Saltzer

MPL-52

TO: Multics Performance Log
FROM: M. Schroeder
DATE: June 25, 1970
SUBJECT: Analysis of 645 CPU Associative Memory Operation and Discussion of Alternative Associative Memory Designs

Introduction

The preceding report in this series, MPL-51, documents the results of several measurements of the 645 CPU with electronic counters. The primary purpose of these measurements was to determine the effectiveness of the associative memory (AM) for SDWs and PTWs that is part of a 645 CPU. The AM prevents the appending cycle of the CPU from slowing the CPU's instruction execution rate in proportion to the product of the memory cycle time and number of SDWs and PTWs referenced, by remembering in flip-flop register copies of the 16 most recently used SDWs and/or PTWs. It is the purpose of this report to analyze the results documented in MPL-51, and to derive from those results alternative AM designs. The alternatives include a simple change (two wires) to the 645 AM to improve performance, and two AM designs for the follow-on processor to the 645 CPU. In addition, the results of several previously unreported measurements are given.

This report is organized in four sections. The first describes in some detail the operation of the current AM. This provides necessary background information for understanding the performance analysis presented in the second section. Following that, the results of several experiments in which the number of AM registers were varied are presented. Finally, these results and the analysis are applied to derive alternative AM designs.

Operation of current AM

The current AM consists of 16 flip-flop registers of 60 bits and some control logic. Each associative register (AR) may contain a copy of an SDW or a PTW, as well as tags which identify the segment, or the segment and page, associated with the contained SDW or PTW, respectively. Figures 1 and 2 present the formats of an AR for these two cases.

AM search cycle -- Each time the CPU enters an appending cycle (to perform a virtual memory reference for a pair of instructions, an indirect pair, or an operand) the AM is searched to determine if it contains a copy of the SDW or PTW which is needed. The input to the search is the segment number and word number address of the item being referenced and a flag indicating whether or not the reference will be a write or a read/alter/rewrite. The search proceeds in up to three passes.

In PASS1 (216ns) the contents of every AR whose Empty/Full bit is "1" and whose segment number matches that given are ORed together. Because the values of bits 27, 28 are always the same for all valid AR's with the same segment number tag, if one or more segment number matches occur it is possible to determine from the ORed result whether or not the segment is paged and the page or block size (see Figure 1). Three results of PASS1 are possible:

1.0 No segment number match occurred. In this case the AM contains no useful SDW or PTW and all appending words will have to be retrieved from memory.

1.1 A segment number match occurred, and bit 28 indicates the segment is not paged. In thiscase the ORed search output must be the contents of the single AR containing the SDW for the non-paged segment. No further appending words are required.

1.2 A segment number match occurred, and bit 28 indicates the segment is paged. In this case the ORed search output may be the result of more than one segment number match (ARs containing both the SDW and PTWs for the segment will match). Bit 29 will indicate large or small pages properly, and bit 18 will be "1" if and only if the match set includes the SDW, but all other bits of the ORed result may be meaningless. A more selective search is thus required.

Address	1	Boundary	Desc.	Segment number	F	A	UC
0	18		27	36 53			56 59

Address field -- Address from the SDW. Contains the quotient mod 64 of the base address of the segment.

Bit 18 -- Always "1". This is the flag that indicates the AR contains an SDW (instead of a PTW).

Boundary field -- Size field from the SDW. Contains the total number of pages in the segment if it is paged, or the number of blocks in the segment if it is not paged.

Descriptor field -- Descriptor field from the SDW. SDWs with bit 32 set to "1" are not put in the AM, so bit 32 is always "0". A specification of each bit follows:

27 if a "1" the segment consists of 64 word pages or blocks
   if a "0" the segment consists of 1024 word pages or blocks
28 if a "1" the segment is not paged
   if a "0" the segment is paged
29 not used
30 if a "1" the segment may be written
   if a "0" the segment may not be written
31 if a "1" the segment may be accessed in both Master and Slave mode
   if a "0" the segment may be accessed in Master mode only
32 always "0"
33-35 class code from the SDW. Allowed values are:
   001 Data
   011 Procedure only
   010 Procedure Slave
   100 Procedure Master

Segment number field -- Contains the segment number associated with the SDW.

Full/Empty bit -- Internal control bit:
if a "1" the information in this AR is valid
if a "0" the information in this AR is not valid

Adjust usage counter bit -- Internal control bit. Whenever it is "1" the usage counter for this AR will be decremented by 1 on an adjust usage counters cycle.

Usage counter field -- Records the order of most recent use for the ARs. The most recently used AR has a usage counter value of "1111" and the least recently used AR has a usage counter value of "0000". Whenever a new entry is added to the AM in is put in the AR with a usage counter value of "0000".

Figure 1: Format of an AR that contains an SDW

Address	0	Page num.	Desc.	Segment number	F	A	UC
0	18		27	36	53		56 59

Address field -- Address from the PTW. Contains the quotient mod 64 of the base absolute address of the page.

Bit 18 -- Always "0". This is the flag that indicates the AR contains a PTW (instead of an SDW).

Page number field -- Contains the page number associated with the PTW.

Descriptor field -- All bits in the field have the same meaning as for an SDW except bit 29. The descriptor contained in this field is generated from both the SDW and PTW descriptors as indicated below:

27 copy of bit 27 of SDW descriptor

28 copy of bit 28 of SDW descriptor

29 the modified bit. Initially contains a copy of bit 26 of the PTW. If this is "0" then it will be set to "1" when the page is modified (as will bit 26 of the PTW in memory).

30 set from the AND of the corresponding bit in the SDW and PTW descriptors

31 set from the AND of the corresponding bit in the SDW and PTW descriptors

32 always "0"

33-35 set to match the most restrictive of the class codes from the descriptor in the SDW and the PTW.

Segment number field -- Contains the segment number associated with the PTW.

Full/Empty bit -- Same as for an AR containing an SDW.

Adjust usage counter bit -- Same as for an AR containing an SDW.

Usage counter -- Same as for an AR containing an SDW.

Figure 2: Format of an AR that contains a PTW

If result 1.2 (above) occurs, then the PASS2 search (234ns) is necessary. This is a search for the PTW required (it is not necessarily in the AM). To match on this pass an AR must have the Empty/Full bit set to "1", must match the given segment number, and must match the page number which is derived from the given word number and bit 28 of the PASS1 result. (Note: it is necessary to know whether large or small pages are the case before the page number can be derived.) At most one AR can match. Two results of PASS2 are possible:

2.0 A match occurs. The search result must be the needed PTW. No further appending words are required as the matching AR contains the relevant description information derived from both the PTW and the SDW of the containing segment.

2.1 No match occurs. The needed PTW is not in the AM. All may not be lost, however. If bit 18 of the PASS1 search result was "1", then the needed SDW must be contained in some AR. PASS3 is initiated to find it. If bit 18 was "0", then all appending words will have to be retrieved from memory.

The PASS3 search (216ns) follows result 2.1 (above) whenever bit 18 of the PASS1 search result was"1" (indicating the SDW is somewhere in the AM). To match on this pass an AR must have the Empty/Full bit set to "1", must contain the given segment number, and must have bit 18 set to "1". A match must occur. After the search is completed, the final appending word (the PTW) must be retrieved from memory.

Before describing the relation of the various search results to the rest of the appending cycle in more detail, it is useful to consider the AM clear, write, read, and adjust usage counter cycles.

AM clear cycle -- The clear cycle is used to empty the AM. It occurs as a result of executing an LDBR or a CAM instruction. Clearing is accomplished by unconditionally setting the Empty/Full bit in all 16 ARs to "0". The usage counter of the 16 ARs are unconditionally set with the 16 values

"0000" through "1111", a different value being placed in each AR. Note that immediately after a clear cycle no search of the AM can find a match.

AM write cycle -- The empty AM which results from a clear cycle is of no use to the appending cycle. SDWs and PTWs must be written into ARs where they can subsequently be found on a search cycle. Each time an unfaulted SDW or a PTW is retrieved from memory during a normal appending cycle, a copy is written into the AM. In the case of an SDW the relevant SDW fields (see Figure 1) are presented unmodified to the AM write cycle, along with the segment number. These are written into the AR with a usage count of "0000", and bits 18 and 54 of the AR are set to "1". In the case of a PTW, a descriptor is generated from the SDW and PTW descriptors as implied by Figure 2, and the relevant PTW fields thus modified are presented to the AM write cycle, along with the segment number and the page number. These are written into the AR with a usage count of "0000", bit 18 of the AR is set to "0", and bit 54 to "1". An adjust usage counters cycle always immediately follows a write cycle.

AM read cycle -- If the relevant SDW or PTW is found in an AR during a search cycle, the read cycle is entered. Bits 0-35 of that AR are placed on the data line to the rest of the appending cycle. An adjust usage counters cycle always immediately follows a read cycle.

Adjust usage counters cycle -- This cycle is entered immediately after a read or write cycle. The effect is to set the usage counter in the AR read or written to "1111", and to decrement by 1 the usage counters of all ARs whose usage counter values were greater than the original value in that AR. The cycle is implemented by associating a 4-bit subtractor/comparator with each AR. All usage counters are updated simultaneously. Note that this maintains the least recently used replacement descipline in the AM. The first 16 write cycles after a clear will fill up all 16 ARs. After this the AR containing the SDW or PTW that has gone the longest without being used will be overwritten by a new entry.

Relation of AM to rest of appending cycle -- Figure 3 is a flow diagram of the entire appending cyle, including the AM. The center column of Figure 3a shows the three passes of the AM search discussed above, and the read and adjust usage counters (AUC) cycles which occur after an SDW or PTW is found. Figure 3b details the branch logic which follows PASS1 and PASS2. The various paths out of the AM search are labeled with the name of CPU logic lines for reference later. These labels can be ignored now, as can the references to the modified bit in Figure 3b.

