

Session 13114

Getting Comfortable in your SLIP/PERs



SHARE
Technology • Connections • Results



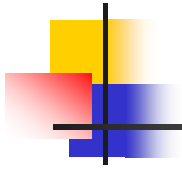
MVS Core Technologies Project – February 6th, 2013

Evan Haruta haruta@us.ibm.com
Patty Little plittle@us.ibm.com

IBM Poughkeepsie

© 2013 IBM Corporation

SHARE San Francisco, February 2013



Trademarks

The following are trademarks of the International Business Machines Corporation in the United States and/or other countries.

- MVS
- OS/390®
- z/Architecture®
- z/OS®

* Registered trademarks of IBM Corporation

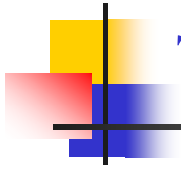


Table of Contents

- What is SLIP/PER? 4
- Events monitored by PER 6
- The power of PER 7
- PER processing and implementation 9
- Performance considerations in designing PER traps . . 11
- Defining PER range to be monitored 13
- Filters, actions, and tailorability 14
- Other PER trap controls 17
- Range pairs 18
- Triplets 19
- Examples 20



What is SLIP/PER? Debugging Power!

- SLIP – Serviceability Level Indication Processing
 - Used to trap **abends** and **messages**
 - Primarily software-driven
- PER – Program Event Recording
 - Used to trap hardware events
 - **Instruction Fetch**
 - **Storage Alteration**
 - **Successful Branch**
 - Hardware **detects event** and invokes SLIP software
 - SLIP software **applies filters** and **takes action(s)**
- SLIP/PER is often generically referred to as SLIP

SLIP/PER is the premiere tool for trapping programming events in z/OS. It can trap software events such as abends or messages, or it can trap hardware events such as the execution (fetching) of an instruction or the alteration of an area of storage. Its power is in its flexibility both in filtering on event environments and in generating documentation.

The monitoring of hardware events is accomplished by the PER part of SLIP/PER processing. When a monitored hardware event occurs, hardware detects this and signals SLIP software. SLIP software handles the additional filtering and taking of action as appropriate.

SLIP software also provides filtering for the software-generated events, and will take action as appropriate.



What we won't talk about today

- Non-PER SLIP traps

Examples:

- SLIP SET,C=0C4,JOBNAME=TEST,
A=TRACE,TRDATA=(STD,REGS),END
 - Write a GTF trace record when an ABEND0C4 occurs in job TEST
- SLIP SET,MSGID=IEA421I,A=SVCD,END
 - Take an SVC dump when message IEA421I is issued

While we won't talk about SLIPs for abends and messages, we will talk (in context) about the SLIP software which supports the PER hardware, doing filtering and taking action. This is the same code that provides filtering and takes action for non-PER SLIPs.



Events Monitored by PER

- Capability to **monitor** certain hardware events
 - **Instruction Fetch (IF)** – Fetching of instructions within a specified instruction range
 - **Storage Alteration (SA)** – Alteration of storage within a specified range
 - **Successful Branch (SBT)** – Branches made into or within a specified instruction range

There is actually a 4th event that is monitored by PER as well: SAS catches stores done by the STURA instruction as well as other alterations.



The Power of PER

- What PER SLIP traps can do:
 - Monitor for IF, SA, or SBT over a range that can be hardcoded, coded through indirection, or identified as a module + offset
 - Filter events to a great level of specificity
 - Generate tailored documentation
 - Trap on a sequence of PER events (“dynamic PER”)
 - Update storage and register content
 - Pseudo-IF/THEN/ELSE logic
- What PER SLIP traps can't do
 - Have more than one PER range being monitored at a time
 - Hit a moving target
 - Detect “DAT off” events
 - Issue a command or drive an exit
 - Arithmetic
 - Handle circular dynamic PER chains

NOTE: Some modules run with PER processing disabled to prevent recursion.

SLIP/PER's robust filtering capabilities allow for the trapping of very specific problem scenarios.

