Analysis and Discussion of

TXF Implementation

"Fish" vs. "BobClean"

transact.h:

In this file I simply moved the TXF structs out of hstructs.h where Bob originally had them and into its own separate header file, and #defined some needed constants (e.g. ZCACHE_LINE_SIZE, etc). This header is used mostly by transact.c.

Bob had redefined the existing CACHE_LINE_SIZE constant in hmalloc.h, which is *wrong!*

The existing CACHE_LINE_SIZE constant in hmalloc.h defines the Hercules ***host*** cache line size, e.g. the cache line size of the Intel host processor that Hercules is running on, *not* the cache line size of the z/Architecture mainframe that Hercules is emulating! Thus the need for a separate #define.

Another thing I changed was the TDB struct. Bob originally defined the fields in the TDB using U16, U32, U64, which is also wrong. The TDB is a z/Architecture structure, not a Hercules structure. Thus, it needs to use HWORD, FWORD and DBLWRD, etc, and needs to be populated using the STORE_HW, STORE_FW and STORE_DW macros. You can see me doing this in the transact.c source file around line 800.

Another minor change was the addition of a U64 logicaladdr; /* original guest logical address */ field in the TPAGEMAP struct. This value is set to the vaddr argument passed to the txf_maddr_l function, eventually making its way into the TDB tdbconfict field via regs->txf_conflict whenever a conflict is detected.

channel.c:

I inserted the missing TXF hooks needed for conflict detection between the channel subsystem and TXF via the insertion of TXF_FETCHREF and TXF_STOREREF macros in key places (e.g. in the CCW, IDAW and MIDAW fetch routines as well as the all-important copy_iobuf function that copies data from the channel (device) into Hercules mainstor and vice-versa.

cpu.c:

Added the missing PGM_SPECIFICATION_EXCEPTION case to the TXF program interrupt

handling logic.

Fixed the restart interrupt abort code from ABORT_CODE_IO (wrong!) to ABORT_CODE_MISC (right), and added the missing statement to set the condition code to TXF_CC_TRANSIENT (cc 2).

Replaced #ifdef guarded code with less verbose ALLOC_TXFMAP and FREE_TXFMAP macros instead. (code itself lives in transact.c)

Removed the clearing of the low-order bits of CR(2) from the cpu_init function since it doesn't belong there. The only place it should be is in the initial_cpu_reset function in ipl.c (which is where he already had it).

Added support for having two separate instruction execution loops -- fastloop and slowloop -- depending on whether any TXF transaction was active or not on the CPU in question.

dat.h:

Moved all of the TXF code out of the "maddr_l" function (where Bob originally had it) and into its own separate function in transact.c instead. (The maddr_l function is static inline and should be as short and sweet as possible! Not some huge function!)

featall.h:

Added code for new OPTION_TXF_SLOWLOOP #define build option that enables the two previously mentioned separate instruction execution loops (fastloop and slowloop) in the run_cpu function in cpu.c.

Hercules_VS2008.vcproj, etc, and Makefile.am:

Added new file transact.h

hexterns.h:

Defined externs for some transact.c functions and fixed an #ifdef for FEATURE_ECPSVM (should be _FEATURE_ECPSVM (underscore) instead).

hinlines.h:

#defined the UPDATE_SYSBLK_TRANSCPUS macro to atomically update the "sysblk.txf_transcpus" counter. It's *critical* that this value be updated atomically since it's relied on by the TXF conflict checking logic.

(Refer to the "txf_maddr_l" function in transact.c where the very first statement is checking the sysblk.txf_transcpus value and exiting immediately (thereby bypassing all TXF conflict checking logic) if it's zero, meaning no transactions are in progress on any CPU in the system)

hstructs.h:

Moved the NTRANTBL, TPAGEMAP and TDB structs out of hstructs and into a separate transact.h header file instead, and rearranged/grouped the txf_ fields more sensibly. Also redfined the txf_transcpus variable in SYSBLK as a signed "int" (instead of as an unsigned U32 like Bob had it) and added explanatory comments as to why.

inline.h:

Moved the abort_transaction function out of here and into the transact.c source file where it belongs.

version.c:

Added "With" or "Without" "Transactional-Execution Facility support" to the Hercules startup version messages.

opcode.h:

Moved the #defines for the _PSW_IA, PSW_IA, SET_PSW_IA, UPD_PSW_IA, etc. macros from the build architecture INdependent section down to the build architecture DEPENDENT section of opcode.h, as explained in my GitHub comment on the subject.

