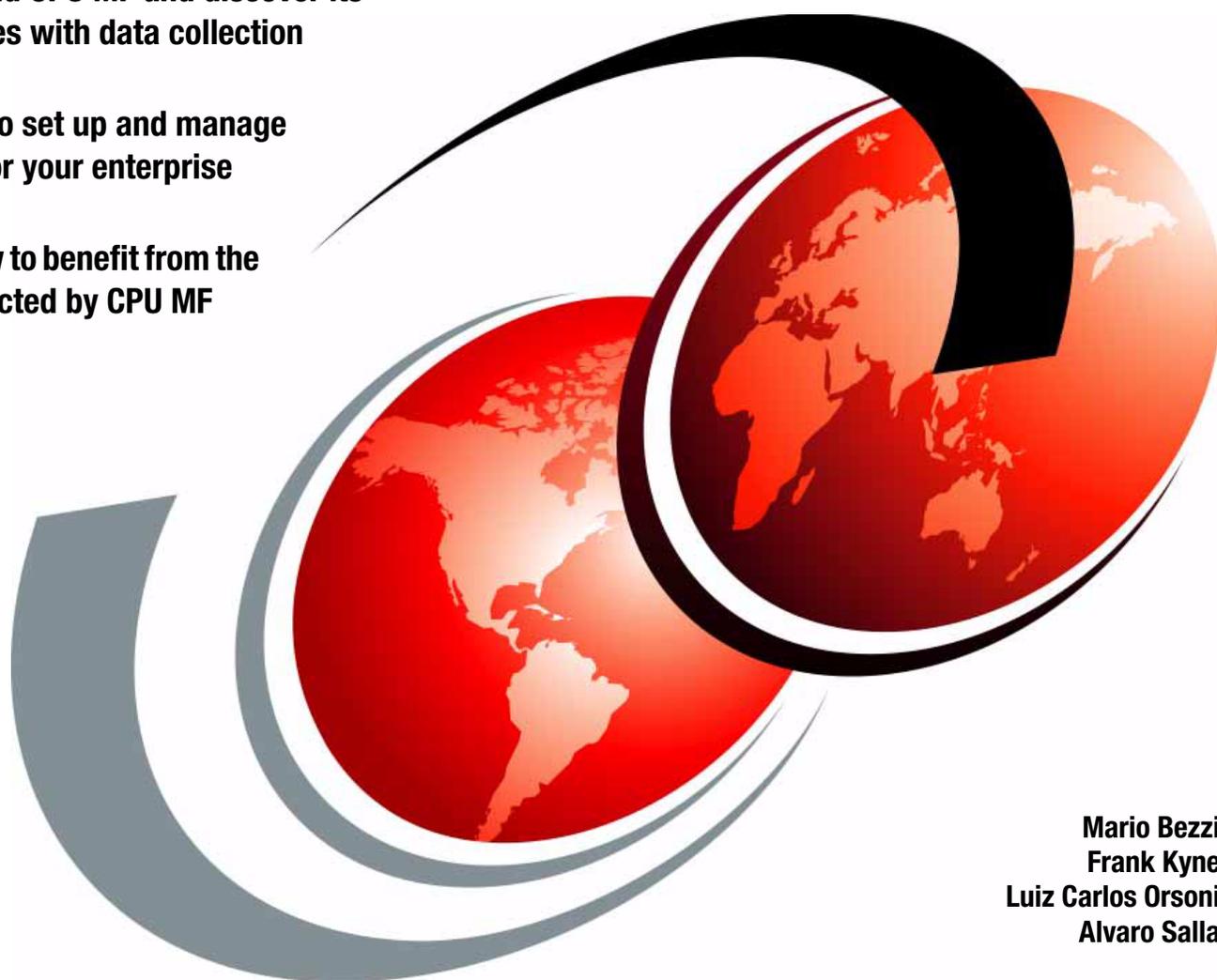


Setting Up and Using the IBM System z CPU Measurement Facility with z/OS

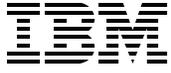
Understand CPU MF and discover its capabilities with data collection

See how to set up and manage CPU MF for your enterprise

Learn how to benefit from the data collected by CPU MF



Mario Bezzi
Frank Kyne
Luiz Carlos Orsoni
Alvaro Salla



International Technical Support Organization

**Setting Up and Using the IBM System z CPU
Measurement Facility with z/OS**

May 2011

Note: Before using this information and the product it supports, read the information in “Notices” on page v.

First Edition (May 2011)

This edition applies to Version 1, Release 12, Modification 0 of z/OS (product number 5694-A01).

This document created or updated on May 12, 2011.

© Copyright International Business Machines Corporation 2011. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	v
Trademarks	vi
Preface	vii
The team who wrote this paper	vii
Now you can become a published author, too!	viii
Comments welcome	viii
Stay connected to IBM Redbooks	viii
Chapter 1. Introduction to CPU MF	1
1.1 Working with the CPU Measurement Facility	2
1.2 CPU MF counters	2
1.2.1 Understanding the processor cache hierarchy	4
1.2.2 Using the counters data	6
1.3 Sampling with CPU MF	7
1.3.1 How sampling works	8
1.3.2 Using the sampling data	8
1.4 Controlling CPU MF	9
1.4.1 Interfacing with HIS	10
Chapter 2. Setting up and managing CPU MF data collection	11
2.1 Configuring CPU MF for data collection	12
2.2 Setting up your environment for CPU MF data collection	12
2.2.1 Prerequisites	12
2.2.2 Authorizing the collection of the CPU MF data	13
2.2.3 RACF security authorization	17
2.2.4 System Management Facilities set up	18
2.2.5 Workload Manager service class	19
2.2.6 UNIX System Services file allocation	20
2.3 Modifying the SMF archiving process	21
2.4 Initializing HIS	21
2.5 Controlling CPU MF data collection	22
2.5.1 Collecting map data for CICS programs	27
2.5.2 CPU cost of enabling CPU MF	28
Chapter 3. CPU MF counters data	31
3.1 Working with counters data	32
3.2 Understanding counters data	34
3.2.1 Contents of the CNT file	34
3.2.2 CPU MF SMF records	40
3.3 Using the counters data	42
3.3.1 Key metrics	42
3.4 Examples of using counters data	47
3.4.1 Categorizing a workload	47
3.4.2 Analyzing the impact of HiperDispatch by using CPU MF	49
3.4.3 Data in memory	50
3.4.4 Impact of 1 MB pages	50
Chapter 4. Understanding your application behavior by using CPU MF	51

4.1 Working with sampling data	52
4.2 Understanding the sampling data	52
4.2.1 Format of the sampling data	54
4.2.2 Working with the sampling data	57
4.3 Understanding map data	61
4.3.1 Exceptions	61
4.3.2 Formatting map records	63
4.3.3 MAP file usage considerations	65
4.4 Reporting on sampling data	66
4.4.1 IBM service offering	68
4.4.2 Creating your own reports	68
Appendix A. Extended counters for z10 and z196	71
Extended counters for z10	72
Extended counters for z196	74
Related publications	77
IBM Redbooks	77
Other publications	77
Online resources	77
Help from IBM	77
Index	79

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

IBM, the IBM logo, and [ibm.com](http://www.ibm.com) are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

CICS®	Redbooks®	System z®
DB2®	Redpaper™	WebSphere®
IBM®	Redbooks (logo)  ®	z/OS®
IMS™	Resource Measurement Facility™	z/VM®
MVS™	RMF™	z10™
Parallel Sysplex®	S/390®	
RACF®	System z10®	

The following terms are trademarks of other companies:

Java, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Preface

This IBM® Redpaper™ publication can help you install and manage the IBM System z® CPU Measurement Facility (CPU MF) capability. In this paper, you can learn how CPU MF gathers data and how this data can be used to more effectively manage your mainframe environment. You can also learn how this data can be used by IBM to help you more accurately characterize your workloads in preparation for capacity upgrades.

The topics covered in this paper target system programmers, capacity planners, or other technical personnel with working knowledge of systems and capacity.

The team who wrote this paper

This paper was produced by a team of specialists from around the world, working at the International Technical Support Organization (ITSO) Poughkeepsie Center.

Mario Bezzi is an IT Specialist with IBM Italy. He has 27 years of experience in Multiple Virtual Storage (MVS™) system programming, both for client support, and in various technical support positions. His areas of expertise include system performance, Parallel Sysplex®, Assembler and C language programming, system technology, and direct access storage device (DASD) hardware technology. Mario joined IBM in 1999. He now works part-time with the ITSO team in Poughkeepsie, NY, teaching and writing books.

Frank Kyne is an Executive IT Specialist and Project Leader at the IBM ITSO Poughkeepsie Center. He writes extensively and teaches IBM classes worldwide on all areas of Parallel Sysplex and high availability. Before joining the ITSO 12 years ago, Frank worked in IBM Ireland as an MVS Systems Programmer.

Luiz Carlos Orsoni is an IBM Mainframe Consultant, working with the Maffei Consulting Group in Brazil for 13 years. He has 41 years of experience in the electronic data processing field. His areas of expertise include performance analysis and capacity planning. He has written about and given classes extensively on S/390®, z Architecture, Parallel Sysplex, and performance. He holds a degree in mathematics from Faculdade de Filosofia, Ciências e Letras de Moema.

Alvaro Salla is an IBM retiree. He worked in IBM for more than 30 years, in large systems. He currently teaches ITSO workshops on z/OS® performance around the world. Alvaro coauthored many IBM Redbooks® publications and spent many years teaching about large systems, from the S/360 to the S/390. He has a chemistry engineering degree from the University of Sao Paulo, Brazil.

Thanks to the following people for their contributions to this project:

Rich Conway
Bob Haimowitz
ITSO Poughkeepsie Center

John Burg
Gary King
Bob St. John
IBM US

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and client satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our papers to be as helpful as possible. Send us your comments about this paper or other IBM Redbooks publications in one of the following ways:

- ▶ Use the online **Contact us** review Redbooks form found at:

ibm.com/redbooks

- ▶ Send your comments in an email to:

redbooks@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- ▶ Find us on Facebook:

<http://www.facebook.com/IBMRedbooks>

- ▶ Follow us on Twitter:

<http://twitter.com/ibmredbooks>

- ▶ Look for us on LinkedIn:

<http://www.linkedin.com/groups?home=&gid=2130806>

- ▶ Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>

- ▶ Stay current on recent Redbooks publications with RSS Feeds:

<http://www.redbooks.ibm.com/rss.html>



Introduction to CPU MF

In this chapter, you can find information about the CPU Measurement Facility (CPU MF), which is new capability that was initially delivered on IBM z10™ processors. It is available on both the enterprise class (EC) and business class (BC) ranges of processors. Additionally, this chapter provides a high level description of what CPU MF is, how it is controlled, the type of data that CPU MF can collect, and why you might want to use it.

This chapter includes the following topics:

- ▶ Working with the CPU Measurement Facility
- ▶ CPU MF counters
- ▶ Sampling with CPU MF
- ▶ Controlling CPU MF

Subsequent chapters include topics on how to manage and benefit from this facility and the data it provides.

1.1 Working with the CPU Measurement Facility

The CPU Measurement Facility provides optional hardware-assisted collections of information about the logical CPUs work that is executed over a specified interval in selected logical partitions (LPAR).

This powerful new capability was not available previously. However, CPU MF does not replace existing functions or capabilities, but provides an enrichment of the existing capabilities.

CPU MF consists of two important, but independent, functions:

- ▶ The collection of counters that maintain counts of certain activities
- ▶ The collection of samples that provide information about precisely what the CPU is doing at the time of the sample

The counters function is intended to be run on a constant basis to collect long-term performance data, in a similar manner to how you collect other performance data.

The sampling function is a short duration, precise function that identifies where CPU resources are being used, to help you improve application efficiency.

CPU MF runs at the LPAR level, so that you can collect counter data in one LPAR, counter and sample data in another LPAR, and not use CPU MF at all on a third LPAR. The information that CPU MF gathers pertains to just the LPARs where you enabled and started CPU MF.

The implementation of CPU MF is nondisruptive. As long as the prerequisite hardware and software are in place, you can start CPU MF data collection with no LPAR deactivations or activations. As a result, it is not necessary to perform an initial program load (IPL) on the system that CPU MF is used with.

CPU MF can run in multiple LPARs simultaneously and can be used with central processors (CPs), IBM System z Integrated Information Processor (zIIP), and IBM System z Application Assist Processor (ZAAP).

The remainder of this chapter introduces CPU MF functions and summarizes how you can use and control them, see 1.3, “Sampling with CPU MF” on page 7 and 1.4, “Controlling CPU MF” on page 9.

1.2 CPU MF counters

The type of workload that runs on a processor, such as IBM System z with Large Systems Performance Reference (LSPR), plays a large role in determining the effective relative capacity of a given model.

For more information about LSPR for System z, see “Large Systems Performance Reference for IBM System z” at:

<https://www-304.ibm.com/servers/resourceLink/lib03060.nsf/pages/lsprindex?OpenDocument>

Figure 1-1 shows how the types of work scale and have a relative capacity of 1 for a one-way processor.

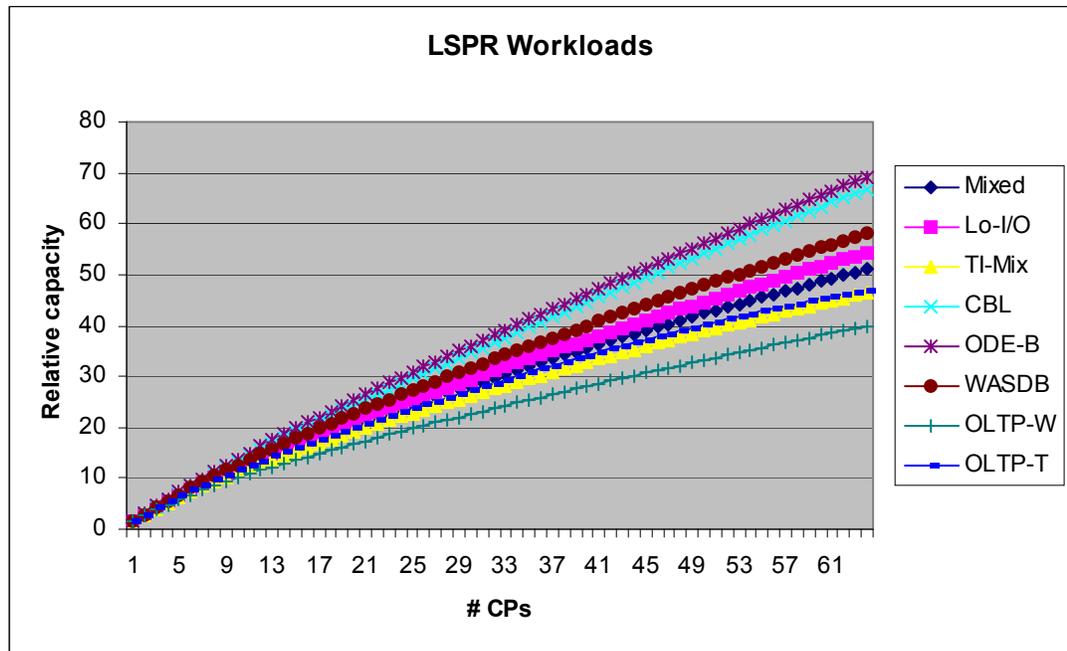


Figure 1-1 System z LSPR workloads

Based on this type of scaling, you can understand how important it is to accurately characterize the work that is running on a processor when you are considering adding capacity. Certain workloads use other methods to interact with processor hardware, causing workloads scale differently.

