Robert Rannie — Northern Illinois University - DeKalb, IL
Milica Kozomara — Texas Instruments - Dallas, TX

Mainframe Operating Systems "Boot Camp"

The General Purpose Computer and Interrupts

Part 1

Session #2895
SHARE 112 in Austin, March 2009

About this Series

These sessions are derived from the System Programming course at NIU. This course makes extensive use of the ASSIST program (John R. Mashey, The Pennsylvania State University) and the extension of ASSIST to create ASSIST-V, An Environmental Simulator for IBM 360 Systems Software Development (Charles E. Hughes & Charles E. Pfleeger, University of Tennessee).

Both of these programs are available in the public domain. Thanks to Michael Stack they may be found at:

http://www.kcats.org/assist/ and http://www.kcats.org/assist-v/

About this Series

- Student Operating System (SOS) and other assignments
  - Assist-V emulation environment
  - Comparative programs: emulation & MVS
  - XDAP/EXCP Channel Program: MVS DASD

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

"Tell 'em what you're gonna tell 'em"

- Basic Instruction Fetch (BIF) Loop
- The Interrupt Process
- The Software Phase of Interrupt Processing

Basic Instruction Fetch (BIF) loop

- Operates in a loop
- Depends on / is controlled by PSW
- Acquires and processes every instruction to be executed by CPU
- Handles interrupts (if any) prior to every instruction fetch
- Consists of 6 steps

Basic Instruction Fetch (BIF) loop

Step 1 of 6 (Top of the Loop)

1. While ("processable" interrupts pending)
2. {
   Process highest priority Interrupt
   }
3. Proceed to Step 2

Note: To better introduce the BIF loop, we begin by assuming there are no Interrupts to be processed and we review the other parts of the loop.

Basic Instruction Fetch (BIF) loop

Step 2 of 6 (Instruction Fetch)

1. Obtain address of instruction to be fetched from PSW
2. Determine length of instruction:
   - Examine first two bits pointed to by PSW (Op. Code)

bits 0 & 1 of instruction	length of instruction
0 0	2 bytes
0 1, 1 0	4 bytes
1 1	6 bytes

Basic Instruction Fetch (BIF) loop

Step 2 of 6 (Instruction Fetch)

- Example 1 - Add
  - PSW points to instruction at X'000004'
  - We know machine code: 5A50C080

```
000000                  L       5,NUM1
000004                  A       5,NUM2
000008                  ST      5,RESULT1
00000C                  L       4,NUM3
000010                  SR      4,5
000012                  ST      4,RESULT2
```


Basic Instruction Fetch (BIF) loop

Step 2 of 6 (Instruction Fetch)

- Example 1 - Add
  - Op. Code X'5A' = B'01011010'
  - Implied length?

bits 0 & 1 of instruction	length of instruction
0 0	2 bytes
0 1, 1 0	4 bytes
1 1	6 bytes

Basic Instruction Fetch (BIF) loop

Step 2 of 6 (Instruction Fetch)

- Example 2 - 80 byte buffer
  - PSW points to "bogus" instruction at X'000016'
  - 80CL1' ' = X'40404040...'

```
00000C                  L       4,NUM3
000010                  SR      4,5
000012                  ST      4,RESULT2
000016          BUFFER  DC      80CL1' '
000066                  AR      5,4
000068                  ST      4,RESULT3
```


Basic Instruction Fetch (BIF) loop

Step 2 of 6 (Instruction Fetch)

- Example 2 - 80 byte buffer
  - "Op. Code" X'40' = B'01000000'
  - Implied Length?

bits 0 & 1 of instruction	length of instruction
0 0	2 bytes
0 1, 1 0	4 bytes
1 1	6 bytes

Basic Instruction Fetch (BIF) loop

Step 2 of 6 (Instruction Fetch)

- Example 2 - 80 byte buffer
  - X'40404040' = STH 4,64(0,4)
  - Will be considered "valid" code
  - No SOC1 - may be executed
  - Debugging Problem - It's all just bytes

Basic Instruction Fetch (BIF) loop

Step 3 of 6 (Set up ILC, Update PSW Address)

1. Set Instruction Length Code (CODE) in PSW
2. Increment address in PSW accordingly (2/4/6)

bits 0 & 1 of ILC in PSW	bytes (just fetched)
0 0	n/a
0 1	2 bytes
1 0	4 bytes
1 1	6 bytes

Basic Instruction Fetch (BIF) loop

Step 4 of 6 (Branching?)