On the right side of Figure 3a the rest of the appending cycle appears. In order to understand this portion, assume that PASS1 of the AM search finds no matching ARs (result 1.0 described earlier). This corresponds to the path labeled "QNIA1" leaving the branch after PASS1 in Figure 3 (a and b). For this result the entire appending cycle must be used, and all appending words retrieved from memory.

Beginning at the top of the right column of Figure 3a, then, the descriptor segment base register is checked to determine if the SDW needed is out-of-bounds in the descriptor segment. If an out-of-bounds fault is not generated, it is determined if the descriptor segment is paged, and if so the descriptor segment PTW if read from memory with a RRS-SP memory cycle (1.20μs). If the DSPTW contains a directed fault, the fault logic is entered; otherwise the addressed SDW is read from memory with a RRS-SP memory cycle. Note that the DSPTW was not written into the AM. They are never saved in the AM. The SDW is checked for a directed fault and (if none appears) an AM write cycle followed by an adjust usage counters cycle is initiated to write the SDW into the AM. Concurrently, if the SDW indicates a non-paged segment (left branch), access privilege and bounds are checked against the SDW and if no fault is generated, the final operand, indirect word, or instruction reference is made to memory. If the SDW indicates a paged segment (downward branch), a bounds check is made, and if no fault is generated, the PTW is retrieved from memory with a RRS-SP memory cycle. The PTW is checked for a directed fault, and if none occurs, the used and modified bits in the PTW are updated if necessary. This is done in the following

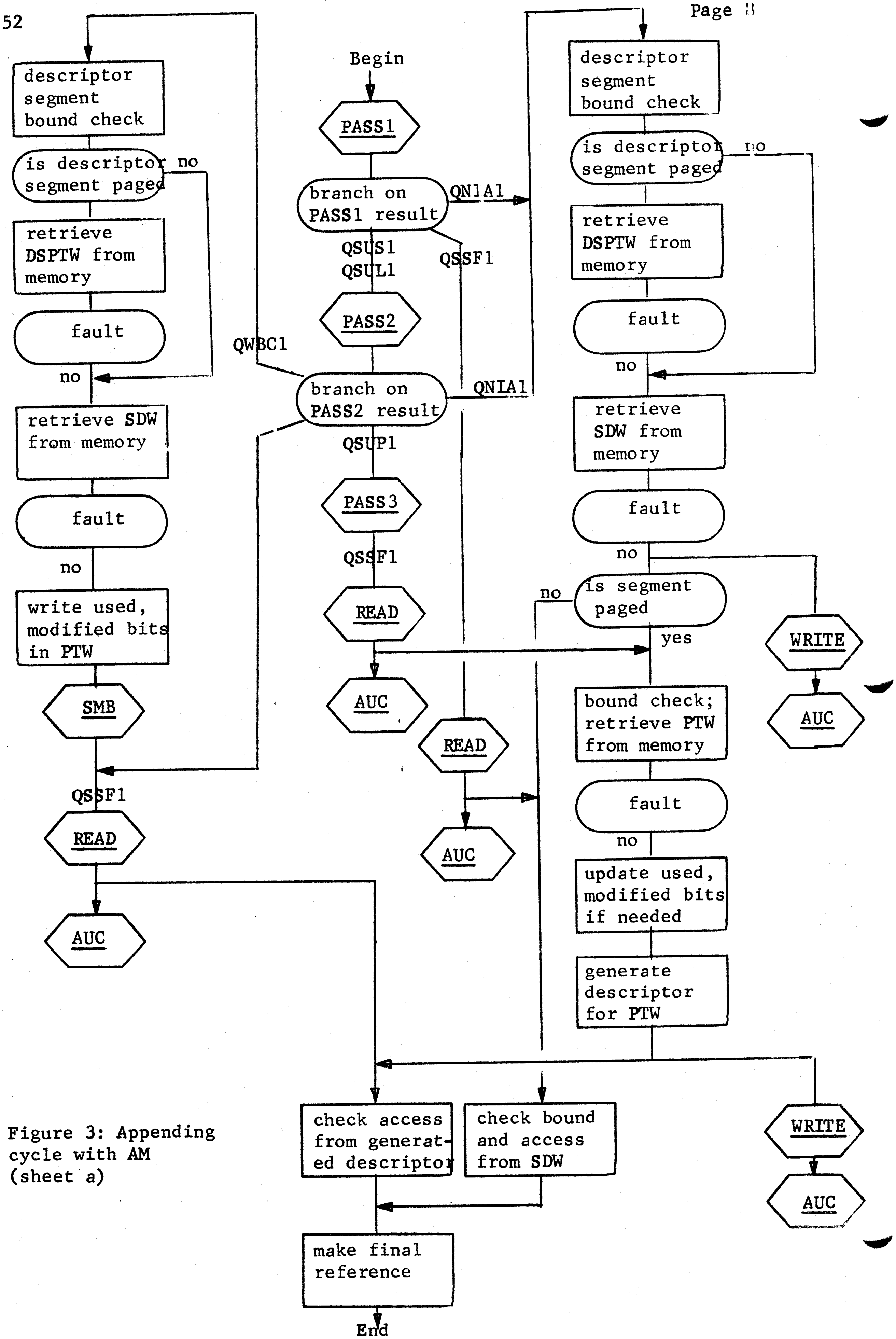

Begin
PASS1
branch on PASS1 result
QNIA1
QSUS1
QSUL1
QSSF1
PASS2
QWBC1
branch on PASS2 result
QNIA1
QSUP1
PASS3
QSSF1
READ
AUC
READ
AUC
descriptor segment bound check
is descriptor segment paged
no
retrieve DSPTW from memory
fault
no
retrieve SDW from memory
fault
no
write used, modified bits in PTW
SMB
QSSF1
READ
AUC
descriptor segment bound check
is descriptor segment paged
no
retrieve DSPTW from memory
fault
no
retrieve SDW from memory
fault
no
is segment paged
no
yes
WRITE
AUC
bound check; retrieve PTW from memory
fault
no
update used, modified bits if needed
generate descriptor for PTW
check access from generated descriptor
check bound and access from SDW
WRITE
AUC
make final reference
End

Figure 3: Appending cycle with AM (sheet a)

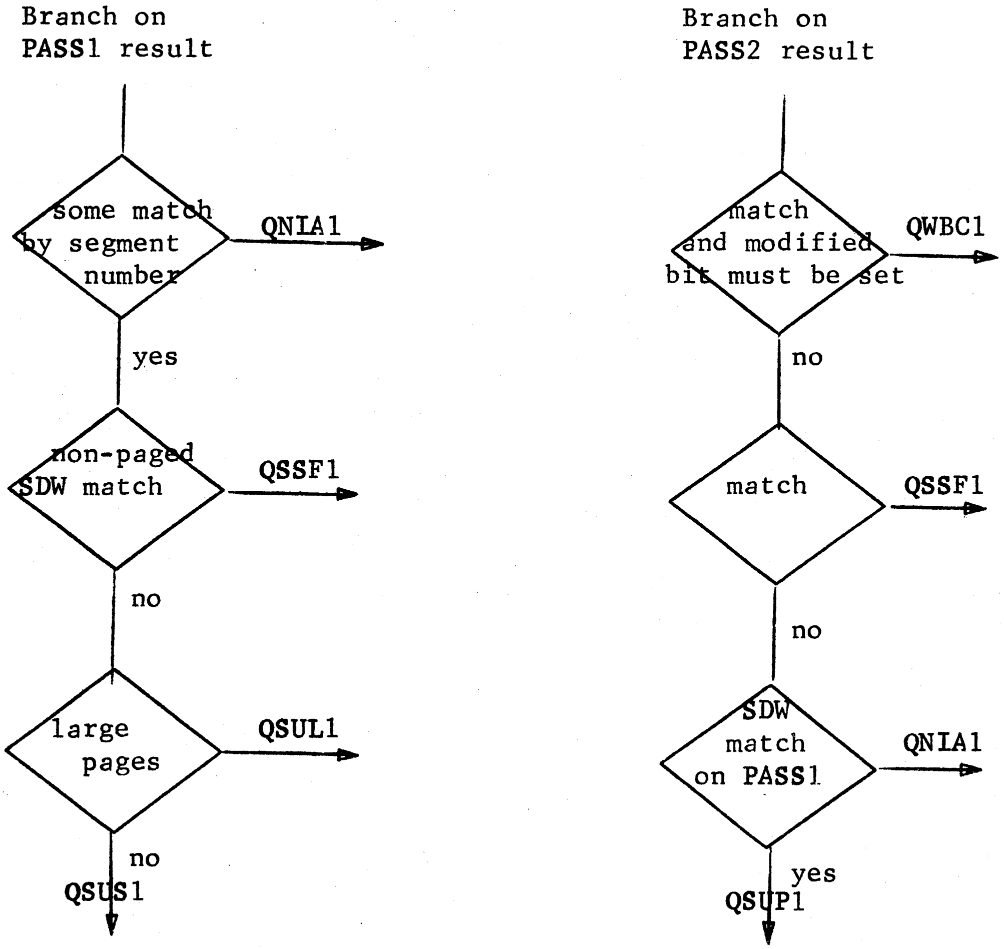


Figure 3: Logic for branching at the conclusion of AM PASS1 and PASS2 (sheet b)

way: the new value for the used bit should be "1". The new value for the modified bit should be "0" unless the old value is "1" or a write or read/alter/rewrite operation is to be performed, in which case it should be "1". If the new values for the used and modified bits are not the same as the old, then the PTW in memory is updated with the new values by a zone write to bits 24-26. Once this is taken care of, a descriptor is generated by comparing the descriptors in the PTW and the previously retrieved SDW, and generating the result implied by Figure 2. (Note: bit 29 in this descriptor will be set to "1" if the modified bit was or became "1" above.) Once this is done, the PTW with the modified descriptor is written into the AM and the usage counters adjusted. Concurrently, access is checked against the generated descriptor, and if no fault results, the final reference is made.

