Introduction to the z/Architecture Transactional-Execution Facility

Richard Cebula
riccebu@uk.ibm.com

Introduction

Introduced on the IBM System z EC12, the Transactional-Execution Facility is a suite of hardware instructions which allow multiple updates to be made to storage which either appear as a single operation or do not appear to occur as observed by programs. A series of instructions can be grouped together into a transaction and any instructions which are used to update memory will be committed to memory at the transaction's end.

Why is the Transactional-Execution Facility Important?
The toughest challenge for programmers in a multi-processor system is the issue of concurrency. The programmer is completely unaware of the order of execution between two threads and this can only be determined whilst the threads are running by the processor. If two threads are operating on the same data or are waiting for events to occur, it is important to serialise the operations of the threads so that the program's logic is able to guarantee the state of the threads. This is normally performed by using software-based locking methods which involve the operating system. As a result, a program that uses software-based locking can be much slower than one that relies on hardware locks. This also means that the program is dependent on the services and APIs provided by the operating system, possibly reducing the portability of the program.
The z/Architecture also provides a number of hardware-locking instructions such as COMPARE AND SWAP (which can be used to attempt an atomic update to memory) and PERFORM LOCKED OPERATION (which is an incredibly complex instruction).

What is the Transactional-Execution Facility used for?
The Transactional-Execution Facility can be used to provide:

- Atomicity – Transactions are either committed or aborted. Committed transactions will store all their updates to memory at transaction end. Aborted transactions will have no updates to memory stored at transaction end (except for non-transactional stores)
- Performance – Using the Transactional-Execution Facility can be much quicker than using software-based locking methods. If a program is permitted to use the Transactional-Execution Facility, then no operating system interaction is needed in order to have "hardware-style locking" governing a series of updates to memory.
- Reducing software complexity – series of COMPARE AND SWAP-style instructions can be replaced by code encapsulated inside a transaction.
- Improved diagnostics – The TDB (Transaction Diagnostics Block) can be examined in order to determine why a transaction was aborted and the updates to memory failed.

What is provided by the Transactional-Execution Facility?
The Transactional-Execution Facility provides the following:

- Three special-purpose controls – the Transaction Abort PSW, the Transaction Diagnostic Block address and the Transaction Nesting Depth.
- Five control register bits – these are used generally by the control program to govern what happens during an interrupt, what happens when transactions are aborted and if the Transactional-Execution Facility is available for programs to use.
- Six general purpose instructions:
  - `TBEGIN` – Start a non-constrained transaction
  - `TBEGINC` – Start a constrained transaction
  - `TABORT` – Abort a transaction
  - `TEND` – End a transaction

- `NTSTG` – Non-Transactional Store
- `ETND` – Extract Transaction Nesting Depth

Prerequisites for using the Transactional-Execution Facility
In order to use the EC12's transactional memory facility, the machine must be equipped with the Transactional-Execution facility installed in hardware. Facility indicator bit 73 indicates whether the Transactional-Execution Facility is available. If both bits 73 and 50 are on, then the Constrained Transactional-Execution Facility is installed.
In order to use the instructions provided by the Transactional-Execution Facility, the PTFs for APAR PM49761 (or RPM asma90-1.6.0-18.rpm or higher) must be applied to HLASM.

Assembling a program that uses the Transactional-Execution Facility
Assembly of a program that uses the Transactional-Execution Facility is the same as that for any program. However, HLASM provides the listing exit ASMAXTXP which can be used to check whether the program being assembled violates any of the constrained transaction rules. Please note that this is an *assembly time* check and not a *runtime* check and therefore is limited to checking code as if it was to be run sequentially.
ASMAXTXP will check code for:

- Use of a zero base register for constrained transactions
- Allow AR modification control being 0 for constrained transactions
- Whether or not a transaction exceeds its instruction and/or byte limit
- Restricted instructions between the issue of a `TBEGINC` and `TEND` instruction

To enable the ASMAXTXP exit, use HLASM with the option `EXIT(PRTEXIT(ASMAXTXP))`.
If any of the above checks fail, then the transaction may not execute correctly due to transaction violations. ASMAXTXP will issue a warning message for any failing check with a return code 4.

Terminology
Constrained Transaction
A constrained transaction is started by the `TEBGINC` instruction. Updates to memory will be committed upon the end of the transaction specified by `TEND`. Constrained transactions are limited to 256 bytes in length between the TBEGINC and TEND instructions, many instructions are classified as *restricted* and are not permitted inside a constrained transaction, and no more than 32 instructions may be executed by the transaction.

Non-constrained Transaction
A non-constrained transaction is started by the `TBEGIN` instruction. At the end of the transaction specified by the TEND instruction, an attempt will be made to commit any updates to memory. If the updates to memory are unsuccessful, the transaction will be *aborted*; control passes to the instruction following the TBEGIN and the condition code is set to non-zero.
If a non-constrained transaction is aborted, general purpose registers specified by the GRSM (General-purpose Register Save Mask) operand of the TBEGIN instruction are restored to their previous values.

Non-transactional Store
The `NTSTG` instruction allows a store to memory to be performed whilst inside a transaction in a non-transactional way. This means that if the transaction is aborted, the non-transactional store would still have been committed.

Using the Transactional-Execution Facility

The rest of this Technote demonstrates how to use the Transactional-Execution Facility by changing a program which currently uses software-based locking methods to update a linked list.

The program consists of:

- A master process which performs the program's initialisation, spawns child processes and waits for them to finish their processing before terminating the program.
- Each child process creates a linked-list which are chained together with the lists from other child processes. In order to do this, the child processes currently use the z/OS ISGENQ macros to obtain and release software locks to perform safe updates to the chain of linked-lists.
- After attaching its linked-list into the chain, the child process waits for a predetermined amount of time in order to simulate processing after which it goes through the chain removing its linked-list.

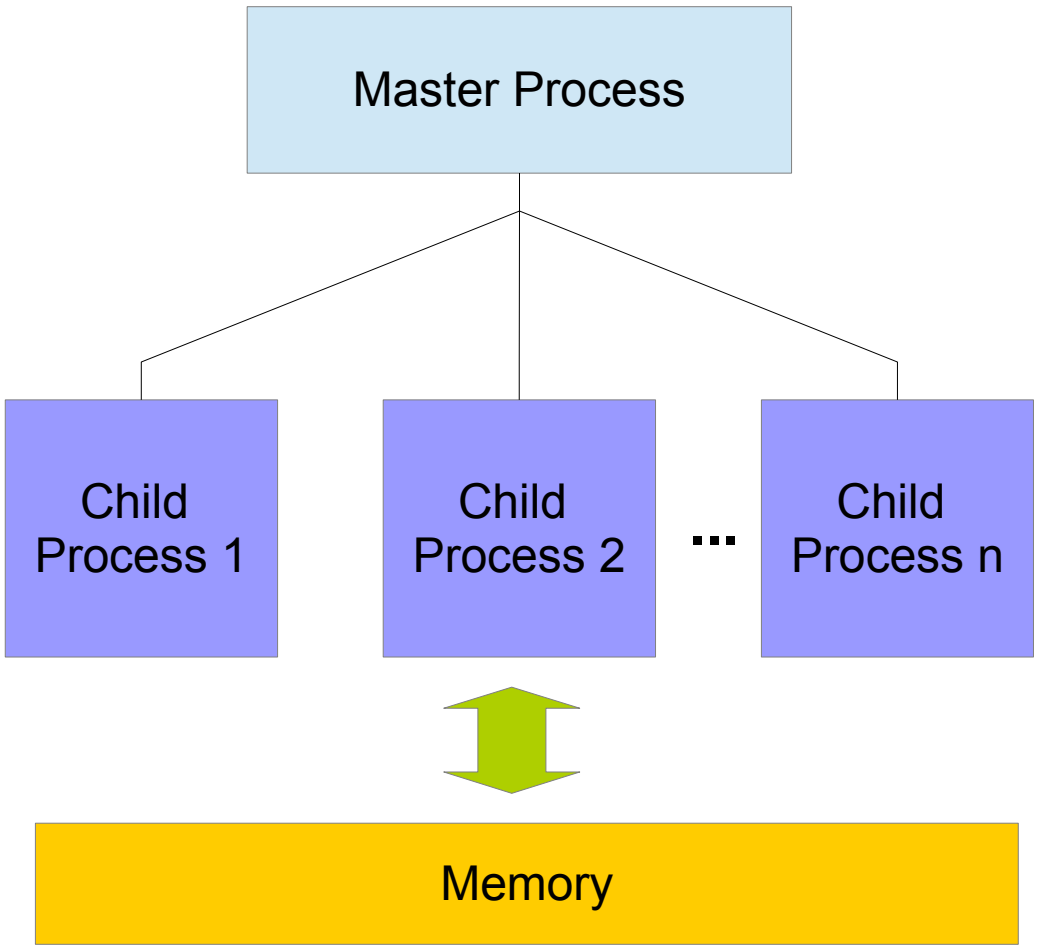

Figure 1 – Example master and child processes

Changing the master process

During initialisation the master process can check whether or not the Transactional-Execution Facility is available for use. This can be done by examining the output of the STFLE instruction. Under z/OS, the PSA already stores the output of the STFLE instruction.