Historically, LSPR workload capacity curves have been named after particular application types or been identified by a software characteristic, such as the following names:

- ▶ CICS®
- ▶ IMS™
- ▶ OLTP-T
- ▶ CB-L
- ▶ LoIO-mix
- ▶ TI-mix

However, capacity performance has always been more closely associated with how a workload uses and interacts with a particular processor hardware design. The challenge has been that there was no ability to get insight into the interaction of workload and hardware design. CPU MF addresses this challenge by providing information about the interaction that was not available previously.

System z is a complex environment, and it might take time for IBM to fully integrate all the information that CPU MF can provide. The first step in the process is to produce LSPR workload capacity curves based on the underlying hardware sensitivities. The eight workload categories that were used by LSPR previously have been replaced by three new workload capacity categories. These three workload capacity categories are based largely on how the workload interacts with the cache hierarchy in the processor.

The traditional tools to help you identify which LSPR workload most closely matched your workload include Resource Measurement Facility™ (RMF™) CPU activity reports, information about the transaction, and input/ output (I/O) rates.

IBM zPCR and zCP3000 tools have been enhanced to add counters information to the metrics that are already used. These enhancements further improve the accuracy of capacity planning tools and can help you identify the *right* LSPR workload category to describe your workload.

To download a copy of the zPCR tool, see “Getting Started with zPCR (IBM's Processor Capacity Reference)” at:

<http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/PRS1381>

For more information about how you can use zPCR and other IBM tools to categorize your workload, see Chapter 3, “CPU MF counters data” on page 31.

1.2.1 Understanding the processor cache hierarchy

In an effort to achieve an acceptable balance of performance and cost, processors contain memory with various levels of performance and various levels of sharing. The memory known as *Level 1 cache* is typically the fastest, but also the most expensive.

As you move further away from the microprocessor, the cost per megabyte *decreases*, but the amount of time required to read or write to that memory from the microprocessor *increases*.

For example, if two numbers that are available in the Level 1 cache are added to a program, the instruction completes quickly. If the numbers must be retrieved from memory in the processor, it takes longer to retrieve the data, causing the instruction to run longer.

The raw hardware speed of any chip is measured in cycles per second, or more commonly, in cycles per microsecond. The duration of an instruction can be described in terms of the number of cycles that complete when the instruction is running.

Two *identical* add instructions might run for a separate number of cycles, depending on where the data needs to be retrieved from.

The number and size of the various types of memory varies from one generation to another. Figure 1-2 shows the various cache hierarchies in z10 and z196.

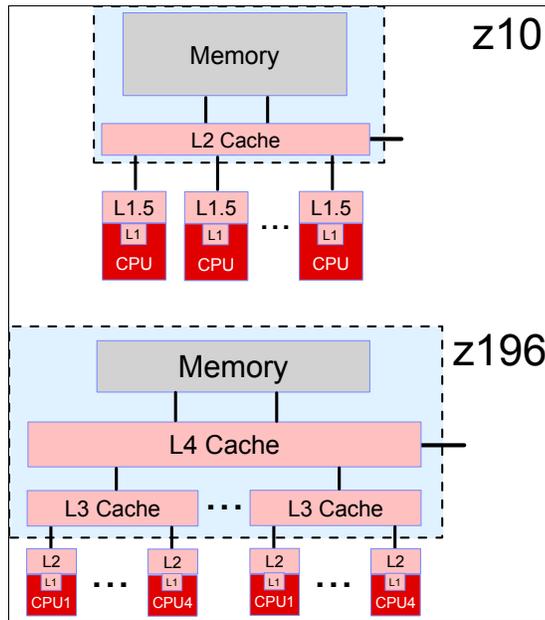


Figure 1-2 IBM z10 and z196 cache hierarchies

Given the various types and sizes of cache on z196 compared to z10, the elapsed time to complete certain types of work depends on the processor speed, and how that work interacts with the cache hierarchy.

In general, a program that finds its data and instructions in the Level 1 cache completes in less time, as compared to one that has to go higher up the cache hierarchy to find its data and instructions. For example, consider a batch job and an online transaction. The batch job is more likely to be processing data sequentially, but the online transaction is more likely to access data randomly.

Because of its sequential access, it is reasonable to expect that the next piece of data already resides somewhere within the processor memory for the batch job, than for the online transaction. Also, a batch program is more likely to execute code that loops through data, with the same instructions being executed over and over. As a result, it is more likely that the next instruction is found somewhere in a cache closer to the microprocessor, than it is for the transaction, where each instruction might only be executed one time.

Given the raw speed of modern processors, the attribute of a workload that possibly has the largest impact on how that workload performs on a given processor, is how it interacts with the memory hierarchy. Specifically, you might need to know how often a program is able to find its instructions and data in levels 1 and 2 cache for z196, or levels 1 and 1.5 cache for z10, and how often the instructions and data had to be retrieved from *the nest*. The nest consists of Level 3 or Level 4 cache or memory for z196, or Level 2 cache or memory for z10. The level of activity to the nest is called the *relative nest intensity* (RNI).

1.2.2 Using the counters data

There are two main purposes for using counters data. The first purpose is to collect counters information on a long-term basis, just as you do with other performance data, such as RMF, DB2®, CICS, and so on. Collecting counters information enables you to trend the use of your processor memory resources and to provide representative data for input to IBM capacity planning tools.

The second main purpose is for use as a secondary performance analysis tool. For example, if you detect a change in performance using RMF or DB2 Performance Expert, you can use the information in the counters data to see if there was any change in how the processor cache resources are being exploited that corresponds to the performance change you have observed. You can also use the counters data for additional insight into the effect of configuration changes, such as enabling HiperDispatch.

To understand what type of information is collected in the counters, how it helps with characterizing your workload, and what that has to do with the RNI, consider the following information:

- ▶ Currently five *types* of counters, called *counter sets*, are available.
- ▶ Counters are grouped into counter sets.
- ▶ Counter controls are defined at the counter-set level.

For example, when the operating system activates a counter set, all counters in the set are activated. Each counter set can be individually activated.

CPU MF provides the following counter sets for each logical CPU:

- ▶ Basic counter set, which includes the following details:
 - The number of executed cycles
 - The number of executed instructions
 - Level 1 cache usage information
 - The *speed*, which is presented in cycles per microsecond, of the processor that CPU MF is running on
- ▶ Problem-state counter set, which is a subset of the values in the basic counters. It contains the same information as the basic counter set, however this information only presents at the time that the logical CPU was in problem state, for example, when bit 15 is set to 1 in the *program status word* (PSW).
- ▶ A group counter set, which consists of information about a number of coprocessors that can be attached to one or more processor unit (PU). At the time of writing, the group counter is not used by CPU MF. Therefore, do not activate this function.
- ▶ Crypto-activity counter set, which contain counts and cycles about the crypto *central processor assist for cryptographic function* (CPACF). This counter set is similar to a Coprocessor-group counter set because the CPACF crypto processor is attached and shared by two PUs.
- ▶ Extended counter set, which are model dependent. They contain more detailed cache hierarchy and virtual addressing translation look-aside buffer (TLB) information than is provided in the basic or problem-state counters.

The meaning of the extended counters varies from one generation of processors to another, so any programs that process CPU MF extended counter data must take this into account when processing data from multiple systems.

For a lot of the data, particularly data that relates to the use of the processor caches, an installation can do little at the application level to change those numbers. For example, if you determine that it takes X number of cycles to complete every update to the Level 1 cache directory, you might not be able to use this information. Particularly with the high level languages used today, such as Java™, the scope for an application programmer to affect the use of the cache resources is limited.

However, the counters can be useful, if you are interested in how configuration changes, such as enabling HiperDispatch, affect the efficiency of the processor cache. Overall, the counters information is more likely to be used to see the effect of a change, rather than in advance of a change, to indicate that a change is needed. In the case of performance monitors, counters information are typically used to help detect hot spots in systems that require tuning, such as a particularly busy DASD volume.

The counters also provide another source of information that is input as part of your capacity planning calculations. IBM capacity planning tools, zPCR and zCP3000, have been enhanced to use this data.

For information about how you can use the counters data to help characterize your workloads and to gain insight into the effect that various configuration changes might have on the RNI of your workloads, see Chapter 3, “CPU MF counters data” on page 31.

1.3 Sampling with CPU MF

In today’s ultra-competitive business environment, with the relentless drive to reduce costs, every business strives to get the most work from their IT resources at the lowest cost. This means that business processes must be as efficient as possible.

Users who have been using System z since its early days might remember the efforts that programmers used to put into making their programs as efficient as possible. However as hardware and software costs decreased and the cost of labor increased, part of the focus on efficiency was shifted to making new applications available as quickly as possible, with less regard for the amount of resources they consumed.

Availability might not have been as important when transactions were executed thousands of times a day. However huge increases in business volumes mean that certain transaction programs now get executed millions or even tens of millions of times a day. With such large volumes, the savings that can potentially be made by improving the efficiency of frequently used programs can be significant.

Products are already available that gather information about application behavior and resource usage. However, by their nature, these products subtly alter the performance of the applications they are monitoring. For example, inserting counter variables into a program changes the compiler register allocation. It also increases the size of the executable, therefore potentially altering cache behavior. This strategy is appropriate for finding the most frequently executed loops, but more subtle effects can go undetected. There are also limitations related to monitoring CPU activity when the system is not enabled for interrupts.

CPU MF sampling provides the possibility to enrich the existing mechanisms for analyzing program behavior with a level of information that was not previously available. Because CPU MF sampling has a low overhead, assuming that appropriate sampling intervals are used, it is possible to collect millions of samples over a relatively short interval, meaning that the activity of even little-used code can be observed. Also, because samples can be taken even when the CPU is not enabled for interrupts, it is possible to gather information about processing that normally cannot be monitored.

1.3.1 How sampling works

The CPU MF capability is built into the PU. A new z/OS component called *hardware instrumentation services* (HIS) sets up buffers that the hardware then uses to store the sampling data. When a number of buffers are filled, the hardware generates an interrupt. This enables HIS to asynchronously collect the sampling information and save it to a file in the UNIX® file system. It also provides the ability for the samples to be gathered without the software, that is responsible for collecting the data, having to run at the highest Workload Manager (WLM) priority level.

At the end of each interval, the CPU stores sampling data into the buffers allocated by HIS. Each 32-byte sample includes the following information:

- ▶ The instruction address that is currently being executed
- ▶ The primary address space identifier number (ASID)
- ▶ Various state information about the central processor (CP), such as whether it is currently in supervisor or problem state

For both counters and sampling data, CPU MF marks the measured data as being related to problem-state or supervisor-state CPU activities so that application and system software activity can be independently identified.

The default collection frequency results in approximately 800 000 samples per minute, which provides fine-grained information about the system activity during the measurement interval.

As a result of the frequency for when samples are collected, the volume of sampling data collected by CPU MF is considerably large. An LPAR using dedicated PUs can create roughly 25 MB of sampling data per minute when using the default sampling frequency. If the LPAR has multiple logical CPUs, the samples are divided across the logical CPUs so that each CPU gathers roughly $800\,000/n$ samples, where n is the number of logical CPUs in that LPAR. The outcome is that the cost of collecting the sampling data, as a percent of total capacity, gets smaller as you add more logical CPUs to the LPAR. Partially because of the volume of data being collected, but also because its use is likely to be heavily targeted, the default measurement period when collecting sampling data is ten minutes.

1.3.2 Using the sampling data

Because it is the hardware that is responsible for collecting the samples, there is no way for the information in the sample to include the name of the program being executed. Therefore, HIS includes a capability to create a file that maps the programs that are resident at each location in virtual storage, in each address space. The *map* function optionally runs at the end of the sampling collection run and saves its information to a UNIX file.

By merging the sampling data, which contains virtual storage addresses and address space IDs, with the information in the MAP file, you can identify the programs that are the large users of CPU. Further, because the start and end addresses for each program are contained in the MAP file, you can report not only which programs are being executed, but also which parts of those programs account for most of the time. If the HIS address space has Resource Access Control Facility (RACF®) access to the data sets containing the executing programs, the MAP file includes the program names and addresses, and the names and addresses of the control sections (CSECTs) within the programs.

For more information about sampling records and how you can use them, see Chapter 4, “Understanding your application behavior by using CPU MF” on page 51.

1.4 Controlling CPU MF

CPU MF is the *hardware* side of the data collection process. The software that is responsible for retrieving the data from CPU MF, and controlling CPU MF, is the HIS component of z/OS, as shown in Figure 1-3. HIS was integrated into z/OS 1.11 and delivered back to z/OS 1.8 with an *authorized program analysis report* (APAR). For information about software prerequisites, see “Software” on page 13.

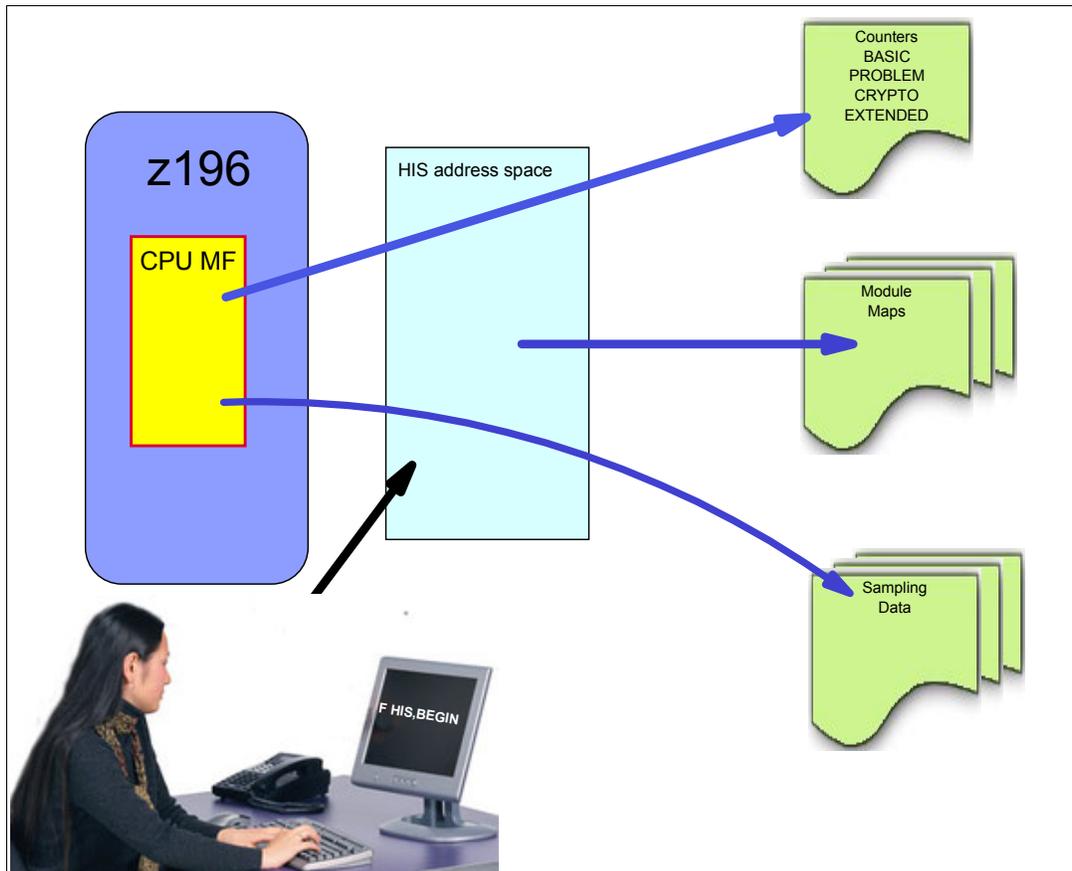


Figure 1-3 Inter-relationship between CPU MF hardware, HIS software, and HIS output files

HIS runs as a started task. Figure 1-3 shows how a MODIFY command is used to get the HIS address space to start CPU MF data collection. HIS then collects the data from the buffers and writes it to one or more UNIX files and *System Management Facilities* (SMF) records, depending on the type of CPU MF data being collected.