Added OPTION_TXF_SLOWLOOP support to the EXECUTE_INSTRUCTION et al. macros to speed-up(?) the cpu.c "run_cpu" function for build architectures for which TXF does not apply (s370 and s390) as well as for z/Arch when no transaction is active.

The basic idea is the CHECK_TXFCTR macro logic before each instruction only needs to be done when a transaction is active but not otherwise. This greatly reduces the code that gets generated for the EXECUTE_INSTRUCTION macro for when there isn't a transaction active thereby (hopefully!) speeding things up a tad.

(And as I said, the code that gets generated for s370 and s390 builds of EXECUTE_INSTRUCTION etc. have *absolutely no TXF logic* in them whatsoever, so they both *should* continue run at the same speed as before; TXF support should have *zero* impact on either one.)

transact.c:

Added Copyright for Bob Wood (You deserve credit for your hard work!).

"**ETND: extract_transaction_nesting_depth**":

Removed the silly "if" for (_GEN_ARCH != 900). It's not needed. The instruction doesn't even exist for any build architecture other than z900 (z/Arch) and all of this is handled automatically by opcode.c's "GENx..." statements for each TXF instruction, e.g. "/*B2EC*/ "___x___x900 (extract_transaction_nesting_depth, RRE, "ETND")".

Similarly, the check for the facility bit is not needed either (and besides, you're doing it wrong anyway!), as this is also handled automatically by our facility design. If the facility is not enabled the instruction automatically program-checks. This is handled by the "instr73" function in facility.c. With our current facility design no more "FACILITY_ENABLED" macros (which is what you should have used) are needed for any instruction that only exists for a given facility. The facility.c code (via the FT2 table) handles all of this automatically.

The regs "gr" fields should never be accessed directly either. Instead, use one of our #defined "GR" macros instead (e.g. "GR_L(n)").

"**TEND: transaction_end**":

Unneeded GEN_ARCH != 900 and facility bit tests removed.

Changed use of hregs to just regs instead, as explained in my GitHub comments.

Fixed a PSW condition code bug. Instruction sets CC2 and otherwise does nothing if there's no transaction to end.

Added code to set the txf_conflict value to the saved pmap->logicaladdr if a conflict is detected (to make its way into the TDB tdbconfict field).

Moved the decrementing of UPDATE_SYSBLK_TRANSCPUS to the very end of the function where a successful transaction actually ends instead of at the beginning where Bob originally had it (which is wrong; if a conflict is detected and the transaction aborts, "abort_transaction" is called and UPDATE_SYSBLK_TRANSCPUS would end up being

called twice!).

"**TABORT: transaction_abort**":

Removed unneeded GEN_ARCH != 900 and facility bit tests.

Added missing PGM_SPECIFICATION_EXCEPTION if effective_addr2 <= 255 and PGM_SPECIAL_OPERATION_EXCEPTION if TXF is not enabled (CR0_TXC bit is zero).

Added missing statement to properly set the condition code in the Abort PSW based on operand2 bit 63.

"**NTSTG: nontransactional_store**":

Again, unneeded GEN_ARCH != 900 and facility bit tests removed.

Changed use of hregs to just regs.

Added missing DW_CHECK.

Added PROGRAMMING NOTE to explain why we're using "vstorec" to update guest storage instead of using STORE_DW.

"**TBEGIN: transaction_begin**":

Unneeded GEN_ARCH != 900 and facility bit tests removed!

Changed use of hregs to just regs.

Added missing DW_CHECK if b1 non-zero and ignoring TDBA if zero.

Added missing TRAN_EXECUTE_INSTR_CHECK (PGM_EXECUTE_EXCEPTION if instruction is executed).

"**TBEGINC: transaction_begin_constrained**":

GEN_ARCH != 900 and facility bit.

regs, not hregs.

TRAN_EXECUTE_INSTR_CHECK.

Added missing PGM_SPECIFICATION_EXCEPTION if b1 non-zero.

Ignore i2 TXF_CTL_FLOAT and TXF_CTL_PIFC bits.

"**process_tbegin**": (called by both TBEGIN and TBEGINC)

regs, not hregs.

Got rid of the i2union crap for regs->txf_gprmask and regs->txf_ctlflag.

Fixed code to *properly* honor CR2 TDC flags based on CR2 Transaction Diagnostic Scope (TDS) bit and PSW Problem State bits.

"**abort_transaction**":

Added test to make sure function is never called if no transaction is active.

Changed use of hregs to just regs.