- Not covered in this session

Basic Instruction Fetch (BIF) loop

Step 5 of 6 (Executing the Instruction)

- Probably most common place for Program Check Interrupt
- Confirms earlier point about PSW address field:

  If instruction fails where will PSW likely point?

Basic Instruction Fetch (BIF) loop

Step 6 of 6 (Return to the Top of the BIF Loop)

Please Note: The test for any pending interrupts will be made following processing of each instruction AND interrupt. It will therefore be made prior to the fetching and execution of EVERY instruction processed by the CPU.

Basic Instruction Fetch (BIF) loop

- Let us now consider the BIF loop with Interrupt Processing:
  - Remember test for pending interrupts at top of BIF loop?
  - Let's now assume there are pending interrupts to be processed

The Interrupt Process

Types of Interrupts (6)

Name of Interrupt	Example of this type of interrupt
External	Signal from an external processor. Expiration of a time period.
Supervisor Call	The executing program has issued an SVC X'xx' instruction (X'0Axx').
Program Check	The executing program has caused a program interruption (X'xxxx').
Machine Check	The executing program has encountered a hardware failure.
Input/ Output	An I/O operation has completed some portion(s) of its operation.
Restart	A switch on the processor has been activated.

The Interrupt Process

- Composed of Hardware Component and Software Component
- Hardware Component
  - 4 steps "hardwired" into CPU
    - Not programmable
    - First X'80' bytes of memory
    - Known as PSW swap

The Interrupt Process

- Low Core
  - Memory location 0 - X'7F'
  - First 4K

The Interrupt Process

The Four-Step Hardware Phase

1. Current PSW is stored into appropriate old PSW location
2. Interrupt Code (IC) is stored into appropriate 'low core' location
   - BC Mode: A(OLD PSW) + 2
   - Other Modes: refer to green card for fixed storage location

The Interrupt Process

The Four-Step Hardware Phase

3. If I/O interrupt - store Channel Status Word into X'40'
4. Load current PSW from appropriate new PSW location

The Interrupt Process

First X'80' Bytes of Memory

X'0'	IPL PSW	
	Restart New PSW	
X'8'	IPL CCW1	
	Restart Old PSW	
X'10'	IPL CCW2	
	A(MYCVT)	

X'18'	External Old PSW	
X'20'	Supervisor Call Old PSW	
X'28'	Program Check Old PSW	
X'30'	Machine Check Old PSW	
X'38'	Input/Output Old PSW	
X'40'	Channel Status Word	
X'48'	CAW	A(MYCVT)
X'50'	CLOCK	Trace Info

X'58'	External New PSW
X'60'	Supervisor Call New PSW
X'68'	Program Check New PSW
X'70'	Machine Check New PSW
X'78'	Input/Output New PSW

The Interrupt Process

First Level Interrupt Handlers (FLIH)

- New PSWs (normally) contain A(FLIH)
- FLIH - set of instructions for handling the interrupt
- An FLIH for each interrupt type supported by CPU

The Interrupt Process

First Level Interrupt Handlers (FLIH)

- FLIH PSWs:
  - Masks - disabled for I/O and External
  - SVC, Restart and Program Check can't be disabled
  - Machine Check and Program Mask Interrupts Enabled
  - Key = 0; Supervisor State; Wait/Run = Run