The authorization that determines whether a given LPAR can gather a particular counter set is controlled on the processor Support Element (SE), and is specified at the LPAR, rather than the whole processor, level. If three LPARs are sharing a single physical PU, it is possible that CPU MF collects counter information for one LPAR, counter and sampling information for another, and not doing any collections for the third LPAR. The first LPAR experiences whatever small CPU cost there is for counters collection, the second LPAR experiences the cost associated with collecting sampling information, and the third LPAR does not see any cost. When a logical CPU is in the wait state, the CPU MF collection stops for that logical CPU.

1.4.1 Interfacing with HIS

Figure 1-4 summarizes the commands you can use to start and stop HIS. Starting the HIS address space does *not* automatically start any data collection. When you are ready to start collecting data, you can use the MVS MODIFY command to pass a set of keywords to HIS, indicating what actions you require, as shown in Figure 1-4.

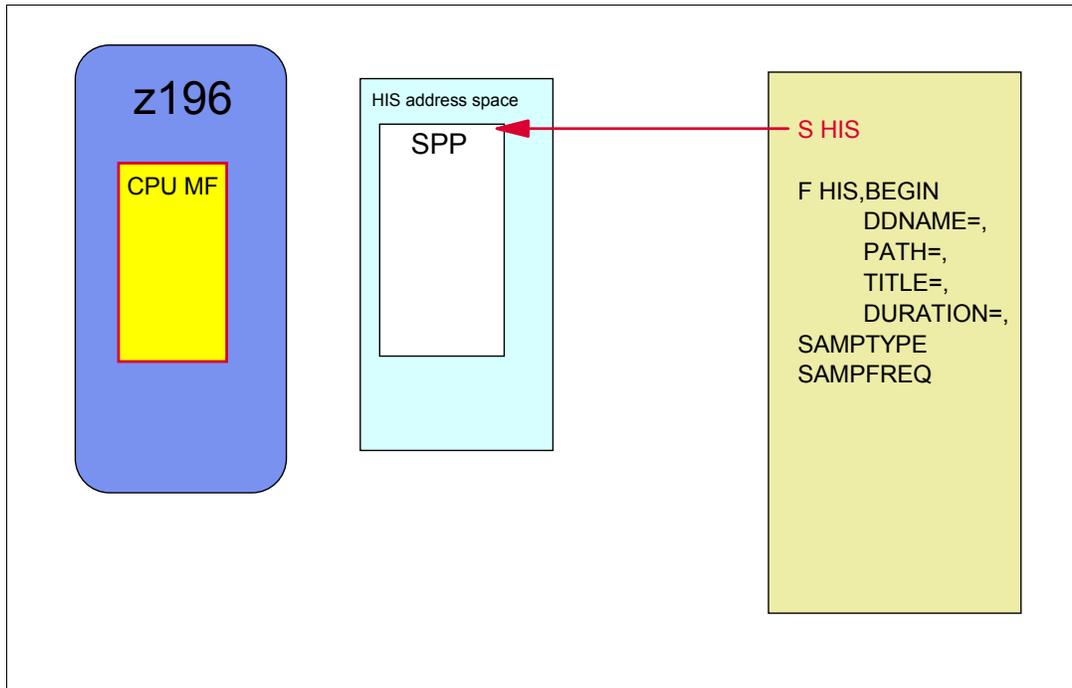


Figure 1-4 Using commands to activate CPU MF functions through HIS

You can pass the following control information to HIS:

- ▶ A data definition name (DDNAME) for a sequential data set that contains the keywords you want to pass to HIS
- ▶ A PATH pointing to the directory where you want the HIS UNIX files to be placed
- ▶ A TITLE to distinguish this collection of data from others done previously
- ▶ A DURATION in minutes that controls how long the measurement period lasts
- ▶ A type of counter or sampling data that you want to collect
- ▶ A frequency or the number of samples per minute that you want be collected

Based on the parameters passed to HIS, HIS then passes the information to the hardware to control what information is collected and where it is to be stored.

HIS externalizes the data to SMF and a file in the UNIX file system for counters data, or just to the UNIX file system for sampling and map data.

You can find more information about how to start, stop, and manage HIS in Chapter 2, “Setting up and managing CPU MF data collection” on page 11.



Setting up and managing CPU MF data collection

This chapter provides information about how you can set up and manage CPU Measurement Facility (CPU MF). It covers the changes that are required to the Hardware Management Console (HMC) and the Support Element, including the set up that is required to start the hardware instrumentation services (HIS) task in z/OS.

This chapter includes the following topics:

- ▶ Configuring CPU MF for data collection
- ▶ Setting up your environment for CPU MF data collection
- ▶ Modifying the SMF archiving process
- ▶ Initializing HIS
- ▶ Controlling CPU MF data collection

2.1 Configuring CPU MF for data collection

Setting up the environment and the processes to collect CPU MF data is simple. Many clients are provided with these instructions on one day and then have CPU MF data ready for analysis by the next morning.

As explained in Chapter 1, “Introduction to CPU MF” on page 1, the two main aspects of CPU MF are the collection of sampling data and the collection of counters data. The use of the data and the format of the data is not the same between the two types. However, all of the setup work that you need to do to collect the data is common to both types. The only difference is in the parameters you pass to the HIS started task, indicating what type of data you want it to collect. For this reason, this chapter describes the setup of the environment without reference to whether you are going to be collecting counters data, sampling data, or both.

2.2 Setting up your environment for CPU MF data collection

Setting up your environment to enable and collect the CPU MF data can be completed non-disruptively, and in little time. However, because of the type of changes that are required, such as SE customization, RACF definitions, and UNIX file system directory definitions, you might need to involve a number of people from your technical support department.

Follow these steps to set up your environment for CPU MF data collection:

1. Ensure that the prerequisite hardware and software service levels are installed.
2. Authorize the collection of CPU MF data at the logical partition (LPAR) level using the HMC or SE.
3. Define a RACF user ID for the HIS started task.
4. Ensure that SMF is set up to allow the collection of CPU MF SMF records.
5. Ensure that the HIS started task has an appropriate WLM service class.
6. Set up the UNIX file system that will contain the HIS files.
7. Modify your SMF archiving processes to save the SMF type 113 records.

2.2.1 Prerequisites

Several hardware and software prerequisites must be in place to enable CPU MF data collection.

Hardware

The CPU MF capability was initially provided on the IBM System z10[®] processors and is supported on z10 and later processors. Both EC and BC ranges are supported. The processor must be on Driver 76D, bundle 20 or later. If you are unsure of the service level of your processor, contact your IBM service representative for this information.

Software

The software side of CPU MF, HIS, was integrated in z/OS 1.11 and rolled back to z/OS 1.8 with APARs. Ensure that at least the following APARs are applied:

- ▶ OA25750
- ▶ OA25755
- ▶ OA25773
- ▶ OA27623
- ▶ OA30429
- ▶ OA30486
- ▶ OA32113
- ▶ OA34485
- ▶ PM08568 - CICS TS 3.2 support for building map information for CICS-loaded modules
- ▶ PM08573 - CICS TS V4 support for building map information for CICS-loaded modules

For a list of all the HIS-related APARs, do a search on component ID 5752SCHIS. To get a list of the APARs related to the map service, do a search for the following component IDs on:

- ▶ 5752SC142 for z/OS 1.9 and 1.10
- ▶ 5752SCPFA for z/OS 1.11 or later

The map service and the Predictive Failure Analysis (PFA) function use the same component ID. Therefore, searching on these component IDs result in a list of the PFA APARs, and the APARs for the map service.

Tip: z/OS running under z/VM® does not support CPU MF.

2.2.2 Authorizing the collection of the CPU MF data

After you apply all of the required hardware and software service levels, authorize the collection of the CPU MF data, which is specified on an LPAR basis.

Prior to making any changes to the HMC or SE, set the authority permissions for the LPAR and CPU MF, and the types of CPU MF data that is to be collected.

Authorizing the collection of CPU MF data on the SE or HMC does *not* immediately trigger the collection of that data. With the changes on the HMC or SE, only the CPU MF data collection can be turned on and off for that LPAR under the control of a program running on z/OS.

Enabling CPU MF on the HMC or SE

This section provides information about how you can authorize the collection of CPU MF data dynamically, using the SE, in a nondisruptive method. You can also authorize the collection by updating the LPAR profile on the HMC and deactivating and then reactivating the LPAR. However, this method is disruptive.

To authorize the collection of CPU MF data dynamically using the SE, log on to the SE for the processor that contains the LPAR that you want to authorize to use CPU MF, with a user ID that has SYSPROG authority.

1. Log on to the Hardware Management Console and select the processor.
2. In the Hardware Management Console, expand **Recovery** and select **Single Object Operations** (Figure 2-1).

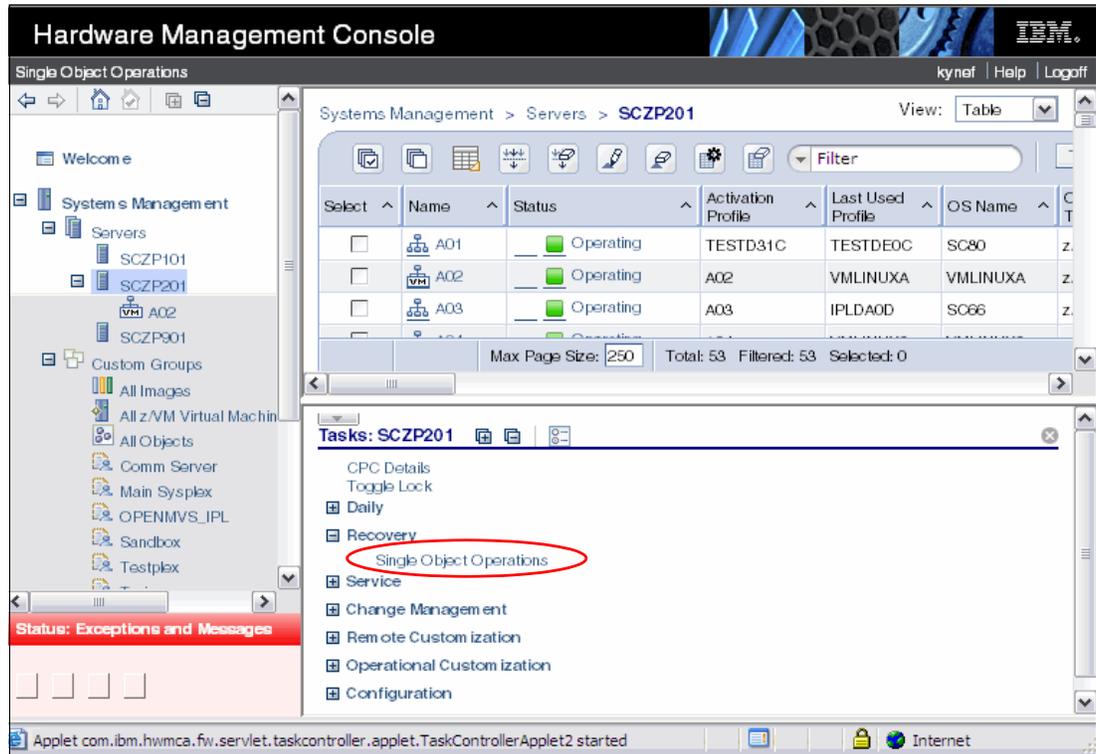


Figure 2-1 Selecting the Single Object Operations option from HMC

3. In the Support Element panel, expand **System Management** and select the processor (Figure 2-2).
4. In the bottom, right side pane of the Support Element panel, expand **CPC Operational Customization** (Figure 2-2).
5. CPU MF is protected by the LPAR security settings, therefore, select **Change LPAR Security**.

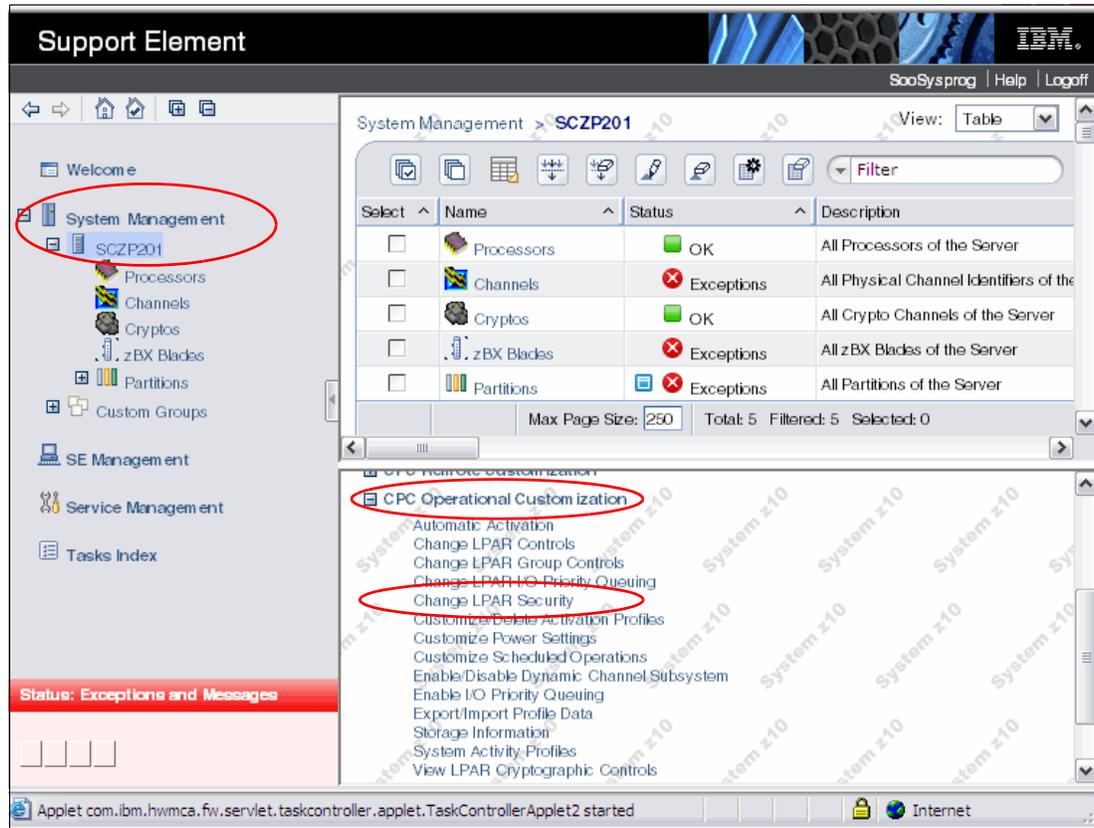


Figure 2-2 Initial SE panel

Next steps: Before proceeding to the next steps, you must understand which type of counters you want to enable for collection, if you are going to use sampling, and which LPs you want to enable for these capabilities.

- In the Change Logical Partition Security panel (Figure 2-3), select the check boxes for the combination of counters and sampling you want to use.