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8

* INCLUDE THE IHAPSA MACRO IN ORDER TO EXAMINE THE PSA
         IHAPSA  ,
* Address PSA and examine facility bits
         XR    R0,R0
         USING PSA,R0
         TM    FLCEFACILITIESLISTBYTE9,X'40' Test for trans facility
         IF    (O)                           Is it on?
           MVI   TB_TRAN_SUPPORT,C'Y'        Yes - non-trans enabled
           TM    FLCEFACILITIESLISTBYTE6,X'20' Constr trans available?
           IF    (O)
             MVI   TB_TRAN_SUPPORT,C'C'       Yes - constr trans on
           ENDIF ,
         ELSE  ,
           MVI   TB_TRAN_SUPPORT,C'N'         Indicate no trans
         ENDIF ,
         DROP  R0

* Each child process can now examine the data passed to it in
* TB_TRAN_SUPPORT to determine whether or not it should run in
* transactional, constrained or non-transactional mode.  It is
* important that all child processes run in the same mode so that no
* problems occur should a child process using the Transactional-Execution
* Facility does not get into a conflict with a child process using
* software based locking.
```

At this stage of the program, each child process will be able to examine the state of TB_TRAN_SUPPORT to determine which "mode" to run in. The program has been written for portability so that the same object code can run on machines with and without the Transaction-Execution Facility being available. The value of TB_TRAN_SUPPORT is one of:

- C → Constrained transactions are available for use
- Y → Transaction-Execution facility is available for use
- N → No transaction facility is available for use

Child process code – Chaining the lists together

Each child process creates its own linked-list in memory. This part of the processing does not affect other child processes since these lists are created independently. Once the list is created, it must be chained together with the other linked-lists in memory and it is this operation which must be conducted in an atomic way to prevent other child processes attempting to update the chain as they add their list at the same time.

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8

* Examine WS_TRAN_SUPPORT (child working storage copy of
* TB_TRAN_SUPPORT) and start the chaining operation.
         CLI   WS_TRAN_SUPPORT,C'C'
         IF    (EQ)
           J     CTRACT          Run the constrained transaction
         ENDIF ,
         CLI   WS_TRAN_SUPPORT,C'N'
         IF    (EQ)
           XR    R5,R5             Clear transaction retry count
           LARL  R4,NTRACTS        Address of transaction retry
           J     NTRACT          Run non-constrained transaction
         ELSE  ,                 Get ENQ and software lock
         ISQENQ REQUEST=OBTAIN,                                        X
               QNAME=S_ENQ_MAJOR_NAME,                                 X
               RNAME=S_ENQ_MINOR_NAME,                                 X
               RNAMELEN=S_ENQ_RNAME_LENGTH,                            X
               CONTROL=EXCLUSIVE,                                      X
               ENQTOKEN=WS_ENQ_TOKEN,                                  X
               SCOPE=STEP,                                             X
               MF=(E,WS_ISGENQ_S)
           J     COM_Q           Skip transaction initialisation
         ENDIF ,
```

The child process can now either initialise its transaction or begin the common chaining code. The common chaining code is at the label COM_Q.

Starting the non-constrained transaction

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8

* Registers 6,7,8 and 9 will be modified during the transaction.
* Should the transaction abort for any reason, these should be
* restored to their previous values.
REGS69   EQU   B'00011000'       Create the GRSM for regs 6-9
*
* Start the non-constrained transaction
*
NTRACTS  TBEGIN WS_TRAN_TDB,REGS69
         JNZ    TRAN_ABORT   Setup abort handler
         J      COM_Q        Jump to common code
```

Starting a non-constrained transaction involves setting up the transaction abort routine, providing a Transaction Diagnostic Block (TDB) and also the General-purpose Register Save Mask (GRSM).
If the transaction is aborted, control will pass to the instruction immediately after the TBEGIN instruction. This instruction is used to jump to the common transaction abort routine TRAN_ABORT. The general-purpose registers indicated in the GRSM will be restored to their values previous to their values prior to the transaction starting.

Starting the constrained transaction

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8

*
* Start a constrained transaction
*
CTRACT   TBEGINC 0,0
         JNZ     TRAN_ABORT   Setup abort handler if we don't start
```

If the constrained transaction facility is available to the program, then the program will make use of it since using a constrained transaction *always* commits any updates it makes. However, constrained transactions have far more limitations placed upon them both in terms of which instructions may be used within them and the number of instructions which are to be placed inside them.

Common chaining code

Once the transactions have been started, or the ENQ has been obtained for the critical section of code, then the common chaining code can begin:

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8

*
* Common chaining code for chaining 2 lists into 2 chains
*
COM_Q    L       R8,TB_CELL_CHAIN_1    Get head 1 into working storage
         L       R9,TB_CELL_CHAIN_2    Get head 2 into working storage
         ST      R8,WS_CELL_1HEAD
         ST      R9,WS_CELL_2HEAD
         ST      R6,TB_CELL_CHAIN_1    Save new head 1
         ST      R7,TB_CELL_CHAIN_2    Save new head 2
         L       R6,WS_CELL_1TAIL      Get last cell 1
         L       R7,WS_CELL_2TAIL      Get last cell 2
         ST      R8,SP1_F_PTR          Join pointer
         ST      R9,SP2_F_PTR
         L       R8,TB_ENQ_COUNT_1     Get chain counts
         L       R9,TB_DEQ_COUNT_1
         AFI     R8,1                  Increase chain counts
         AFI     R9,1
         ST      R8,TB_ENQ_COUNT_1
         ST      R9,TB_DEQ_COUNT_1
         L       R8,TB_CELL_COUNT_1
         L       R9,TB_CELL_COUNT_2
         A       R8,WS_CELL_COUNT_1
         A       R9,WS_CELL_COUNT_2
         ST      R8,TB_CELL_COUNT_1
         ST      R9_TB_CELL_COUNT_2
*
* Check if running a transaction and if so then end it.
* If not then release the ENQ
*        CLI     WS_TRAN_SUPPORT,C'N'
         IF      (NE)
           TEND                          End the transaction to commit
         ELSE    ,                       Release the ENQ
         ISGENQ REQUEST=RELEASE,                                       X
               ENQTOKEN=WS_END_TOKEN,                                  X
               MF=(E,WS_ISGENQ_S)
         ENDIF   ,
```

In the common code, a number of counters and pointers were updated inside the transaction. Doing the same operation without a software-based lock and without the transactional-execution facility would become very difficult.
If changing existing code to make use of transactional-execution rather than using other software-based locking methods, critical section code may need to be restructured so that it can be executed within a transaction. The difficulty in doing this should be considered by the programmer before attempting to modify their existing code.
More information about the restrictions of using the Transactional-execution Facility can be found in the z/Architecture Principles of Operations (SA22-7832-09) – Chapter 5 – Program Execution.

Creating a Transaction-Abort Routine

If a transaction fails to initialise or it is aborted either by the hardware or by issuing the TABORT instruction, control passes to the instruction immediately following the corresponding TBEGIN / TBEGINC instruction and the condition code is set to non-zero. This is why in the example program, a JNZ TRAN_ABORT is used immediately following the TBEGIN and TBEGINC instructions.

The transaction abort routine may include logic for examining the Transaction Diagnostic Block (TDB). The format of the TDB is documented in Principles of Operation and contains many useful fields for diagnosing why a transaction was aborted and the state of the Transaction-Execution Facility at the time of the abort such as the Transaction Nesting Depth (TND), Transaction Abort Code (TAC), Aborted-Transaction Instruction Address (AITA).

The PERFORM PROCESSOR ASSIST (PPA) instruction can be used during the transaction abort processing in order to try and increase the likelihood of the transaction committing successfully. Before starting the non-constrained transaction, the child process uses register 5 as a transaction retry count and register 4 as the transaction retry address.

The transaction abort routine code is as follows:

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8

* ********************************************************
* Transaction abort routine TRAN_ABORT
* ********************************************************
*
* If the transaction fails to initialise or if it is aborted
* then control will pass here.
* The TDB may contain information that will help to diagnose
* why the transaction failed.
* The number of times that the transaction has failed is
* stored in register 5.
* The transaction retry address is stored in register 4.
*
TRAN_ABORT DC    0H
         WTO     MF=(E,WS_WTO_TABORT)  Show transaction has aborted
         CLI     WS_TRAN_SUPPORT,C'Y'
         IF      (NE)
*          SOMETHING WENT SERIOUSLY WRONG!!
           LA      R3,WS_TRAN_TDB        Make TDB easy to access
           ABEND   20,DUMP               ABEND and dump
         ENDIF   ,
*
* All the code for examining the TDB for any useful information
* should go here...
*
* Update the transaction fail count
        AHI     R5,1
        CIJNL   R5,20,DISTRAN      Failed 20 times? - Yes, give up
        PPA     R5,0,1             No - ask for help
        BR      R4                 Redrive transaction
DISTRAN MVI     WS_TRAN_SUPPORT,C'N'    Disable transaction support
        J       TIDYUP             Go tidy up storage and exit
```