The Interrupt Process

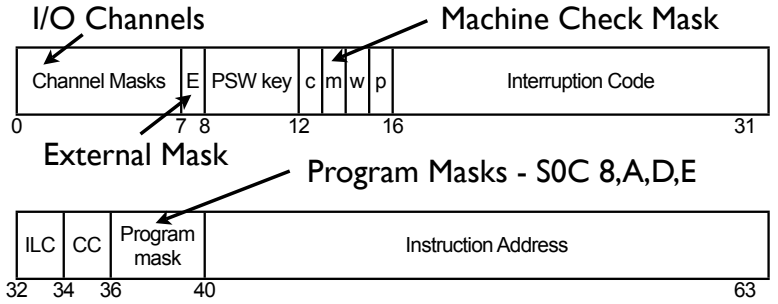
To Process or Not?

- "Masks" in PSW indicate whether or not to process a particular "class" of interrupts
  - Mask Bit = 1 - Allow/Accept Interrupts
    - referred to as "Masked On" / "Enabled"
  - Mask Bit = 0 - Don't Allow/Accept Interrupts
    - referred to as "Masked Off" / "Disabled"

The Interrupt Process

To Process or Not?

- Masks in PSW



The Interrupt Process

To Process or Not?

- Handling sequence, from high priority (1) to low priority (7)

1. Exigent Machine Check
2. Supervisor Call
3. Program Check
4. Repressible Machine Check
5. External
6. I/O
7. Restart

Note: After the processing of each one of the pending interrupts we return to the top of the BIF loop for a new evaluation of all pending Interrupt.

The Interrupt Process

To Process or Not?

- "Disposition" of interrupts which occur while corresponding type is masked off

External	Remain Pending
Supervisor Call	N/A Cannot be masked off
Program Check	N/A (for those which cannot be masked off)
Program Mask	Masked off are lost Applies to 8, A, D, E
Machine Check	Depends on nature of machine check
I/O	Remain Pending
Restart	N/A Cannot be masked off

The Interrupt Process

To Process or Not?

- New Concepts
  - PSW state => "Don't run", i.e. "Wait"
  - Privileged Instruction
    - Results in SOC2
    - Typically not encountered by Application Programmers

The Software Phase of Interrupt Processing

Beyond the PSW Swap: the "Essence" of a Program

- What is the "Essence" of a Program?
  - Registers
  - PSW
  - Memory

The Software Phase of Interrupt Processing

Role of the First Level Interrupt Handler

- Duties:
  1. Save the Essence of the Interrupted Program
  2. Direct execution to appropriate module
  3. Direct execution to code to restore Program from Essence

The Software Phase of Interrupt Processing

Fundamental Control Blocks (CBs)

- CBs involved in Essence saving
  - Communication Vector Table (CVT)
  - Task Control Block (TCB)
  - Request Block (RB)

The Software Phase of Interrupt Processing

Fundamental Control Blocks (CBs)

- Only 1 CVT
  - Address stored at X'10' and X'4C'
- TCB and RB for each unit of work
  - created in dynamically allocated memory
- CVT, TCB and RB defined in DSECTs provided by IBM/RPR
  - Some points about SOS

The Software Phase of Interrupt Processing

Characteristics of Principal Control Blocks

Control Block	DSECT Name	Location
Communication Vector Table	CVT	Low Core
TCB Pointer also 'TCBWORDS'	IEATCB	Low Core
Task Control Block	TCB	Dynamic
Request Block	RB	Dynamic

The Software Phase of Interrupt Processing

Control Block Linkage

MYCVT+0	A(TCBWORDS)
TCBWORDS+4	A(current TCB)
TCB+0	A(RB)
TCB+X'30'	A(TCBGRS)
RB+X'10'	A(RBOPSW)

The Software Phase of Interrupt Processing

How the FLIH Saves the Essence

- Register usage convention established for FLIH of SVCs

```
R3  <- A(CVT)
R4  <- A(current TCB)
R5  <- A(associated RB)
```

- Establish Addressability

```
USING CVT,3              Get addressability of CVT
USING TCB,4              Get addressability of TCB
USING RB,5               Get addressability of RB
```