Save the regs->bear breaking event address register to be later moved into the TDB "tdbbreakeventaddr". I think Bob might have gotten a little confused regarding the BEAR. He was saving the PSW instruction address (IA) into a local variable he called "breakaddr" and then storing that value (breakaddr) into the TDB's "tdbinstaddr" field (which is the PSW instruction address where the abort occurred) but was otherwise not filling in the TDB "tdbbreakaddr" at all. Both values are now properly saved and used to fill in the TDB.

PLEASE NOTE my comments regarding the TDB address too. I think my code might still be wrong as I believe we *may* need to support the ability to fill in *two separate* TDBs for the unfiltered program check case, like Bob was doing. (This may be a case where Bob's was right and my code is now wrong.)

The manual mentions at the bottom of page 5-93: *"The CPU places status into one **or** two TDBs, as follows:"* (with emphasis on the word "or"), and then goes on to state *"...the TBEGIN-specified TDB is **always** stored on a transaction abort."* (with emphsis on "always"). A few paragraphs later where it talks about the Program-Interruption TDB it states that the low-core TDB at 0x1800 is stored *"...when a transaction is aborted due to a program interruption."* This leads me to believe that *POSSIBLY* **both** TDBs should be filled in when a program interrupt occurs, not just one or the other like I'm currently doing. I might need to fix this.

Another thing I need to fix is that, according to the footnote 1 in Figure 5-13 on page 5-94, *some* TDB fields are stored ONLY for TBEGIN-specified TDBs, and then *only* under certain conditions. Otherwise they are reserved (which I'm presuming means they shouldn't be touched). I'm not currently doing that (and neither was Bob either), so that probably needs to be fixed too.

I did fix the switch(regs->psw.asc) case values however. Bob's code was *very* wrong in that regard. This is another example of why #define constant values should be used and not hard-coded values like Bob was using.

I also question Bob's bypassing the populating of the TBEGIN-specified TDB unless the tdb->tdbformat field has 0x01 bit on. First of all, the tdbformat field is described as a numeric value, not as a bit field, so testing for the 0x01 bit is wrong. Secondly, the manual saying *nothing* about the format field needing to be set by the user! Rather, I believe the format field is set by the *system* (i.e. the TXF hardware microcode) when it populates the TDB. Thus I removed that test from my code and am always setting the format to numeric '1' instead.

The tdb->tdbflags now have both the TDB_CTI and TDB_CTV flags set appropriately. Bob for some reason was always setting it to 0x00.

The STORE_DW, etc, macros are now used to fill in the TDB, not using simple "=" assignment like Bob was doing. This is of course to ensure Hercules host big-endian/little-endian compatibility. It appears Bob was always presuming a little endian (Intel) Hercules host, which of course is wrong.

The setting of the PSW condition code is now fixed: for abort code 255 the CC is set to TXF_CC_TRANSIENT or to TXF_CC_INDETERMINATE for abort codes >=256.

"**alloc_txfmap**", "**free_txfmap**":

These are simply the same code Bob had hard coded in the "run_cpu" and "cpu_uninit" functions in cpu.c which are now called via the simple ALLOC_TXFMAP and FREE_TXFMAP macros instead.

"**txf_conflict_chk**":

Called by "txf_maddr_l" function (which itself is called via the dat.h "maddr_l" DAT function). The code was originally in the "txf_maddr_l" function itself, but I broke it out into a separate function instead so I could check both hostregs (sysblk.regs[i]) ***and*** guestregs for a conflict for SIE compatibility (as explained in my GitHub comment). This is also where I save the address of the conflict in regs->txf_conflict to eventually be placed into the TDB "tdbconfict" field as already explained.

"**txf_maddr_l**":

This is simply the TXF code that Bob originally had coded in the dat.h "maddr_l" DAT function (invoked via the MADDR/MADDRL macros), which dat.h's "maddr_l" function

now simply calls as a normal function call. I did this because I wanted to keep the dat.h "maddr_l" DAT function short and sweet since it's declared static inline.

I'm not sure I like treating arn == USE_REAL_ADDR as being interpretted as an instruction fetch call. Isn't USE_REAL_ADDR used for fetching and storing from real storage as opposed to dat-enabled virtual storage? This looks like a bug to me but I didn't change it.

Again, REGS use was changed to use just "regs", not "hregs".

In the conflict checking logic where we're "for" looping through each CPU (sysblk.regs [i]), I'm now calling the previously mentioned "txf_conflict_chk" function for both hostregs *and* guestregs for SIE compatibility.

The logic for mapping and saving of the real page being accessed was not changed, nor was the refreshing of the cache lines.

So that's it I guess!

If anything is unclear or if I've missed anything or you have any questions, let me know!

Thanks for your hard work Bob! MUCH appreciated! :)