Change Logical Partition Security - SCZP201											
Input/output configuration data set (IOCDS): A0 IODF28											
Logical Partition	Active	Performance Data Control	I/O Config Control	Cross Partition Authority	Partition Isolation	Basic Counter	Problem State Counter	Crypto Activity Counter	Extended Counter	Group Counter	Basic Sampling
A0A	Yes	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
A0B	Yes	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
A0C	Yes	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
A0D	Yes	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
A0E	Yes	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
A0F	Yes	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
A01	Yes	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
A02	Yes	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
A03	Yes	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
A04	Yes	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
A05	Yes	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
A06	No	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
A07	Yes	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
A08	No	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
A09	No	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
A1A	No	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
A1B	No	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
A1C	Yes	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
A1D	Yes	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
A1E	Yes	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Buttons: Save and Change, Change Running System, Save to Profiles, Reset, Cancel, Help

Figure 2-3 Selecting counters and sampling data in the Change Logical Partition Security panel

After selecting the counters and sampling data, the activation profile is updated for each LPAR to reflect the new authority settings, and the change is activated dynamically for each LPAR.

- Optional: Click the **Change Running System** button to update the current instances of the LPs, but *not* update the activation profiles.
- Optional: Click the **Save to Profiles** button to update the activation profiles, but *not* change the running systems.

As shown in Figure 2-3 on page 16, five counter types are displayed on the right side of the panel, along with Basic Sampling. Authorizing the counters or basic sampling does not result in any processing until you issue the F HIS,BEGIN command in the associated LPAR. You might want to authorize the counters and sampling in all LPs before proceeding, rather than having to come back and make more changes in the future.

Group counter: At this time, IBM does not recommend enabling the Group counter.

z/OS LPs: Collect the basic and extended counters on a permanent basis, and enable those two counters for all z/OS LPs.

9. Click the **Save and Change** button to save your changes to generate the status confirmation, as shown in Figure 2-4.

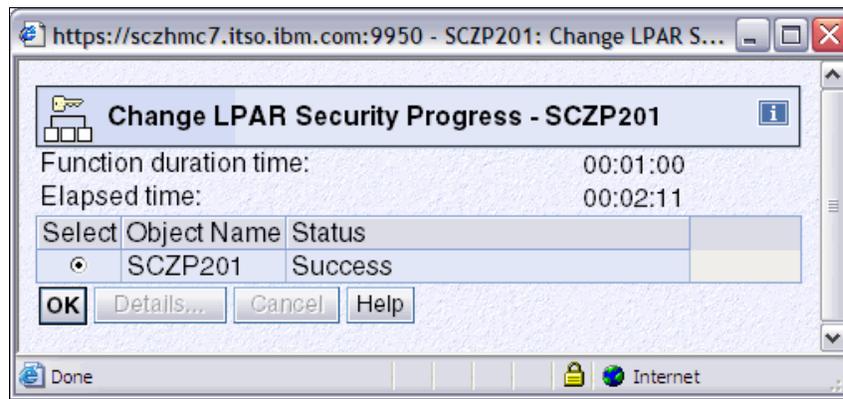


Figure 2-4 Response to Save and Change selection

At this point, all the LPs that you selected are now *authorized* to collect CPU MF data. If you did not authorize an LPAR for CPU MF counters collection and you try to start counter collection in that LPAR, you might see the following message:

```
HIS026I MODIFY HIS COMMAND CANNOT BE PROCESSED. BASIC COUNTER SET IS
UNAUTHORIZED
```

The next stage is to prepare the software side with RACF security authorization, so you can turn the collection on and off.

2.2.3 RACF security authorization

Before you can start the HIS address space, you must define a RACF user ID that the HIS address space can use. Also, you must specify a home directory for HIS. The home directory determines where the files that HIS creates for the UNIX file system are placed.

There are no special considerations for the HIS user ID, but you define the user ID for RACF, and assign that user to the HIS started task in the RACF STARTED class. The default job control language (JCL) for the HIS started task does not access any traditional Multiple Virtual Storage (MVS) data sets. Therefore, you do not need to grant the user access to any data set profiles.

A RACF command to define the HIS user ID might look similar to the following example:

```
ADDUSER hisproc DFLTGRP(sysprog) OMVS(AUTOUID HOME('/u/his'))
```

The following information provides an explanation for each segment of the command:

- ▶ `hisproc` is the name of the HIS started task.
- ▶ You can use any name that you want to for the started task. The profile in the STARTED class ties the name of the HIS started task to a RACF user ID and group.
- ▶ `sysprog` is the RACF group that the administrators that manage HIS are part of. By placing HIS in the same RACF group as those users, they are able to access the UNIX files created by HIS.
- ▶ `u/his` is the default directory where the CPU MF files are saved.

If you want to use AUTOUID in the ADDUSER command, you must set up the RACF database to support *application identify mapping*. If your RACF is not set up to support this, you must explicitly specify a unique user ID on the ADDUSER command.

During a sampling collection run, HIS attempts to gather CSECT information about the load modules that are loaded in virtual storage, if the MAPASID or MAPJOB parameter is used. In this case, additional authority is required to be able to read the data sets containing the load modules. One option is to grant RACF READ authority for the HISPROC user to the data sets and z/OS UNIX System Services directories that are used when loading modules from the following locations:

- ▶ LPA storage
- ▶ The LNKST concatenation
- ▶ A joblib/steplib/tasklib concatenation
- ▶ A concatenation identified by a program-specified data control block (DCB)

However, a more practical method of providing the right level of authorization is to specify the TRUSTED parameter in the STDATA segment of the STARTED profile for the HISPROC started task in RACF. This avoids the possibility of failure for the load modules mapping due to missing authorization. You can use a command similar to the following example to set this method up:

```
RDEFINE STARTED hisproc.* STDATA(USER(HIS) GROUP(sysprog) TRUSTED(YES))
```

2.2.4 System Management Facilities set up

If you request HIS to collect counters data, SMF Type 113 subtype 2 records are created by the HIS started task for the following instances:

- ▶ At the start of the data collection
- ▶ At 15 minute intervals, although this can be overridden using the SMFINTVAL option on the F HIS command
- ▶ Again, when you end collection

The only setup that is required is to ensure that you are not suppressing the Type 113 records. The easiest way to verify that you are not suppressing Type 113 records is to issue a D SMF,O command. The response (Example 2-1 on page 19) shows which record types are enabled or disabled for each of the following elements:

- ▶ TSO
- ▶ JES2
- ▶ STC flow
- ▶ SYS

Example 2-1 shows the response of SMF recording options.

Example 2-1 Displaying SMF recording options

```
D SMF,0
IEE967I 16.50.58 SMF PARAMETERS 371
      MEMBER = SMFPRMZ1
      SMFDLEXIT(USER3(IRRADU86)) -- DEFAULT
      SMFDLEXIT(USER2(IRRADU00)) -- DEFAULT
      MULCFUNC -- DEFAULT
      BUFUSEWARN(25) -- DEFAULT
      MEMLIMIT(00002G) -- DEFAULT
      DUMPABND(RETRY) -- DEFAULT
      SUBSYS(TSO,NOTYPE(4,5,20,34,35,40,80,99)) -- PARMLIB
      SUBSYS(TSO,NODETAIL) -- PARMLIB
      SUBSYS(TSO,INTERVAL(SMF,SYNC)) -- PARMLIB
      SUBSYS(TSO,EXITS(IEFUSI)) -- PARMLIB
      SUBSYS(JES2,NOTYPE(4,5,20,34,35,40,80,99)) -- PARMLIB
      SUBSYS(JES2,NODETAIL) -- PARMLIB
      SUBSYS(JES2,INTERVAL(SMF,SYNC)) -- PARMLIB
      SUBSYS(JES2,EXITS(IEFUSI)) -- PARMLIB
      SUBSYS(STC,NOTYPE(4,5,20,34,35,40,80,99)) -- PARMLIB
      SUBSYS(STC,NODETAIL) -- PARMLIB
      SUBSYS(STC,INTERVAL(SMF,SYNC)) -- PARMLIB
      SUBSYS(STC,EXITS(IEFUSI)) -- PARMLIB
      SYS(NOTYPE(4,5,20,34,35,40,80,99)) -- PARMLIB
```

If the system is not enabled to record Type 113 records, update the SMFPRMxx Parmlib member and issue a SET SMF=xx command to activate the new parameters. The Type 113 records do not need to be enabled for TSO, JES2, or STC, but they must be enabled on the SYS keyword in SMFPRMxx, as shown in the last line of Example 2-1.

2.2.5 Workload Manager service class

If HIS is being used solely to collect counters data, it uses little CPU time because by default, it only collects data from the buffer one time every 15 minutes. Therefore, you might assume that the WLM service class assigned to the HIS started task is not important. However, if you want the data collection to start as soon as you issue the F HIS,BEGIN command, assign HIS to a service class to ensure that it gets access to CPU resources in a timely manner when HIS needs it.

If you plan on using the sampling capability of CPU MF, remember that the default sampling interval results in the generation of 25.6 MB of data per minute. Therefore, if you use HIS to collect sampling data, ensure that the HIS started task is assigned a WLM service class that supports the ability to move the data to the UNIX files in a timely manner. If you do *not* do this, you are likely to lose samples. Example 2-2 shows the messages that you receive if sampling records are lost.

Example 2-2 Determining if sampling records are lost

```
HIS019I EVENT COUNTERS INFORMATION VERSION 1
FILE NAME: SYSHIS20100304.223617.CNT
COMMAND: MODIFY HIS,B,CTR=ALL,ST=D,DUR=18
LOST SAMPLES: 13126090
COUNTER VERSION NUMBER 1: 1   COUNTER VERSION NUMBER 2: 1
```

If the HIS started task does not have sufficient CPU resources, it is possible that sampling data is overwritten before it can be moved to disk. In this case, a data loss might occur that is reported in the CNT file. With the F HIS,BEGIN command, you can control how HIS reacts to a data loss situation. The default is to ignore the lost data and continue collecting data, but you can instruct HIS to terminate the collection if any data is lost.

The storage area used to hold the sampling data is a maximum of 1024 4 KB buffers. If you do not specify a number of buffers with the F HIS,BEGIN command, the system dynamically determines the number.

2.2.6 UNIX System Services file allocation

Regardless of which options you specify when you start CPU MF data recording, HIS always creates at least one type of UNIX file. The directory that these files are written to is controlled by the home directory associated with the HIS user ID, or through the PATH option on the F HIS,B command. The path for the UNIX file system is *not* specified anywhere in the HIS JCL. Also, remember that HIS does not do any data collection, and therefore, does not allocate any files until you issue the F HIS,B command.

Example 2-3 provides sample JCL to allocate and format a data set for the file system of the HIS directory. Customize this JCL to match your environment and naming conventions.

Example 2-3 JCL to set up directory for HIS

```
//DEFINEZ EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//AMSDUMP DD SYSOUT=*
//SYSIN DD *
        DEFINE CLUSTER (NAME(HIS.SYSA.ZFS) -
                LINEAR CYLINDERS(1000 500) SHAREOPTIONS(3) -
                STORCLAS(SCCOMP))
/*
//CREATEZ EXEC PGM=IOEAGFMT,
//          PARM=(' -aggregate HIS.SYSA.ZFS -compat
//          -owner HIS -group SYSPROG -perms o770')
//SYSPRINT DD SYSOUT=*
//STDOUT DD SYSOUT=*
//STDERR DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//CEEDUMP DD SYSOUT=*
//*
```

This job grants read, write, and execute access to the HIS user ID, and to the SYSPROG RACF group, with no access to any other ID or group.

You also must consider how the file is mounted. You can use an AUTOMOUNT policy to mount HIS.SYSA.ZFS at /u/his. If you do not use an AUTOMOUNT policy to mount the file system, you must have another mechanism to ensure that the file system is mounted at the path. The file system must be mounted either by default to the home directory of the user ID specified in the STARTED class profile, or specified in the PATH= option of the MODIFY command.

Large file sizes: These UNIX files can be large, for example, 256 MB for a default 10 minute sampling run. Considering the size, make sure that the directory you specify can handle files of larger sizes, particularly if you expect to be doing several runs.

zFS files reside in Virtual Storage Access Method (VSAM) linear data sets (LDSs). Non-extended address ability LDSs have a maximum of 4 GB. Therefore, it does not take long to fill the file if you are doing several sampling runs. For this reason, assign the file an System Managed Storage data class that has the extended format and extended address ability capabilities enabled.

The files created for the counters information are small, typically about 3 KB per processor unit (PU), regardless of the length of the measurement period.

If you are going to be running CPU MF collection in multiple systems concurrently, you must make the following actions:

- ▶ Define multiple file systems, one for each system.
- ▶ Have a unique mount point for each one.
- ▶ Mount each one on the system that writes to that file system.

Then, when you start data collection with the F HIS,BEGIN command, specify the PATH keyword to point HIS at the file system for that LPAR. This is to eliminate the performance impact of using the UNIX file system sysplex sharing support for those files.

2.3 Modifying the SMF archiving process

If you follow the IBM recommendation to run counter collection on a permanent basis, you must update your SMF archiving processes so that the Type 113 records are saved along with your other performance data.

Although RMF does not support the Type 113 records, keeping those records in the same data set as the RMF SMF records is probably the most logical repository for them.

2.4 Initializing HIS

HIS is the z/OS component that collects CPU MF hardware event data, such as counters and sampling, for IBM System z10 or later machines. HIS runs in its own address space and is started using an MVS START command.

The HIS started task, JCL, is provided in the SYS1 . IBM . PROCLIB library. If this library is not part of your standard PROCLIB concatenation, copy the HIS JCL over to a library that contains your other started tasks.

The HIS JCL provided by IBM is shown in Example 2-4. When the HIS started task is started, this does *not* automatically cause the PU to start updating counters or gathering samples. Both the PU processing to create the data, and the collection of the data from the buffers, is only started or stopped, with commands that you pass to HIS using a MODIFY HIS console command *after* the HIS started task has been started.

Example 2-4 HIS JCL provided by IBM

```
//HIS      PROC  
//HIS      EXEC PGM=HISINIT,REGION=0K,TIME=NOLIMIT
```

```

/**
/** You can specify an MVS command file to contain some or all of
/** the settings for the instrumentation run. The command file
/** must have fixed-length LRECL=80 records.
/**
/** If this option is desired,
/** 1. Replace 'DUMMY' below with the name of the MVS command
/**    file and its DISPOSITION.
/** 2. Specify the DDNAME keyword on the 'MODIFY HIS' command.
/**    For example:
/**      "MODIFY HIS,BEGIN,DDNAME=CMDFILE1"
/**
/**CMDFILE1 DD DUMMY
/**CMDFILE2 DD DUMMY
/**SYSPRINT DD SYSOUT=*
/**

```

When you are finished with your data collection, you can stop the HIS task by issuing the P HIS command, or you can leave HIS running and simply stop data collection by issuing the F HIS,END command. The name of the HIS started task is completely flexible, so you can use any valid started task name you want.

If you follow the IBM recommendation and run counters collection on a permanent basis, you must add to your system startup automation the start of the HIS started task, and the F HIS,BEGIN,CTRONLY,CTRSET=(B,E),SI=SYNC command. Similarly, your system shutdown process must be modified to add a P HIS command to stop HIS in an orderly manner, as part of the system shutdown process.

2.5 Controlling CPU MF data collection

To get HIS to start or stop collecting CPU MF data, use the MVS MODIFY command. You can either pass the information to HIS on the MODIFY command, or you can use the MODIFY command to tell HIS which DDNAME it can use to obtain the commands. Example 2-5 shows the commands that can be passed to HIS.