When to use transactions

Although the Transaction-Execution Facility provides a very useful, they cannot be used to solve every locking problem. Transactions are good for short updates to memory that require more than a single instruction. Transactions should not be used for preparing to enter a critical section, timing / synchronising threads and large updates to memory.

Transactions and software-based locking methods should not be used together for the same critical sections of code. Since the Transaction-Execution Facility is based in hardware, it takes no notice of any software-based locks that are managed by the operating system. Consider the following:

- Processes A and B are attempting to update the same set of counters in memory.
- Process A acquires a software-based lock for the region of memory with exclusive access.
- Process B does not wait for the lock and instead uses a transaction to update the memory and commits its changes.
- Process A (which has the software lock) writes its changes to memory after Process B had finished its transaction and therefore overwrites the changes that Process B had made corrupting the data.

The problem occurs because Process B is unaware to the software-based lock held by Process A and therefore acts as if no locking was used over the data.

Summary

The Transactional-Execution Facility provided by System z can be used to simplify code and improve performance in the correct circumstances when needing to provide a level of atomicity to small critical sections of code.

Constrained transactions have severe limitations placed upon them, they always commit when they reach the TEND instruction and so can be used to guarantee the state of data if the transaction started.

Non-constrained transactions do not suffer with the same limitations as constrained transactions however they may be aborted if they cannot commit their data and therefore require a suitable transaction abort routine to be prepared should this occur. Any general-purpose registers specified in the GRSM (General-purpose Register Save Mask) upon entry to the non-constrained transaction will be restored to their previous values. The transaction abort routine can use the PERFORM PROCESSOR ASSIST (PPA) instruction in order to increase the likelihood of a non-constrained transaction committing its changes to data.

Appendix – JCL and Program Source Code

This technote provides a downloadable zip file which contains the source code, JCL and the assembler options file XOPT.txt

Contents of the zip file

\JCL
- \JCL\RUN.txt ← JCL used to run the executable load module
- \JCL\TRNSASX.txt ← JCL used to build the executable load module

\SOURCE
- \SOURCE\TBCHILD.asm ← Child program source
- \SOURCE\TBDATA.asm ← Common data copybook
- \SOURCE\TBMSTR.asm ← Master program source

\XOPT.txt ← HLASM options data set

Required data sets on z/OS

The following data sets should be created on your z/OS system under a suitable high-level qualifier before uploading the contents of the zip file to the system:

&myhlq..HLASM.TRNSACTN.JCL
This is the PDS into which the JCL is to be uploaded

&myhlq..HLASM.TRNSACTN.LOAD
This PDS will contain the executable load module built by the JCL

&myhlq..HLASM.TRNSACTN.SOURCE
This is the PDS into which the source files are to be uploaded

&myhlq..HLASM.TRNSACTN.XOPT
This sequential data set will contain the HLASM options.

Contents of HLASM options data set

The HLASM options data set (specified by the ASMAOPT DD statement) allows for a convenient place for assembly options to be located.

```
INFO
OBJ
EXIT(PRTEXIT(ASMAXTXP))
```

In particular interest for building programs that use the transactional-execution facility, the HLASM-supplied print exit ASMAXTXP should be used to check that your program adheres to the restrictions of the transactional-execution facility.
For more information on using ASMAXTXP, refer to the Appendix P in the HLASM Programmer's Guide (SC26-4941-06).

JCL used to assemble programs

The program is designed to assemble and issue WTOs so that it is clear to see how the program operates. To disable this, use an alternative library which has an inert WTO macro in it during assembly.

Note that the sample program makes use of the HLASM Structured Programming Macros which are available as part of the High Level Assembler Toolkit Feature. See the High Level Assembler Release 6 Toolkit Feature Users' Guide (GC26-8710-10) for more information at: http://www.ibm.com/software/awdtools/hlasm/library.html

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8

//TRNSASX  JOB NOTIFY=&SYSUID
//*
//* ************************************************************
//* THIS JCL IS USED TO BUILD AND LKED AN ASSEMBLER PROGRAM.
//* THE FOLLOWING IS ASSUMED WHEN USING THIS JCL:
//*      THE SOURCE FOR THE PROGRAM RESIDES IN ----> &SRCLIB
//*      THE EXECUTABLE LOADMOD IS OUTPUT TO ------> &LODLIB
//*      THE LISTING IS OUTPUT TO -----------------> &PRFXNM..Z.&PRGNM
//*      THE PROGRAM NAME IS ----------------------> &PRGNM
//*      THE ASSEMBLER OPTIONS USED ARE IN --------> &PRFXNM..XOPT
//* ************************************************************
//*
//* CHANGE THE FOLLOWING LINE TO REFLECT THE PROGRAM NAME
// SET PRGNM=TBMSTR
//*
//* CHANGE THE FOLLOWING LINE TO REFLECT THE HLQ FOR THE SOURCE
// SET PRFXNM=&SYSUID..HLASM.TRNSACTN
//*
// SET SRCLIB=&PRFXNM..SOURCE
// SET LODLIB=&PRFXNM..LOAD
// SET OPTLIB=&PRFXNM..XOPT
//*
//* CHANGE THE FOLLOWING TO REFLECT THE HLQ FOR HLASM AND SYSTEM
//* DATA SETS
// SET HLQ=SYS1
// SET HLQASM=SYS1.ASM
//*
//* COMPILE AND LINK STEP - PRODUCE THE EXECUTABLE LOAD MOD
//COMPLNK  PROC
//ASMMSAMP EXEC ASMACL
//STEPLIB    DD DISP=SHR,DSN=&HLQASM..SASMMOD1
//           DD DISP=SHR,DSN=&HLQASM..SASMMOD2
//SYSLIB     DD DISP=SHR,DSN=&SRCLIB
//           DD DISP=SHR,DSN=&HLQ..MACLIB
//           DD DISP=SHR,DSN=&HLQASM..SASMMAC1
//           DD DISP=SHR,DSN=&HLQASM..SASMMAC2
//C.SYSIN    DD DISP=SHR,DSN=&SRCLIB(&PRGNM)
//C.SYSPRINT DD DISP=OLD,DSN=&PRFXNM..Z.&PRGNM
//C.ASMAOPT  DD DISP=SHR,DSN=&OPTLIB
//L.SYSLMOD  DD DISP=SHR,DSN=&LODLIB(&PRGNM)
//         PEND
//* ************************************************************
// SET PRGNM=TBMSTR
//CP1      EXEC COMPLNK
//* ************************************************************
// SET PRGNM=TBCHILD
//CP2      EXEC COMPLNK
//* ************************************************************
```

TBDATA – DSECT used by TBMSTR and TBCHILD

The TBDATA DSECT is used by both the master (TBMSTR) and child (TBCHILD) programs for providing a common data structure in which the master program can communicate with each child program created.

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8

* ****************************************************************
* TBDATA - DSECTs for TBMASTER and TBCHILD programs
*
* Copyright IBM(UK) Ltd 2014
* HLASM - High Level Assembler and Toolkit (5696-234)
*
* IBM HLASM - Transactional-Execution Facility Technote
*
* This COPYBOOK is used to provide the TBMSTR and TBCHILD programs
* with a common set of data structures.
*
* The TB_MASTER DSECT is the data area which is passed from the
* master program to the child programs.
*
* The format of memory areas obtained from the 1st and 2nd subpool is
* defined by the TB_SP1_PARM and TB_SP2_PARM DSECTs and are used by
* the child programs in order to build storage chains.
*
* ****************************************************************
TB_MASTER       DSECT ,                  Area passed to children tasks
TB_EYE          DC CL(C_EYE_LENGTH)' '   Eye catcher
TB_CELL_ID_1    DC F'0'                  cell poolid
TB_CELL_ID_2    DC F'0'                  cell poolid
TB_ENQ_COUNT_1  DC F'0'                  Number of ...
TB_DEQ_COUNT_1  DC F'0'                  Number of ...
TB_CELL_COUNT_1 DC F'0'                  Number of cells
TB_CELL_COUNT_2 DC F'0'                  Number of cells
TB_ATTA_COUNT   DC F'0'                  Attach count
TB_DETA_COUNT   DC F'0'                  Dettach count
TB_CELL_CHAIN_1 DC A(0)                  cell poolid
TB_CELL_CHAIN_2 DC A(0)                  cell poolid
TB_TRAN_SUPPORT DC C'N'                  Trans support availability
                DC 0D                    align eye
TB_FCLTY_LIST   DC 2FD'0'                Faciliy list
TB_EYE_E        DC CL(C_EYE_LENGTH)' '   Eye catcher
                DC 0D
TB_LENGTH       EQU *-TB_MASTER
* ****************************************************************
TB_SP1_PARM    DSECT ,                   Subpool 1
SP1_EYE        DC CL(C_EYE_LENGTH)' '    Eye catcher
SP1_TCB        DC A(0)                   Pointer to TCB
SP1_F_PTR      DC A(0)                   Pointer to next cell
               DC 0D
SP1_LENGTH     EQU *-TB_SP1_PARM
* ****************************************************************
TB_SP2_PARM    DSECT ,                   Subpool 2
SP2_EYE        DC CL(C_EYE_LENGTH)' '    Eye catcher
SP2_TCB        DC A(0)                   Pointer to TCB
SP2_F_PTR      DC A(0)                   Pointer to next cell
               DC 0D
SP2_LENGTH     EQU *-TB_SP2_PARM
```