Now relax the assumption that no match occurred on PASS1 of the AM search. As Figure 3a indicates, the various possible AM search results listed earlier cause various subsequences of the full appending cycle to occur (with the exception of the mysterious sequence on the far left of Figure 3a).

Referring to Figure 3b, the result 1.0 path (labeled QNIA1) has just been explored. The result 1.1 path is labeled QSSF1 in the PASS1 branch logic. Recall that in this case a non-paged SDW is found in the AM. Following the path in Figure 3a, the SDW is read from the AM and the usage counters adjusted. The append cycle is entered following this read just at the point where the bounds and access in a non-paged SDW are checked. The SDW from the AM takes the place of the SDW that would have been retrieved from memory. The result 1.2 path is labeled QSUL1 or QSUS1, depending on whether the PASS1 output indicated large or small pages, respectively, and leads to PASS2 of the AM search.

Result 2.0 is labeled both QSSF1 and QWBC1 in the PASS2 branch portion of Figure 3b. The difference is subtle. It has to do with a special circumstance which can occur. If the final operand reference is to be a write or a read/alter/rewrite and if the modified flag (bit 29 of the matched AR) in the PTW found by PASS2 is not yet "1", then both the copy of this flag in the AM, and in the PTW in memory (bit 26 of the PTW)must be set to "1". This case is labeled QWBCl. If this does not occur, then the path labeled QSSFl is followed. The latter reads the PTW from the AM, adjusts the usage counters and enters the appending cycle at the point where the descriptor generated from an SDW and PTW is checked for access permission. The PTW from the AM simply takes the place of the PTW which would have been retrieved from memory. Since it already contains the generated descriptor, no further information is needed.

The QWBCl path proceeds to the left-most column of Figure 3a. This is a special appending cycle to locate the PTW in memory so its modified bit (?6) can be set to "1" also. It is in every way the same as the portion of the normal cycle shown parallel to it in the right column of Figure 3a as far as the sixth step. Here, instead of putting the retrieved SDW in the AM, the SDW is used to make a write reference to the proper PTW in memory. The write is a zone write to bits 24-26 of the PTW which sets the modified bit to "1". A special AM cycle (SMB) is then entered which sets the modified bit in the AR to "1". This path then rejoins the QSSFl path. The case of result 2.1 in which an SDW match did not occur in PASSl is labeled QNIAl, and is handled the same way as result 1.0 (which has the same label). The other case of result 2.1 (an SDW was found on PASSl) is labeled QSUPl and leads to PASS3 of the AM search.

After PASS3, the found SDW is read from the AM, the usage counters adjusted, and the appending cycle entered at the point where the bounds check on a paged SDW is performed. The SDW from the AM takes the place of the SDW that would have been retrieved from memory.

Observation -- This is probably a good point to note several observations about the current 645 CPU AM. Clever advantage has been taken of the shape of SDWs and PTWs to make efficient use of the storage space available.

The address field in both is the same length, and completely overlaps. Thus, it can occupy the same bits of an AR in either case. PTWs do not contain a bound field, and since the bound field in an SDW specifies the length of a segment only in units of a page, it is not necessary to copy the SDW bound into an AR containing a PTW. (The fact that an unfaulted PTW exists guarantees that the words in a page are within the bound for the segment.) Thus, the space in an AR which contains the bound field for an SDW can contain the page number tag for a PTW. (Both fields require the same number of bits.) Both SDWs and PTWs require segment number tags in their ARs, so there is no problem here. All that remains is the descriptor field in SDWs and PTWs. In the 645, the descriptors in PTWS are used in the access check performed before each reference to a word in paged segment. The access check is made against a descriptor that is generated from the descriptor fields in both the SDW and the PTW. It is thus necessary to save a descriptor in an AR regardless of whether an SDW or PTW occupies it. All three descriptors (SDW, PTW, and generated) are the same length. All that is needed to be able to use a PTW found in the AM even when the corresponding SDW is not there is to save the generated descriptor in ARs which contain PTWs rather than just a copy of the PTW descriptor. Finally, the "used" bit from PTWs need not appear in ARs. The fact that a PTW is in the AM implies that the "used" bit is "1".

The point of making these observations now is to allow some of them to be contrasted later with the situation in the 645 follow-on CPU.

Performance of current AM with Multics running

In this section an analysis of the performance of the current 16 register AM is presented. The analysis is based in part on the data presented in MPL-51, and in part on additional data. All measurements were taken while Multics was executing on a hardware configuration that included 1 CPU and 256K of core memory. See MPL-51 for a description of how the data was obtained.

Model for analysis -- The rather detailed description of the operation of the AM presented in the first section of this MPL provides the basis for interpreting the data collected. As a first step in this interpretation a tabular summary of the possible results of an AM search is presented. This will serve as a model for part of the analysis of the data. In Figure 3 the paths corresponding to the various search results were labeled with the names of logic lines in the CPU. A pulse appears on the named line whenever the indicated result occurs. The table in Figure 4 summarizes the correspondence between the possible results and the pulses generated. The columns correspond to the logic lines, and the rows to the possible search results. An X'ed table entry indicates the occurrence of a pulse during a search cycle. As can be seen, a combination of pulses is needed to identify each result.

Pulse counts -- The average number of pulses per second observed on each of these lines during the electronic counter measurements of the 645 CPU are listed below. The counts of QSTA1 and QNIA1 were taken simultaneously. The counts for the other four lines were taken one after another on another day.

QSTA1[1]	414,668/s
QNIA1[1]	5,180/s
QSUL1[2]	194,209/s
QSUS1[3]	1,059/s
QSUP1[1]	826/s
QSSF1[4]	381,088/s

Percentage of occurrence of search results -- The measurement results themselves do not indicate directly how the AM is being used. Of real interest is the percentage of searches which produce each result indicated in Figure 4.

[1] reported in MPL-51

[2] not previously reported, see Figure A1 in the appendix

[3] not previously reported, see Figure A2 in the appendix

[4] not previously reported, see Figure A3 in the appendix

logic lines result

QSTA1	QNIA1	QSUL1	QSUS1	QSUP1	QSSF1	
100	1.25	50.32	.27	.21	98.75	
X	X					no segment number match from PASS1
X	X	X				only incorrect PTWs found on PASS2, no SDW from PASS1, large pages
X	X		X			only incorrect PTWs found on PASS2, no SDW from PASS1, small pages
X					X	non-paged SDW found on PASS1
X		X		X	X	paged SDW found on PASS3, large pages
X			X	X	X	paged SDW found on PASS3, small pages
X		X			X	correct PTW found on PASS2, large pages
X			X		X	correct PTW found on PASS2, small pages
start AM search	no useful match	large pages indicated by PASS1	small pages indicated by PASS1	no PTW match on PASS2, but SDW match found on PASS1	some useful match	

Figure 4: Correspondence between AM search results and pulses generated

While it is apparent that this cannot be derived exactly from the available data (the table has eight rows but only six columns, five of which are linearly independent) a reasonable approximation can be made. The task is further complicated by the fact that the data from the two different days of measurement represented are inconsistent (QSSF1 + QNIA1 = QSTA1 should be true, but 381,088 + 5,180 ≠ 414,668). This inconsistency can be resolved by assuming that the percentage of occurrence of each result was the same for both sets of measurements.

To adjust the measurements, start with QSTA1 and QNIA 1 which were counted simultaneously. QSTA1 - QNIA1 = QSSF1, which means the QSSF1 count for that period was 409, 488 each second. This is 107.45 percent of the measured count from the next day. So multiply the QSUS1, QSUL1, QSUP1, and QSSF1 counts by 1.0745. The adjusted counts and the percentage of total searches per second each represents are:

QSTA1	494,668/s	100.00
QNIA1	5,180/s	1.25
QSUL1	208,678/s	50.32
QSUS1	1,138/s	.27
QSUP1	888/s	.21
QSSF1	409,488/s	98.75

The adjusted percentages are listed at the head of the columns in Figure 4.

After observing that the data confirms the predominance of large pages among paged segments (the only paged segment with small pages in Multics is SCAS), things can be simplified by removing the large page/small page distinction. A new table is produced by merging columns QSUL1 and QSUS1. The percentage of occurrence to associate with the merged column is the sum of the percentages for the component columns because pulses never occur on both lines on the same search cycle. The new table is shown in Figure 5.

The corrected data immediately indicates that the results represented by rows 1 and 2 of Figure 5 together occur on 1.25% of the searches, and by row 4 on .21% of the searches. The result represented by row 3 of Figure 5 must occur on a minimum of 48.16% of the searches, because an upper bound

	logic lines					result
	QSTA1	QNIA1	QSUS1 QSUL1	QSUP1	QSSF1	
	100	1.25	50.59	.21	98.75	
1.25	X	X				no segment number match from PASS1
	X	X	X			only incorrect PTWs found on PASS2, no SDW from PASS1
*48.16 - 49.41	X				X	non-paged SDW found on PASS1
.21	X		X	X	X	paged SDW found on PASS3
*49.13 - 50.38	X		X		X	correct PTW found on PASS2

percentage of occurance

*must add together to give 98.54

Figure 5: Percentage of occurance of AM search results

on percentage of occurrence for row 1 is 1.25%, and row 2, 4, and 5 together total 50.59%. The same result must occur on a maximum of 49.41% of the searches, because a lower bound on percentage of occurrence for row 1 is 0%. Adding to these upper and lower bounds for row 3 the values for row 1 + row 2 and row 4, the result represented by row 5 is seen to occur on between 49.13% and 50.38% of the searches. These bounds are summarized on the left of Figure 5.

What comes out must have gone in -- Two more previously unreported measurements are available. SDWs are written into the AM (using the write cycle discussed in the first section) at an average rate of[1] 6,364/s. The similar figure for PTWs for segments with large pages is[2] 3.158/s. No data is available on PTWs for segments with small pages, but the pulse count for the QSUS1 line discussed earlier suggests that the number of small PTWs written into the AM per second is very low. Observe that these rates do not account for all SDW and PTW references to memory. They include only SDWs and PTWs which are written into the AM. Not counted are DSPTWs (which are never written into the AM), SDWs and PTWs which contain directed faults (which are not written in the AM), and DSPTWs, SDWs, and PTWs referenced in connection with maintaining the used and modified bits in PTWs.

