“Tell ‘em what you’re gonna tell ‘em”

- SVC First Level Interrupt Handler (FLIH)
- Scheduling a Supervisor Request Block (SVRB)
- Some Type 1 SVCs
- SVC 8 (Type 2) - Loader
- SVC 13 (Type 4) - Abend

SVC First Level Interrupt Handler

<table>
<thead>
<tr>
<th>Type</th>
<th>Runs?</th>
<th>Resident</th>
<th>Number of Loads</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Disabled</td>
<td>Yes</td>
<td>N/A</td>
</tr>
<tr>
<td>2</td>
<td>Enabled</td>
<td>Yes</td>
<td>N/A</td>
</tr>
<tr>
<td>3</td>
<td>Enabled</td>
<td>No</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>Enabled</td>
<td>No</td>
<td>&gt;1</td>
</tr>
</tbody>
</table>

Labels
- Type
- Runs?
- Resident
- Number of Loads

Types of SVCs

Our Agenda for the Week

- #2895 - Part 1: The General Purpose Computer and Interrupts
- #2896 - Part 2: From IPL to Running Programs
- #2897 - Part 3: SVCs and More SVCs
- #2898 - Part 4: Program Interrupts (You Want An Exit With That?)
- #2899 - Part 5: FLIH: I/O INTERRUPTS
- #2894 - Mainframe Operating System Boot Camp: Highlights
SVC First Level Interrupt Handler

**SVC Table**

<table>
<thead>
<tr>
<th></th>
<th>00: Type 1</th>
<th>10: Type 2</th>
<th>11: Type 3/4</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL4(SVC000)</td>
<td>00......</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AL4(SVC008)</td>
<td>10......</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AL4(SVC255)</td>
<td>11......</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Address of the SVC Module | Other SVC information

**Logic of SVC (all Types)**

- First steps of FLIH must have PSW disabled for interrupts
- On entry to FLIH, current PSW has just been swapped in
- Some SVC routines must run disabled for their full length

**SVC First Level Interrupt Handler**

**Logic of Type 1 SVCs**

- Save the 'Essence' of the interrupted program
- Determine which SVC was called and BALR R14,R6 to that module.
- Following return, store the output parameters into the appropriate TCBGRS.
- Branch to the dispatcher.

**SVC First Level Interrupt Handler**

```
IC 2,SVOPSW+3
SLL 2,3
L 7,CVTSVCTA
LA 7,IC(2,7)
L 6,IC(7)
BALR 14,6
STM 0,1,TCBGRS
ST 15,TCBGRS+15*4
L 12,CVTS0DS
BR 12
```

Get Interruption Code from OldPSW
IC*8 = index of Addr. in SVC
get loc(A(SVCMOD)) - R7 (convention)
Address of this entry in table get A(SVCMOD) - R6 (convention)
transfer control to SVCMOD
Make new values R0, R1, R15 available to user
Get A(IDISPATCH)
Return to dispatcher
Scheduling a Supervisor Request Block

Reason for Multiple Types

- Not all SVCs run enabled
- Some SVCs need to be enabled for interrupts (at all times) and to be resident in the nucleus of the operating system. Why?
- RECALL: All dispatched programs run under control of a PSW with an I/O-External mask of X'FF': enabled for interrupts.

Definition of Process

- To minimize the number of SVCs running disabled for Interrupts, run most SVCs (Types 2, 3, and 4):
  - Dispatched
  - Enabled for interrupts
  - by “Scheduling an SVRB”

Logic of SVC

- In FLIH, after obtaining A(SVC module):
- Determine SVC as Type 1 or Type 'Other'
- If (SVC is type ‘other’)
  - Acquire an SVRB
  - Point RBLINK in SVRB to PRB
  - Point TCBRB to SVRB
Scheduling a Supervisor Request Block

Before Acquiring SVRB - TCB & PRB

After Acquiring SVRB - TCB, PRB & SVRB

Logic of SVC

- Copy TCBGRS and TCBFRS from TCB into SVRB
- Set PSW in SVRB
  - I/O Enabled
  - Key 0
  - Supervisor State
  - Pointing to SVC Module
Scheduling a Supervisor Request Block

After Acquiring SVRB - TCB, PRB & SVRB

17

Logic of SVC

Scheduling a Supervisor Request Block

Logic of SVC

• For SVCs that are dispatched with SVRBs, a different restoration process is necessary:
  1. Return param registers to TCBGRS locations
  2. 'Kill' the SVRB & return to dynamic CB pool
     • We need the Type 1 SVC 3 (RB Killer) to accomplish this
     • Let’s examine four Type 1 SVCs

18

• Store 'standard' SVC FLIH registers (3, 4, 5, and 6) into TCBGRS
• Store A(CVTEXIT) into TCBGRS+(14*4).
• Branch to the Dispatcher!