TBMSTR – The master program

The purpose of the master program is to detect the presence of the transactional-execution facility and create child processes which create and maintain memory subpools.
The value of C_CHILD_COUNT determines the number of child processes to create.
By uncommenting the lines in the master program marked with the comment ####### will cause the master program not to determine the presence of the transactional-execution facility.

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8

TBMSTR   RSECT
TBMSTR   RMODE  ANY
TBMSTR   AMODE  31
* ******************************************************************
* TBMSTR - Master transaction program
*
* Copyright IBM(UK) Ltd 2014
* HLASM - High Level Assembler and Toolkit (5696-234)
*
* The purpose of this program is to spawn child processes in order to
* build linked lists.
* The number of child processes spawned is determined by the
* C_CHILD_COUNT equate.
*
* ******************************************************************
         COPY  ASMMSP
         ASMDREG
         ASMMREL
C_EYE_LENGTH  EQU 16
C_CHILD_COUNT EQU 100              Amount of child threads to create
         PRINT OFF
         SYSSTATE ARCHLVL=1
         IEABRCX  DEFINE
         PRINT ON
         IHAPSA   ,
*
* ******************************************************************
* Start of mainline code
* ******************************************************************
*
TBMSTR   RSECT    ,
         PRINT ON
         SAVE (14,12),,*
         LARL  R9,STATICAREA
         USING (STATICAREA,STATICAREA_E),R9
         LA    R2,WS_LENGTH
         STORAGE OBTAIN,LENGTH=(R2),ADDR=(R3),COND=NO
         USING (WORKING_STORAGE,WORKING_STORAGE+WS_LENGTH),R3
         ST    R13,WS_SAVEA+4       save callers savearea
         LA    R13,WS_SAVEA         address my savearea
         ST    R13,WS_SAVEA+8       .. and save it
         MVC   WS_EYE,=CL(C_EYE_LENGTH)'TBMSTR W/S->'
         MVC   WS_EYE_E,=CL(C_EYE_LENGTH)'TBMSTR W/S-<'
         USING TB_MASTER,WS_MASTER_DATA
         MVC   TB_EYE,=CL(C_EYE_LENGTH)'TBMSTR MASTER>'
         MVC   TB_EYE_E,=CL(C_EYE_LENGTH)'<TBMSTR MASTER'
         MVI   TB_TRAN_SUPPORT,C'N'
         XC    TB_FCLTY_LIST,TB_FCLTY_LIST
         MVC   WS_LOAD(S_LOAD_L),S_LOAD
```

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8

*
* Setup WTOs showing transactional execution facility
*
         MVC   WS_WTO_SUPPORT_Y(S_WTO_SUPPORT_YL),S_WTO_SUPPORT_Y
         MVC   WS_WTO_SUPPORT_C(S_WTO_SUPPORT_CL),S_WTO_SUPPORT_C
         MVC   WS_WTO_SUPPORT_N(S_WTO_SUPPORT_NL),S_WTO_SUPPORT_N
         MVC   WS_WTO_ENDED(S_WTO_ENDED_L),S_WTO_ENDED
*
         MVC   WS_ATTACHX(S_ATTACHX_L),S_ATTACHX
         MVC   WS_POOL_1(S_POOL_1_L),S_POOL_1
         MVC   WS_POOL_2(S_POOL_2_L),S_POOL_2
*
* Check for the availability of the transaction facility
* This is located in bits 73 and 50 of the result of the STFLE
* instruction.
* Bit 73 must be checked first and indicates the presence of the
* Transactional-Execution Facility.
* Bit 50 may be checked once it is know that the transactional
* execution facility is available on this machine and indicates the
* Constrained transactional execution facility is available.
*
* If you wish to build the program so that it does not use the
* transactional execution facility then uncomment the line marked
* ########
*        AGO   .B_LOAD                          #######
         XR    R0,R0
         USING PSA,R0
         TM    FLCEFACILITIESLISTBYTE9,X'40'   Test for trans facility
         IF    (O)                             Is it on?
           MVI   TB_TRAN_SUPPORT,C'Y'          Yes - non-constr enabld
           WTO   MF=(E,WS_WTO_SUPPORT_Y)
           TM    FLCEFACILITIESLISTBYTE6,X'20' Constr trans avavilable?
           IF    (O)
             MVI   TB_TRAN_SUPPORT,C'C'        Yes - constr trans on
             WTO    MF=(E,WS_WTO_SUPPORT_C)
           ENDIF ,
         ELSE  ,
           CLI   TB_TRAN_SUPPORT,C'N'          Unknown value?
           JNE   CABEND                        Yes - ABEND
           WTO   MF=(E,WS_WTO_SUPPORT_N)       Indicate no tran
         ENDIF ,
         DROP  R0
.B_LOAD  ANOP  ,                                #######
*
* Load and start the child programs TBCHILD.
* After the child programs have been started, TBMSTR will wait until
* the child programs finish their processing.
*
BEGLOAD  LOAD  EP=TBCHILD,                                            X
               SF=(E,WS_LOAD)
         ST    R0,WS_TBCHILD_EP
         ST    R1,WS_TBCHILD_LP
```