Effect of number of AM registers on performance

In this section the results of several experiments are considered in which the number of ARs in the AM were varied. The full AM contains 16 ARs. An AM with 8 ARs was obtained by wiring the high order bit of the usage counters in 8 of the ARs to permanently contain the value "1". An AM with 4 ARs was obtained by wiring the two high order bits of the usage counters in 12 of the ARs to permanently contain the value "11". An AM with no ARs was obtained by turning off the AM using a front panel switch on the CPU.

[1] see Figure A4 in the appendix

[2] see Figure A5 in the appendix

Figure 6 shows the effect of the AM size changes on the rate of occurrence of four events of interest: memory accesses/second, AM searches/second, instruction executions/second, and AM "not founds"/second (QNIAls). All data for Figure 6 was reported in MPL-51.

It is obvious from Figure 6 that the size of the AM has a direct effect on the instruction execution rate of the CPU. All of the decrease in the instruction exectuion rate which occurs as the size of the AM is decreased is caused by the retrieval from memory of more and more SDWs and PTWs. Figure 7 illustrates this point quite well. The number of virtual memory references (AM searches) per instruction remains constant as the size of the AM is decreased, while the number of core memory references per instruction rises dramatically. Both the ratio of core memory accesses to instruction excutions, and the absolute instruction execution rate would be affected even more if there were not so many non-paged segments in the ring 0 portion of the system. From these measurements it appears that 16 was a good choice for the number of ARs in the AM. While more ARs would certainly cause some further increase in the instruction execution rate, the next readily available increment of 16 ARs does not appear to be a good buy. The first derivative of the instruction execution rate with respect to the number of ARs is fairly close to zero with the current size. An AM with 8 ARs is probably too small. The instruction execution rate drops by 5.6% as the number of ARs decreases from 16 to 8, which is significant. More importantly, with only 8 ARs, the system would be more vunerable to changes which affect the size of the average working set, e.g., a new Call-Save-Returns sequence which touched more pages might push the average working set size just over 8 pages, thus causing an 8 register AM to become too small.

Note, however, that the instruction execution rate curve in Figure 6 is not flat at 16 ARs. This suggests that a little more performance might be "squeezed-out" of the 645 CPU with a few more ARs. As indicated above, the performance improvement expected is not great enough to justify doubling the AM size. Moreover, this would be practically and economically impossible to do on the current machine. There is a simple change that could be made to a 645-CPU, however, which would have the effect of adding a few more ARs.

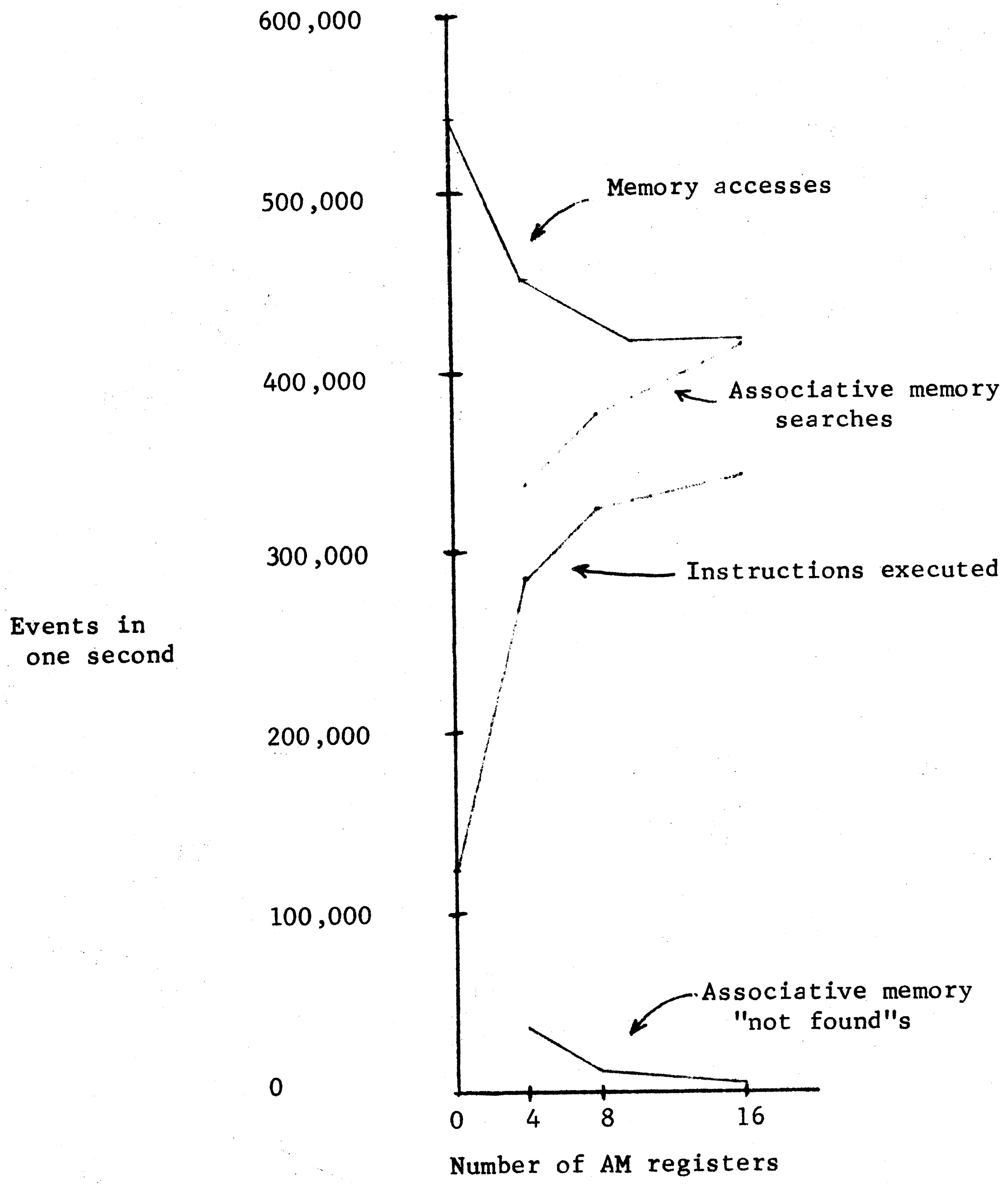


Figure 6: 645 CPU performance with various AM sizes

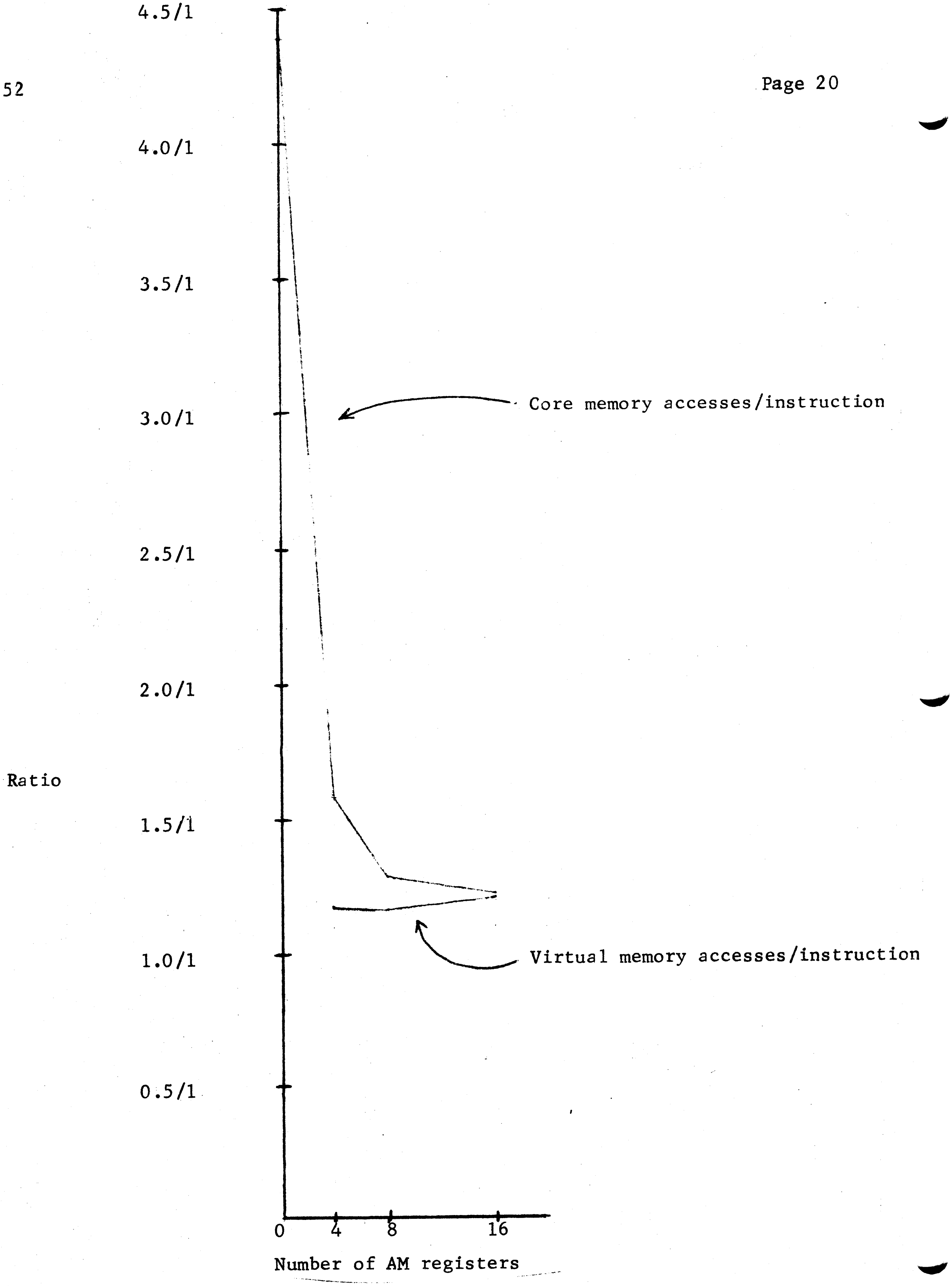


Figure 7: Core and virtual memory accesses per second with various AM sizes

As indicated in Figure 5, only .21% of all AM searches result in a match with a paged SDW, while 98.54% of all AM searchers result in a match with a PTW or a non-paged SDW (which is really a PTW for an arbitrarily sized page).

