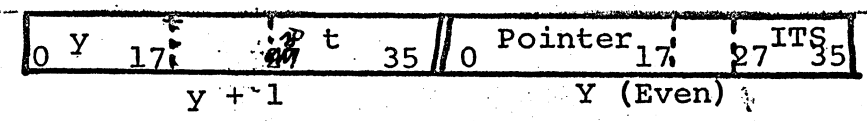
The modifier will appear in the tag field of an even indirect word. Recognition of this modifier will cause the address field of the word in which it appears to be interpreted as the effective pointer. The adjacent higher odd word specifies the effective address of two instructions to be executed. Unless the pair of instructions so addressed are SCU and transfer, with an eventual RCU, execution of the original instruction will not be completed. The format is



The second modifier to be added is:

Indirect to Segment ITS

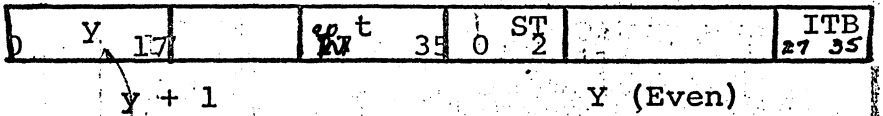
Recognition of this modifier, in an even indirect word will cause the address field of the word to be interpreted as the effective base address pointer. The adjacent higher odd word will be treated as a normal indirect word, relative to that pointer. The format for the pair is



The third modifier to be added is:

Indirect to Base ITB

Recognition of this modifier in an even indirect word will cause the most significant three bits of the address field of that word to be interpreted as a segment tag specifying the base address register that contains the effective pointer. The adjacent higher odd word will be treated as a normal indirect word, relative to that pointer. The format for the pair is:



A fourth IT modifier, FAULT 2, will be added to distinguish the IT fault in the 6363 from that occurring in the 635 programs.

Faults and Interrupts

The present Master/Slave indicator will no longer define the Master state since this is a function of the segment descriptor. It will, however, continue to denote that procedure is executed in absolute address. It will override the Temporary Base Register to declare, if 1, that:

1. All procedure is executed in absolute address unless bit 29 is 1, in which case the bases are appended.
2. Access to memory is unbounded.
3. Procedure is in Master.
4. The interrupt inhibit bit is still effective.
5. Timer run-out is inhibited.

This mode will be referred to as Absolute Mode. All interrupts and faults will be answered in absolute mode. Bit 29 will permit direct reference to the segments accessible to the interrupted user.

Unlike the 635, certain faults can occur during execution of instructions, and on the 636, provision has been made for saving and restoring the Control Unit when this occurs. Accordingly, the facility will be provided for accepting external interrupts and timer run out at the end of a memory cycle instead of at the completion of an odd location instruction.

To permit uniform interpretation of 635 programs, all 635 instruction codes that have no 636 meaning, function as NOP in 635 slave mode, or trap in 635 slave mode will cause a trap to the operating system when encountered in Slave Mode (TSS or equivalent is needed to get out of the absolute mode). The 635 instructions involved are: LBAR, RMFP, SMFP, LDT, TSS. The function of the interrupt inhibit bit (28) will be altered to permit inhibiting timer interrupt when operating in slave mode.