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8

         CPOOL BUILD,PCELLCT=10,SCELLCT=5,                             X
               CSIZE=128,SP=0,LOC=31,                                  X
               CPID=TB_CELL_ID_1,                                      X
               HDR='TBMSTR CPOOL ONE',                                 X
               MF=(E,WS_POOL_1)
         CPOOL BUILD,PCELLCT=8,SCELLCT=12,                             X
               CSIZE=64,SP=0,LOC=31,                                   X
               CPID=TB_CELL_ID_2,                                      X
               HDR='TBMSTR CPOOL TWO',                                 X
               MF=(E,WS_POOL_2)
         XR    R5,R5
         LA    R6,C_CHILD_COUNT
         LA    R7,WS_ECB_CHILD
         LA    R8,WS_TCB_CHILD
NextAttach DC 0H
         ATTACHX EP=TBCHILD,PARAM=TB_MASTER,                           X
               ECB=(R7),                                               X
               SF=(E,WS_ATTACHX),MF=(E,WS_TBCHILD_PL)
         ST    R1,0(,R8)
         LA    R7,L'WS_ECB_CHILD(,R7)
         LA    R8,L'WS_TCB_CHILD(,R8)
         AFI   R5,1
         BCT   R6,NextAttach
         ST    R5,TB_ATTA_COUNT
         XR    R5,R5
         LA    R6,C_CHILD_COUNT
         LA    R7,WS_ECB_CHILD
         LA    R8,WS_TCB_CHILD
NextEndAndDetach DC 0H
         WAIT  1,ECB=(R7)
         DETACH (R8)
         LA    R7,L'WS_ECB_CHILD(,R7)
         LA    R8,L'WS_TCB_CHILD(,R8)
         AFI   R5,1
         BCT   R6,NextEndAndDetach
         ST    R5,TB_DETA_COUNT
CLEANUP  DC    0H
         DELETE EPLOC=WS_TBCHILD_EP
         CPOOL DELETE,                                                 X
               CPID=TB_CELL_ID_1
         CPOOL DELETE,                                                 X
               CPID=TB_CELL_ID_2
*
* At this stage, the child processes have completed processing.
* TBMSTR releases its storage and returns control to the OS.
*
EXIT     DC    0H                    exit label
         WTO   MF=(E,WS_WTO_ENDED)
         L     R13,WS_SAVEA+4       restore callers savearea
         LA    R2,WS_LENGTH
         STORAGE RELEASE,LENGTH=(R2),ADDR=(R3),COND=NO
         RETURN (14,12),,RC=0     return
CABEND         DC 20X'0'
```

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8

*
* ********************************************************************
* Static storage used by TBMSTR
* ********************************************************************
*
STATICAREA       DC 0D'0'           align storage
*
* Static macro definitions
*
S_LOAD           LOAD  SF=L
S_LOAD_L         EQU   *-S_LOAD
S_ATTACHX        ATTACHX SF=L
S_ATTACHX_L      EQU *-S_ATTACHX
S_POOL_1         CPOOL BUILD,MF=L
S_POOL_1_L       EQU *-S_POOL_1
S_POOL_2         CPOOL BUILD,MF=L
S_POOL_2_L       EQU *-S_POOL_2
*
S_CHILD_NAME     DC  CL8'TBCHILD'
*
* WTO messages for static storage
*
S_WTO_SUPPORT_Y  WTO 'Transactionality available',MF=L
S_WTO_SUPPORT_YL EQU *-S_WTO_SUPPORT_Y
S_WTO_SUPPORT_C  WTO 'Constrained Transactionality available',MF=L
S_WTO_SUPPORT_CL EQU *-S_WTO_SUPPORT_C
S_WTO_SUPPORT_N  WTO 'No transactionality available',MF=L
S_WTO_SUPPORT_NL EQU *-S_WTO_SUPPORT_C
S_WTO_ENDED      WTO 'Master process ended',MF=L
S_WTO_ENDED_L    EQU  *-S_WTO_ENDED
*
                 LTORG ,
STATICAREA_E     DC 0D'0'           align storage - end of static
*
* ********************************************************************
* Working storage used by TBMSTR
* ********************************************************************
*
WORKING_STORAGE DSECT             Program dynamic area
WS_EYE          DC CL(C_EYE_LENGTH)' '    Eye catcher
WS_SAVEA        DC 18F'00'                save area
WS_LOAD         DC 0D,XL(S_LOAD_L)'00'    load macro
WS_ATTACHX      DC 0D,XL(S_ATTACHX_L)'00' Attachx
WS_POOL_1       DC 0D,XL(S_POOL_1_L)'00'  cpool 1
WS_POOL_2       DC 0D,XL(S_POOL_2_L)'00'  cpool 2
*
* WS WTO buffers for transaction support
*
WS_WTO_SUPPORT_Y  DC 0D,XL(S_WTO_SUPPORT_YL)'00'
WS_WTO_SUPPORT_C  DC 0D,XL(S_WTO_SUPPORT_CL)'00'
WS_WTO_SUPPORT_N  DC 0D,XL(S_WTO_SUPPORT_NL)'00'
WS_WTO_ENDED      DC 0D,XL(S_WTO_ENDED_L)'00'
*
WS_TBCHILD_EP  DC A(0)            child program address
WS_TBCHILD_LP  DC A(0)            child program address
```

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8

*
* ********************************************************************
* Static storage used by TBMSTR
* ********************************************************************
*
STATICAREA       DC 0D'0'           align storage
*
* Static macro definitions
*
S_LOAD           LOAD  SF=L
S_LOAD_L         EQU   *-S_LOAD
S_ATTACHX        ATTACHX SF=L
S_ATTACHX_L      EQU *-S_ATTACHX
S_POOL_1         CPOOL BUILD,MF=L
S_POOL_1_L       EQU *-S_POOL_1
S_POOL_2         CPOOL BUILD,MF=L
S_POOL_2_L       EQU *-S_POOL_2
*
S_CHILD_NAME     DC  CL8'TBCHILD'
*
* WTO messages for static storage
*
S_WTO_SUPPORT_Y  WTO 'Transactionality available',MF=L
S_WTO_SUPPORT_YL EQU *-S_WTO_SUPPORT_Y
S_WTO_SUPPORT_C  WTO 'Constrained Transactionality available',MF=L
S_WTO_SUPPORT_CL EQU *-S_WTO_SUPPORT_C
S_WTO_SUPPORT_N  WTO 'No transactionality available',MF=L
S_WTO_SUPPORT_NL EQU *-S_WTO_SUPPORT_C
S_WTO_ENDED      WTO 'Master process ended',MF=L
S_WTO_ENDED_L    EQU  *-S_WTO_ENDED
*
                 LTORG ,
STATICAREA_E     DC 0D'0'           align storage - end of static
*
* ********************************************************************
* Working storage used by TBMSTR
* ********************************************************************
*
WORKING_STORAGE DSECT             Program dynamic area
WS_EYE          DC CL(C_EYE_LENGTH)' '    Eye catcher
WS_SAVEA        DC 18F'00'                save area
WS_LOAD         DC 0D,XL(S_LOAD_L)'00'    load macro
WS_ATTACHX      DC 0D,XL(S_ATTACHX_L)'00' Attachx
WS_POOL_1       DC 0D,XL(S_POOL_1_L)'00'  cpool 1
WS_POOL_2       DC 0D,XL(S_POOL_2_L)'00'  cpool 2
*
* WS WTO buffers for transaction support
*
WS_WTO_SUPPORT_Y  DC 0D,XL(S_WTO_SUPPORT_YL)'00'
WS_WTO_SUPPORT_C  DC 0D,XL(S_WTO_SUPPORT_CL)'00'
WS_WTO_SUPPORT_N  DC 0D,XL(S_WTO_SUPPORT_NL)'00'
WS_WTO_ENDED      DC 0D,XL(S_WTO_ENDED_L)'00'
*
WS_TBCHILD_EP  DC A(0)            child program address
WS_TBCHILD_LP  DC A(0)            child program address
```

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8

WS_TBCHILD_PL  DC A(0)            child program address
WS_TCB_CHILD   DC (C_CHILD_COUNT)F'00'           TCB
WS_ECB_CHILD   DC (C_CHILD_COUNT)F'00'           ECB
WS_COUNTER     DC F'00'           ws counter
               DC 0D              align end
WS_MASTER_DATA DC XL(TB_LENGTH)'00'
WS_EYE_E       DC CL(C_EYE_LENGTH)' '    Eye catcher
WS_LENGTH      EQU *-WORKING_STORAGE WS Length
         COPY TBDATA
         END   TBMSTR
```

TBCHILD – The child program

The purpose of the child program is to build linked-lists of acquired storage. Each child process is created by the master program and has passed to it a number of parameters. As the child begins to start its processing, it uses the WS_TRAN_SUPPORT field in order to determine whether it is able to make use of the transactional-execution facility.