19

18

19

• The SVCnnn Module will run enabled for Interrupts, in key 0, and in Supervisor State
• The Base register of all SVC Modules is R6
• All SVCs exit on R14 with return parms in R15, R0, and R1

20
Some Type 1 SVCs

SVC 0 - EXCP/XDAP

- Input parameters:
- R1: A (Input/Output Block)
- SVC 0 uses Assist-V XREAD & XPRINT in explicit format:

  \[ \text{XREAD } 0(0,Rn),(Rm) \quad \text{Rn - A (Buffer)} \]
  \[ \text{XPRINT } 0(0,Rn),(Rm) \quad \text{Rm - Buffer Length} \]
  \[ x = 0 \text{ for XREAD} \]
  \[ x = 2 \text{ for XPRINT} \]

- To proceed, perform three tests:
  - Verify the first 3 bytes of IOB are C'IOB'.
  - Verify the 2-byte Device Address is zero.
  - Verify IOBOPCDE is either X'01' or X'02'.

---

<table>
<thead>
<tr>
<th>I</th>
<th>O</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device Address</td>
<td>01 = Write</td>
<td>02 = Read</td>
</tr>
<tr>
<td>Buffer Address</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Buffer Length</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**Notes**

- SVC 0 - EXCP/XDAP
- SVC 0 uses Assist-V XREAD & XPRINT in explicit format.
- To proceed, perform three tests:
  - Verify the first 3 bytes of IOB are C'IOB'.
  - Verify the 2-byte Device Address is zero.
  - Verify IOBOPCDE is either X'01' or X'02'.

---

**Code Snippet**

```
SVC0MOD DS 0H
    Begin SVC0 Module
    USING SVC0MOD,6
    USING IO8,1
    USING CVT,3
    *
    DOEZIO SR,15,15
    L 7,IOBUFADD
    *LH 8,IOBUFLEN
    * * Check the IOB operation code
    CLI IOBOPCDE,X'01'
    BE SVC0PRNT
    CLI IOBOPCDE,X'02'
    BE SVC0READ
    B SVC00119
    No other operation code is allowed
```
Some Type 1 SVCs

**SVC 0 - EXCP/XDAP**

- Assist-V’s XREAD CC
  - 0 - successful read
  - 1 - End Of File encountered
- SVC 0 provides Return Code (0 or 4) in R15

**SVC 1 & 2 - Wait & Post**

- Input Parameters, SVC 1:
  - R0 - Number of events
  - R1 - A(Event Control Block)
- Input Parameters, SVC 2:
  - R0 - Post Code in bits 2-31
  - R1 - A(Event Control Block)
Some Type 1 SVCs

SVC 1 & 2 - Wait & Post
ECBCC(1)  ECBRBA(3)

WP xx x x x AL3(RB)

SVC 1 - Wait

SVC1MOD DS 0H
USING SVC1MOD,6
USING RB,5
C 0,SVC1NE
BNE SVC1119

Some Type 1 SVCs

SVC 1 & 2 - Wait & Post

<table>
<thead>
<tr>
<th>WP</th>
<th>Action</th>
<th>WP</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>Change WP to '10'</td>
<td>0 0</td>
<td>Change WP to '01'</td>
</tr>
<tr>
<td></td>
<td>A(RB) from R5</td>
<td></td>
<td>No change to RBWCF</td>
</tr>
<tr>
<td></td>
<td>ST AL3(RB) in ECB+1</td>
<td></td>
<td>Go to set Post Code in bits 2-31 of ECB</td>
</tr>
<tr>
<td></td>
<td>RBWCF = RBWCF + 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1</td>
<td>No-Op</td>
<td>0 1</td>
<td>No-Op</td>
</tr>
<tr>
<td>1 0</td>
<td>S301 Abend</td>
<td>1 0</td>
<td>Change WP to '01'</td>
</tr>
<tr>
<td></td>
<td>L Rn from ECB &amp; Using RB, Rn</td>
<td></td>
<td>Go to set Post Code in bits 2-31 of ECB</td>
</tr>
<tr>
<td></td>
<td>RBWCF = RBWCF - 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>No-Op</td>
<td>1 1</td>
<td>No-Op</td>
</tr>
</tbody>
</table>

SVC 1 - Wait

...
Some Type 1 SVCs

**SVC 2 - Post**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVC2MOD 05</td>
<td>Get addressability on the SVC</td>
</tr>
<tr>
<td>A0</td>
<td>WP</td>
</tr>
<tr>
<td>IS</td>
<td></td>
</tr>
<tr>
<td>T register tests:</td>
<td></td>
</tr>
<tr>
<td>TM 0(1),X'40'</td>
<td>0 or 1 (not non-dispatchable)</td>
</tr>
<tr>
<td>BOR 14</td>
<td>11: NoOp, return to caller</td>
</tr>
<tr>
<td>IOM 15,7,1(1)</td>
<td>Get A(RB) from ECB</td>
</tr>
</tbody>
</table>

...  

**SVC 3 - Exit, RB Killer**

- No input or output parameters
- SVC 3 will "Kill" the current RB by disconnecting its CB from the TCB and returning that CB to the dynamic memory pool.
- The TCBRB+1 field is updated with the contents of the RBLINK+1.