Example 2-5 Supported HIS commands

```

F hisproc,{BEGIN | B}
    [{TITLE | TT}='textdata']
    [PATH='pathname']
    [{DDNAME | DD}=ddname]

COUNTERS keywords:
    [{CTRONLY}]
    [{CTRSET | CTR}={ALL | (B[,P[,C[,E]]]}]
    [{DURATION | DUR}=duration_value in minutes]

SAMPLINGS keywords:
    [{BUFCNT | BUF}=bufcnt from 4 to 1024 4Kb pages]
    [{DATALOSS | DL}={IGNORE | STOP}]
    [{SAMPFREQ | SF}=freq up to 800000]
    [{SAMPTYPE | ST}=samptype either B| D]
    [{DURATION | DUR}=duration_value in minutes | 10]

```

MAPS keywords:

```
[,{MAPONLY }]  
[,{MAPASID | MAS}={ALL | (asid1,asid2,...asid32)}]  
[,{MAPJOB | MJOB}=(job1,job2,...jobn)]  
[,{MAPVERBOSE | MAPV}]
```

Misc keywords:

```
[,{SMFINTVAL | SI}={SYNC|int}]  
[,{STATECHANGE | SC}={SAVE|STOP|IGNORE}]
```

F hisproc,{END | E}

Important: You can find the latest set of HIS commands for your release of z/OS in the appropriate level of the *z/OS MVS System Commands*, SA22-7627. Check this document frequently to verify you have the latest commands, because it is possible that new releases or new APARs might add new commands. The commands in this book are presented only to provide you with information about how you can control the HIS started task processing and the types of output that are created based on which HIS commands are issued.

Example 2-5 on page 22 shows one set of keywords associated with counters, another set associated with sampling, and another set associated with the map function. Example 2-5 on page 22 also shows the BEGIN command, used to start data collection, and an END command, used to terminate data collection.

If you issue a F HIS,BEGIN command with no other parameters, HIS initiates the collection of basic and problem state counters and basic sampling data, but no MAP file is created.

If you want to *only* collect counters, or *only* sampling data, or create the MAP file, then additional keywords must be specified. Depending on which keywords you specify, HIS creates varying output.

If you request, or default to, counters collection, HIS creates the following output:

- ▶ SMF Type 113 records. One record gets created for each logical CPU that is online to the LPAR at the start of the collection period. Then, one more record is created for each logical CPU after every 15 minutes or at the end of each SMF interval if you specify that on the SMFINTVAL parameter. Finally, a record is created for each logical CPU at the end of the collection period.

The SMF records contain ever-increasing cumulative counts. To work out the count for a given interval, subtract the counts in the SMF record at the start of the interval from the values in the SMF record for the end of the interval.

If you specify CTRONLY, confirming that you only want counters to be collected, collection runs until you stop it by stopping the HIS address space or by issuing a F HIS,E command.

- ▶ A UNIX file in the directory specified on the HOME statement for the user ID that is assigned to HIS or in the PATH statement you specified on the F HIS,B command. Unlike the SMF records, which contain ever-increasing counts, and where you get one SMF record for each logical CPU, you get only one UNIX file for the counter data. This file contains the delta values for the entire collection period. It also contains the counts for all online PUs. An example of part of the contents of this file is shown in Example 2-6.

Example 2-6 Extract from counters UNIX file

```

HIS019I EVENT COUNTERS INFORMATION VERSION 1
FILE NAME: SYSHIS20100221.121354.CNT
COMMAND: MODIFY HIS,B,CTRONLY
COUNTER VERSION NUMBER 1: 1   COUNTER VERSION NUMBER 2: 1

COUNTER SET= BASIC
COUNTER IDENTIFIERS:
  0: CYCLE COUNT
  1: INSTRUCTION COUNT
  2: L1 I-CACHE DIRECTORY-WRITE COUNT
  3: L1 I-CACHE PENALTY CYCLE COUNT
  4: L1 D-CACHE DIRECTORY-WRITE COUNT
  5: L1 D-CACHE PENALTY CYCLE COUNT

START TIME: 2010/02/21 12:13:54  START TOD: C592FB9B1DB68F92
END TIME:   2010/02/21 12:20:49  END TOD:   C592FD26A2CA7092
COUNTER VALUES (HEXADECIMAL) FOR CPU 00 (CPU SPEED = 4404 CYCLES/MIC):
  0-  3 00000006663C2773 000000013562F9B1 00000000020BFA21 00000000769FBCC0
  4-  7 0000000003D2CA56 0000000026DA59633 -----
```

The files that HIS creates that contain counters data use the following naming convention:

SYSHISyyyymmdd.hhmmss.CNT

Each of the separate type of files that HIS creates has a separate low level qualifier. However, the CNT file type is always used for counters files.

If you request or default to the collection of sampling data, HIS creates one file for each active logical CPU in the system:

One file for each active logical CPU in the system. The file is created at the beginning of the collection period, but not closed until the end of the collection period. Unlike the counters file, where you only get one file regardless of how many logical CPUs are in the LPAR, for sampling, you get one file per logical CPU. HIS uses the following naming convention for these files:

SYSHISyyyymmdd.hhmmss.SMP.cpu#

Where .SMP, indicates that this is a sampling file, and cpu# is the logical CPU number in hexadecimal. Sampling data contains the addresses of the instructions being executed and the state information about the associated logical processor.

The sampling data is written out to the file continuously as the data is stored in the buffer by the PU. The sampling file is all binary data and is not in a human-readable format similar to the counters file.

The sampling data is only written to the UNIX file and is not written to SMF records.

If IBM support requests you to enable the collection of diagnostic sampling information, the resulting file names are the same, and contain both basic and diagnostic samples. However, the amount of data created are about three times the amount that can be created for an equivalent basic sample.

If you request the collection of map information, HIS creates a human-readable file containing the start and end address of every program in the MVS common area. This file contains the start and end address of every program in the private area of every swapped-in address space, depending on which options you specify. The file also contains information about the system that HIS was run on and a storage map showing the various parts of virtual storage. The virtual storage can be in the common service area (CSA), extended common service area (ECSA), nucleus, and so on. HIS uses the following naming convention for these files:

SYSHISyyyymmdd.hhmss.MAP

The first part of the file name follows the same convention as the counters and sampling files. The low level qualifier is MAP. HIS only creates one file, regardless of the number of logical CPUs in the LPAR, or the number of address spaces that were requested on the MAPASID keyword.

Example 2-7 shows an extract from the beginning of a MAP file. You can find the meaning of the various records in 4.3.2, “Formatting map records” on page 63.

Example 2-7 Extract from MAP file

```
I SYS #@$2
I SMFI#@$2
I OS z/OS
I FMIDHBB7760
I DATE10053
I TIME23392786
I MAP V1R1
I LPID00000011
I MACH00002097
B BDY PRIVATE 0000000007FFFFFF
B BDY CSA      0080000000C63FFF
B BDY CSAALLO0004C14802EEB328
B BDY CSACONVT0000000000000000
B BDY MLPA     00C6400000C64FFF
B BDY FLPA     0000000000000000
B BDY PLPA     00C6500000E35FFF
B BDY SQA      00E3600000FD5FFF
B BDY SQAALLO000E9DA801706878
B BDY RWNUC    00FD600000FE386F
B BDY RON      00FE400000FFFFFF
B BDY ERON     01000000199945F
```

HIS keyword considerations

This section provides information and considerations about the keywords that you can pass to the HIS address space.

CTRSET keyword

The CTRSET keyword controls which types of counter data are collected. The default counter sets, if you do not specify a CTRSET keyword, are basic counters and problem state counters.

However, for normal counters collection, collect basic and extended counters, using the following command:

```
CTRSET=(B,E)
```

If you plan on making configuration changes that can impact how the processor memory hierarchy is used, you might want to also start collecting the *problem state* (P) counters for a period before and after the change. For example, changes that can impact processor memory are turning on HiperDispatch and upgrading a processor.

Type 113 records: If IBM support requests Type 113 records from you, ensure that the basic and extended counters are being collected. Include B and E in the CTRSET=(a,b) parameter on the F HIS,B command.

DURATION keyword

The default value for the DURATION keyword depends on the type of data being collected. If you are collecting *only* counters data, the default is that the collection continues until you issue an F HIS,END command or stop the HIS address space.

If you are collecting *only* sampling data, *or* sampling and counters data, the default DURATION is 10 minutes, after which the collection of both counters and sampling data stops. Regardless of what you are collecting, if you explicitly specify a value on the DURATION keyword, the collection stops after that time.

Sampling data: Even if you want sampling data for more than 10 minutes, collect sampling data for 10 minutes, then capture map data. Then run for another 10 minutes and capture the map data again. The reason for this collection method is because the map data represents the end of the collection interval. If you run sampling for an extended time period and then capture the map data, the map data might not accurately reflect the samples taken at the start of the collection interval.

SAMPFREQ keyword

The SAMPFREQ keyword controls the sampling frequency. The default target value is 800 000 samples per minute.

Sampling frequency: It is important to understand the relationship between the sampling frequency and the number of online logical CPUs in the LPAR, and the percentage of time that those logical CPUs are dispatched on physical CPUs.

For example, if you use the default sampling frequency of 800 000 in an LPAR with just one dedicated CPU, you can expect to receive approximately 800 000 samples in one minute.

If the LPAR has 10 dedicated CPs, you might receive approximately 800 000 samples per minute, with about 80 000 samples per logical CPU.

If the LPAR is using shared engines, the number of samples you might get is determined by what percent of the time each logical CPU is dispatched on a physical CPU. If the LPAR has one logical CPU, and that logical CPU is dispatched 50% of the time, then you can expect to get about 400 000 samples per minute.

As a result, for a given sampling frequency, the overhead of running sampling, as a percentage of the total available capacity, gets lower as the number of logical CPUs increases. It also means that the granularity of sampling, for a given sampling frequency, decreases as the number of logical CPUs increases.

If the processor is one of the smaller business class (BC) machines, you might want to specify a smaller SAMPFREQ value, possibly as low as 20,000 for a small BC. For example, if your processor is half the speed of a 1-way full capacity model, then specify a SAMPFREQ of 400 000. The sampling frequency does not have to reflect the exact relative capacity of your processor compared to a IBM z10 EC (2097-701) processor. However, specifying a smaller SAMPFREQ helps keep the overhead at acceptable levels for smaller models.

Special considerations for the online or offline status of logical CPUs are as follows:

- ▶ Sampling data is only collected for logical CPUs that are online to the LPAR at the beginning of the hardware data event collection run. Therefore, ensure that any PUs that the system uses during the collection period are online *before* you issue the F HIS,BEGIN command.
- ▶ The handling of online or offline logical CPUs is not the same for counters data, because a counters run might last for many hours or days. However, a sampling collection typically only run for minutes. Consider the following information when working with a sampling collection:
 - If a logical CPU is offline at the end of a collection run, you get no end time. However, you might get the following message:
PROCESSOR WAS CONFIGURED OFFLINE DURING RUN.
 - The start time of a logical CPU is based on the last time that CPU had its counters enabled. If the run starts at 12:00 with an SMF interval of 15 minutes, the following time intervals apply:
 - If CPU 1 is offline at the start and brought online at 12:05, the start time for CPU 1 is 12:15.
 - If CPU 2 is online at the start, brought offline at 12:10, and then brought back online at 12:20, the start time for CPU 2 is 12:30.
 - If CPU 3 is online at the start, brought offline at 12:10, and brought back online at 12:13, the start time for CPU 3 is 12:15.
 - For CPU 2 and 3, the data that was collected between 12:00 and 12:10 is lost.
 - If Intelligent Resource Director (IRD) takes a logical CPU offline, naturally, collection for that logical CPU stops at that point. If IRD brings the CPU back online, collection does not resume until the next time HIS prepares to create the next Type 113 SMF record. For more information, see “Start, configure, and stop hardware event data collection” in *z/OS MVS System Commands, SA22-7627*.

2.5.1 Collecting map data for CICS programs

CICS APARs PM08568 and PM08573 add the ability to create information about programs that have been loaded by the CICS loader domain in a format that is compatible with the map output from HIS.

To get the map, use the new CICS CLDM transaction. The transaction sends its output either to a file in a UNIX file system or to SYSOUT, depending on the parameter used with the transaction. If the output is sent to a file, the UNIX file is placed in the file system that is specified on the PATH parameter. The file name is in the following format:

```
SYSHISTyyyymmdd.hhmmss.asidx.jobname.CICSMAP
```

For more information about APAR PM08568, see “Special installation information for CICS TS 3.2 APAR PM08568” at:

<http://www-01.ibm.com/support/docview.wss?uid=swg21449741>

2.5.2 CPU cost of enabling CPU MF

One of the most common questions in relation to any performance monitoring or profiling tool is “What is the overhead?” The CPU cost of CPU MF depends on what type of information you are gathering.

For counters data, the CPU cost is so small as to be unmeasurable. Therefore, turn on the collection of counters data in all LPs on a permanent basis.

For the sampling data, the cost is a little higher. The IBM guideline is 1 to 2%. That is based on the default sampling frequency of 800 000 samples a minute, and is based on a 2097-701. The CPU cost can vary from that guideline of 1 to 2% if one of the following situations apply:

- ▶ You change the sampling frequency. For example, if you specify a target frequency of 400 000 samples a minute, you can expect a cost of .5 to 1%.
- ▶ You have more than one CPU in the LPAR. As the number of logical CPUs in the LPAR increases, the overhead, as a percentage of the total capacity of the processor, decreases from these values.
- ▶ The CPU speed is not the same as for a 2097-701. The overhead value is related to the speed of the CPU. For a fixed sampling interval, the overhead is lower on a faster CPU and higher on a slower CPU. In order to stay within the expected CPU cost, you might want to reduce the sampling frequency, if your system is running on a z10 BC model. If your system is running on a z196, with a higher CPU speed, you can use the default frequency and have a lower CPU cost, or you can run at a higher sampling frequency at the same cost.

Table 2-1 contains values to help you determine a sampling rate for other models of z10 that can result in a similar overhead value.

Table 2-1 Suggested sampling rates for various z10 model

Model	Sampling rate per minute
2097-701	800,000
2097-601	550,000
2097-501	405,000
2097-401	190,000
2098-Z01	580,000
2098-Y01	530,000
2098-X01	470,000
2098-W01	420,000
2098-V01	370,000
2098-U01	330,000
2098-T01	290,000
2098-S01	265,000
2098-R01	230,000
2098-Q01	210,000
2098-P01	190,000

Model	Sampling rate per minute
2098-O01	165,000
2098-N01	145,000
2098-M01	130,000
2098-L01	110,000
2098-K01	100,000
2098-J01	85,000
2098-I01	80,000
2098-H01	70,000
2098-G01	65,000
2098-F01	50,000
2098-E01	50,000
2098-D01	40,000
2098-C01	35,000
2098-B01	30,000
2098-A01	20,000



CPU MF counters data

This chapter provides information about the contents of the CPU MF counters data and how you can use that information.

This chapter includes the following topics:

- ▶ Working with counters data
- ▶ Understanding counters data
- ▶ Using the counters data
- ▶ Examples of using counters data

3.1 Working with counters data

As discussed in Chapter 1, “Introduction to CPU MF” on page 1, one of the two types of data that CPU MF provides is called counters data. This chapter provides information about the fields in the counters data and the various locations that the data is written to. You can also find information about this function and examples of how the counters data can be used in this chapter.

It is beyond the scope of this paper to provide details about the cache hierarchy implementation in all the generations of System z processors and how that cache is managed. However, to fully understand the information presented in the CPU MF counters, it is valuable to gain an understanding of the cache hierarchy in the processor generations that you operate. For more information, consult the following resources:

- ▶ For z10, see the IBM Journal of Research and Development, Volume 53, Number 1. In particular, see the articles “IBM System z10 performance improvements with software and hardware synergy” and “IBM System z10 processor cache subsystem microarchitecture”. You can find the IBM Journal of Research and Development at the following web address:

<http://www.research.ibm.com/journal/rdindex.html>

- ▶ For z196, see *IBM zEnterprise System Technical Guide*, SG24-7833.

z196: Currently, there is a plan to publish an issue of the IBM Journal of Research and Development that is focused on the z196.