SLIP/PER provides a wide variety of actions that can be taken. SLIP provides parameters for tailoring of the documentation that is asked to produce.

Dynamic PER is supported through the A=TARGETID parameter.

Updating of register and storage content is supported through the A=REFBEFOR/REFAFTER parameter.

Pseudo-IF/THEN/ELSE logic is supported through the A=SUBTRAP parameter.

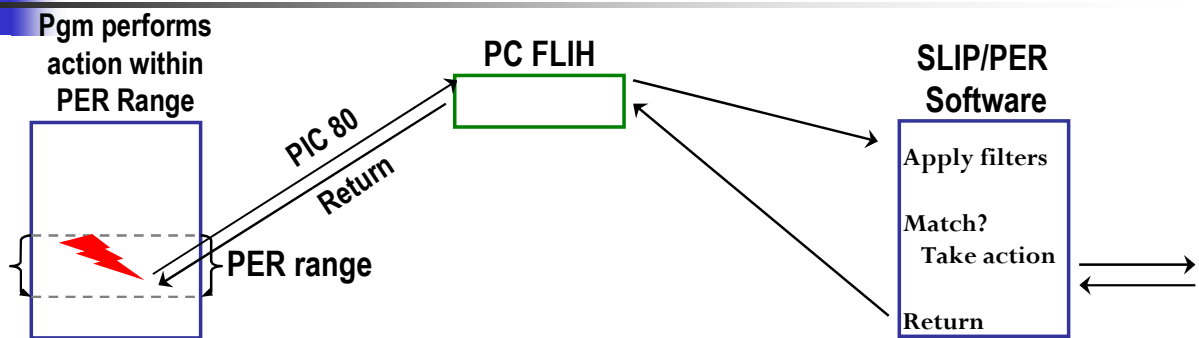


Disclaimers

- We can't tell you everything about PER in an hour!
- We will discuss the most useful keywords but will not be comprehensive.
- You may not choose to use some of these features on your own, but you should be aware of SLIP's wide range of capabilities when working problems with IBM support.
 - SLIP L2 welcomes questions from software service representatives.
- **We love questions!** We will try to answer your question right away. But SLIP is complex and sometimes subtle. If we don't know the answer of the top of our heads, we will get back to you!
- For all things SLIP, see the (very large) section on SLIP in the **MVS System Commands** manual.

Due to bullet #1, we will not be covering the REMOTE parameter.

PER Processing



- Hardware detects a monitored event occurring and issues a program interrupt (PIC 80)
- The operating system's Program Check First Level Interrupt Handler (PC FLIH) receives control and routes control to SLIP/PER software
- SLIP/PER software checks filters of active PER trap to see if environment at the time of interrupt meets all criteria
 - YES – Takes requested action(s)
 - Usually operating system returns control to point of PER interrupt
 - NO – Operating system returns control to point of PER interrupt

After most SLIP actions, control will be returned to the program that experienced the PER interrupt. An exception is A=RECOVERY which instead targets the interrupted unit of work with an ABEND06F.



PER Implementation

- When a PER trap is activated:
 - Control Reg9 on all CPs is set to indicate event being monitored
 - Control Regs 10 and 11 on all CPs are set to reflect the range being monitored
 - PSW bit 1 is turned on, indicating to hardware that it should monitor for the event defined via Control Regs 9 thru 11
 - Whenever feasible based on the requested SLIP, the operating system will limit the setting of this bit to units of work in designated address spaces
- Can only monitor one PER range at a time on a system
- PER monitoring by hardware is very efficient
 - Poorly performing PER traps are the result of poor trap design


PER processing can be enabled and disabled under a running unit of work simply by turning on and off the PER bit in the PSW. Some operating system code turns off this bit while it is running in order to prevent recursion scenarios. However, most operating system code runs enabled for PER.