Clearly, paged SDWs are not very useful. If paged SDWs were not remembered in the AM then all 826 matches per second with paged SDWs (average) would be replaced by memory references for a DSPTW and the SDW, requiring about 826*(1.2μs + 1.2μs) = 1982 μs out of each second, which would slow the instruction execution rate by approximately .2%. This is an absolute upper bound on the performance decrease that could result. On the other hand, not remembering paged SDWs in the AM would free ARs to remember more non-paged SDWs and PTWs, effectively increasing the useful size of the AM. Just how many ARs, on the average, are occupied by paged SDWs is hard to estimate. But currently, for each of the 3,158 PTWs per second written into the AM, the corresponding SDW must be in the AM when the write occurs, and it takes at least 16 more writes to the AM to push the SDW out the bottom. So the change would appear to free some ARs. The net effect should be a small performance increase, especially while programs with large working sets are executing.

The change suggested could be made quite easily. All that is needed is to make the writing of an SDW into the AM by the appending cycle conditional on the paged/not paged bit of the SDW. J. Ammons indicates that this can be done by removing one wire from each of two pins on the AM door of the CPU. No further changes would be necessary. This change might be good for a small performance increase with the current hardware until a follow-on processor is available.

Design of an AM for the 645 follow-on processor

MHDM-5 is a draft of a proposal for a follow-on processor to the 645 CPU. This new CPU would embody all the essential features of the 645 CPU, as well as certain refinements and additions. As the data collected in the counter measurements of the 645 CPU have shown, an AM is a necessary unit of a CPU which performs address appending of the sort required by Multics. Thus, an AM would be an important part of the follow-up processor. In this section two possible designs for that AM are discussed, and the design considered to be most suitable by this author is identified.

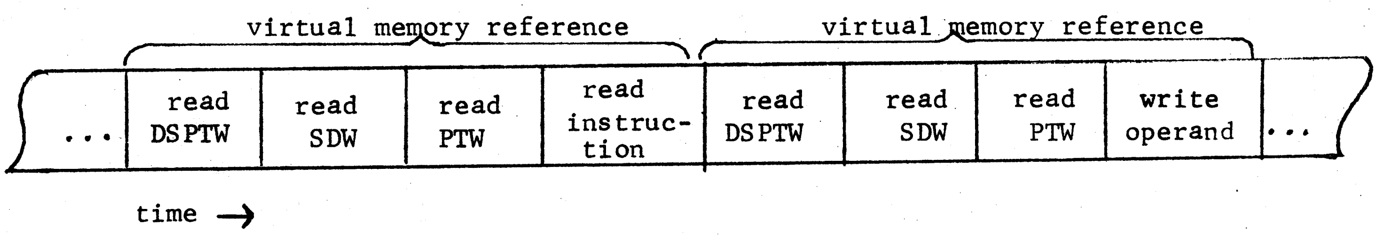
The following discussion assumes familiarity with the proposals of MHDM-5 for a follow-on processor. Several aspects in which the follow-on processor would be different from the 645 CPU affect AM design. These are summarized below:

1. The follow-on processor would recognize only one page size. This size would be adjustable between the limits of 64 words and 4096 words by a field engineer. Two fixed page sizes are recognized by the 645 CPU.
2. A fairly elaborate protection mechanism would be built into the CPU. This would result in 72 bit SDWs with very long description fields. PTWs would still be 36 bits, but contain no descriptors.
3. The bound field of an SDW would specify length in units of 8 words.
4. The address fields in SDWs and PTWs would have different lengths and precisions.
5. Ring crossing would occur without a fault, thus tending to increase the length of time a processor may run without clearing the AM.

The effect of each of these differences on AM design will be brought out later.

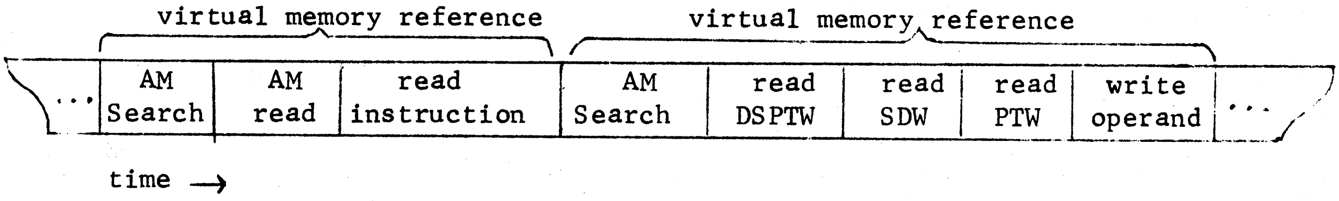
General considerations -- Before considering specific design proposals, several general characteristics that any AM design for the proposed follow-on processor must have are discussed.

Consider the stream of memory references produced by a 645-like CPU executing Multics with the AM disabled. Ignoring the occasional direct references by absolute address which occur, this stream would appear at one grain as a string of virtual memory references. Within each virtual memory reference would appear a DSPTW and SDW retrieval, and perhaps a PTW retrieval, as well as the instruction, indirect word, or operand reference.



A well designed processor will make every effort to minimize the gap between individual core references by buffering and decoding several instructions at once, and performing arithmetic and logical operations in parallel with memory references, for memory references take quite long when compared with the internal speed of a CPU. The amount of parallelism which can occur, however, is constrained by the fact that a reference may depend upon data derived from the preceeding reference, or the result of the preceeding instruction. Thus, sequence must be maintained. A completely densely packed string of core references is the limit on processor speed.

A 645-like associative memory removes some of the appending word accesses in some of the virtual memory references by inserting the AM search cycle before all virtual memory references. The stream of memory references generated by the processor running with the AM in operation would reflect the AM search and read cycles.



The first virtual memory reference above illustrates a match on the AM search. The following AM read cycle replaces 3 appending word references to memory. The second virtual memory reference above illustrates a "not found" on the AM search. The normal appending word references to memory follow.

Not shown above are the AM write and adjust usage counters cycles. These occur in parallel with memory references, and are thus transparent from this point of view. That the write and adjust usage counters cycles can both occur in the time of a single memory reference is all that is necessary. (Because of this, the adjust usage counters cycle of the 645 AM may be overly fast. Instead of using 16 subtractor/comparators and updating all usage counters simultaneously, there may be enough time to use only one subtractor/comparator and update them sequentially. This should be investigated with respect to the AUC cycle in the AM of the follow-on procession.) The AM search and read cycles, however, must largely occur in sequence with the memory references. By greatly increasing the complexity of the internal logic of a CPU, the AM search and read cycles can sometimes be made to occur in parallel with memory references, but it is impossible to always hide it. In the same way that the memory cycle time limited CPU speed, then, the AM search and read cycle times have a limiting effect on CPU speed. It is therefore very important that the AM search and read cycles be as fast as possible.

The times for the three passes of the 645 CPU AM search are 216ns, 234ns, and 216ns, respectively. Multiple passes are required in the case of the 645 because two page sizes are recognized. It is necessary to know the page size used by a segment before the proper page number can be calculated from the given word number. PASS1 determines the page size from which a page number is calculated for PASS2. PASS3 is never entered unless a match is assured, so as long as the time for this pass is shorter than a memory cycle (which it is) it is helping to increase the instruction execution rate.

The follow-on processor would recognize only one page size. Thus, a single pass AM search is possible, and should be required. Again, this search should be as fast as possible.

A second general consideration is the size of the AM for the follow-on processor. It should not be smaller than the AM for the 645 CPU. Several points are relevant. The experiment described in the previous section of this report indicated that cutting the 645 CPU AM from 16 ARs to 8 ARs lowers the instruction execution rate by 5.6%, a significant amount. Moreover, in the proposed follow-on system the ratio of memory cycle time to AM search time may be as high as 5:1 (hopefully), compared to ~ 3:1 for the 645 system. Because of this, the relative penalty of a "not found" search result is greater with the follow-on system. Next, the page size for the follow-on system may be as small as 64 words. With such a small page size (the current 645 page size is 1024 words) more pages will be included in the working set of a process, and thus more AM space required for the PTWs.

Finally, the proposed hardware ring switching mechanism for the follow-on machine would have the effect of increasing the average time between AM clear cycles. It will no longer be necessary to clear the AM each time rings are switched, either as the result of a call, a fault, or an interrupt. This may have the effect of moving to the right the point where the curve of the instruction execution rate with respect to AM size becomes flat.

but environment changes.

Direct extension of 645 CPU AM design -- The most obvious design for the AM of the 645 follow-on is a direct extension of the current AM. This approach is now explored.

Some of the changes that must be made are caused by the different SDW and PTW formats. An SDW for the follow-on system would have the following fields:

- ADDR, 24 bits, contains the full absolute address of the base of the segment.
- BOUND, 15 bits, contains the length of the segment in 8 word blocks.
- DESC, 30 bits, contains access keys, a paged/unpaged flag, and a 15-bit "call limiter" which defines those words of a segment which are entry points.
- FAULT, 3 bits, contains a directed fault flag and code.

A PTW for the follow-on system would have the following fields:

- ADDR, 18 bits, contains the high order 18 bits of the 24-bit absolute page address
- USED, 1 bit, indicates whether or not the page has been used since the last time this flag was reset.
- MODIFIED, 1 bit, indicates whether or not the page has been used since the last time this flag was reset.
- FAULT, 3 bits, contains a directed fault flag and code.

