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/

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)

<table>
<thead>
<tr>
<th>bits 0 &amp; 1 of instruction</th>
<th>length of instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>2 bytes</td>
</tr>
<tr>
<td>0 1, 1 0</td>
<td>4 bytes</td>
</tr>
<tr>
<td>1 1</td>
<td>6 bytes</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>000000</th>
<th>L 5,NUM1</th>
</tr>
</thead>
<tbody>
<tr>
<td>000004</td>
<td>A 5,NUM2</td>
</tr>
<tr>
<td>000008</td>
<td>ST 5,RESULT1</td>
</tr>
<tr>
<td>00000C</td>
<td>L 4,NUM3</td>
</tr>
<tr>
<td>000010</td>
<td>SR 4,5</td>
</tr>
<tr>
<td>000012</td>
<td>ST 4,RESULT2</td>
</tr>
</tbody>
</table>

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...’

<table>
<thead>
<tr>
<th>00000C</th>
<th>L 4,NUM3</th>
</tr>
</thead>
<tbody>
<tr>
<td>000010</td>
<td>SR 4,5</td>
</tr>
<tr>
<td>000012</td>
<td>ST 4,RESULT2</td>
</tr>
<tr>
<td>000016</td>
<td>BUFFER DC 80CL1’’</td>
</tr>
<tr>
<td>000066</td>
<td>AR 5,4</td>
</tr>
<tr>
<td>000068</td>
<td>ST 4,RESULT3</td>
</tr>
</tbody>
</table>

Basic Instruction Fetch (BIF) loop

*Step 2 of 6 (Instruction Fetch)*

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

<table>
<thead>
<tr>
<th>bits 0 &amp; 1 of instruction</th>
<th>length of instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>2 bytes</td>
</tr>
<tr>
<td>0 1, 1 0</td>
<td>4 bytes</td>
</tr>
<tr>
<td>1 1</td>
<td>6 bytes</td>
</tr>
</tbody>
</table>

Basic Instruction Fetch (BIF) loop

*Step 2 of 6 (Instruction Fetch)*

- Example 2 - 80 byte buffer
  - Implied Length?

<table>
<thead>
<tr>
<th>bits 0 &amp; 1 of instruction</th>
<th>length of instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>2 bytes</td>
</tr>
<tr>
<td>0 1, 1 0</td>
<td>4 bytes</td>
</tr>
<tr>
<td>1 1</td>
<td>6 bytes</td>
</tr>
</tbody>
</table>
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)

<table>
<thead>
<tr>
<th>bits 0 &amp; 1 of ILC in PSW</th>
<th>bytes (just fetched)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>n/a</td>
</tr>
<tr>
<td>0 1</td>
<td>2 bytes</td>
</tr>
<tr>
<td>1 0</td>
<td>4 bytes</td>
</tr>
<tr>
<td>1 1</td>
<td>6 bytes</td>
</tr>
</tbody>
</table>

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.

The Interrupt Process

Types of Interrupts (6)

<table>
<thead>
<tr>
<th>Name of Interrupt</th>
<th>Example of this type of interrupt</th>
</tr>
</thead>
<tbody>
<tr>
<td>External</td>
<td>Signal from an external processor: Expiration of a time period.</td>
</tr>
<tr>
<td>Supervisor Call</td>
<td>The executing program has issued an SVC X’xx’ instruction (X’0Ax’).</td>
</tr>
<tr>
<td>Program Check</td>
<td>The executing program has caused a program interruption (X’xxx’).</td>
</tr>
<tr>
<td>Machine Check</td>
<td>The executing program has encountered a hardware failure.</td>
</tr>
<tr>
<td>Input/ Output</td>
<td>An I/O operation has completed some portion(s) of its operation.</td>
</tr>
<tr>
<td>Restart</td>
<td>A switch on the processor has been activated.</td>
</tr>
</tbody>
</table>