Performance Considerations in Designing PER Traps

- A PER trap does not have to match to affect performance
 - Every instruction executed in an IF range triggers PER processing
 - Every alteration of storage within a SA range triggers PER processing
 - Every branch to/within a SB range triggers PER processing
- Avoid setting a PER trap over a large range
- Be careful when setting a PER trap in a frequently executed or frequently altered range
- Be careful of setting a PER trap in a performance path
- When in doubt, use PRCNTLIM parameter! (default=10%)

The PRCNTLIM (Percent Limit) parameter limits how much CPU a PER trap is allowed to consume on the system. If SLIP/PER CPU usage exceeds the indicated percentage, the offending PER trap is disabled.



PER Performance and the **JOBNAME** Parameter

- SLIP/PER provides a **JOBNAME** parameter which allows for **filtering of an event to only those units of work running under a particular job**
 - SLIP/PER will only turn on the PSW PER bit for units of work belonging to the specified job
 - PER interrupts will not be experienced by work running under other jobs
 - Use **JOBNAME** whenever feasible

When using PVTMOD, it is advisable to use **JOBNAME=**, and also **MODE=HOME** whenever feasible. If you are trying to catch an event that is occurring under a job in a cross memory environment, **MODE=HOME** may not be feasible. However, be sensitive to possible system performance issues that could result. Code a low **PRCNTLIM** (1-2%) to be on the safe side.



Defining PER Range to Be Monitored

- IF, SA, and SBT
 - **RANGE** RANGE=(beginningaddr,endingaddr)
 - Loaded into Control Registers 10 & 11 at time the SLIP is **SET**
 - RANGE is required on SA SLIPs

- IF and SBT only, instead of RANGE
 - **NUCMOD** or **NUCEP**
 - **LPAMOD** or **LPAEP**
 - **PVTMOD** or **PVTEP** } =(modname,startoffset,endoffset)

- SA only – identifies space where monitored storage resides
 - **ASIDSA** ASIDSA='JES2' or ASIDSA=3F
 - **DSSA** DSSA=('MYJOB'.MYDSPAC1)

RANGE can be used to define the range to be monitored for SB, IF, or SA. SLIP permits indirection to be specified in the RANGE parameter. This is resolved at the time the SLIP is set, and the resulting address is placed into the control registers used by SLIP. Note that the ending address can be omitted, in which case only the single byte indicated by the beginning address will be monitored.

NUCMOD & NUCEP, LPAMOD & LPAEP, PVTMOD & PVTEP can be used in a SBT or IF PER trap instead of RANGE. For example, one can specify:

```
NUCMOD=(IEAVEPST,40,4F)
```

on an IF PER trap to monitor the fetching of instructions in the POST code between offset +40 and offset +4F. This is much more flexible than having to hardcode the addresses in the RANGE parameter. Note that specification of a starting offset and ending offset are optional. If neither is specified, then the entire module is monitored. If ending offset is omitted, then only the instruction at the starting offset is monitored.

Use ASIDSA or DSSA on every SA PER trap. If monitoring a private storage range in an address space for alteration, specify the jobname or ASID owning that storage on ASIDSA. If monitoring global storage, specify ASIDSA=SA. If monitoring data space storage, use DSSA.



Filters

- Filtering to an address space level
 - JOBNAME
 - ASID
- Filtering on what code triggered the PER event (SA PER traps only)
 - ADDRESS
 - NUCMOD or NUCEP
 - LPAMOD or LPAEP
 - PVTMOD or PVTEP
- Filtering on environment
 - MODE
 - PSWASC
- Filtering on storage and/or register content
 - DATA

Filtering is done primarily by software, although some JOBNAME filtering may occur at the hardware level. Software gets control as a result of the PER interrupt and examines the defined traps in LIFO order to determine what filters to apply. After JOBNAME/ASID, the DATA parameter is probably the most commonly used FILTER. It provides a large degree of filtering capability, including bit checks and/or byte checks in registers or in storage. Multiple checks can be made as part of the filtering process.