An AR which contains an SDW must have room for all SDW fields with the exception of the 3-bit FAULT field. This totals 69 bits. In addition, a 15-bit segment number tag, an SDW/PTW bit, an Empty/Full bit, an adjust usage counters bit, and a usage counter are required. With a 16 AR AM a 4-bit usage counter is necessary, for a grand total of 91 bits in each AR.

An AR which contains a PTW must contain the 18-bit ADDR field and the MODIFIED bit from the PTW. In addition, all of the 30-bit DESC field from the associated SDW (except perhaps the paged/unpaged bit) and the 15-bit bound field must be included. Finally, a 15-bit segment number tag, a 12-bit page number tag (the page size can be set to as small as 64 words) plus the 7 bits of control fields listed above must be included. The grand total here is 97 bits.

It is not clear, however, that the job could be easily done with the maximum of these two values as the AR length. To do so would require that the field divisions in ARs for SDWs and PTWs be substantially different. This would complicate the control logic of the search, read, and write cycles. Aligning the fields to remove this complication would increase the AR length from 97 bits to approximately 104 bits.

Now consider the required single pass search applied to this AM. Since the page size is known in advance, the search would be for a segment number and page number match on a PTW, a segment number match on a non-paged SDW, or a segment number match on a paged SDW if no PTW match was found. The control logic for performing such a complex search in one pass may be complex and expensive. More importantly, it may be slow. The complication is that several sorts of matches must be looked-for on the same search cycle. Applying the simplification of not remembering paged SDWs in the AM, as was suggested for the 645, would reduce the complexity of the search by eliminating the possibility of multiple matches on the same search cycle instance, but non-paged SDWs would still have to be contended with. Eliminating paged SDWs may not be a good idea, however, for in the follow-on system the page size can be as small as 64 words. Under these circumstances more PTWs are required to describe a given number of words of virtual memory, and paged SDWs might be used often enough to warrent keeping them in the AM.

A second proposal -- Another objection to the above AM design is that it makes inefficient use of the storage capacity of the AM. In the case of the 645, each PTW contains a possibly different descriptor which is consulted on each use of the PTW in the appending cycle. It is therefore necessary that individual descriptors be remembered in ARs which contain PTWs. In the case of the follow-on processor PTWs do not contain descriptors. The descriptor saved in an AR which contains a PTW is a direct copy of the descriptor from the SDW of the containing segment. If more than one PTW for the same segment is in the AM, then each will carry a copy of the same descriptor. This suggests that more efficient use of AM storage capacity could be made by factoring the descriptors out of PTWs for the same segment, something that is not possible for the 645, but would be for the follow-on processor.

This observation leads to an AM design for the follow-on processor which allows factoring and has none of the objectionable characteristics of the direct extension proposal above. The proposal is to provide the follow-on processor with two AMs; one for SDWs and one for PTWs. This design is presented below, and the arguments in its favor are developed.

Figure 8 illustrates a format that contains all necessary fields for an AR in AM_{sdw}. Bits 0 through 68 contain direct copies of all SDW fields except the directed fault flag and its associated code. Since SDWs with directed faults are never put in AM_{sdw}, it is not necessary to save these fields in an AR. Following the SDW information is a segment number tag, an empty/full bit, an adjust usage counters bit (possibly unnecessary), and a usage counter. The length of the latter depends on the size of AM_{sdw}.

A search of AM_{sdw} involves finding the AR whose empty/full bit has the value "1" and whose segment number tag matches the given segment number. Zero or one ARs can match.

Figure 9 illustrates a format that contains all necessary fields for an AR in AM_{ptw}. The address field and the modified flag from a PTW occupy the first 19 bits. Other PTW fields need not be included. Following the PTW information is a page number tag, segment number tag, and the same internal control fields that appear in AM_{sdw}.

A search of AM_{ptw} involves finding the AR whose empty/full bit has the

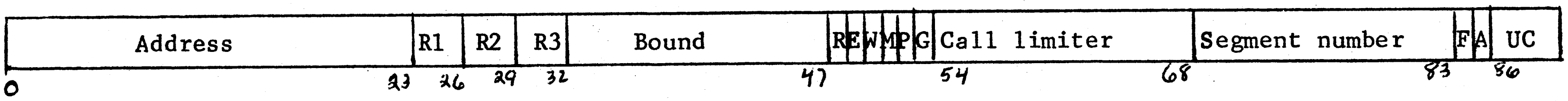


Figure 8: Format for an AR in AM_{sdw}

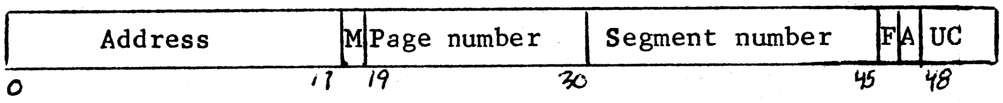


Figure 9: Format for an AR in AM_{ptw}

value "1" and whose segment number and page number tags match the given segment and page number. Zero or one ARs can match.

The read, write and adjust usage counter cycles for the two AMs are the same, respectively, and very similar to those cycles in the 645 AM. The read cycle places the contents of the AR found by a search (if any) on the AR output lines. The write cycle places a new entry into the AR with usage count "0000". The adjust usage counters cycle decrements by one the usage counters of all ARs whose usage counters are greater than the usage counter of the AR just read or written.

Figure 10 illustrates how the two ARs would be used in the appending cycle of the follow-on processor. The appending cycle begins at the upper left with a simultaneous search of both AMs. A match in either or both of AM_{sdw} and AM_{ptw} is followed by read and adjust usage counter cycles in either or both, respectively. If AM_{sdw} finds no match, the appending cycle is entered (top of right column) and the needed SDW retrieved if no faults or out-of-bounds conditions are encountered. If the SDW is unfaulted, then it is written into AM_{sdw} and the usage counters for AM_{sdw} are adjusted. Control is now at the point where the match and no match paths from the AM_{sdw} search meet. The SDW obtained by either means is checked for an out-of-bounds reference. If a fault does not result, it is determined whether or not the AM_{ptw} search was successful. If so (left branch), the modified flag in the PTW from AM_{ptw} is checked to see if it must be set to "1". This will be the case if it is currently "0" and the operation about to be performed includes a write. Setting can be easily accomplished by a zone write to the PTW in memory (the address field from the associated SDW is always available at this point) and a set modified bit cycle to AM_{ptw}. Once this is done (if it is necessary), access is checked against the SDW descriptor, and the final reference made if allowed.

If the AM_{ptw} search failed (downward branch), then the SDW is checked to determine if the segment is paged. If not (left branch), access is checked and the final reference made. If it is paged (downward branch),

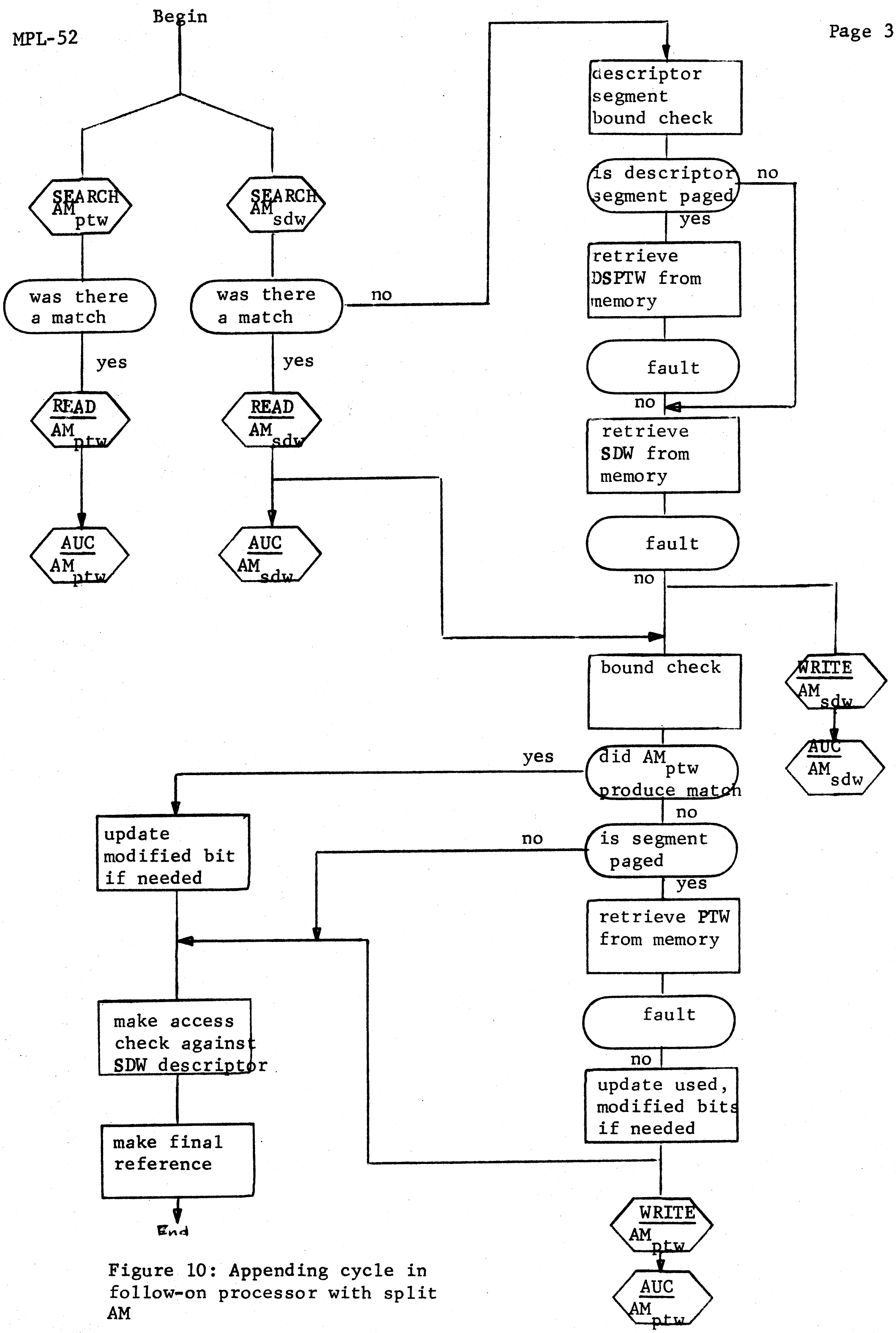


Figure 10: Appending cycle in follow-on processor with split AM

amount of logic may
exceed amount of storage

the PTW needed is retrieved from memory. If unfaulted, the used and modified bits are inspected. If either must be changed, this is done by a zone write to memory. The modified bit in the copy of the PTW to be saved in AM_{ptw} is set to the updated version, and the AM_{ptw} write and adjust usage counter cycles initiated. The final access check and reference are then performed.