At a high level, the ideal situation is for data and instructions to be sourced from a cache that is as close to the microprocessor as possible. The deeper into the cache hierarchy the processor needs to go to obtain the data or instructions that it is using, the longer that process takes. Figure 3-1 shows the relative amount of time required to retrieve data from the cache and memory locations for z10 and z196.

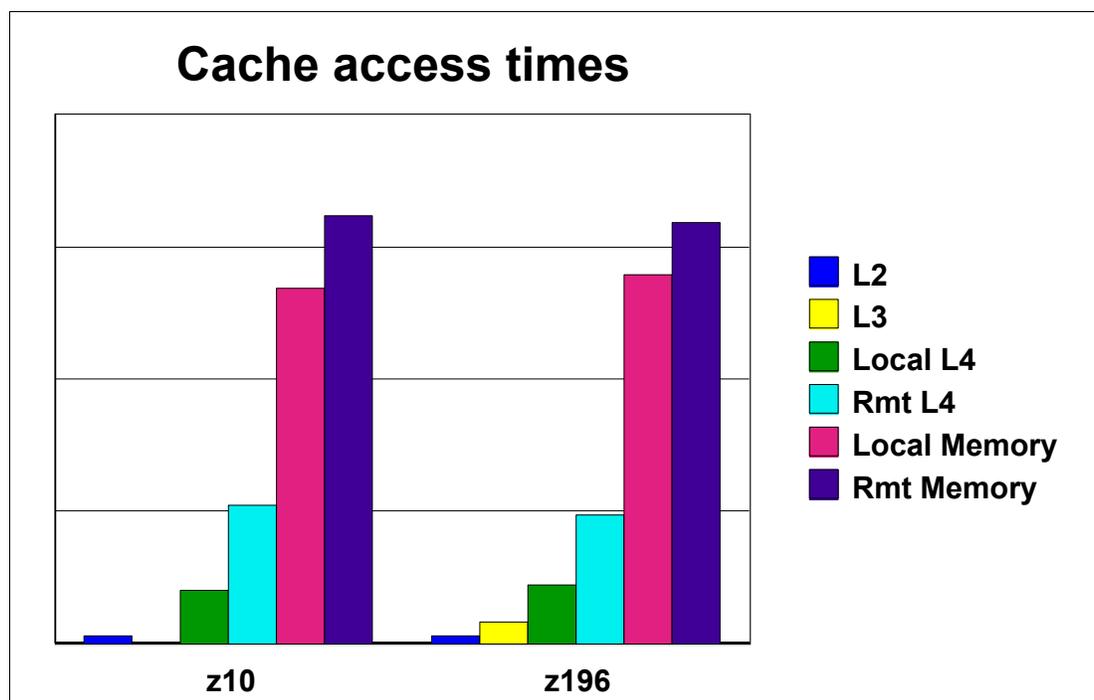


Figure 3-1 Access times to retrieve data from cache and memory

The actual access times are less important than the relative amount of time required to source data or instructions from one level of cache versus another, for example, L2 cache compared to L4 cache. The difference in access times from Local L4 cache compared to remote L4 cache on another book, is also interesting and one of the reasons why IBM introduced HiperDispatch. On processors with multiple books, the storage for each LPAR is distributed across the books, ensuring consistent average access times no matter which book a given instruction happens to be dispatched in.

If access times remain similar, but processor speeds increase, the relative cost of having to source data or instructions from deep in the memory hierarchy increases, if all other factors are equal. This is fundamentally the same challenge experienced with DASD performance over the years, where CPU speeds are increasing at a faster pace than DASD response times are reducing. In the case of DASD, this disparity is offset by ever-larger caches in the DASD subsystems. Also, the combined size of the caches in a z196 are larger than those in a z10, as shown in Figure 3-2.

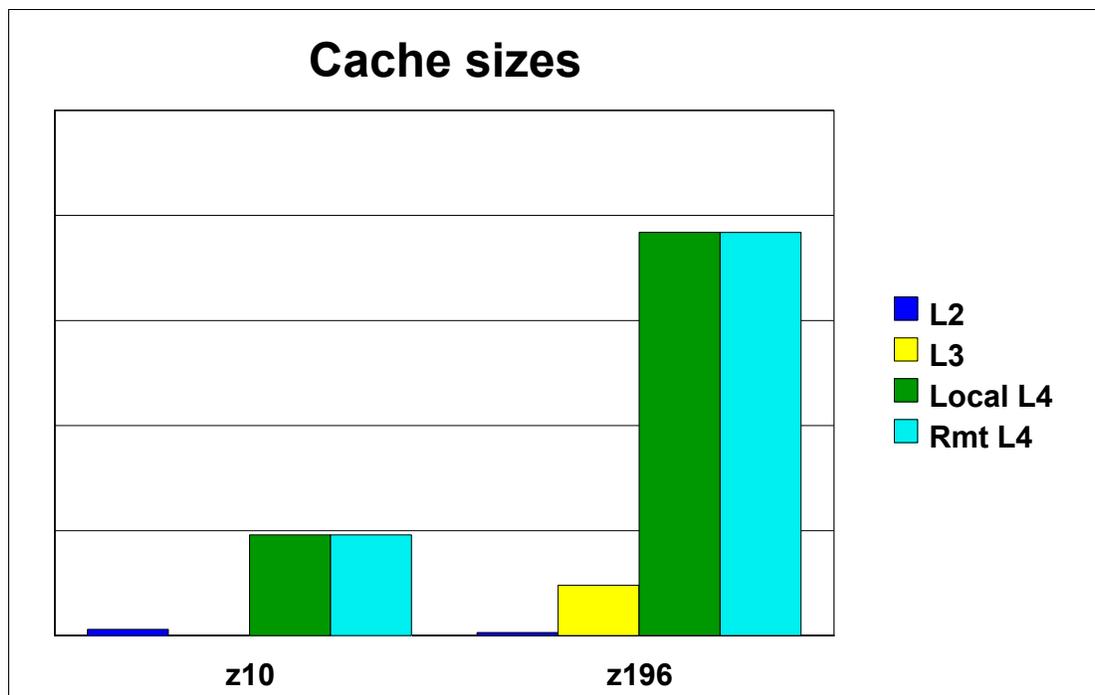


Figure 3-2 Comparison of cache sizes on z10 and z196

Given the difference in access times, you might wonder why Level 1 caches are not larger. In terms of cost, providing larger caches is not justified by the performance improvement that it can deliver. Also, space on the chip is becoming so dense that there is a physical limit to how much cache you can place on a chip, considering all the other functions that also need to be on the chip.

Driving optimum performance from a given processor requires a combination of technology, effective management, and exploitation of that technology. The information provided in the various counters, especially the extended counters, can help you see how configuration changes that you make impact how effectively the processor cache hierarchy is being exploited. In general, you want to see changes that you make result in a higher percentage of data and instructions being sourced from the lower numbered caches, such as L1, L2, and L3. You also want to see a smaller percentage being sourced from L4 and memory.

3.2 Understanding counters data

Unlike the CPU MF sampling data which is written only to the UNIX file system, the counters data is written to both SMF records and the UNIX file system. You can control whether or not the counters data is created. However, if you specify that you want the counters data or default to the counters data being collected, you cannot stop the creation of both the SMF records and the UNIX file.

One UNIX file is created for the entire collection period and covers all logical CPUs. The file name is in the following format:

```
SYSHISyyyymmdd.hhmmss.cnt
```

The following definitions explain each segment of the file format:

SYSHIS	Is fixed
yyyy	Is the year that the collection period started
mm	Is the month when the collection period started
dd	Is the day when the collection period started
hh	Is the hour when the collection period started
mm	Is the minute when the collection period started
ss	Is the second when the collection period started
CNT	Is fixed

The information in the CNT file represents the entire measurement period.

3.2.1 Contents of the CNT file

Because it is presented in human-readable format, the CNT file is easier to use to illustrate the counters data than the SMF Type 113 records. However, the SMF data is probably easier to use if you want to process the data using a program. In this section, we describe the meaning of the fields in the CNT file. This data was collected on a z10, in a z/OS LPAR with just one logical CPU.

The first few lines in the file (Figure 3-3) are self-explanatory. The line that shows the exact command that was used to initiate the collection of the CPU MF data is provided. This output can be helpful if you are subsequently trying to understand why you did or did not get particular data in the report.

Also be aware of the meaning of the COUNTER VERSION NUMBER fields. These fields indicate the generation of processor that the counters data was created on. In Figure 3-3, the information was captured on a z10. On a z196 at driver 86, the version number is 2. Future generations might have other version numbers.

```
HIS019I EVENT COUNTERS INFORMATION VERSION 1
FILE NAME: SYSHIS20100302.220948.cnt
COMMAND: MODIFY HIS,B,CTRONLY,CTR=ALL
COUNTER VERSION NUMBER 1: 1   COUNTER VERSION NUMBER 2: 1
```

Figure 3-3 CNT file contents (part 1 of 5)

As discussed in 1.2, “CPU MF counters” on page 2, the following counter sets, among others, are available:

- ▶ Basic counter set
- ▶ Problem-state counter set
- ▶ Group counter set (not currently being used)
- ▶ Crypto-activity counter set
- ▶ Extended counter set

All counter sets that were included in the collection run are included in the file. Within each set, information is provided for each logical CPU in the LPAR where appropriate.

You can find detailed information about the contents of the basic counters, problem-state counters, and the crypto-activity counters in *Set-Program-Parameter and CPU-Measurement Facilities*, SA23-2260.

Basic counters

The next section down in the file (Figure 3-4) provides information about the basic counters.

```
COUNTER SET= BASIC
COUNTER IDENTIFIERS:
  0: CYCLE COUNT
  1: INSTRUCTION COUNT
  2: L1 I-CACHE DIRECTORY-WRITE COUNT
  3: L1 I-CACHE PENALTY CYCLE COUNT
  4: L1 D-CACHE DIRECTORY-WRITE COUNT
  5: L1 D-CACHE PENALTY CYCLE COUNT

START TIME: 2010/03/02 22:09:48  START TOD: C59ED19632573984
END TIME:   2010/03/02 23:10:39  END TOD:   C59EDF3076465604
COUNTER VALUES (HEXADECIMAL) FOR CPU 00 (CPU SPEED = 4404 CYCLES/MIC):
  0- 3 000003C962FC79FE 0000009520177728 000000027C2A7719 0000006A4D55A056
  4- 7 000000018AFC0EED 00000198EEFF3D82 -----
```

Figure 3-4 CNT file contents (part 2 of 5)

The first line identifies this part of the CNT file as the section that contains the basic counters.

The next few lines of the CNT file (Figure 3-4) provide the meaning of each field in the counter values that appear a few lines later. The basic counter set has the following counters (0-5):

Cycle count

The number of *used* CPU cycles on this logical CPU in the measurement period.

Instruction count

The number of instructions executed by the logical CPU in the measurement period.

L1 instruction-cache directory write count

The total number of Level 1 instruction cache directory writes. A directory write means that the cache directory is updated due to the following instances:

- A cache line is brought into the cache as a result of a cache miss.
- The state of a cache line is changed.

	Cache misses might not result in a directory write. For example, cache misses caused by incorrect branch prediction might not result in a directory write, if the branch is resolved in time.
L1 instruction-cache penalty cycle count	The total number of cycles spent loading instructions into the Level 1 instruction cache.
L1 data-cache directory write count	The total number of Level 1 data-cache directory writes.
L1 data-cache penalty cycle count	The total number of cycles spent loading the Level 1 data cache.

The next two lines of the CNT file (Figure 3-4 on page 35) show the start and end date and time of the measurement period. This is followed by the counter values for each CPU. Unlike the SMF records, where you get one record for each logical CPU, the CNT file contains information for *all* the logical CPUs in the LPAR during the measurement period. This line also shows the announced speed value for the CPU in terms of cycles per microsecond. This number is the inverse of the gigahertz rating of the processor.

Figure 3-4 on page 35 shows the following values:

- ▶ The measurement period from start to end is 3651 seconds (22:09:48 to 23:10:39).
- ▶ Only 1 logical CP (CPU 00) in the LPAR.
- ▶ The average utilization of the logical CPU over the measurement period is 25.8%. The utilization is calculated as x'3C962FC79FE' cycles at 4404 cycles per microsecond equates to roughly 945 seconds, divided by 3651 seconds in the interval. You must make adjustments for shared CPUs. If this LPAR has a weight that is equivalent to 25.8% of the total of all weights and all LPs are running at full capacity, the logical CPU might be 100% from the perspective of z/OS in that LPAR.
- ▶ The number of instructions executed over the period is 640,488,535,848.
- ▶ The number of directory updates to the Level 1 instruction cache directory is 10,673,092,377.
- ▶ The number of CPU cycles that are used to load the Level 1 cache is 456,563,990,614.
- ▶ The number of updates to the data cache directory is 6,626,741,997.
- ▶ The number of cycles that are used to load the Level 1 data cache is 1,756,356,361,602.

Currently, the basic counters have the same meanings for z10 and z196 processors. However, see the latest edition of *Set-Program-Parameter and CPU-Measurement Facilities*, SA23-2260 for the most current information.

The basic counters currently have IDs of 0 to 5. The problem state counters have IDs of 32 to 37. The crypto-activity counters have IDs of 64 to 79, and the extended counters have IDs of 128 to 156. These unique IDs enable the counters to be referred to using a shortened annotation. For example, counter B0 is counter 0 in the basic counter set. Counter E128 is counter 128 in the extended counter set, and so on.

Problem-state counters

Further down the in the CNT file, the next few lines is the problem-state section. This provides essentially the same information as the basic counters, except that the values represent the subset of the measurement period when the logical CPUs were in problem state. Figure 3-5 shows the problem state counters for the same interval as the counters that were shown in Figure 3-4 on page 35.

```
COUNTER SET= PROBLEM-STATE
COUNTER IDENTIFIERS:
 32: PROBLEM-STATE CYCLE COUNT
 33: PROBLEM-STATE INSTRUCTION COUNT
 34: PROBLEM-STATE L1 I-CACHE DIRECTORY-WRITE COUNT
 35: PROBLEM-STATE L1 I-CACHE PENALTY CYCLE COUNT
 36: PROBLEM-STATE L1 D-CACHE DIRECTORY-WRITE COUNT
 37: PROBLEM-STATE L1 D-CACHE PENALTY CYCLE COUNT

START TIME: 2010/03/02 22:09:48  START TOD: C59ED19632573984
END TIME:   2010/03/02 23:10:39  END TOD:   C59EDF3076465604
COUNTER VALUES (HEXADECIMAL) FOR CPU 00 (CPU SPEED = 4404 CYCLES/MIC):
 32- 35 000001B0D9031290 000000389CFBFE46 00000000654EA890 000000198598D76E
 36- 39 000000009948AB65 0000011AA8E769C6 -----
```

Figure 3-5 CNT file contents (part 3 of 5)

Because the basic counters reflected both supervisor and problem state during the measurement period, you might expect the problem state counters to be less than the corresponding basic counters. Consider the following information for the problem state counters, as shown in Figure 3-5:

- ▶ The number of seconds of problem state CPU time over the measurement period is 422, meaning that 44.7% of the *used* CPU time is in problem state. Figure 3-5 shows that CPU 00 is busy for a total of 945 seconds over the interval.
- ▶ The number of instructions executed in problem state over the period is 243,151,928,902 out of a total 640,488,535,848 instructions (37.9%).
- ▶ The number of directory updates to the Level 1 instruction cache directory when in problem state is 1,699,653,776, compared to a total of 10,673,092,377.
- ▶ The number of CPU cycles that are used when loading the Level 1 instruction cache if in problem state is 109,615,568,750, compared to a total of 456,563,990,614 for the entire measurement period.
- ▶ The number of updates to the data cache directory when in problem state is 2,571,676,517, compared to the total of 6,626,741,997.
- ▶ The number of cycles that are used to load the Level 1 data cache is 1,214,014,515,654, compared to a total of 1,756,356,361,602 over the interval.