Actions

- For SVC dumps:
 - SVCD, SYNCSVCD, TRDUMP, STDUMP
- For GTF trace:
 - TRACE, TRDUMP, STOPGTF
- For system trace:
 - STRACE, STDUMP
- Other
 - REFBEFOR, REFAFTER to update storage/registers
 - TARGETID to activate a new PER trap
 - RECOVERY to force an abend
 - SUBTRAP to simulate “IF-THEN-ELSE” logic
 - IGNORE to ignore the PER event
 - WAIT to enter restartable wait state

Note that some actions can be combined.

SVCD takes an asynchronous (scheduled) SVC dump. SYNCSVCD takes a synchronous SVC dump, which means that the unit of work requesting the dump does not continue running until the dump capture completes. A synchronous SVC dump can only be taken if the PER event occurs in an Enabled Unlocked Task (EUT) environment. There are other restrictions as well. Note that if A=SYNCSVCD is requested on a PER trap and the environment at the time the trap matches is invalid for a synchronous SVC dump, then a regular (scheduled) SVC dump will be taken instead.

For a SLIP defined to disable after N matches (via the MATCHLIM parameter), TRDUMP writes a GTF trace record for all N matches, and also takes an SVC dump on the Nth match. STOPGTF will stop GTF trace.

For a SLIP defined to disable after N matches (via the MATCHLIM parameter), STDUMP writes a system trace record for all N matches, and also takes an SVC dump on the Nth match.



Tailorability

- SVC dumps
 - JOBLIST, ASIDLST, DSPNAME dump address/data spaces
 - SDATA dump particular types of storage areas
 - SUMLIST, LIST dump specified storage ranges
 - STRLIST dump structures
 - REMOTE take dumps on other systems in sysplex
- GTF trace
 - TRDATA trace specified storage ranges
- System trace
 - STDATA trace specified storage ranges

Multiple jobs, ASIDs, and data spaces may be requested. Note that JOBLIST and ASIDLST are not mutually exclusive.

SUMLIST gathers the specified data ranges during summary dump processing. LIST gathers the specified data ranges during later dump processing.

Indirection may be specified in the LIST, SUMLIST, TRDATA, and STDATA parameters.

Wildcarding may be used in JOBLIST and DSPNAME.



Other PER trap controls

■ Parameters

- ENABLE, DISABLE to define trap enabled/disabled on SET
- MATCHLIM to disable trap after N matches
- PRCNTLIM to disable trap if it uses >N pct of CP
- ID to name SLIP trap (up to 4 characters)

■ Commands

- SLIP SET,
- SLIP MOD,ENABLE,ID=
- SLIP MOD,DISABLE,ID=
- SLIP DEL,ID=
- D SLIP
- D SLIP=

A SLIP trap can be set as disabled, then enabled at some later point. When entering multiple SLIP traps with interdependencies, it is advisable to enter them all disabled with a similar SLIP ID, then to enable them simultaneously through the SLIP MOD,ENABLE,ID= command, using wildcarding in the ID to target the multiple SLIPs.

A SLIP must be disabled before it can be deleted.

D SLIP will show the names of all SLIP traps defined to the system, along with an indication of whether the SLIP is enabled or disabled. To see the parameters of a specific SLIP, use: D SLIP=xxxx .

Range Pairs

- Syntax for range pairs: $\equiv(\text{beginaddr}, \text{endaddr})$
 - **Beginaddr, endaddr** may be address or indirect address

 - Parameters that support range pairs:
 - RANGE, ADDRESS
 - TRDATA, STDATA
 - LIST, SUMLIST
- } **Support multiple range pairs**
-
- What is an indirect address?
 - Combination of address or register notation, indirection symbols (% , ? , !), and/or displacements, symbolics (BEAR, BPER)
 - $\text{LIST} = (\overbrace{1R?+4?-1C}, \overbrace{1R?+4?+F}, \overbrace{13R?+0}, \overbrace{13R?+3F})$
 - $\text{LIST} = (\overbrace{1R?+4?-1C}, \overbrace{+F}, \overbrace{13R?}, \overbrace{+3F})$ shorthand for previous example

Root

Note: These are displacements, NOT LENGTHS!!