Functionally, AM_{sdw} and AM_{ptw} are almost identical. They are similar enough that both may be able to use identical control circuits. Since individually they are less complex than the AM derived by direct extension of the 645 AM above, this split AM design is simpler than the direct extension proposal. The appending cycle itself is also simpler, particularly with respect to updating the modified bit in memory in the case where the PTW has been obtained from the AM (see Figure 3). The design also makes efficient use of AM space by factoring descriptors out of PTWs as discussed earlier. For example, a 16 register AM of the first design (104 bit ARs) would require a total of 1624 bits of register. It would have the capacity to remember 16 PTWs and/or SDWs. The combination of a 16 register AM_{ptw} and an 8 register AM_{sdw} would require a total of 1552 bits of registers. It would have the capacity to remember 16 PTWs and 8 SDWs. Finally, paged SDWs are saved without increasing the complexity of the one pass search. This may provide a significant performance increase if page sizes smaller than 1024 are eventually used in the follow-on system.

Close examination of Figure 10 reveals that the circumstance can arise that a PTW is found in AM_{ptw}, but the corresponding SDW is <u>not</u> found in AM_{sdw}. Because the PTW lacks the required descriptor the SDW must be retrieved from memory in this case. This is not a serious shortcoming of the split AM design. First of all, because the usage counters of both AMs are kept up to date with each use, the order of SDWs in AM_{sdw} will always reflect the order of PTWs in AM_{ptw}. If only paged segments are represented by entries in the two AMs then it is impossible for the circumstance to occur. The trouble comes when non-paged SDWs are written into AM_{sdw}. These will overwrite paged SDWs without pushing the PTWs out of AM_{ptw}, leading to a situation where the circumstance can occur. Consider the behavior of a split AM with an n register AM_{sdw} and an n register AM_{ptw}, and a 645-like AM with n registers in this situation. The non-paged SDWs push out the entries that allow access to the same pages in both cases. Thus, where the split AM may subsequently find a PTW but not an SDW, the 645-like AM will find no entry at all. More memory references result from the latter. If AM_{sdw} is smaller than the 645-like AM, then the circumstances could occur in the split AM alone, but this would not happen frequently enough to be a problem.

This brings up the question of the size of AM_{sdw}/AM_{ptw}. 16/8 would probably be sufficient. But 16/16 has the advantage of making the two AMs closer to identical (identical control logic circuit modules could perhaps be used for both), as well as guaranteeing that the split AM would always work as well or better than a 16 register AM of the 645 kind. (In the case that the AM contained appending words for 16 different segments, the 16/8 split AM would have a disadvantage with respect to the 16 register 645 like AM, but the 16/16 split AM would not. The 16/16 size also insulates the system against changes brought about by smaller pages and longer periods beween AM clears.

It is this author's recommendation that a 16/16 split AM design be incorporated in the follow-on processor.

An acknowledgement

To John Ammons for his considerable part in gathering the data on which this report is based, and his comments and suggestions on its contents.

1 CPU, 256K Memory
6/10/70 35 users

Number of QSUL1 pulses observed in 10 seconds

1,921,378
1,825,322
2,043,797
2,464,247
2,661,798
2,347,921
2,297,742
1,600,369
2,358,562
1,771,143
2,694,500
1,650,913
1,627,625
1,599,409
1,739,144
1,199,164
1,807,508
2,126,573
1,755,372
2,565,405
1,804,127
2,242,706
2,131,598
1,401,512
1,405,891
1,395,848
1,996,850

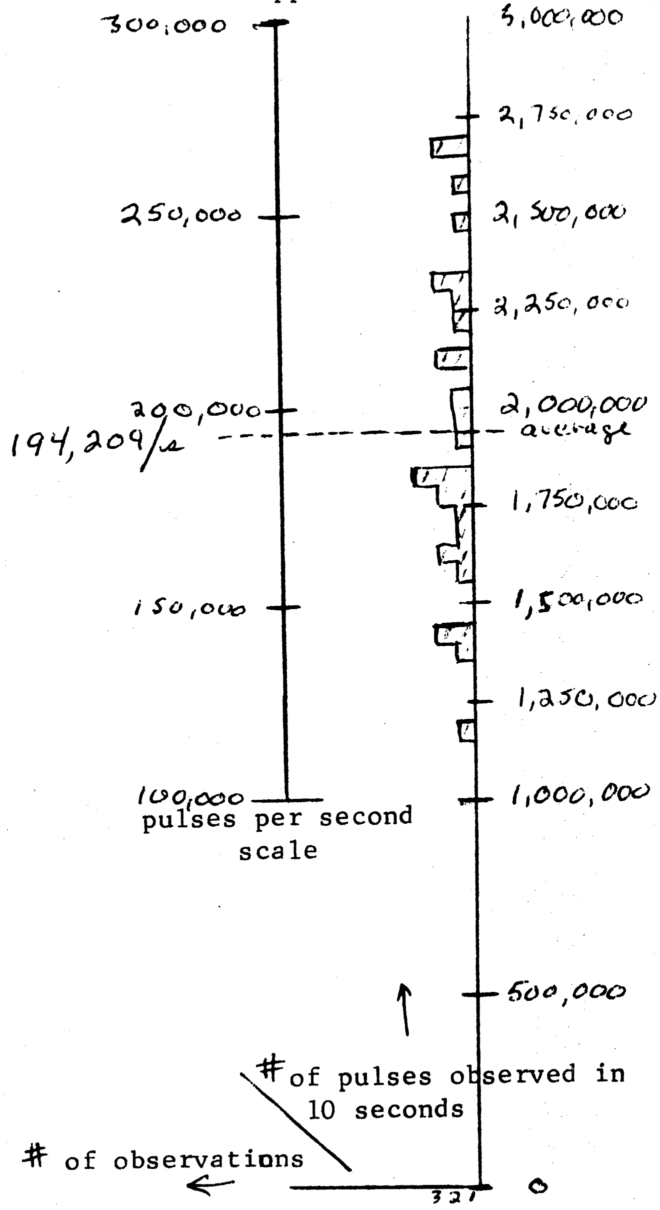


Figure A1 : QSUL1 pulses from a 16 Register Associative Memory

1 CPU, 256K Memory
6/10/70 35 users

Number of QSUS1 pulses
observed in 10 seconds

2,993
8,298
5,954
5,723
7,665
6,994
8,682
14,134
10,981
15,961
14,552
13,320
15,460
9,156
10,566
12,261
15,499
13,877
13,036
13,075
12,210
10,231
8,604
9,443
9,655
9,963
7,669

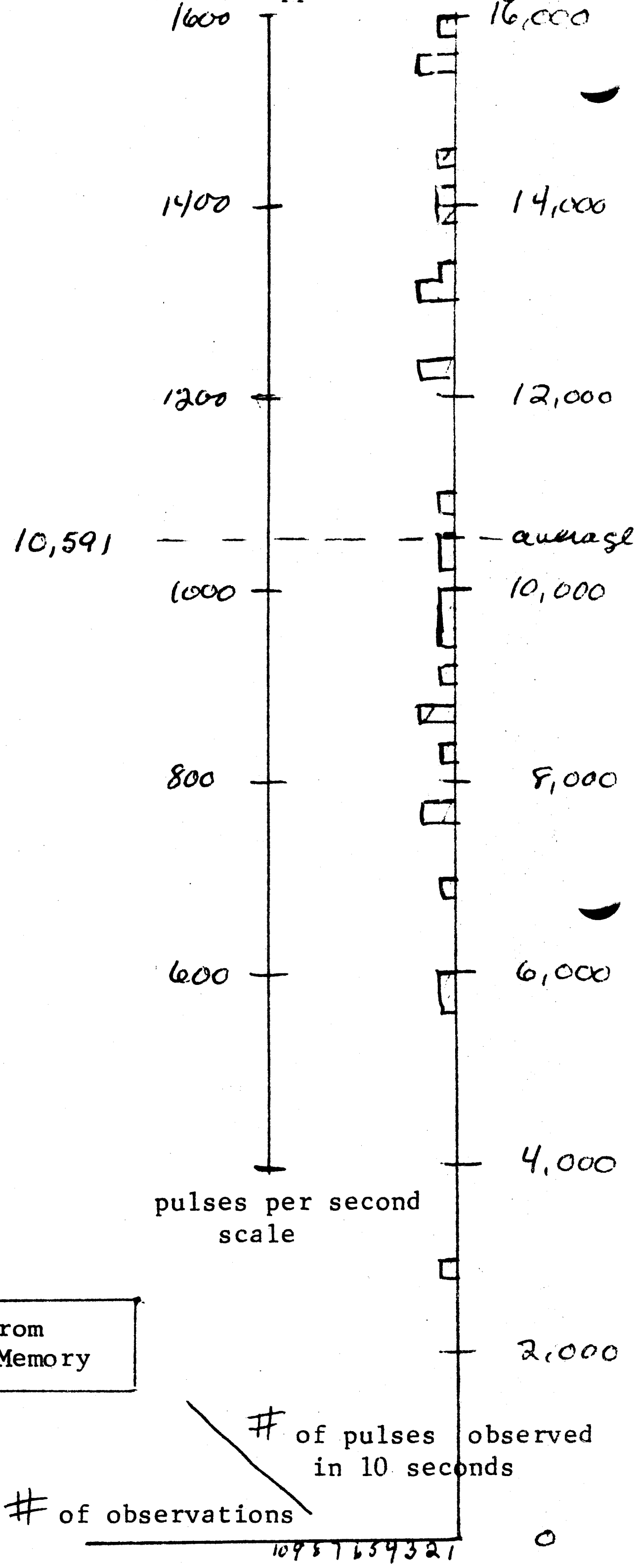


Figure A2 : QSUS1 pulses from
a 16 Register Associative Memory

1 ÇPU, 256K Memory
6/10/70 35 users

Number of QSSF1 pulses
observed in 10 seconds

3,880,019
3,863,283
3,753,168
4,221,004
3,804,773
3,886,770
3,902,891
3,713,319
3,828,071
3,835,681
3,856,371
3,711,402
3,907,148
3,973,497
3,774,415
3,803,441
3,789,912
3,865,503
3,907,352
3,762,611
3,782,990
3,846,606
3,844,223
3,784,223
3,828,801
3,955,427