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8

TBCHILD  RSECT
TBCHILD  RMODE  ANY
TBCHILD  AMODE  31
         COPY ASMMSP
         ASMDREG
         ASMMREL
* ******************************************************************
* TBCHILD - Child transaction program
*
* Copyright IBM(UK) Ltd 2014
* HLASM - High Level Assembler and Toolkit (5696-234)
*
* The purpose of this program is to link a number of storage areas
* into a shared linked list.
*
* ******************************************************************
C_EYE_LENGTH  EQU 16
C_POOL1_COUNT EQU 16              Amount of cells to create in pool1
C_POOL2_COUNT EQU 32              Amount of cells to create in pool2
C_LOOP_COUNT  EQU 1024
         PRINT OFF
         SYSSTATE ARCHLVL=1
         IEABRCX  DEFINE
         PRINT ON
*
* Prepare program execution environment
*
         SAVE (14,12),,*
         LARL  R9,STATICAREA
         USING (STATICAREA,STATICAREA_E),R9
         L     R10,0(,R1)           Point to input parms
         USING TB_MASTER,R10
*
* Ensure that the data we have been passed by the caller is correct
*
         CLC   TB_EYE,=CL(C_EYE_LENGTH)'TBMSTR MASTER>'
         BNE   E_IS_NOK
         CLC   TB_EYE_E,=CL(C_EYE_LENGTH)'<TBMSTR MASTER'
         BE    E_IS_OK
E_IS_NOK DC    0H
         DC    A(9)
E_IS_OK  DC    0H
         LA    R2,WS_LENGTH
         STORAGE OBTAIN,LENGTH=(R2),ADDR=(R3),COND=NO
         USING (WORKING_STORAGE,WORKING_STORAGE+WS_LENGTH),R3
         ST    R13,WS_SAVEA+4       save callers savearea
         LA    R13,WS_SAVEA         address my savearea
         ST    R13,WS_SAVEA+8       .. and save it
```

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8

*
* Prepare working storage eyecatchers
*
         MVC   WS_EYE,=CL(C_EYE_LENGTH)'TBCHILD W/S->'
         MVC   WS_EYE_E,=CL(C_EYE_LENGTH)'TBCHILD W/S-<'
         MVC   WS_ISGENQ_S(S_ISGENQ_S_L),S_ISGENQ_S
         MVC   WS_WTO_ENDED(S_WTO_ENDED_L),S_WTO_ENDED
*
* Setup transaction support messages
*
         MVC   WS_TRAN_SUPPORT,TB_TRAN_SUPPORT
         MVC   WS_WTO_SUPPORT_Y(S_WTO_SUPPORT_YL),S_WTO_SUPPORT_Y
         MVC   WS_WTO_SUPPORT_C(S_WTO_SUPPORT_CL),S_WTO_SUPPORT_C
         MVC   WS_WTO_SUPPORT_N(S_WTO_SUPPORT_NL),S_WTO_SUPPORT_N
         MVC   WS_WTO_TABORT(S_WTO_TABORT_L),S_WTO_TABORT
         MVC   WS_WTO_DIS_TRAN(S_WTO_DIS_TRAN_L),S_WTO_DIS_TRAN
         MVC   WS_WTO_TEND(S_WTO_TEND_L),S_WTO_TEND
         MVC   WS_WTO_TNCS(S_WTO_TNCS_L),S_WTO_TNCS
         MVC   WS_WTO_TNCE(S_WTO_TNCE_L),S_WTO_TNCE
         MVC   WS_WTO_TCS(S_WTO_TCS_L),S_WTO_TCS
         MVC   WS_WTO_TCE(S_WTO_TCE_L),S_WTO_TCE
         MVC   WS_WTO_WAITS(S_WTO_WAITS_L),S_WTO_WAITS
         MVC   WS_WTO_WAITE(S_WTO_WAITE_L),S_WTO_WAITE
         MVC   WS_TRAN_EYE_S,=CL(C_EYE_LENGTH)'TRANSACTION>>>>>'
         MVC   WS_TRAN_EYE_E,=CL(C_EYE_LENGTH)'<<<<<TRANSACTION'
         XC    WS_TRAN_TDB(C_TDB_LEN),WS_TRAN_TDB   Clear the TDB
*
         USING PSA,0
         MVC   WS_PSATOLD,PSATOLD
         DROP  0
*
* Start of mainline code
*
         LA    R11,C_LOOP_COUNT
StartProcess DC 0H
         ST    R11,WS_LOOP_CNTR           Save loop counter
         XR    R5,R5
         LA    R7,WS_CELL_1P_GET
         LA    R8,C_POOL1_COUNT
*
* Start building a chain from the 1st subpool
*
GetPool1 DC    0H
         USING TB_SP1_PARM,R1
         CPOOL GET,                                              X
               CPID=TB_CELL_ID_1
         ST    R1,0(,R7)
         LA    R7,4(,R7)
         AFI   R5,1
         MVC   SP1_EYE,=CL(C_EYE_LENGTH)'TBCHILD SP1->'
         MVC   SP1_TCB,WS_PSATOLD
         BCT   R8,GetPool1
         DROP  R1
         ST    R5,WS_CELL_COUNT_1
         XR    R5,R5
         LA    R7,WS_CELL_2P_GET
         LA    R8,C_POOL2_COUNT
```

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8

*
* Start building a chain from the 2nd subpool
*
GetPool2 DC    0H
         USING TB_SP2_PARM,R1
         CPOOL GET,                                                    X
               CPID=TB_CELL_ID_2
         ST    R1,0(,R7)
         LA    R7,4(,R7)
         AFI   R5,1
         MVC   SP2_EYE,=CL(C_EYE_LENGTH)'TBCHILD SP2->'
         BCT   R8,GetPool2
         DROP  R1
         ST    R5,WS_CELL_COUNT_2
*
* Chain the cells together
*
         LA    R6,WS_CELL_1P_GET
         LA    R8,C_POOL1_COUNT
         BCTR  R8,0
         USING TB_SP1_PARM,R7
Chain1   DC    0H
         L     R7,0(,R6)          current cell
         LA    R6,4(,R6)          get next one
         ST    R6,SP1_F_PTR       save forward cell
         BCT   R8,Chain1          next cell
         L     R7,0(,R6)          last cell
         ST    R7,WS_CELL_1TAIL   save tail cell
         DROP  R7
         LA    R6,WS_CELL_2P_GET
         LA    R8,C_POOL2_COUNT-1
         USING TB_SP2_PARM,R7
Chain2   DC    0H
         L     R7,0(,R6)          current cell
         LA    R6,4(,R6)          get next one
         ST    R6,SP2_F_PTR       save forward cell
         BCT   R8,Chain2          next cell
         L     R7,0(,R6)          last cell
         ST    R7,WS_CELL_2TAIL   save tail cell
         DROP  R7
         LA    R6,WS_CELL_1P_GET
         LA    R7,WS_CELL_2P_GET
         USING TB_SP1_PARM,R6
         USING TB_SP2_PARM,R7
         L     R6,0(,R6)                       get new head
         L     R7,0(,R7)                       get new head
*
* Check for the presence of the transaction facilty
* If we can run a constrained transaction on the data rather
* than using a full ENQ, then do so.
*
         LR    R2,R9                     Free up register 9
         DROP  R9
         USING (STATICAREA,STATICAREA_E),2
         CLI   WS_TRAN_SUPPORT,C'C'
```

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8

         IF    (EQ)                      Run constr transaction?
           WTO     MF=(E,WS_WTO_SUPPORT_C)
           J     CTRACT
         ENDIF ,
         CLI   WS_TRAN_SUPPORT,C'Y'      Run non-constr transaction?
         IF    (EQ)
           WTO     MF=(E,WS_WTO_SUPPORT_Y)
           J     NTRACT
         ENDIF ,
* If control flows through here, then run non-transactional code
GETENQ   WTO     MF=(E,WS_WTO_SUPPORT_N)
         ISGENQ REQUEST=OBTAIN,                                        X
               QNAME=S_ENQ_MAJOR_NAME,                                 X
               RNAME=S_ENQ_MINOR_NAME,                                 X
               RNAMELEN=S_ENQ_RNAME_LENGTH,                            X
               CONTROL=EXCLUSIVE,                                      X
               ENQTOKEN=WS_ENQ_TOKEN,                                  X
               SCOPE=STEP,                                             X
               MF=(E,WS_ISGENQ_S)
         J     COM_Q                   Skip transaction initialisation
*
* The transactional code will modify registers 6,7,8 and 9.
* Should the transaction abort for any reason, then these resgisters
* well be restored to their previous values.
REGS69   EQU   b'00011000'    Transaction GRSM for registers 6-9
*
* Start a non-constrained transaction
*
NTRACT   XR     R5,R5                 Set register 5 as trns abort count
         LARL   R4,NTRACTS            Set address of transaction retry
         LARL   R11,GETENQ            Set address of non trans retry
NTRACTS  TBEGIN WS_TRAN_TDB,REGS69
         JNZ   TRAN_ABORT             Setup abort handler
         J     COM_Q                  Jump to common code
*
* Start a constrained transaction
*
CTRACT   TBEGINC 0,0
         JNZ   TRAN_ABORT             Setup abort handler
COM_Q    L     R8,TB_CELL_CHAIN_1     Get head 1 into working storage
         L     R9,TB_CELL_CHAIN_2     Get head 2 into working storage
         ST    R8,WS_CELL_1HEAD
         ST    R9,WS_CELL_2HEAD
         ST    R6,TB_CELL_CHAIN_1     save head
         ST    R7,TB_CELL_CHAIN_1     save head
         L     R6,WS_CELL_1TAIL       get last cell
         L     R7,WS_CELL_2TAIL       get last cell
         ST    R8,SP1_F_PTR                join chain
         ST    R9,SP2_F_PTR                join chain
         L     R8,TB_ENQ_COUNT_1
         L     R9,TB_DEQ_COUNT_1
         AFI   R8,1
         AFI   R9,1
         ST    R8,TB_ENQ_COUNT_1
         ST    R9,TB_DEQ_COUNT_1
```