The Software Phase of Interrupt Processing

How the FLIH Saves the Essence

```
STM   0,15,TCBGRS          Put callers registers into TCB
STD   0,TCBFRS             Store FPR 0
STD   2,TCBFRS+8           Store FPR 2
STD   4,TCBFRS+16          Store FPR 4
STD   6,TCBFRS+24          Store FPR 6
MVC   RBOPSW(8),SVOPSW     Put callers PSW into RB
```


The Software Phase of Interrupt Processing

After Saving the Essence

- FLIH transfers control to appropriate O/S module
- Control returns to FLIH (normally)
- FLIH restores program Essence

The Software Phase of Interrupt Processing

How to Restore the Essence

- Reverse the process.
- Assume R3 contains A(CVT)

```
L     3,0(0,3)          R3 now points to 'TCBWORDS'
ST    4,4(0,3)          store A(TCB) into TCBWORDS+4
LD    0,TCBFRS+0        Reload Floating Point Registers
                         and FPRs 2, 4, and 6
                         from TCBFRS + 8, 16, and 24
LM    0,15,TCBGRS       Reload General Purpose Registers
LPSW  RBOPSW            Reload the PSW
```


The Software Phase of Interrupt Processing

A Small Problem

- R3, R4, and R5 must be used to address CVT, TCB and RB:
  - What happened to original program values?
  - Stored in save area, maybe?
    - Then what about original value of base register used to address that save area?

The Software Phase of Interrupt Processing

Answer: Low Core!

- Why GPR 0 is ignored in D(X,B) and D(B):
  - Built-in feature
  - No register needed to address first 4K (0 through X'FFF')
- Now we have:

```
FIRST4K  CSECT                   BEGIN control seciont FIRST4K
         USING FIRST4K,0         Get addressability
*                                   NOTE: R0 is specified as base to
*                                         allow addressability to low
*                                         core without the use of a
*                                         base register.
```


The Software Phase of Interrupt Processing

Utilizing Low Core

- Any label in first 4K can be addressed implicitly
- Generated machine code uses GPR 0 as base
  - Contents of GPR 0 ignored
  - Only displacement is used
  - Largest displacement is X'FFF' = 4095

The Software Phase of Interrupt Processing

Utilizing Low Core

```
STEP 2 : Store callers registers in low-core
     STM   0,15,SVINTSAV      Store current registers in low-core

STEP 3 : Get address of CVT and the current TCB and RB
     L     3,76               R3 <- A(CVT)
     L     4,0(,3)            R4 <- A(TCBWORDS)
     L     4,4(,4)            R4 <- A(TCB)
     L     5,0(,4)            R5 <- A(RB)

STEP 5 : Move users GPs and FPs to TCB
     USING CVT,3              Get addressability of CVT
     USING TCB,4              Get addressability of TCB
     USING RB,5               Get addressability of RB

     MVC   RBOPSW(8),SVOPSW       Put callers PSW into RB
     MVC   TCBGRS(16*4),SVINTSAV  Put callers registers into TCB
```


The Software Phase of Interrupt Processing

Utilizing Low Core

- No problem reloading FPRs from TCBFRS and GPRs from TCBGRS
- Problem:
  - Addressability of RBOPSW is lost (R5 altered)
  - Can't perform LPSW
- Solution:
  - Move RBOPSW to Low Core location before destroying R5

The Software Phase of Interrupt Processing

Utilizing Low Core

The hardware feature that ignores the contents of GPR zero in address calculations enables the saving of Registers and restoring of PSWs that is ESSENTIAL to the operation of the GPC!

- That's why GPR zero is ignored in D(X,B) and D(B) addressing!

Systems Programmers must understand the logic of Interrupts on the GPC!

Sneak Preview

Part 2: From IPL to Running Programs

- What you need to RUN a program: (Basic O/S)
  - Initial Program Load (IPL)
  - Dispatcher
  - SVC-FLIH
    - SVC 1 (Wait), SVC 8 (Loader)
  - Master Scheduler

Questions

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