OpenVMS Asynchronous System Trap (AST) Internals

Robert Gezelter Software Consultant
35 – 20 167th Street, Suite 215
Flushing, New York 11358 – 1731
United States of America

+1 718 463 1079
gezelter@rlgsc.com

Wednesday, November 13, 1996
1:00 pm – 1:50 pm
Room B1

Fall 1996 US DECUS Symposium
Anaheim Convention Center
Anaheim, California
There’s absolutely no reason for being rushed along with the rush. Everybody should be free to go very slow ... What you want, what you’re hanging around in the world waiting for, is for something to occur to you.

- Robert Frost

What is an AST?

An AST (Asynchronous System Trap) is a subroutine call executed outside the main thread of execution.
Common Factors – VAX and ALPHA

- ASTs are FIFO by access mode
- critical to internal functioning of OpenVMS
- managed by combination of hardware, firmware, and software
- common Programmer Interface

Common User Interface
Reference Sources:

Book:
VAX/VMS Internals and Data Structures,
OpenVMS ALPHA Internals and Data Structures,
VAX Architecture Handbook
Digital Press
ALPHA Architecture Handbook

Manuals:
VMS System Services
Guide to Creating Modular Procedures
I/O Users Manual
RTL Library
VMS Device Support

Creation of ASTs

DCLAST System Service
Events (I/O Complete, Timer, Locks, etc.)
Internal System Processing (Process Context)
External Requests (GETJPI, etc.)
AST Queueing

Simulated FIFO by Access Mode
Oldest Kernel Mode First
Newest User Mode Last

ASTs executing at elevated modes (Kernel, Executive, and Supervisor) have complete access to Process Context.
OpenVMS for VAX

AST processing is managed by a combination of VAX hardware and VMS.
AST Support provided by VAX Hardware

**REI Instruction**

PR$_ASTLVL$ – Processor AST Level Register
IPL$_ASTDEL$ – AST Delivery IPL Level

REI Instruction

Checks PR$_ASTLVL$ and destination Access Mode to see if ASTs are deliverable.
If ASTs are deliverable, generate Software Interrupt at IPL$_ASTDEL$ (IPL 2)
PR$_\text{ASTLVL}$ – Processor AST Level Register

3 Bits wide
0 - Kernel Mode AST Pending
1 - Executive Mode AST Pending
2 - Supervisor Mode AST Pending
3 - User Mode AST Pending
4 - No ASTs are Pending

IPL$_\text{ASTDEL}$ – AST Delivery IPL Level

Only IPL requested by VAX CPU microcode, not accessed by the MTPR instruction.
AST Queueing

FIFO by Access Mode
Oldest Kernel Mode First
Newest User Mode Last
One queue, ACBs are inserted in order
User Mode ASTs can be directly inserted at the end of the ACB chain.

AST Delivery

Verify there are ASTs pending.
Remove first AST from queue.
Check if AST is eligible.
Check if an AST is already active.
Check if ASTs are enabled.
Indicate that AST is active.
Release Quota.
Set ASTLVL to Current+1.
Build Parameter List.
Deallocate ACB.
Use REI to switch to actual Mode.
Call routine using CALLG.
**AST Queueing**

Processed by SCH$QAST.
Check Target Process.
Validate Request.
Insert ACB in queue.
Compute new ASTLVL.
Make Process computable.

**AST Entry**

Invoked by CALLG instruction.
Must conform to VAX Calling Standard.
Static Structures are undefined.
Register Contents are unpredictable.
Safe Storage: Stack, Variables used only from AST state at current level.
AST Exit

Control returned using RET.
EXE$ASTRET cleans stack.
CHMK to switch to Kernel Mode.
Recomputes ASTLVL.
Out of Kernel Mode.
Restores R0, R1.
AST processing is managed by a combination of ALPHA hardware, PAL Code, and OpenVMS.

AST Support provided by ALPHA Hardware

Hardware Privileged Context Block & Processor Registers
- ASTSR - AST Summary Register
- ASTEN - AST Enable Register

Corresponding save areas in HPCB

4 dedicated interrupts on IPL
- IPL$_ASTDEL AST (IPL 2)
- ASTDelivery IPL Level

NOTES
Dedicated IPL 2 Interrupt Entries

SCH$USER_ASTDEL
SCH$SUPER_ASTDEL
SCH$EXEC_ASTDEL
SCH$KERNEL_ASTDEL

CALL_PAL REI Instruction

Checks ASTSR and ASTEN to determine if destination Access Mode has a deliverable AST

If an AST is deliverable, generate corresponding IPL$_ASTDEL (IPL 2) interrupt
**ASTSR – AST Summary Register**

4 Bits wide
1 - Kernel Mode AST Pending
2 - Executive Mode AST Pending
4 - Supervisor Mode AST Pending
8 - User Mode AST Pending

**ASTEN – AST Enable Register**

4 Bits wide
1 - Kernel Mode AST Enabled
2 - Executive Mode AST Enabled
4 - Supervisor Mode AST Enabled
8 - User Mode AST Enabled
IPL$_{\text{ASTDEL}}$ –
AST Delivery IPL Level

Dedicated to AST Delivery

AST Queueing

FIFO by Access Mode
5 Queues
Special Kernel
Kernel
Executive
Supervisor
User
AST Delivery

Verify new IPL < IPL$_ASTDEL
Locate Most Privileged Pending AST
Check for Delivery Enabled
Clear Pending Bit in ASTSR
Save Volatile Registers
Checks to see if AST already active.
If so, returns with "not delivered"
Else, calls SCH$ASTDEL

AST Queueing

Processed by SCH$QAST.
Check Target Process.
Validate Request.
Insert ACB in correct queue.
Compute new ASTSR.
Make Process computable.
AST Entry

Invoked by Standard CALL. Must conform to ALPHA Calling Standard.
Static Structures are undefined. Register Contents are unpredictable.
Safe Storage: Stack, Variables used only from AST state at current level.

AST Exit

Control returned using RET. EXE$AST_EXIT cleans stack.
CHMK to switch to Kernel Mode. Recomputes ASTSR.
XQP is a good example of the power of ASTs. ASTs allow XQP to relinquish control while waiting for a resource or an asynchronous event.
Questions?

Robert Gezelter Software Consultant
35 – 20 167th Street, Suite 215
Flushing, New York 11358 – 1731
United States of America

+1 718 463 1079
gezelter@rlgsc.com