% - interpret address as 24-bit

? – interpret address as 31-bit

! – interpret address as 64 bit

Use R to represent a 31-bit general purpose register and G to represent a 64-bit general purpose register.

The symbolic BEAR refers to the address where the last flow-altering instruction occurred. This includes branches, PC, PR, LPSW, etc.

The symbolic BPER refers to the beginning address in a PER range. It is helpful for DATA comparisons.



Triplets (data comparison, storage refresh)

- Syntax for triplets: $\text{=(target,operator,source)}$
 - **Target** may be address, indirect address, or register
 - **Source** may be address, indirect address, register, or constant (max 8 bytes)
 - **Operator** options: EQ, NE, LT, NL, GT, NG
 - If 'A' appended (e.g. EQA) – operator acts on the **address** of the source
 - If 'C' appended (e.g. EQC) – operator acts on the **content** of the source
 - If used without 'A' or 'C' appended, the source must be a **constant**
 - **“(b)” following target address** indicates a bit position within the targeted byte or register; indicates operation is on binary data
 - **“(n)” following operator** indicates how many bits or bytes to operate on for Address and Contents operations
- Parameters: REFBEFOR, REFAFTER, DATA
 - All support multiple triplets, separated by AND or OR (default is “AND”)
 - $\text{DATA}=\text{(1R?+8(0),EQ,1,OR,15R,EQC(4),13R?+10,OR,0R,GT,0)}$
 - $\text{REFBEFOR}=\text{(FEBC0,EQ,C1C2C3C4,15R,EQ,0)}$

In addition to the operators listed above, there is also NE, NL, NG, etc.

A SLIP is operating on binary data if and only if there is a “(b)” value specified after the target. Based from 0, this indicates the beginning bit position that is to be altered or compared within the target register or address. If, in addition, a “(n)” is specified after the operator, this indicates how many bits are to be moved or compared. If no “(n)” is specified when performing a “C” (Contents) or “A” (Address) type of operation, the default is 1 bit.

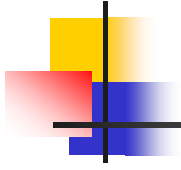
If there is no “(b)” after the target, and the operator is followed by “(n)”, this indicates how many bytes are to be altered or compared. This syntax should be used when performing a “C” (Contents) or “A” (Address) type of operation. The default is 4 bytes.

When SLIP does a contents or an address refresh (alter), it refreshes the first *n* bytes of storage and the last *n* bytes of a register.

If doing an EQ, GT, or LT, the number of bytes to be altered or compared is determined by the number of bytes in the constant supplied as the source.

The first example above (DATA=) will match if bit 0 at the storage location pointed to by Reg1+8 (perhaps the start of the 3rd word in a parameter list?) is on, or if the content of Reg15 is the same as the data found at address 13R?+10, or if Reg0 is greater than 0.

The second example above (REFBEFOR=) will update the storage at FEBC0 to contain eyecatcher 'ABCD', and will set Reg15 to contain 0. This is done BEFORE any other requested actions, most likely because the damage that we are repairing would otherwise prevent successful completion of other actions.



SLIP Examples



Case 1

- **A real customer case from mid-January:**

The product FMID information in the CVTPRODI field of the CVT is being overlaid, with surprisingly devastating system effect. This 8-byte field is in the CVT prefix at offset -X'20'. This field content is established at IPL and should never get changed. Take a dump when this field gets altered. (Note: Low core location X'10' points to the CVT. The CVT lives in common storage.)

- `SLIP SET,SA,ASIDSA=SA,RA=(10?-20,10?-19),
A=SYNCSVCD,END`

- **NOTES**

- **ASIDSA=SA** is used when monitoring alteration of common storage.
- The indirection in the **RANGE** is resolved when the SLIP is enabled.
- We could have written the range as: **RA=(10?-20,-19)**.
- **See APAR OA41190!**

A=SYNCSVCD and A=SVCD default to a match limit (ML) of 1, meaning the SLIP will disable after 1 match.