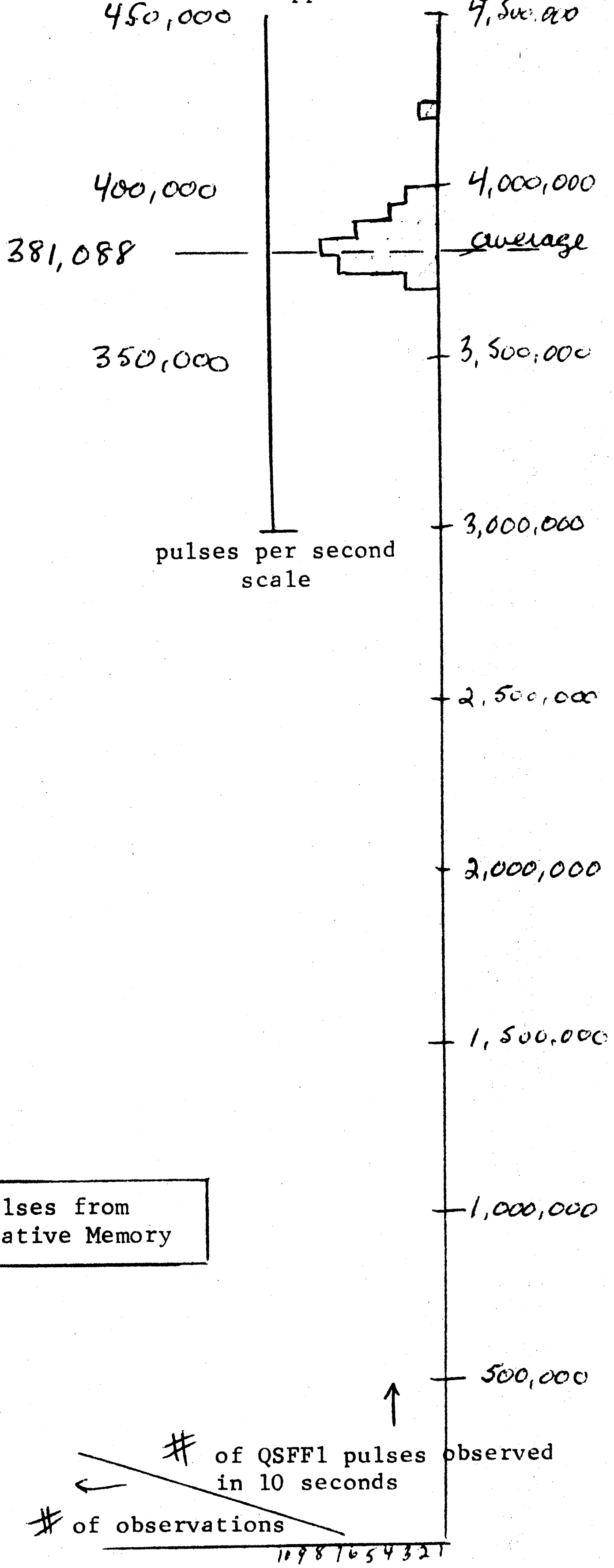


Figure A3 : QSSF1 pulses from
a 16 Register Associative Memory

1 CPU, 256K Memory
6/10/70 35 users

Number of SDWs written
into the AM in 10 seconds

35,249
55,238
45,200
56,992
58,380
58,173
44,057
71,343
67,699
56,487
58,993
85,897
56,373
61,845
80,200
80,589
86,994
77,126
66,086
67,691
68,967
54,072
54,268
75,985
53,097
77,541

counter probe on CPU
line BSDW

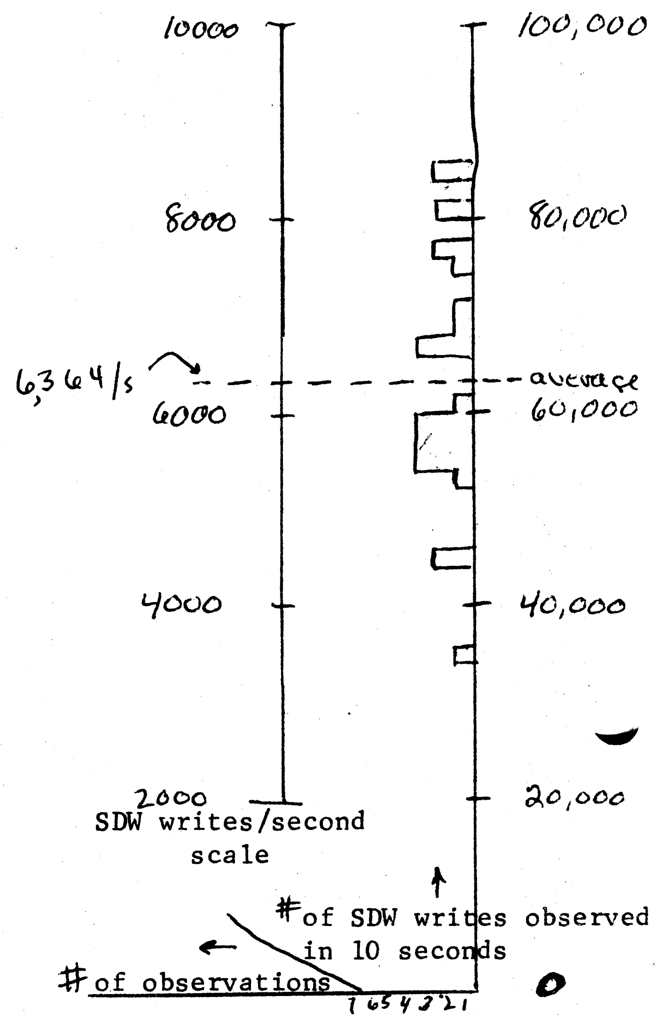


Figure A4 : SDWs written into
a 16 Register Associative Memory

1 CPU, 256K Memory
6/10/70 35 users

Number of PTWs for large pages written into the AM in 10 seconds

39,620	23,957
38,309	39,922
33,435	30,581
29,785	26,771
30,644	40,180
38,191	45,419
18,883	34,826
34,222	21,484
25,678	27,849
21,951	35,114
13,746	28,496
26,728	32,873
38,173	27,368
40,401	42,096
21,092	34,917
35,817	29,552
41,490	28,163
30,239	30,285
26,325	31,208
35,561	32,144
36,963	28,032
45,884	29,099
26,574	31,188
28,212	24,034
41,689	30,861
30,844	28,052
28,550	31,539

counter probe on CPU line BPTL

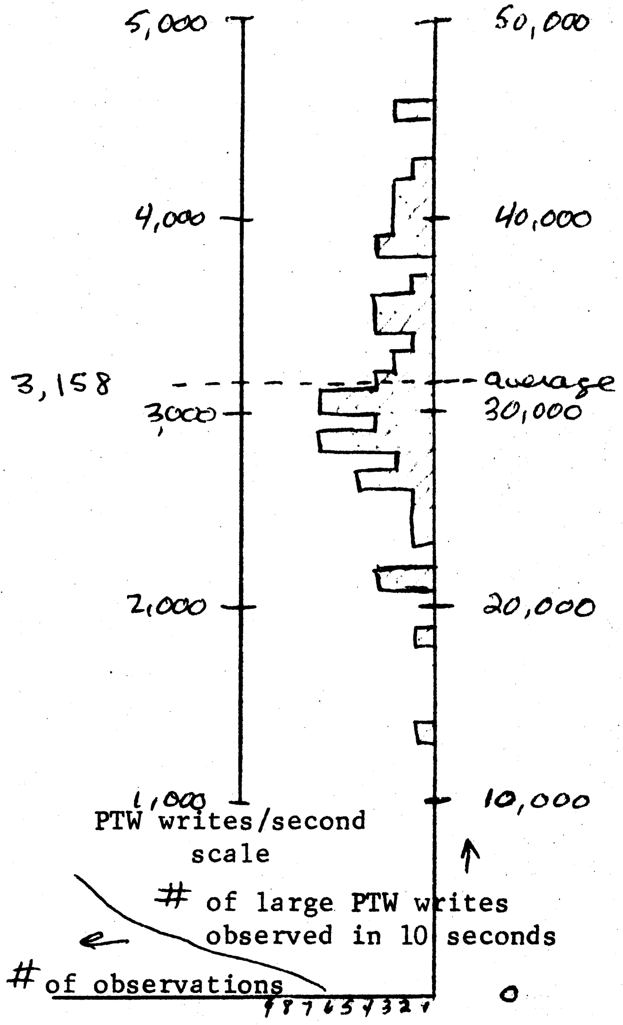


Figure A5 : PTWs for large pages written into a 16 Register Associative Memory