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8

         AFI   R8,1
         AFI   R9,1
         ST    R8,TB_ENQ_COUNT_1
         ST    R9,TB_DEQ_COUNT_1
         L     R8,TB_CELL_COUNT_1
         L     R9,TB_CELL_COUNT_2
         A     R8,WS_CELL_COUNT_1
         A     R9,WS_CELL_COUNT_2
         ST    R8,TB_CELL_COUNT_1
         ST    R9,TB_CELL_COUNT_2
*
* Check if we're running in a transaction and if we are not, then
* jump over the TEND instruction
*
         CLI   WS_TRAN_SUPPORT,C'N'
         IF    (NE)
* End the transaction and don't release an ENQ
           TEND
           WTO     MF=(E,WS_WTO_TEND)
         ELSE  ,
           DROP  R6
           DROP  R7
*
* Non-transactional ENQ release
*
           ISGENQ REQUEST=RELEASE,                                    X
               ENQTOKEN=WS_ENQ_TOKEN,                                 X
               MF=(E,WS_ISGENQ_S)
         ENDIF ,
*
* Simulate processing by waiting a short interval before continuing
*
         WTO     MF=(E,WS_WTO_WAITS)
MCONT    STIMER WAIT,TUINTVL=S_WAIT_TIME
         WTO     MF=(E,WS_WTO_WAITE)
*
* After the simililated processing wait is over, start releasing the
* storage and decoupling the chains.
*
*
* Prepare registers before iterating through
*
         MVC   WS_ISGENQ_S(S_ISGENQ_S_L),S_ISGENQ_S
         LA    R6,TB_CELL_CHAIN_1     Get top of chain
         LA    R8,C_POOL1_COUNT       number of entries to release
         LA    R7,WS_CELL_1P_REL      save pointers here
*
* Check the state of the transaction-execution facility and change
* into non-constrained transational mode if it is available.
*
         CLI   WS_TRAN_SUPPORT,C'N'
         IF    (NE)
           MVI   WS_TRAN_SUPPORT,C'Y'
         ENDIF ,
         CLI   WS_TRAN_SUPPORT,C'C'
```

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8

         IF    (EQ)
           WTO     MF=(E,WS_WTO_TCS)
           J     CTRACTDQ
         ENDIF ,
         CLI   WS_TRAN_SUPPORT,C'Y'
         IF    (EQ)
           WTO     MF=(E,WS_WTO_TNCS)
           J     NTRACTDQ
         ENDIF ,
* Else run in non-transactional mode
DQENQ    ISGENQ REQUEST=OBTAIN,                                        X
               QNAME=S_ENQ_MAJOR_NAME,                                 X
               RNAME=S_ENQ_MINOR_NAME,                                 X
               RNAMELEN=S_ENQ_RNAME_LENGTH,                            X
               CONTROL=EXCLUSIVE,                                      X
               ENQTOKEN=WS_ENQ_TOKEN,                                  X
               SCOPE=STEP,                                             X
               MF=(E,WS_ISGENQ_S)
         J     COM_DEQ
*
* Start dequeuing chain 1 items
*
         USING TB_SP1_PARM,R6
COM_DEQ  DC    0H
         CLI   WS_TRAN_SUPPORT,C'N'   Are we running in non-trans mode?
         IF    (NE)
           CLI   WS_TRAN_SUPPORT,C'C'   Run in constr trans mode?
           IF    (NE)
NTRACTDQ     XR    R5,R5                No - clear trans count
             LARL  R4,NTRACTSQ          Prepare retry address
             LARL  11,DQENQ
NTRACTSQ     TBEGIN WS_TRAN_TDB,REGS69  Start unconstrained trans
             JNZ   TRAN_ABORT           Prepare abort handler
           ELSE ,
CTRACTDQ     TBEGINC 0,0
             JNZ   TRAN_ABORT
           ENDIF ,
         ENDIF ,
DQCOM    ST    R6,0(,R7)
         LA    R7,4(,R7)
         L     R6,SP1_F_PTR
         CLI   WS_TRAN_SUPPORT,C'N'
         IF    (NE)
           TEND
         ENDIF ,
         BCT   R8,COM_DEQ
         ST    R6,TB_CELL_CHAIN_1
         DROP  R6
         CLI   WS_TRAN_SUPPORT,C'N'
         IF    (NE)
           TEND
         ENDIF ,
```

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8

*
* Start dequeuing chain 2 items
*
         L     R6,TB_CELL_CHAIN_2     get top of chain
         USING TB_SP2_PARM,R6
         LA    R8,C_POOL2_COUNT       number of entries to release
         LA    R7,WS_CELL_2P_REL save pointers here
UncPool2 DC    0H
         CLI   WS_TRAN_SUPPORT,C'N'   Non-transactional mode?
         IF    (NE)
           CLI   WS_TRAN_SUPPORT,C'C'   Constrained transactional mode?
           IF    (EQ)
             TBEGINC 0,0                Yes - start constrained trans
             JNZ   TRAN_ABORT
           ELSE ,
             TBEGIN WS_TRAN_TDB,REGS69  No - start unconstrained trans
             JNZ    TRAN_ABORT          Prepare abort handler
           ENDIF ,
         ENDIF ,
uncp2_c  ST    R6,0(,R7)
         LA    R7,4(,R7)
         L     R6,SP2_F_PTR
         CLI   WS_TRAN_SUPPORT,C'N'
         IF    (NE)
           TEND
           BCT   R8,UncPool2     Branch and reinitialise trans
         ELSE  ,
           BCT   R8,uncp2_c      Just loop back under the lock
         ENDIF ,
         ST    R6,TB_CELL_CHAIN_2
         DROP  R6
*
* If no transaction had taken place, then release the ENQ.
*
         CLI   WS_TRAN_SUPPORT,C'N'
         IF    (EQ)
RELDQENQ ISGENQ REQUEST=RELEASE,                                       X
               ENQTOKEN=WS_ENQ_TOKEN,                                  X
               MF=(E,WS_ISGENQ_S)
         ENDIF ,
*
* Now that the chains have been uncoupled from the main chain, the
* storage for them can be released.
*
MCONT2   LR    R9,R2                  Restore register 9 for Wrk Strg
         DROP  R2
         USING (STATICAREA,STATICAREA_E),R9
         LA    R7,WS_CELL_1P_GET
         LA    R8,C_POOL1_COUNT
FrePool1 DC    0H
         L     R6,0(,R7)
         CPOOL FREE,                                                   X
               CPID=TB_CELL_ID_1,                                      X
               CELL=(R6)
         LA    R7,4(,R7)
```

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8

         BCT   R8,FrePool1
         LA    R7,WS_CELL_2P_GET
         LA    R8,C_POOL1_COUNT
FrePool2 DC    0H
         L     R6,0(,R7)
         CPOOL FREE,                                                   X
               CPID=TB_CELL_ID_2,                                      X
               CELL=(R6)
         LA    R7,4(,R7)
         BCT   R8,FrePool2
*
* If the process has more work to do, then loop around and start again
*
         L     R11,WS_LOOP_CNTR    reload loop counter
         BCT   R11,StartProcess    loop around
*
* ********************************************************************
* Mainline return point
* ********************************************************************
*
EXIT     DC    0H                   exit label
         WTO     MF=(E,WS_WTO_ENDED)
         L     R13,WS_SAVEA+4       restore callers savearea
         LA    R2,WS_LENGTH
         STORAGE RELEASE,LENGTH=(R2),ADDR=(R3),COND=NO
         RETURN (14,12),,RC=0     return
*
* ********************************************************************
* TRANSACTION ABORT ROUTINE
* ********************************************************************
*
* If the transaction fails to initialise or if it is aborted,
* then control will pass here to the TRAN_ABORT label.
*
* If the program was running in a non-constrained transaction, then
* we examine the Transaction Diagnostic Block (TDB)
*
* The number of times that a transaction attempts to redrive itself
* is determined by register 5.
*
* The transactional retry address is stored in register 4.
*
* The non-transactional retry address (once register 5 has reached a
* certain threshold), is stored in register 11.
*
TRAN_ABORT DS  0H
         WTO   MF=(E,WS_WTO_TABORT)
         CLI   WS_TRAN_SUPPORT,C'Y'     Non-constrained transaction?
         IF    (NE)
*          SOMETHING WENT SERIOUSLY WRONG!!
           LA    R3,WS_TRAN_TDB
           ABEND 20,DUMP
DISTRAN    MVI   WS_TRAN_SUPPORT,C'N'   Disable tran support for child
           WTO   MF=(E,WS_WTO_DIS_TRAN)
           BR    R11                    Redrive update logic
         ENDIF ,
```