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

- 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

**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

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

---

### First X'80' Bytes of Memory

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'0'</td>
<td>IPL   PSW Restart New PSW</td>
</tr>
<tr>
<td>X'8'</td>
<td>IPL   CCW1 Restart Old PSW</td>
</tr>
<tr>
<td>X'10'</td>
<td>IPL   CCW2 A(MYCVT)</td>
</tr>
<tr>
<td>X'18'</td>
<td>External Old PSW</td>
</tr>
<tr>
<td>X'20'</td>
<td>Supervisor Call Old PSW</td>
</tr>
<tr>
<td>X'28'</td>
<td>Program Check Old PSW</td>
</tr>
<tr>
<td>X'30'</td>
<td>Machine Check Old PSW</td>
</tr>
<tr>
<td>X'38'</td>
<td>Input/Output Old PSW</td>
</tr>
<tr>
<td>X'40'</td>
<td>Channel Status Word</td>
</tr>
<tr>
<td>X'48'</td>
<td>CAW</td>
</tr>
<tr>
<td>X'50'</td>
<td>CLOCK Trace Info</td>
</tr>
<tr>
<td>X'58'</td>
<td>External New PSW</td>
</tr>
<tr>
<td>X'60'</td>
<td>Supervisor Call New PSW</td>
</tr>
<tr>
<td>X'68'</td>
<td>Program Check New PSW</td>
</tr>
<tr>
<td>X'70'</td>
<td>Machine Check New PSW</td>
</tr>
<tr>
<td>X'78'</td>
<td>Input/Output New PSW</td>
</tr>
</tbody>
</table>
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

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

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?

- Handling sequence, from high priority (1) to low priority (7)
- Exigent Machine Check
- Supervisor Call
- Program Check
- Repressible Machine Check
- External
- I/O
- 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 interrupts.

The Interrupt Process

To Process or Not?

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

<table>
<thead>
<tr>
<th></th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>External</td>
<td>Remain Pending</td>
</tr>
<tr>
<td>Supervisor Call</td>
<td>N/A Cannot be masked off</td>
</tr>
<tr>
<td>Program Check</td>
<td>N/A (for those which cannot be masked off)</td>
</tr>
<tr>
<td>Program Mask</td>
<td>Masked off are lost Applies to B, A, D, E</td>
</tr>
<tr>
<td>Machine Check</td>
<td>Depends on nature of machine check</td>
</tr>
<tr>
<td>I/O</td>
<td>Remain Pending</td>
</tr>
<tr>
<td>Restart</td>
<td>N/A Cannot be masked off</td>
</tr>
</tbody>
</table>

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

Fundamental Control Blocks (CBs)

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

Characteristics of Principal Control Blocks

<table>
<thead>
<tr>
<th>Control Block</th>
<th>DSECT Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication Vector Table</td>
<td>CVT</td>
<td>Low Core</td>
</tr>
<tr>
<td>TCB Pointer also ‘TCBWORDS’</td>
<td>IEATCB</td>
<td>Low Core</td>
</tr>
<tr>
<td>Task Control Block</td>
<td>TCB</td>
<td>Dynamic</td>
</tr>
<tr>
<td>Request Block</td>
<td>RB</td>
<td>Dynamic</td>
</tr>
</tbody>
</table>
The Software Phase of Interrupt Processing

Control Block Linkage

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MYCVT+0</td>
<td>A(TCBWORDS)</td>
</tr>
<tr>
<td>TCBWORDS+4</td>
<td>A(current TCB)</td>
</tr>
<tr>
<td>TCB+0</td>
<td>A(RB)</td>
</tr>
<tr>
<td>TCB+X'30'</td>
<td>A(TCBGRS)</td>
</tr>
<tr>
<td>RB+X'10'</td>
<td>A(RBOPSW)</td>
</tr>
</tbody>
</table>

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

- 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 ACTB into TCBWORDS+4
LD 0,TCBFRS+0  Reload Floating Point Registers
               and FPRs 2, 4, and 6
LM 0,15,TCBGRS  from TCBFRS + 8, 16, and 24
LPSW RBOPSW  Reload General Purpose Registers
            Reload the PSW
```

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

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,SWINTSAV 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 4,0C,3 R4 <- A(TCBWORDS)
L 4,4C,4 R4 <- A(TCB)
L 5,0C,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),SWINTSAV Put callers registers into TCB

• 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