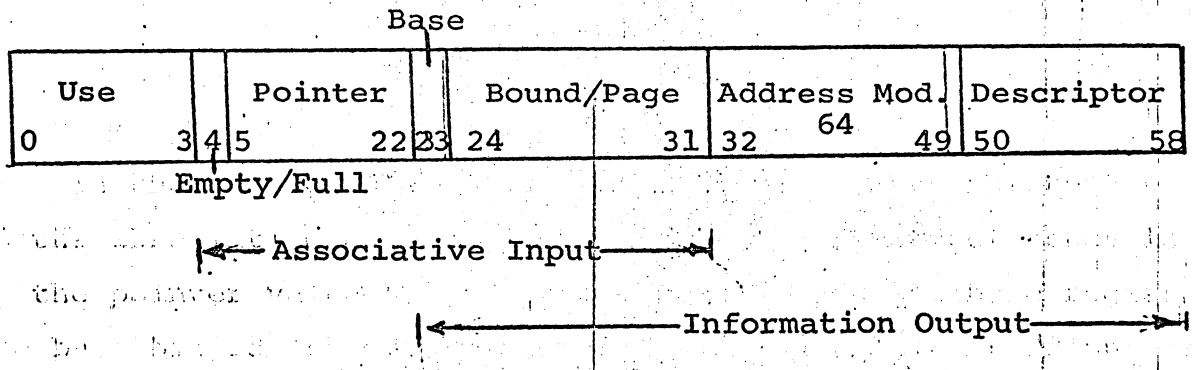
Associative Memory

A minimum of 16 words of rapid access associative memory is provided to capture the segment descriptor and page table words when first used in the execution of a process. Each word is then available for a period determined by its relative frequency of use and the interval between use. The associative memory normally performs a look-aside function except during Absolute mode operation when it is quiescent.

Memory Management

In the course of address preparation, several references to the associative memory may be made. The first reference is by the pointer value held in the effective address base register. The base bit and the descriptor information will be read out if present for subsequent processing. Failure to match indicates the absence of both the page desired and its segment descriptor and results in a reference to the Descriptor Segment for the descriptor. The second reference is by the pointer value held in the effective address base register and page number to determine the presence of the page word. If present, address preparation proceeds. If the page word is absent, a reference by pointer and base bit determines the presence of the appropriate page table base address. If present, the base is used to fetch the desired page table word. Otherwise, the appropriate descriptor is accessed in the descriptor segment. The first entry of a page table word into the associative memory will cause the USED bit in the related descriptor field to be set in main memory. In a like fashion, the first time a page is written into by a store command, the written bit in the descriptor will be set to one in both the associative memory and main memory page table word.

Word Format



Use Contains integer usage value.

Empty/Full If set to 1 indicates valid information contained in word.

Pointer An 18 bit address which refers to a word within the descriptor segment.

Base If set to 1, the information field contains a segment descriptor; if set to 0, the information field contains a page table word.

Boundary Contains a boundary for the segment descriptor or a page number for a page table word.

Address Holds the 18 bit address (mod 64) for the page table or page.

Descriptor Contains the 9 bit descriptor field (see segment descriptor and page table word format description). Bit position 52 (normally blank in a segment descriptor) will contain the page table written bit.

Usage Alogrithm

The use field contains an integer whose value reflects the both frequency and recency of use for the related memory word. A value of 15 is assigned to the most recently used word. A value of zero indicates the least recently used word. The use value is modified according to the following rules:

New Entry into A.M.

a. New word replaces contents of the word whose use value is zero; a new use value of 15 is assigned to this word.

b. The use value of all other words is decremented by 1.

New Usage of Word

a. The use value of the selected word is noted and a new use value of 15 assigned.

b. For all words having a greater use value than that noted for the selected word, decrement the use value by 1.

Associative Memory Instructions

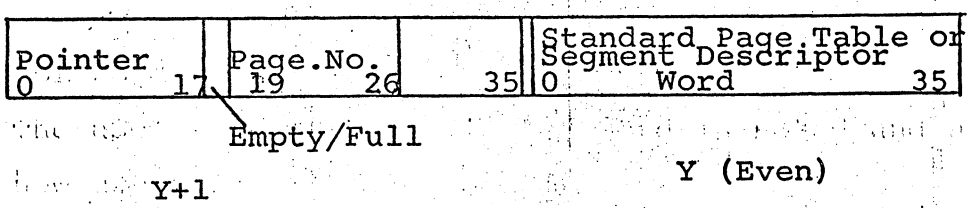
New instructions are provided for controlling the associative memory. They may be executed only in Master mode.

Clear Associative Memory

The Empty/Full bit in each word will be set to zero.