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8

* ### CODE FOR TDB EXAMINATION GOES HERE
* All diagnostics have been performed on the TDB
* Redrive the program logic with processor assist for this child
* process.
         AHI   R5,1                     Increase trns abort count
         CIJNL R5,20,DISTRAN            Failed 20 times?
         PPA   R5,0,1                   Perform proc assist
         BR    R4                       Redrive transaction
*
* ********************************************************************
* Child process static storage
* ********************************************************************
*
STATICAREA          DC 0D'0'           align storage
S_ISGENQ_S          ISGENQ MF=(L,S_,0D)
S_ISGENQ_S_L        EQU *-S_ISGENQ_S
S_ENQ_MAJOR_NAME    DC CL8'TBMSTR'
S_ENQ_MINOR_NAME    DC CL(32)'TBCHILD Enqueue Res'
S_ENQ_RNAME_LENGTH DC AL1(L'S_ENQ_MINOR_NAME)
S_WAIT_TIME DC F'1'      1 timer interval - approx 26 micro secs
*
* WTO macros
*
S_WTO_SUPPORT_Y  WTO 'Running in transactional mode',MF=L
S_WTO_SUPPORT_YL EQU *-S_WTO_SUPPORT_Y
S_WTO_SUPPORT_C  WTO 'Running in constrained trans mode',MF=L
S_WTO_SUPPORT_CL EQU *-S_WTO_SUPPORT_C
S_WTO_SUPPORT_N  WTO 'Running in non-transactional mode',MF=L
S_WTO_SUPPORT_NL EQU *-S_WTO_SUPPORT_N
S_WTO_TABORT     WTO 'Transaction aborted',MF=L
S_WTO_TABORT_L   EQU *-S_WTO_TABORT
S_WTO_DIS_TRAN   WTO 'Disabling transactional mode',MF=L
S_WTO_DIS_TRAN_L EQU *-S_WTO_DIS_TRAN
S_WTO_TEND       WTO 'Transaction ended',MF=L
S_WTO_TEND_L     EQU *-S_WTO_TEND
S_WTO_TNCS       WTO 'Non-constrained transaction started',MF=L
S_WTO_TNCS_L     EQU *-S_WTO_TNCS
S_WTO_TNCE       WTO 'Non-constrained transaction ended',MF=L
S_WTO_TNCE_L     EQU *-S_WTO_TNCE
S_WTO_TCS        WTO 'Con-constrained transaction started',MF=L
S_WTO_TCS_L      EQU *-S_WTO_TCS
S_WTO_TCE        WTO 'Con-constrained transaction ended',MF=L
S_WTO_TCE_L      EQU *-S_WTO_TCE
S_WTO_ENDED      WTO 'Child process ended',MF=L
S_WTO_ENDED_L    EQU *-S_WTO_ENDED
S_WTO_WAITS      WTO 'Child process starting wait',MF=L
S_WTO_WAITS_L    EQU *-S_WTO_WAITS
S_WTO_WAITE      WTO 'Child process finished wait',MF=L
S_WTO_WAITE_L    EQU *-S_WTO_WAITE
*
                 LTORG
STATICAREA_E     DC 0D'0'           align storage - end of static
```

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8

*
* ********************************************************************
* Child process working storage
* ********************************************************************
*
WORKING_STORAGE DSECT ,           Program dynamic area
WS_EYE          DC CL(C_EYE_LENGTH)' '    Eye catcher
WS_SAVEA        DC 18F'0'                 save area
WS_ISGENQ_S     DC 0D,XL(S_ISGENQ_S_L)'00'  enq
WS_ENQ_TOKEN    DC CL(32)' '               enq token
WS_CELL_COUNT_1 DC F'0'
WS_CELL_COUNT_2 DC F'0'
WS_CELL_1P_GET  DC (C_POOL1_COUNT)A(0)     pointer to cell 1
WS_CELL_2P_GET  DC (C_POOL2_COUNT)A(0)     pointer to cell 2
WS_CELL_1P_REL  DC (C_POOL1_COUNT)A(0)     pointer to cell 1
WS_CELL_2P_REL  DC (C_POOL2_COUNT)A(0)     pointer to cell 2
WS_CELL_1HEAD   DC A(0)                    head of chain
WS_CELL_2HEAD   DC A(0)                    head of chain
WS_CELL_1TAIL   DC A(0)                    temp tail
WS_CELL_2TAIL   DC A(0)                    temp tail
WS_PSATOLD      DC A(0)                    tcb
WS_WTO_ENDED    DC 0D,XL(S_WTO_ENDED_L)'00'
WS_LOOP_CNTR    DC F'0'
*
* ********************************************************************
* TRANSACTIONAL-EXECUTION FACILITY DATA AREA
* ********************************************************************
*
WS_WTO_SUPPORT_Y  DC 0D,XL(S_WTO_SUPPORT_YL)'00'
WS_WTO_SUPPORT_C  DC 0D,XL(S_WTO_SUPPORT_CL)'00'
WS_WTO_SUPPORT_N  DC 0D,XL(S_WTO_SUPPORT_NL)'00'
WS_WTO_TABORT     DC 0D,XL(S_WTO_TABORT_L)'00'
WS_WTO_DIS_TRAN   DC 0D,XL(S_WTO_DIS_TRAN_L)'00'
WS_WTO_TEND       DC 0D,XL(S_WTO_TEND_L)'00'
WS_WTO_TNCS       DC 0D,XL(S_WTO_TNCS_L)'00'
WS_WTO_TNCE       DC 0D,XL(S_WTO_TNCE_L)'00'
WS_WTO_TCS        DC 0D,XL(S_WTO_TCS_L)'00'
WS_WTO_TCE        DC 0D,XL(S_WTO_TCE_L)'00'
WS_WTO_WAITS      DC 0D,XL(S_WTO_WAITS_L)'00'
WS_WTO_WAITE      DC 0D,XL(S_WTO_WAITE_L)'00'
WS_TRAN_EYE_S     DC CL(C_EYE_LENGTH)' ' Transaction support eyecatcher
WS_TRAN_SUPPORT   DC C'N'                Transaction support
*
WS_TRAN_TDB    DC 0D         Transaction Diagnostic block       +0
WS_TDB_FORM    DC XL.8'0'    TDB FORMAT                         +0
WS_TDB_FLAGS   DC XL.8'0'    TDB FLAGS                          +0
               DC 4C'0'    ! Reserved                           +0
WS_TDB_TND     DC XL.16'0'   TDB TND (Nesting depth)            +0
WS_TDB_TAC     DC XL8'0'     TDB TRANSACTION ABORT CODE         +8
WS_TDB_CONFL   DC XL8'0'     TDB CONFLICT LOCATION              +16
WS_TDB_ATIA    DC XL8'0'     TDB ABORTED TRANS INSTR ADDR       +24
WS_TDB_EAID    DC XL.8'0'    TDB EAID                           +32
WS_TDB_DXC     DC XL.8'0'    TDB DXC                            +32
               DC 2X'0'    ! Reserved                           +32
WS_TDB_PI_ID   DC XL.32'0'   TDB PROGRAM INTERRUPTION ID        +32
```

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8

WS_TDB_EXCP_ID DC XL8'0'     TDB TRANSLATION EXCEPTION ID       +40
WS_TDB_BEA     DC XL8'0'     TDB BREAKING EVENT ADDRESS         +48
               DC 72X'0'   ! Reserved                           +56
WS_TDB_GPR     DC 120X'0'    TDB GPRs                           +128
C_TDB_LEN      EQU *-WS_TRAN_TDB
*
WS_TRAN_EYE_E  DC CL(C_EYE_LENGTH)' '    Transaction support eyecatcher
*
* ********************************************************************
* END OF TRANSACTIONAL-EXECUTION FACILITY DATA AREA
* ********************************************************************
*
WS_EYE_E       DC CL(C_EYE_LENGTH)' '    Eye catcher
               DC 0D              align end
WS_LENGTH      EQU *-WORKING_STORAGE WS Length
*
* Other DSECTs and COPYBOOKS
*
         COPY  TBDATA
         IHAPSA DSECT=YES,LIST=YES
         END   TBCHILD
```