Case 2

- Consider Case 1. If the field is static, we should be able to refresh it, thus preventing the system impact.

Refresh the field back to its original content of 'HBB7780', before taking a dump.

- SLIP SET,SA,ASIDSA=SA,RA=(10?-20,10?-19),
A=(REFBEFOR,SYNCSVCD),
REFBEFOR=(10?-20,EQ,C8C2C2F7_F7F8F040),END

NOTES

- An additional **REFBEFOR** triplet could be added prior to the existing one if we want to **copy the corruption**, thereby preserving it for diagnostic purposes, prior to refresh:
 - **REFBEFOR=(targetaddr,EQC(8),10?-20,...)**
- Use REFAFTER in the case of an overlay where the damage does not need to be immediately corrected, so that you can see the content of the overlay in the dump.
- SLIP will update storage a max of 8 bytes at a time.
- The underscore within the **REFBEFOR** source value is optional and strictly cosmetic.

REFBEFOR and REFAFTER are extremely powerful. We recommend using this option under the guidance of an experienced support representative.



Case 3

- You have an application named BADLUCK which runs in its own address space of the same name. The eyecatcher of the application's “WXYZ” control block keeps getting overlaid. This control block lives in common storage at address 1E123000, and its eyecatcher is at offset +0. Take a dump when the storage gets corrupted.
- SLIP SET,SA,ASIDSA=SA,RA=(1E123000,1E123003),
DATA=(1E123000,NE,E6E7E8E9),A=SYNCSVCD,END
- NOTES
 - ASIDSA=SA is used when monitoring alteration of common storage.
 - We don't want to add JOBNAME to the SLIP as that would filter out an overlay done by another job.
 - The DATA compare will make sure that we don't match on the case where job BADLUCK is initializing the control block eyecatcher.



Case 4

- You have an application named BADLUCK which runs in its own address space of the same name. The eyecatcher of the application's "ABCD" control block is being overlaid. You believe the application itself is responsible for the overlay. The overlaid control block always lives at private storage address 6000, and the eyecatcher of "ABCD" is at offset +0. Take a dump when the storage gets corrupted.
- SLIP SET,SA,ASIDSA='BADLUCK',RA=(6000,6003),
JOBNAME=BADLUCK,MODE=HOME,
DATA=(6000,NE,C1C2C3C4),A=SYNCSVCD,END
- NOTES
 - **MODE=HOME** restricts a match to a non-cross memory environment. If work in JOB BADLUCK PC's to another address space, then corrupts the control block in private storage of job BADLUCK, this SLIP won't catch that.
 - Length of constant determines how many bytes of **DATA** are compared.

Remember that ASIDSA indicates the address space that the private storage being monitored resides in. This is *not* a filter for who actually overlays the storage. JOBNAME or ASID must be used to filter on who is doing the alteration.

Case 5

- Same as Case 4 except that this time we don't know where the private storage-resident "ABCD" control block lives until after it is GETMAINED. It is GETMAINED by private load module GETSTOR at offset +X'1B0'. On return from a GETMAIN, register 1 contains the address of the GETMAINED storage. Once we have this information, we can set up the SA SLIP.

- SLIP SET,IF,DISABLE,P=(GETSTOR,1B0),ID=SLP1,
JOBNAME=BADLUCK,MODE=HOME,
A=TARGETID,TARGETID=SLP2,END

SLIP SET,SA,DISABLE,RA=(1R?+0,1R?+3),ASIDSA='BADLUCK',
DATA=(BPER?+0,NE,C1C2C3C4),ID=SLP2,
A=SYNCSVCD,END

SLIP MOD,ENABLE,ID=SLP1

In this example, we're assuming that GETSTOR+1B0 points to the return point from the GETMAIN.

If you accidentally try to enable both SLIPs simultaneously, SLIP processing will detect that SLP2 is targeted by SLP1 and will leave it disabled.

A dynamic PER trap chain can consist of two or more traps. The chain cannot be circular.