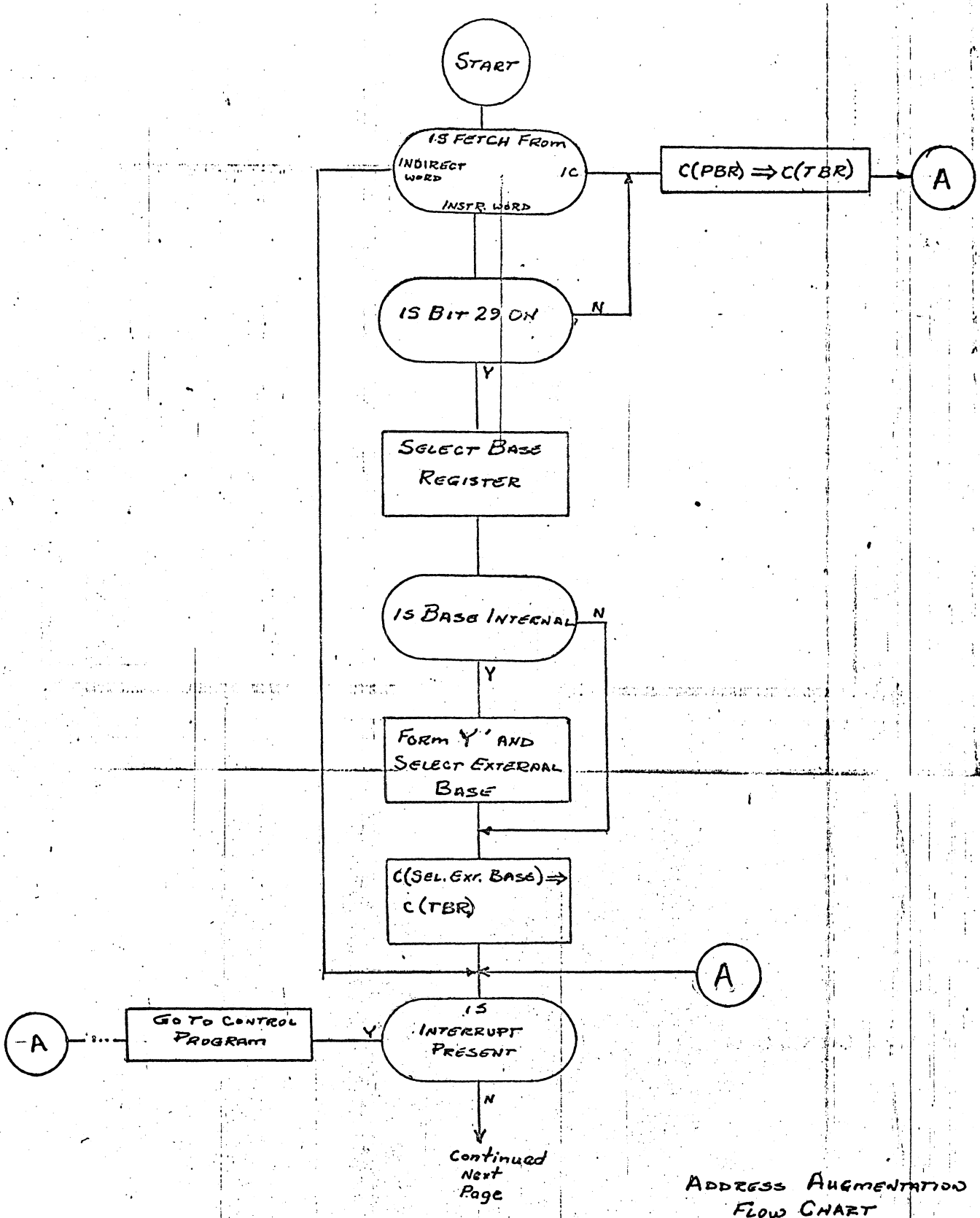
Store Associative Memory

The contents of each associative memory word; exclusive of use value, will be stored in core memory as a word pair. The successive word pairs will be ordered by increasing use value. The format for the word pair is:

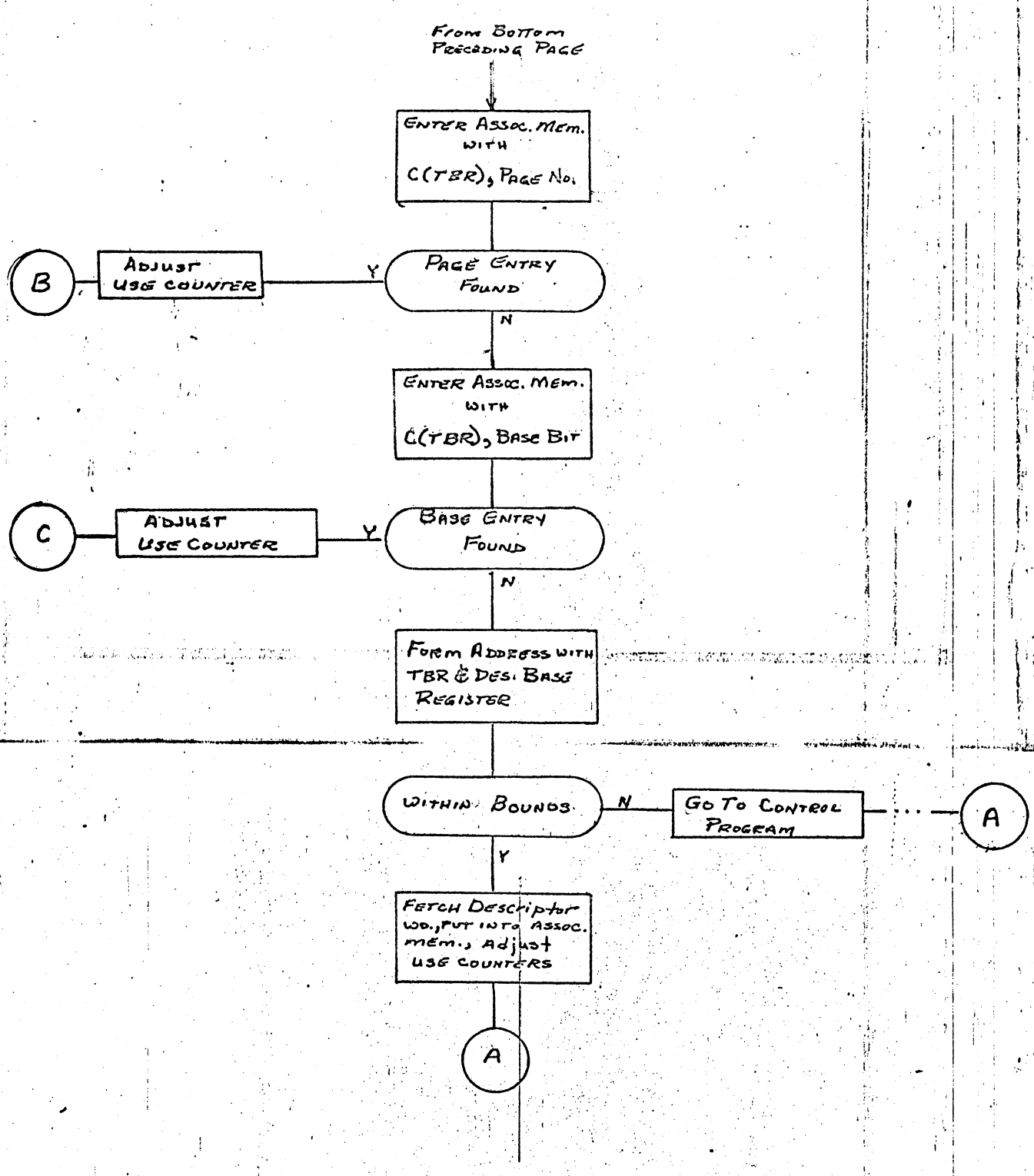


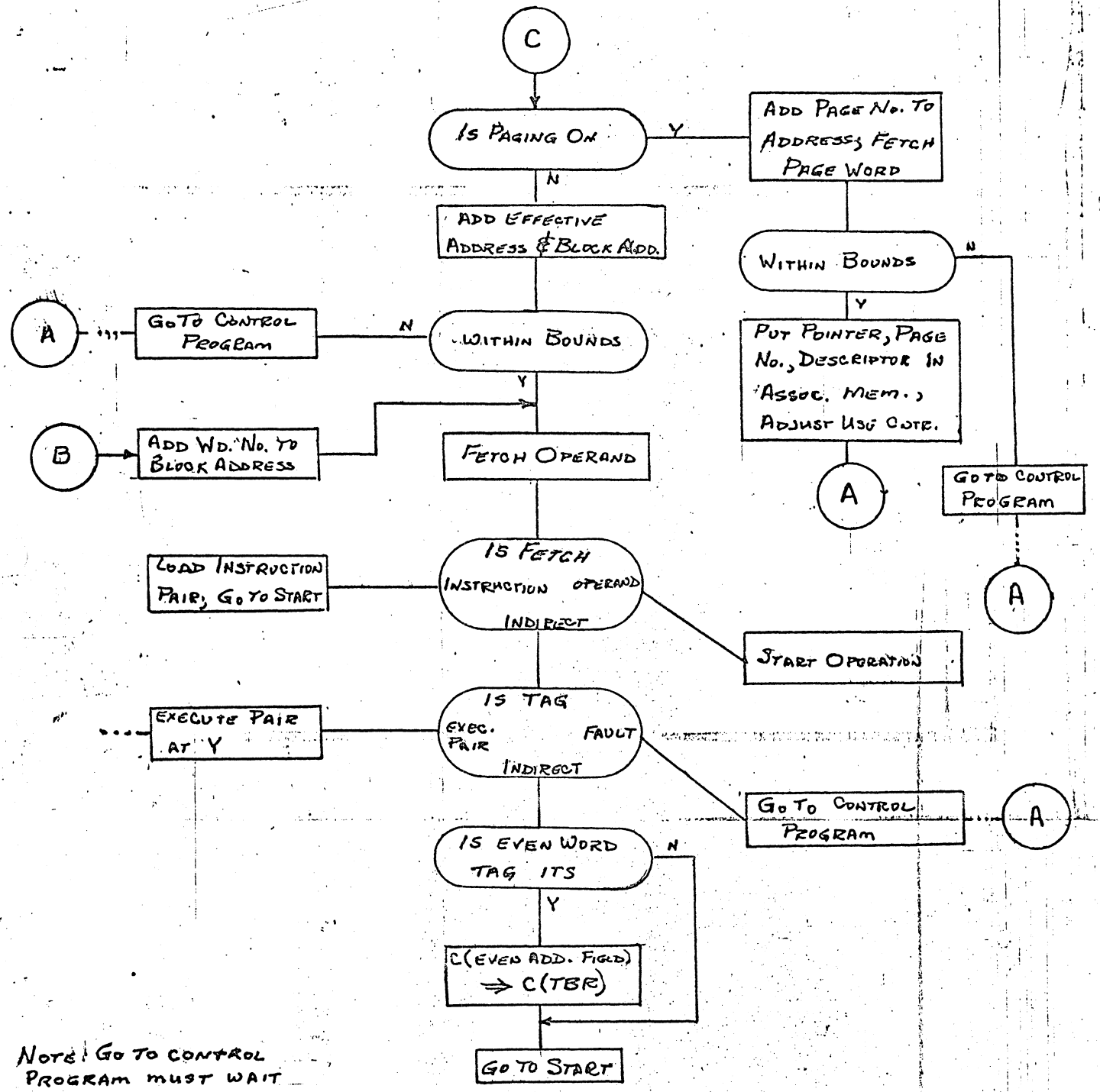
Store Associative Memory Zero

The contents of the associative memory word having zero usage value will be stored in a word pair with the above format.



ADDRESS AUGMENTATION FLOW CHART


From Bottom Preceding Page
Enter Assoc. Mem. with C(TBR), Page No.
Page Entry Found
Y
B
Adjust Use Counter
N
Enter Assoc. Mem. with C(TBR), Base Bit
Base Entry Found
Y
C
Adjust Use Counter
N
Form Address with TBR & Des. Base Register
Within Bounds
N
Go To Control Program
A
Y
Fetch Descriptor wd., Put into Assoc. mem., Adjust use counters
A



NOTE: GO TO CONTROL PROGRAM MUST WAIT UNTIL PRECEDING OPERATION IS COMPLETE. RCU RETURNS TO (A)