Some Type 1 SVCs

**SVC 2 - Post**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVC2MOD 05</td>
<td>Get addressability on the SVC</td>
</tr>
<tr>
<td>A0</td>
<td>WP</td>
</tr>
<tr>
<td>IS</td>
<td></td>
</tr>
<tr>
<td>T register tests:</td>
<td></td>
</tr>
<tr>
<td>TM 0(1),X'40'</td>
<td>0 or 1 (bit 0 and 1 are overwritten)</td>
</tr>
<tr>
<td>BOR 14</td>
<td>11: NoOp, return to caller</td>
</tr>
<tr>
<td>IOM 15,7,1(1)</td>
<td>Get A(RB) from ECB</td>
</tr>
</tbody>
</table>

...  

**SVC 2 - Post**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVC2MOD 05</td>
<td>Get addressability on this RB</td>
</tr>
<tr>
<td>ST 0,0(1)</td>
<td>Get RBXCF</td>
</tr>
<tr>
<td>IC 0,0</td>
<td>decrement it by 1 and save the new value</td>
</tr>
<tr>
<td>BOR 14</td>
<td>11: NoOp, return to caller</td>
</tr>
<tr>
<td>IOM 15,7,1(1)</td>
<td>Get A(RB) from ECB</td>
</tr>
</tbody>
</table>

...  

**SVC 3 - Exit, RB Killer**

- If there are multiple RBs chained off of the TCB via the TCBRB pointer, the current RB will be "Killed", its CB will be returned to the dynamic memory pool, and the next RB in the chain will now be pointed to via the TCBRB field.
Some Type 1 SVCs

*SVC 3 - Exit, RB Killer*

*Multiple RBs*

- If the RB is the only one chained off of the TCB via the TCBRB pointer, the TCBRB field will be zeroed and when an attempt is made to dispatch this TCB, the dispatcher Job End Code will process the termination of this job.
Some Type 1 SVCs

SVC 3 - Exit, RB Killer

- If the RB just unchained was the last RB in the chain
- and the flag bit is one (force local abend otherwise)
- Then set the TCBRB to 0
- To reload GPRS code
- test to confirm flag = 0 (force local abend otherwise)
- GPRS = 0, or next RB in line return from SVC 3
- ASSIST-V: terminate simulation

Why do you kill an RB?

• Remember:
  When the dispatcher finds a TCB with its TCBRB set to zero the dispatcher goes to its 'Job End' code, ends the processing of that job and prepares to process the next job.

• You may be surprised to learn how all of your jobs have ended
• How do programmers end programs?
  • BR 14
  • R14 - a value received on entry
• Now you know how to find A(THECVT)
  • R14 - A(MYCVT+X'50'), What is there?

SVC 3

The output of an Assembler produces at least four types of "Object Module (O/M)" records: ESD, TXT, RLD, and END

These records possess the pertinent information that was contained in the Assembly source program.

A linkage editor or a loader can use the O/M to create an executable version of the program in the computer memory.
**SVC 8 - (Type 2) - Loader-Lite (LL)**

**SVC-8 Input:**
- R0 = Program Load Address
- R1 = Available Memory

**SVC-8 Output:**
- R0 = Entry Point Address
- R1 = Size of Loaded Program
- R15 = Return Code (RC), 0=OK, >0=Not!

**LL typically executes a five-step program.**
1. Use HLASM to assemble SOS SVC-8
2. Copy the SVC-8 O/M to have two O/Ms
3. Use AMBLIST to format & display O/M
4. Use IEBPTPCH to view all bytes of O/M
5. Run Assist-V. Supplied SVC-8 loads SOS SVC-8, that SVC-8 loads 2nd SOS SVC-8, that SVC-8 loads all future test programs.

---

**SVC 13 (Type 4) - Abend**

- **Entry Parameters:**
  - R1
    - #1
    - #2
    - #3
    1. Flag Byte - If Bit 0 = 1 - Dump request
    2. System ABEND Code (3 hex digits)
    3. User ABEND Code (~ 4 decimal digits)

- **Exit Parameters:** None

**SVC 13 (Type 4) - Abend**

- SVC 13 formats and writes out (USING SVC 0) the following:
  - interrupt code (ALL of R1)
  - the PSW
  - twelve bytes of memory “centered” on the A in the PSW
  - address of the first of the 12 bytes
  - GPRs and FPRs from SVRB
SVC 13 (Type 4) - Abend

- Finally, chain through all RBs and point PSWs at the CVTEXIT
- Exit SVC 13 normally with BR R14

Sample Indicative Dump (a REAL one)

A (CVTEXIT) in all PSWs to finish Abend
Sneak Preview

Part 4: Program Interrupts (You want an exit with that?)

- Program Mask: Friend or Foe?
- Program Check Abend with SVC 13: From Inside SVC-FLIH?
- SPIE or NOT?
- SVC 14 in SOS: Tougher than SPIE?
- Safe Return From Exits? Only with "Portia"!

Questions

rrannie@cs.niu.edu
m-kozomara@ti.com
www.cs.niu.edu/~rrannie