Crypto-activity counters

As you move down the in the CNT file, the next section contains the crypto-activity counters, as shown in Figure 3-6.

```
COUNTER SET= CRYPTO-ACTIVITY
COUNTER IDENTIFIERS:
 64: PRNG FUNCTION COUNT
 65: PRNG CYCLE COUNT
 66: PRNG BLOCKED FUNCTION COUNT
 67: PRNG BLOCKED CYCLE COUNT
 68: SHA FUNCTION COUNT
 69: SHA CYCLE COUNT
 70: SHA BLOCKED FUNCTION COUNT
 71: SHA BLOCKED CYCLE COUNT
 72: DEA FUNCTION COUNT
 73: DEA CYCLE COUNT
 74: DEA BLOCKED FUNCTION COUNT
 75: DEA BLOCKED CYCLE COUNT
 76: AES FUNCTION COUNT
 77: AES CYCLE COUNT
 78: AES BLOCKED FUNCTION COUNT
 79: AES BLOCKED CYCLE COUNT

START TIME: 2010/03/02 22:09:48  START TOD: C59ED19632573984
END TIME:   2010/03/02 23:10:39  END TOD:   C59EDF3076465604
COUNTER VALUES (HEXADECIMAL) FOR CPU 00 (CPU SPEED = 4404 CYCLES/MIC):
64- 67 0000000000000000 0000000000000000 0000000000000000 0000000000000000
68- 71 0000000000000000 0000000000000000 0000000000000000 0000000000000000
72- 75 0000000000000000 0000000000000000 0000000000000000 0000000000000000
76- 79 0000000000000000 0000000000000000 0000000000000000 0000000000000000
```

Figure 3-6 CNT file contents (part 4 of 5)

The following cryptographic-related functions are used by the System z crypto engines and the crypto counters report:

PRNG	Pseudo random number generator
SHA	Secure Hash Algorithm
DEA	Data Encryption Algorithm
AES	Advanced Encryption Standard

You can find the meaning of the fields in the crypto section in the following information:

PRNG function count	The total number of PRNG functions requested by the CPU.
PRNG cycle count	The total number of CPU cycles when the DEA/AES coprocessor is busy performing the PRNG functions requested by the CPU.
PRNG blocked function count	The total number of PRNG functions that are requested by the CPU but are blocked because the DEA/AES coprocessor is busy performing a function requested by another CPU.

PRNG blocked cycle count	The total number of CPU cycles blocked for the PRNG functions requested by the CPU because the DEA/AES coprocessor is busy performing a function requested by another CPU.
SHA function count	The total number of the SHA functions requested by the CPU.
SHA cycle count	The total number of CPU cycles when the SHA coprocessor is busy performing the SHA functions requested by the CPU.
SHA blocked function count	The total number of SHA functions that are requested by the CPU but are blocked because the SHA coprocessor is busy performing a function requested by another CPU.
SHA blocked cycle count	The total number of CPU cycles blocked for the SHA functions issued by the CPU because the SHA coprocessor is busy performing a function requested by another CPU.
DEA function count	The total number of the DEA functions requested by the CPU.
DEA cycle count	The total number of CPU cycles when the DEA/AES coprocessor is busy performing the DEA functions requested by the CPU.
DEA blocked function count	The total number of DEA functions that are requested by the CPU but are blocked because the DEA/AES coprocessor is busy performing a function requested by another CPU.
DEA blocked cycle count	The total number of CPU cycles blocked for the DEA functions requested by the CPU because the DEA/AES coprocessor is busy performing a function requested by another CPU.
AES function count	The total number of the AES functions requested by the CPU.
AES cycle count	The total number of CPU cycles when the DEA/AES coprocessor is busy performing the AES functions requested by the CPU.
AES blocked function count	The total number of AES functions that are requested by the CPU but are blocked because the DEA/AES coprocessor is busy performing a function requested by another CPU.
AES blocked cycle count	The total number of CPU cycles blocked for the AES functions requested by the CPU because the DEA/AES Counter coprocessor is busy performing a function requested by another CPU.

Figure 3-6 on page 38 shows all the counters are zero because the workload used did not use any encryption functions.

Extended counters

The final set of counters are the extended counters. These are presented in the CNT file after the crypto counters, as shown in Figure 3-7.

```
COUNTER SET= EXTENDED
COUNTER IDENTIFIERS:
  MODEL DEPENDENT INFORMATION NOT AVAILABLE

START TIME: 2010/03/02 22:09:48  START TOD: C59ED19632573984
END TIME:   2010/03/02 23:10:39  END TOD:   C59EDF3076465604
COUNTER VALUES (HEXADECIMAL) FOR CPU 00 (CPU SPEED = 4404 CYCLES/MIC):
128-131 000000024D1BFBE1 00000000D3A2C2E7 00000000277460D1 000000003B82DE1F
132-135 000000000000C281 0000000000675707 00000000447F60D4 0000000002A34E61
136-139 00000000063DA28E 00000000054ABD5D 000000002F82F908 0000000058F2CEF7
140-143 0000000025387E32 00000000027C5F89 0000000000000000 0000000000000000
144-147 0000000000000000 000000138906B076 000000483B87F284 0000002B2392344B
148-151 0000000000000000 0000000000000000 0000000000000000 0000000000000000
```

Figure 3-7 CNT file contents (part 5 of 5)

Unlike the other counters, descriptive information is not provided for the extended counters. All that is contained in the CNT file for the extended counters is the following information:

```
MODEL DEPENDENT INFORMATION NOT AVAILABLE
```

The meaning of any given extended counter depends on the generation of processor that the data came from. The meanings of the extended counters for both z10 and z196 are contained in Appendix A, “Extended counters for z10 and z196” on page 71. You can find more information about the extended counters in *IBM The CPU-Measurement Facility Extended Counters Definition for z10 and z196*, SA23-2261.

3.2.2 CPU MF SMF records

The other destination for counters data is SMF Type 113 records. Unlike the CNT file, where you only get one file containing information for all the logical CPUs for the entire collection interval, with SMF Type 113 records, you get one SMF record at the beginning of the measurement period for each online logical CPU. You then get one SMF record for each CPU every 15 minutes, by default. Finally, you get one more SMF record per CPU at the end of the measurement period. Another difference from the CNT file is that the SMF records contain cumulative counts. To work out the values for the entire measurement interval, subtract the values in the first SMF record created at the start of the measurement period, from those in the last SMF record.

Both the CNT file and the SMF records contain the same type of information. At the start of a measurement, HIS obtains the counter values from the storage buffer. It writes these values to an SMF record for each logical CPU and keeps a copy for use when it creates the CNT file at the end of the measurement period. The counter values might be zero at the start of the measurement. Every 15 minutes, or every SMF interval, if that is what you request, HIS extracts the latest counter values and saves them to SMF records. At the end of the measurement period, HIS extracts the values again and writes them to SMF. At this point it uses the original values to calculate the delta values for the measurement period, and writes these values to the CNT file.

The format of the SMF Type 113 records is shown in Figure 3-8. You can find information about the layout of these records in *z/OS MVS System Management Facilities (SMF)*, SA22-7630.

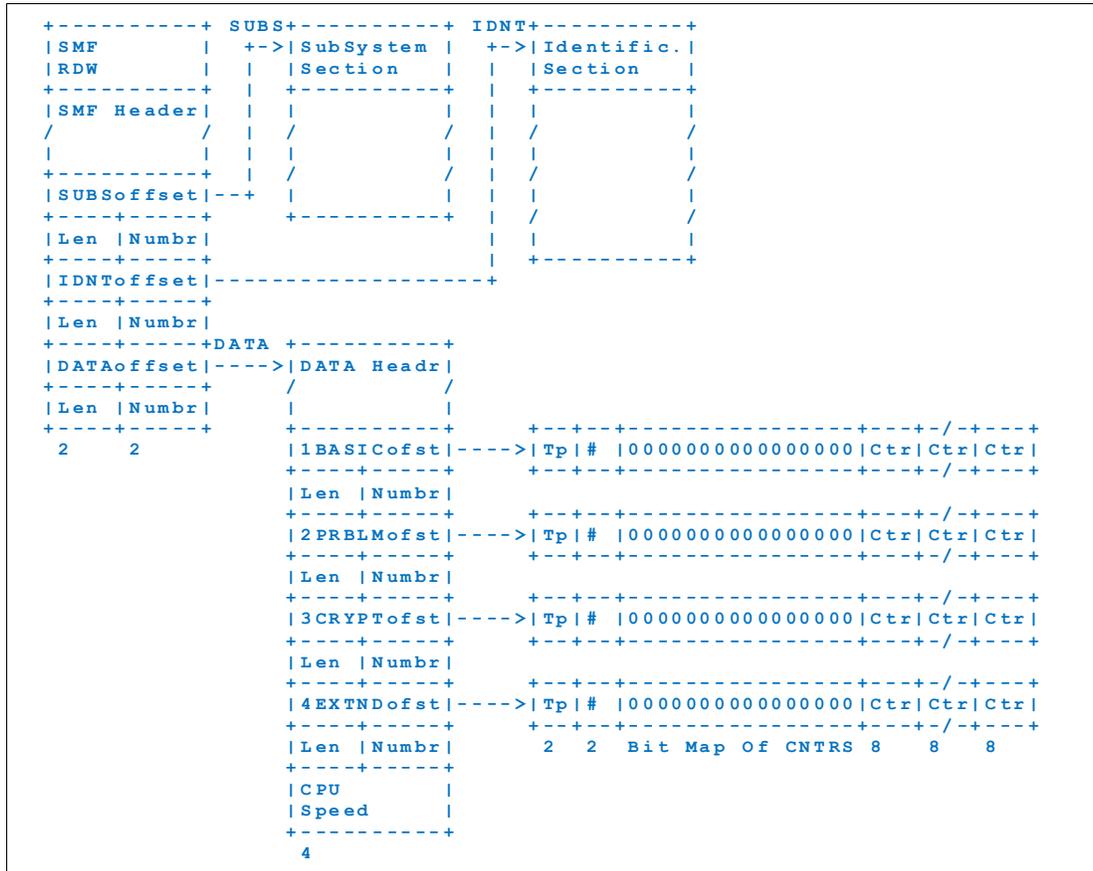


Figure 3-8 Layout of SMF Type 113 records

The layout of the records is familiar to anyone that has experience processing SMF records. The data section of the record, which immediately follows the self-defining section of the SMF record, is the part that contains the counters.

The first part of the data section is the counter set section. Each counter set section consists of 12 bytes, made up as follows:

- ▶ The first byte that indicates the counters set type:
 - 1 indicates that it is a basic counter set
 - 2 indicates that it is a problem state counter set
 - 3 indicates that it is a crypto counter set
 - 4 indicates that it is an extended counter set
- ▶ The next byte is reserved.
- ▶ The next two bytes indicate the number of counters in the counter set.
- ▶ The next eight bytes are a bit pattern, indicating which counters are being recorded.

This might be easier to understand using an example. Example 3-1 shows the first 192 bytes of a sample SMF Type 113 record. Offset x'90' in the record contains the first counter set section.

Example 3-1 IDCAMS print of first 192 bytes of SMF Type 113 record

```

RECORD SEQUENCE NUMBER - 2
000000 DE710047 69DE0110 064F7B7C 5BF2C8C9 E2400002 00000018 00000034 00140001
000020 00000048 00200001 00000068 01FC0001 0000F0F1 C8C9E240 40404040 E2D7F74B
000040 F14BF140 C8C9E240 40404040 00472E93 0110064F C8C9E240 40404040 00000000
000060 00000000 C5A21C4B BBDF2E04 C5A21C4B BCA6A504 00008000 00010001 00000094
000080 000C0004 000000C4 00080034 00001134 01000006 FC000000 00000000 02000006
0000A0 FC000000 00000000 03000010 FFFF0000 00000000 04000018 FFFFFFF0 00000000

```

Because the print Example 3-1 in was produced using IDCAMS PRINT, the first four bytes of each record are not shown, so the offsets need to be adjusted accordingly.

The first byte at offset x'90' contains 01, indicating that this information pertains to a basic counter set. The next byte, the reserved one, contains 00.

The next two bytes contain 0006, meaning that there are 6 basic counters. If you look back at Figure 3-4 on page 35 you can see that there are 6 counters in the basic counters section of the report.

The next four bytes are a bitmap, indicating which counters are included in the counters section. In Example 3-1, x'FC000000' equates to b'1111 1100 0000 0000 0000 0000 0000 0000'. This means that counters 0, 1, 2, 3, 4, and 5 are included in the SMF record. Because the SMF record contains a mapping of which counters are included in the record, it is not necessary to include blank sections for possible future counters that are not being collected. This collection results in SMF records that are not as large as can otherwise be the case.

After the counter set sections, you can find the counters sections. Each counter is eight bytes. The first 48 bytes are the six basic counters, followed by 48 bytes for the six problem-state counters, 128 bytes for the sixteen crypto counters, and so on.

3.3 Using the counters data

As shown in Figure 1-1 on page 3, the effective and relative capacity of any given processor depends on the type of work that is run on that processor. A large factor of the volume of a given type of work that can be completed by a given processor is how that workload interacts with the cache hierarchy in the processor. The counters data can provide you with insight into that interaction.

3.3.1 Key metrics

As provided in 3.2.1, "Contents of the CNT file" on page 34, the counters contain a lot of raw data. In order to easily interpret that data, you must process it down to a few key metrics. Based on experience with using this data, the following values have been determined to be the most useful:

CPI Cycles per instruction, which gives you an indication of the types of instructions being executed and how those instructions are using the memory hierarchy. Typically workloads that move a lot of data, or that require data to be sourced from deeper in the memory hierarchy, can

have a higher CPI value than simpler instructions. This also applies to workloads that source more of their instructions or data from cache resources closer to the microprocessor.

This value is calculated by dividing counter 0 (used cycles) in the basic counter set, by counter 1 (executed instructions) in the same counter set.

PRBSTATE

Percentage of problem state, which provides the percent of all executed instructions that were run if the processor was in problem state.

This value is calculated by dividing counter 33 (problem state executed instructions) in the problem state counter set, by counter 1 (executed instructions) in the basic counter set, and multiplying by 100.

LPARCPU

APPL (application) percentage captured and non-captured by GCPs, zAAPs, and zIIPs, which represents the capacity used by all logical CPUs in the LPAR, expressed as a percent of 1 CP. For example, if the LPAR has five logical CPs and each is consuming 100% of a physical CP, the LPARCPU percent is 500.

LPARCPU is calculated as follows:

$$\left(\frac{1}{\text{CPU speed}/1\,000\,000} \times \text{counter 0} \right) / \text{seconds in interval} \times 100$$

L1MP

Level 1 misses per 100 instructions, which provides you with the number of executed instructions (per 100) that required instructions or data to be sourced from somewhere other than Level 1 cache.

This is calculated by dividing the sum of counters 2 (L1 I-CACHE DIRECTORY-WRITE COUNT) and 4 (L1 D-CACHE DIRECTORY-WRITE COUNT) in the basic counter set, by counter 1 (executed instructions) in the same counter set, and then multiplying by 100.