Supplement to Appendix C

BELL TELEPHONE LABORATORIES
INCORPORATED

SUBJECT: Clarification of Appending Hardware DATE: June 14, 1965

FROM: V. A. Vyssotsky

TO THE SOFTWARE COMMITTEE:

During the first two weeks of June 1965, E. Glaser of MIT and V. A. Vyssotsky of BTL discussed with Messrs. J. F. Couleur and G. Oliver of GE a number of details of 636 hardware which are not clearly described, or are described with minor errors, in J. F. Couleur's memorandum of February 26, 1965 (software committee document 71). This note is a summary of design details gleaned by V. A. Vyssotsky from the discussions. Further information will be available on July 15.

636 Fault Traps

The 635 has 16 distinct fault trap locations; the 636 has 32. Although the detailed mapping of fault conditions into fault traps on the 636 is not frozen, the general structure of 636 fault traps is known, and I shall detail its current form below. Numerical fault code values should not be inferred from this listing; I do not know numerical values of any of the fault codes. All fault traps on the 636 which were not on the 635, except the trouble fault, have priority IV. Priority of the trouble fault has not been determined.

1) Shutdown; one fault. This fault corresponds to the same conditions on the 636 as on the 635.

2) Memory; one fault. This fault occurs in all those cases which would cause it on a 635, and in addition occurs if the appending hardware detects a bounds violation. Whether a bounds violation resulted from testing against the descriptor base register or from testing against a descriptor in the descriptor segment can be determined from the contents of the safe-stored control unit.

3) MME 1; one fault. Same as on 635.

4) Fault tag 1; one fault. Same as on 635.

5) Timer runout; one fault. Same as on 635.

6) Command fault (compatibility fault); one fault. This fault replaces the command fault of the 635. It occurs if a slave mode procedure attempts to execute any of the instructions: CIOC, SMIC, RMCM, SMCM, LDT, TSS.

7) Derail; one fault. Same as on 635.

8) Lockup; one fault. This fault has changed slightly from the 635. The fault occurs if the processor is in a program lockup which inhibits recognizing an execute or timer runout fault or an interrupt for more than 16 milliseconds. Examples of this condition are: TRA * or continuous use of the inhibit bit in absolute or master mode; use of the inhibit to prevent timer runout in slave mode.

why?

9) Connect; one fault. The processor has received a connect from another processor via a system controller.

10) Parity; one fault. Same as on 635

11) Illegal op code; one fault. Same as on 635.

12) Operation not completed; one fault. Same as on 635.

13) Startup; one fault. Same as on 635.

14) Overflow; one fault. Same as on 635.

15) Divide check; one fault. Same as on 635.

16) Execute; one fault. Same as on 635.

17-24) Directed faults 0-7; eight faults. Whether the fault was specified in a segment descriptor, in a page descriptor for the descriptor segment, or in a page descriptor for some other segment can be determined from the contents of the safe-stored control unit.

25-27) MME2,3,4; three faults.

28) Fault tag 2; one fault.

29) 635 instruction deleted fault; one fault. This fault will occur if any procedure attempts to perform a 635 instruction meaningless on the 636. These instructions are: LBAR, SBAR, SMFP, RFMP.

30) Illegal procedure fault; one fault. This fault will occur if a slave procedure attempts to violate descriptor specification of a segment or page as

a) master access only

b) write protected

c) execute only procedure

d) data only (i.e., no execution permitted)

This fault will also occur if a slave procedure attempts to load or alter the contents of a locked base, unless the attempt occurs as part of an LDB instruction. An LDB instruction in slave mode loads all unlocked bases, leaves all locked bases unaltered, and the encountering of locked bases during execution of an LDB does not cause a fault. The illegal procedure fault will also occur if a slave procedure attempts to perform any of the instructions LDBR, SDBR, CLAM, STAM, STAMZ, SCU.

31) Invalid descriptor fault; one fault. This fault occurs if the class of a descriptor being decoded in the appending hardware is not any of the defined classes.