Case 5: Notes

- Notes:

- When one PER trap targets another as it matches and disables, this is called **dynamic PER**.
- We have given each SLIP an ID, which can be a maximum of 4 characters.
- The **RANGE** of the second SLIP is **resolved at the time the SLIP is enabled**, using the value in register 1 at the time the first SLIP matched and disabled.
- The **DATA** parameter of the second SLIP is **resolved when a PER event occurs** for that SLIP, that is, when the specified range is altered.
- The symbolic **BPER** can be used to indicate the **B**eginning address of the **PER** range.
- When using dynamic PER, it is helpful to **set all SLIPs disabled** originally, then **enable the first in the chain**.
- We do not need to specify JOBNAME and MODE on the second SLIP. **The environmental specifications on the first SLIP automatically apply to all SLIPs in a dynamic PER chain.** This cannot be overridden!

There is one other SLIP symbolic: BEAR . This is set up to be the Breaking Event Address from the Breaking Event Address Register. This is the address of the last instruction on this CP to cause a branch or a change in linear flow of execution (e.g. PC or LPSW) prior to the PER interrupt.



Case 6

- You have an application named BADLUCK which runs in its own address space of the same name. The eyecatcher of the application's "ABCD" control block is being overlaid. *This application interfaces with a number of other address spaces, any of which could have done the overlay.* The overlaid control block always lives at private storage address 6000, and the eyecatcher of "ABCD" is at offset +0. Take a dump when the storage gets corrupted.
- **SLIP SET,SA,RA=(6000,6003),ASIDSA='BADLUCK',
DATA=('BADLUCK'.6000,NE,C1C2C3C4),
A=SYNCSVCD,END**
- **NOTES**
 - Since we don't know what address space or cross memory environment the overlayer is running in, *we have removed JOBNAME=BADLUCK and MODE=HOME.*
 - Since we don't know which address space will be current when the SA occurs, we must qualify the address of the **DATA** compare to indicate that address **6000** is within job **BADLUCK**.



Case 7

- CSECT MYMOD lives at offset X'A00' thru X'AFF' in LPA load module MYLOAD. To debug a logic problem, you want to trace the instruction flow through the entire CSECT, writing a GTF record for each instruction. You want to trace the X'20' byte work area pointed to by Register 11, as well as basic environmental information.
- SLIP SET,IF,L=(MYLOAD,A00,AFF),A=TRACE,
TRDATA=(STD,REGS,11R?+0,+1F),END
- NOTES
 - For **A=TRACE**, the default is to have no limit on how many times the SLIP can match. Use the ML parameter if you want a match limit on the SLIP.
 - Always specify **STD** and **REGS** to get fundamental diagnostic information
 - Note that the X'100' byte range that we are trapping is not huge, but if the code executes very frequently or loops, this could cause impact.



Case 8

- The results from the Case 7 SLIP suggest the problem is related to bad parameter list content. Trace the parameter list ONLY on entry to and exit from MYMOD. (Assume the last instruction of MYMOD is offset X'FE' into the CSECT.) The parameter list is pointed to by Register 1, is X'10' bytes long, and lives above the bar. Also trace the last X'20' bytes of the X'30' byte control block pointed to by the second word of the parameter list. This block lives below the bar.

■ SLIP SET,IF,D,L=(MYLOAD,A00,AFF),ID=SLP1,A=TRACE,
TRDATA=(STD,REGS,1G!+0,+F,1G!+4?+10,+2F),END

SLIP SET,IF,D,L=(MYLOAD,A01,AFD),ID=SLP2,
A=IGNORE,END

SLIP MOD,ENABLE,ID=SLP*



Case 8: Notes

- This is not a violation of the rule that you can only monitor one PER range at a time.
 - Software parsing of SLIP syntax detects:
 - The **SLP2 range** is a subset of the **SLP1 range**
 - SLP2 has **A=IGNORE**
 - **PER CR10 and CR11 will hold the range from SLP1**
 - Software filtering will determine whether a PER event has occurred within the subset **range** defined by SLP2 and take action accordingly.
 - PER EVENT ignored for all but the first and last instruction
 - First and last instruction produce trace data
- The order of SLIP entry is important! SLIP software processes traps in a LIFO order, and we want it to encounter the **A=IGNORE** trap first.
- Enter both SLIPs disabled with similar IDs, use wildcarding to enable simultaneously.
- In TRDATA we must use “xG!” instead of “xR?” to perform 64-bit interpretation of the register.