32) Trouble fault; one fault. If, following acceptance of any trap and before successful completion of the SCU, a second trap must be forced, the trouble fault occurs instead of the second trap. The trouble fault will occur, for example, on a parity error in fetching the instruction pair from the fault vector, or on a directed fault during operand addressing for the SCU.

Bounds Test Erratum

The boundary test is incorrectly described on page 15 of the February 26 memorandum. A correct description is:

Let N_B be the contents of the boundary field of a descriptor. Let P^{1024} be the contents of bits 0-7 of an address which is to be translated by the appending hardware using 1024 word paging. Let P^{64} be the contents of bits 4-11 of an

address to be translated by the appending hardware using 64 word paging, and let H^{64} be the contents of bits 0-3 of that address.

Then the bounds test for 1024 word paging is satisfied if $N_B - P^{1024}_{19} \rightarrow \geq 0$. The bounds test for 64 word paging is satisfied if $H^{64} = 0$ and $N_B - P^{64}_{15} \rightarrow \geq 0$.

Time of Generation of Appending Faults

The sequence of calculations and tests in the appending hardware must be known in order that the software may handle faults correctly. I shall therefore specify in as much detail as I can the sequence of calculations and tests relevant to fault handling.

1) The bounds test for the descriptor segment is performed.

2) If the descriptor segment is paged, the appropriate page table entry is tested for directed fault. No other tests are made on a page table entry for the descriptor segment.

3) The segment descriptor word is retrieved.

4) The class of the descriptor is decoded. If the class is directed fault, a direct fault is taken.

5) If the mode is slave, the master access bit of the descriptor is tested.

6) If the mode is slave, tests are made for attempted execution of data, illegal accessing of execute-only procedure, and illegal accessing of master procedure.

7) The bounds check is performed, and if the mode is slave a check is made for write-protect violation. I do not know in which order these two tests are performed.

8) If the segment is paged, the page table entry is tested for directed fault.

9) If the segment is paged and the mode is slave, the master access bit of the page table entry is tested.

10) If the segment is paged and the mode is slave, the page table entry is tested for attempted execution of data, illegal accessing of execute-only procedure, and illegal accessing of master procedure.

11) If the segment is paged and the mode is slave, the page table entry is checked for write-protect violation.

This description corrects and supersedes the description on pages 10 and 11 of the memorandum of February 26.

Definition of Master and Execute Only Classes

On page 11 of the February 26 memorandum, the classes "procedure, execute only" and "procedure, master" are mentioned but not precisely defined. I shall now give what I understand to be the definition of these classes.

If a segment is of class "procedure, execute only," then:

a) If the current mode is slave and if the procedure base register does not point to the descriptor of this "procedure, execute only" segment, any reference to this segment (whether instruction fetch, operand fetch, operand store or indirect address) will cause a fault unless the reference is as the final

effective address of a transfer-type instruction transferring control of location 0 of the execute-only segment.

b) If the procedure base register does point to the descriptor of this "procedure, execute only" segment, references to this segment will be treated as if the class of the segment were "procedure, slave." The description of LBR(n) on page 17 of the February 26 memorandum is incorrect; the words "execute only or" should be deleted.

c) If the current mode is master or absolute, access to a "procedure, execute only" segment is unrestricted.

If a segment is of class "procedure, master," references to it will be treated in exactly the same way as if the class of the segment were "procedure, execute only" except that in case

b) If the procedure base register does point to the descriptor of this "procedure, master" segment, then reference to the segment is unrestricted, because the mode is master.

Fault Vector Relocation

The origin of a fault vector or interrupt vector on the 636 will be determined by manual switch settings. For each fault or interrupt vector the corresponding switches may be set either locally or remotely. The fault vector origins will not be relocatable under program control.

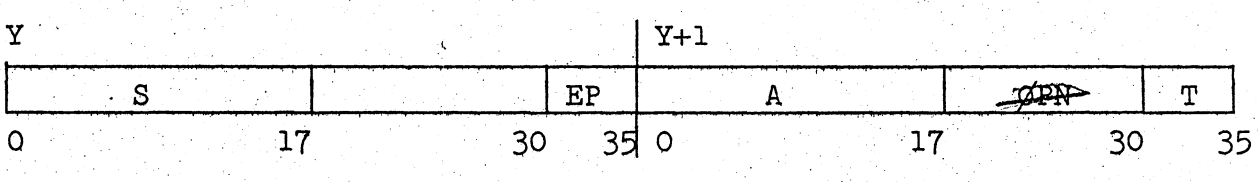
Safestore Contents

Detailed specifications on what control unit information is safe stored at time of fault of interrupt are not known to me. They will be available about July 15.

Execute Pair

The mechanism for the execute pair modifier agreed on by BTL, MIT and GE on February 10, and described in the February 26 memorandum, is inherently unusable. A replacement mechanism has been designed, and is described below. This has not yet been frozen; however, this or some similar technique will be frozen the week of July 6.

The format of an EP is



Location Y must be a location congruent to zero mod 8. When the EP is recognized, the contents of the control unit is safe-stored directly into locations Y+2, ..., Y+7 inclusive. Then execution of the instruction in location Y+1 is forced. Normal address modification will be performed during operand accessing for the instruction in Y+1. The instruction in Y+1 should be a transfer or RET, or an XEC or XED with a transfer or RET as a subject instruction. What the hardware will do if this condition is violated is unpredictable. I do not know what will happen if the instruction in location Y+1 has a 1 in bit 29.

Appending in Absolute Mode

If the current mode is absolute and an instruction is executed with bit 29 set to 1, appending will be done throughout the entire operand addressing computation of the instruction in

question. As soon as that operand addressing chain has been completed, appending will cease. Thus, for example, if bit 29 of an XEC instruction performed in absolute mode is 1, the fetching of the subject instruction will be done with appending, but the operand fetch for the subject instruction will be done without appending unless the subject instruction also has bit 29 set to 1.

The EP, ITS and ITB modifiers always imply appending. Hence, if an ITS or ITB modifier is encountered during an addressing computation in absolute mode, the remainder of that addressing computation will be performed with appending whether or not appending was in effect when the modifier was encountered. If an EP modifier is encountered during an addressing computation in absolute mode, the fetch of the subject instructions of the EP will be performed with appending. I do not know in what mode the subject instructions will be executed.

Miscellany

1) If a descriptor segment is paged, use bits in page table entries for the descriptor segment are set just as are the use bits in page table entries for any other segment.

2) When an internal base is specified, either by bit 29 being 1 or by an ITB modifier, the contents of the internal base is added to the effective address before appending begins. Appending is then done using the associated external base. Any subsequent addressing steps will use the external base but not the internal base.

3) There probably will not be an automatic bootstrap load procedure from the drum.

4) The pictures of the EP, ITS and ITB modifiers on page 23 of the February 26 memorandum are erroneous. All tags occupy bits 30-35 of the word, not bits 27-35. EP, ITS, and ITB are tags just like any other tags.

5) EP, ITS, ITB, FT1 and FT2 modifiers will be recognized after the start of IR indexing, although other IT modifiers are not.

6) The effect of modifiers invalid in context (e.g., TRA A,DU) is unpredicable. Hypotheses based on the assumption that the hardware is systematic are almost certainly incorrect.

7) The effective address of a conditional transfer instruction is not computed unless the condition is satisfied.

8) The operand fetch of an XEC or XED is treated as a data fetch. Hence in slave mode an XEC or XED can be used to execute an instruction from a segment whose class is "data." Since XEC and XED are not transfer-type instructions, in slave mode XEC and XED cannot be used to execute any instructions from a segment whose class is "procedure, execute only" or "procedure, master." If the current mode is master, and if the subject instruction(s) of an XEC or XED are in a segment of class "procedure, slave," the subject instruction(s) will be performed in slave mode.

9) The SD modifier causes the tally field of the indirect word to be incremented, not decremented.

10) If a segment or page is write-protected, an attempt to modify a word in the segment by ID, DI, IDC, DIC, AD, SD, or SC indexing will cause a fault, and the word whose modification was attempted is not fetched from memory.

V. A. Vyssotsky

MH-1373-VAV-AK V. A. VYSSOTSKY