A=IGNORE defaults to no match limit.

Note that an interrupt will occur on every instruction in this PER range, even though we are IGNORE-ing all but the first and last instructions!

Case 9

- What if we want to combine our actions from cases 7 and 8? Let's trace the parameter list on our first and last entries, let's trace the X'20' byte work area pointed to by Register 11 for all the in between entries, and let's take a dump and stop the GTF trace if, on exit from MYMOD, we have a non-zero value in Reg15!

- SLIP SET,IF,D,L=(MYLOAD,A00,AFF),ID=SLP1,A=TRACE,
TRDATA=(STD,REGS,1G!+0,+F,1G!+4?+C,+2F),END

SLIP SET,IF,D,L=(MYLOAD,A01,AFD),ID=SLP2,
A=(SUBTRAP,TRACE),TRDATA=(STD,REGS,11R?+0,+1F),END

SLIP SET,IF,D,L=(MYLOAD,AFE),ID=SLP3,
A=(SUBTRAP,SYNCSVCD,STOPGTF),DATA=(15R,NE,0),END

SLIP MOD,ENABLE,ID=SLP*



Case 9: Notes

- This is not a violation of the rule that you can only monitor one PER range at a time.
 - Software parsing of SLIP syntax detects:
 - The SLP2 and SLP3 RANGES are subsets of the SLP1 RANGE
 - SLP2 and SLP3 have **A=SUBTRAP**
 - **PER CR10 and CR11 will hold the RANGE from SLP1.**
 - Software filtering will determine whether a PER event has occurred within the subset RANGES defined by SLP2 or SLP3 and take action accordingly.
- Once again the order of SLIP entry is important!
- Once again we enter the SLIP set disabled, then use wildcarding to enable all simultaneously.

A=IGNORE defaults to no match limit.

Note that an interrupt will occur on every instruction in this PER range, even though we are IGNORE-ing all but the first and last instructions!



Case 10 (a breather!)

- You are suffering an abend in your **nucleus module SVC201**. An SVC dump is being produced by your recovery. However, in order to better understand the code flow leading up to the abend, you would like to **see all branches** within SVC201 **in the system trace table**. **Include the X'10' byte area pointed to by Register 6** in the trace data.
- SLIP SET,**SBT**,**N=(SVC201)**,
A=STRACE,STDATA=(6R?,+F),ML=10000,END
- **NOTES**
 - **A=STRACE** causes an SPER system trace entry to be written.
 - A maximum of X'14' bytes of data may be written in an SPER entry.
 - Default **MatchLim** for **A=STRACE** varies depending on SLIP parameters specified. For this example, the default is 50, so we code our own much larger
 - When no starting/ending offset is specified on **NUCMOD** (or LPAMOD or PVTMOD), the entire module is monitored.



Case 11

- A rogue program keeps branching into your code at offset +B0 in LPA module MYLOAD. The rogue program moves around in storage, but it has the eyecatcher 'BADGUY' at offset +0 and makes the branch to your code at offset +X'3C'. You want to abend this program whenever it branches to your code.
- SLIP SET, $\overset{\downarrow}{S}BT, \overset{\downarrow}{L}=(MYLOAD, B0), \overset{\leftarrow}{A}=RECOVERY, ML=1000, \overset{\downarrow}{D}ATA=(BEAR?-3C, EQ, C2C1C4C7E4E8), END$
- **NOTES**
 - A=RECOVERY results in the interrupted unit of work being targeted with an ABEND06F.



Are you comfortable now?



Any
Questions?