The following metrics and formulas apply only to z10:

L15P

Percentage of updates sourced from L1.5 cache, which is the percentage of updates to the Level 1 caches (data and instruction) where the update came from the Level 1.5 cache.

This is calculated by dividing the sum of counters 128 and 129 in the extended counter set, by the sum of counters 2 (L1 I-CACHE DIRECTORY-WRITE COUNT) and 4 (L1 D-CACHE DIRECTORY-WRITE COUNT) in the basic counter set, and then multiplying by 100.

Counter 128 writes to Level 1 instruction cache where the instruction was sourced from Level 1.5 cache. Counter 129 writes to Level 1 data cache where the data was sourced from Level 1.5 cache.

L2LP

Percentage of updates sourced from Level 2 Local cache on same book, which is the percent of updates to the Level 1 caches (data and instruction) where the update came from the Level 2 cache in the same book as the microprocessor.

The percentage is calculated by dividing the sum of counters 130 and 131 in the extended counter set, by the sum of counters 2 (L1 I-CACHE DIRECTORY-WRITE COUNT) and 4 (L1 D-CACHE DIRECTORY-WRITE COUNT) in the basic counter set, and then multiplying by 100.

Counter 130 writes to Level 1 instruction cache where the instruction was sourced from the local Level 2 cache. Counter 131 writes to Level 1 data cache where the data was sourced from the local Level 2 cache.

L2RP

Percentages of updates sourced from Level 2 remote cache (on a *different book*), which is the percent of updates to the Level 1 caches (data and instruction) where the update came from a Level 2 cache that was not in the same book as the microprocessor.

The percentage is calculated by dividing the sum of counters 132 and 133 in the extended counter set, by the sum of counters 2 (L1 I-CACHE DIRECTORY-WRITE COUNT) and 4 (L1 D-CACHE DIRECTORY-WRITE COUNT) in the basic counter set, and then multiplying by 100.

Counter 132 writes to Level 1 instruction cache where the instruction was sourced from the remote Level 2 cache. Counter 133 writes to Level 1 data cache where the data was sourced from remote Level 2 cache.

MEMP

Percentage of updates sourced from memory, which is the percent of updates to the Level 1 caches (data and instruction) where the update came from memory. That memory might be in the same book as the microprocessor or in a different book.

MEMP is calculated as follows:

$$\text{MEMP} = ((\text{MEM_ON_BOOK} + \text{MEM_OFF_BOOK}) / \text{ALL}) \times 100$$

See the following calculation for an example:

$$\text{MEM_ON_BOOK} = \text{Counter 134} + \text{Counter 135}$$

$$\text{MEM_OFF_BOOK} = ((\text{Counters 2} + 4 - 128 - 129 - 130 - 131 - 132 - 133 - 134 - 135))$$

$$\text{ALL} = \text{Counters 2} + 4$$

Est SCPL1M

Estimated sourcing cycles per Level 1 cache miss, which is calculated by dividing the sum of counters 3 and 5, by the sum of counters 2 and 4, and multiplying that number by .84.

Counter 3 is the Level 1 instruction cache penalty cycle count. Counter 5 is the Level 1 data cache penalty cycle count. Counter 2 is the Level 1 instruction cache directory write count. Counter 4 is the Level 1 data cache directory write count.

The following metrics and formulas apply only to z196:

L2P

Percentages of updates sourced from Level 2 cache, which is the percent of updates to the Level 1 caches (data and instruction) where the update came from the Level 2 cache.

The percentage is calculated by dividing the sum of counters 128 and 129 in the extended counter set, by the sum of counters 2 (L1 I-CACHE DIRECTORY-WRITE COUNT) and 4 (L1 D-CACHE DIRECTORY-WRITE COUNT) in the basic counter set, and then multiplying by 100.

Counter 128 writes to Level 1 data cache where the data was sourced from the Level 2 cache. Counter 129 writes to Level 1 instruction cache where the instruction was sourced from the Level 2 cache.

L3P	<p>Percentage of updates sourced from Level 3 cache on same chip as microprocessor, which is the percent of updates to the Level 1 caches (data and instruction) where the update came from a Level 3 cache.</p> <p>The percentage is calculated by dividing the sum of counters 150 and 153 in the extended counter set, by the sum of counters 2 (L1 I-CACHE DIRECTORY-WRITE COUNT) and 4 (L1 D-CACHE DIRECTORY-WRITE COUNT) in the basic counter set, and then multiplying by 100.</p> <p>Counter 150 writes to Level 1 data cache where the data was sourced from the Level 3 cache. Counter 153 writes to Level 1 instruction cache where the instruction was sourced from Level 3 cache.</p>
L4LP	<p>Percentage of updates sourced from Level 4 cache in the same book, which is the percent of updates to the Level 1 caches (data and instruction) where the update came from Level 4 cache in the same book.</p> <p>The percentage is calculated by dividing the sum of counters 135, 136, 152, and 155 in the extended counter set, by the sum of counters 2 (L1 I-CACHE DIRECTORY-WRITE COUNT) and 4 (L1 D-CACHE DIRECTORY-WRITE COUNT) in the basic counter set, and then multiplying by 100.</p> <p>Counter 135 writes to Level 1 data cache where the data was sourced from the Level 4 cache in the same book. Counter 136 writes to Level 1 instruction cache where the instruction was sourced from the Level 4 cache in the same book. Counter 152 writes to Level 1 data cache where the data was sourced from a Level 3 cache on a separate chip in the same book. Counter 155 writes to Level 1 instruction cache where the instruction was sourced from a Level 3 cache on a separate chip in the same book.</p>
L4RP	<p>Percentages of updates sourced from a Level 4 cache in another book, which is the percent of updates to the Level 1 caches (data and instruction) where the update came from a Level 4 cache in a different book.</p> <p>The percentage is calculated by dividing the sum of counters 138, 139, 134 and 143 in the extended counter set, by the sum of counters 2 (L1 I-CACHE DIRECTORY-WRITE COUNT) and 4 (L1 D-CACHE DIRECTORY-WRITE COUNT) in the basic counter set, and then multiplying by 100.</p> <p>Counter 138 writes to Level 1 data cache where the data was sourced from the Level 4 cache in a different book. Counter 139 writes to Level 1 instruction cache where the instruction was sourced from the Level 4 cache in a different book. Counter 134 writes to Level 1 data cache where the data was sourced from a Level 3 cache in a different book. Counter 143 writes to Level 1 instruction cache where the instruction was sourced from a Level 3 cache in a different book.</p>
MEMP	<p>Percentage sourced from memory, which is the percent of updates to the Level 1 caches (data and instruction) where the update came from memory. The memory might be in the same book as the microprocessor, or in a different book.</p> <p>MEMP is calculated as follows:</p> $\text{MEMP} = ((\text{MEM_ON_BOOK} + \text{MEM_OFF_BOOK}) / \text{ALL}) * 100$

See the following calculation for an example:

MEM_ON_BOOK = Counters 141 + 142

MEM_OFF_BOOK = ((Counters 2 + 4 - 128 - 129 - 150 - 153 - 135 - 136 - 152 - 155 - 138 - 139 - 134 - 143 - 141 - 142)

ALL = Counters 2 + 4

Est SCPL1M

Estimated sourcing cycles per Level 1 cache miss, which is calculated by dividing the sum of counters 3 and 5 by the sum of counters 2 and 4, and then multiplying that number by the sum of .54 plus 10% of the relative nest intensity (RNI).

Counter 3 is the Level 1 Instruction cache penalty cycle count. Counter 5 is the Level 1 Data cache penalty cycle count. Counter 2 is the Level 1 Instruction cache directory write count. Counter 4 is the Level 1 Data cache directory write count.

For all calculations, the counters for all logical CPUs in the LPAR must be included.

For practice, you can write your own program to process the Type 113 records. An easy way to get started with certain analysis of the counters data is to use data extraction program provided by IBM named CP3KEXTR. To download the CP3KEXTR program, see “Data Extraction Program (CP3KEXTR) for zPCR” at:

<http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/PRS4229>

If you run the CP3KEXTR program against a data set containing SMF Types 23, 70 to 78, and 113 records (Figure 3-9), part of the output contains the metrics described in 3.3.1, “Key metrics” on page 42.

```
CP3KE113: VARIABLE AVG MIN MAX STND_DEV COUNT
CP3KE113: L1MP 0.4 0 2.7 1 7
CP3KE113: MEMP 1.3 0 8.8 3.1 7
CP3KE113: RNI 0.2 0 1.2 0.4 7
CP3KE113: L2P 10.5 0 73.8 25.8 7
CP3KE113: L3P 2.2 0 15.3 5.4 7
CP3KE113: L4LP 0.2 0 1.4 0.5 7
CP3KE113: L4RP 0.1 0 0.6 0.2 7
CP3KEXTR: PROCESSING COMPLETE
```

Figure 3-9 CP3KEXTR report of counters data

The font format of the output can be small, but if you import it into a spreadsheet, you can get a spreadsheet similar to the one in Figure 3-10 on page 47.

The spreadsheet format (Figure 3-10) can be easier to use, as the values and variable names are clearly separated and can be plotted or included in other calculations.

CP3KE113:	VARIABLE	AVG	MIN	MAX	STND_DEV	COUNT
CP3KE113:	L1MP	0.4	0	2.7	1	7
CP3KE113:	MEMP	1.3	0	8.8	3.1	7
CP3KE113:	RNI	0.2	0	1.2	0.4	7
CP3KE113:	L2P	10.5	0	73.8	25.8	7
CP3KE113:	L3P	2.2	0	15.3	5.4	7
CP3KE113:	L4LP	0.2	0	1.4	0.5	7
CP3KE113:	L4RP	0.1	0	0.6	0.2	7
CP3KEXTR:	PROCESSING COMPLETE					

Figure 3-10 CP3KEXTR report in spreadsheet

This report is from a z196, as you can see from the presence of the L3P, L4LP, and L4RP metrics.

The metrics can help you more easily see where the data and instructions are being sourced from, and the amount of CPU resource that was being consumed to do a given volume of work.

3.4 Examples of using counters data

Previous topics provided information about the type of data that is contained in the counters, the metrics that can be derived from that data, and the tools that you can use to extract the metrics from the CPU MF SMF records. This topic provides cases for when you might want to use the counters data to help you analyze the results of changes to your configuration and workloads.

You can use counters data to help with the following work:

- ▶ Using the information provided in the counters can help you more easily categorize your workload into one of the three new LSPR workloads.

One of the challenges for customers has been to identify the LSPR workload that is most like their work. The counters and certain related tools can help you achieve this task in an easier and more accurate method.

- ▶ Determine the impact of system-level changes, such as enabling HiperDispatch.
- ▶ Understand if greater use of data in memory techniques provides more effective exploitation of the processor cache hierarchy.
- ▶ Investigate how the use of 1 MB pages affects use of the processor caches.

3.4.1 Categorizing a workload

As explained in 1.2, “CPU MF counters” on page 2, the number of LSPR workload types has been reduced from eight to three. Also, the information provided in the counters data can be used together with the traditional capacity planning metrics to help you identify which of the three LSPR workload types most closely describes your workload.

The RNI (introduced in 1.2.1, “Understanding the processor cache hierarchy” on page 4) provides you with part of the information to help you categorize your workload. The *nest* is the set of cache and memory resources that are not private to a given microprocessor.

Using the metrics in 3.3.1, “Key metrics” on page 42, you can calculate the RNI for a given workload using the counters data.

For z10, the RNI is calculated as follows:

$$\text{z10 RNI} = ((1.0 \times \text{L2LP}) + (2.4 \times \text{L2RP}) + (7.5 \times \text{MEMP})) / 100$$

For z196, the calculation is similar, but adjusted for the extra level of cache in the z196:

$$\text{z196 RNI} = (1.6 \times (0.4 \times \text{L3P}) + (1.0 \times \text{L4LP}) + (2.4 \times \text{L4RP}) + (7.5 \times \text{MEMP})) / 100$$

Formula updates: The metrics formulas can change. For the latest formulas, see the LSPR workload categories at:

<https://www-304.ibm.com/servers/resourceLink/lib03060.nsf/pages/lsprwork?OpenDocument&pathID=>

Table 3-1 shows the three LSPR workload types.

Table 3-1 Relationship of counter values to LSPR workload types

L1MP	RNI	Workload hint
<3	>=.75 <.75	Average Low
3 to 6	>1.0 .6 to 1.0 <.6	High Average Low
>6	>=.75 <.75	High Average

The counter values and the RNI that is calculated from them are *not* provided as a replacement for the traditional capacity planning tools. The best view of capacity performance remains an SMF Type 72 analysis where the CPU per transaction or CPU per input/output (I/O) for major workloads can be compared. This comparison must include a ratio between machines drawn, weighted by the percentage of each workload type. However, the RNI is easily calculated and can be used as another input to help you categorize your workloads for capacity planning purposes.

HIS does not provide a tool to calculate the RNI values from the counters data, but you can use other IBM and non-IBM tools for this purpose. In particular, the zPCR (Processor Capacity Reference) has been updated to use the counters data as part of its input to help you determine relative capacities of two processors. To download the client version of zPCR, see “Getting Started with zPCR (IBM's Processor Capacity Reference)” at:

<http://www.ibm.com/support/techdocs/atmastr.nsf/WebIndex/PRS1381>

Use the CP3KEXTR program (referenced in 3.3.1, “Key metrics” on page 42) to extract the information from the SMF Type 113 records for input to the zPCR tool. Part of the output from CP3KEXTR is the RNI values for the time period that was input to the program.

3.4.2 Analyzing the impact of HiperDispatch by using CPU MF

As provided in 3.1, “Working with counters data” on page 32, a given workload completes faster if the instructions and data that it needs are sourced from a cache that is closer to the microprocessor. This topic provides a comparison of the two configurations, one closer to the microprocessor and one further from the microprocessor, and the impact of using HiperDispatch.

In the first configuration, a z/OS LPAR has twenty CPs, but currently only needs the capacity of two. Therefore, on average, it uses 10% of each of the twenty CPs. If other LPs are using the other 90% of the CPs, the chances of this LPs data or instructions still being in the Level 1 or Level 2 cache when it is dispatched again are quite small. Also, with 20 CPs, the chances of a given piece of work being re-dispatched on the same physical CP is also quite small.

Compare that to a configuration where the LPAR has just two CPs. In that case, the chance of a piece of work being re-dispatched on the same physical CP is greatly increased. A similar theory is behind the HiperDispatch capability, introduced with the z10 processor. HiperDispatch attempts to reduce the number of physical CPs that the work for a given LPAR is dispatched over. This increases the likelihood of the data and instructions required by the work running on the CPU being found in cache. However, HiperDispatch is not appropriate for every LPAR.

Table 3-2 shows an LPAR with 21 shared CPs on a z10 that was measured with and without HiperDispatch enabled.

Table 3-2 Using counters to understand impact of HiperDispatch

HD Setting	CPI	L1MP	L1.5P	L2LP	L2RP	MEMP
OFF	8.2	3.9	63.7	23.1	6.7	6.6
ON	7.5	3.8	70.3	19.8	3.7	6.3

Table 3-2 shows a general shift in numbers from the right side of the table in the MEMP (memory) and L2RP (Level 2 cache) columns, to the left side of the table in the L1MP (Level 1 cache) and L1.5P (Level 1.5 cache) columns. In order to get representative measurements, data from multiple days was input to the analysis. Table 3-2 also shows that the CPI reduced by 10% after HiperDispatch was enabled.

There are no *right* or *wrong* values. The values for one configuration might not be the same as the values for another configuration. What is more important is how the values move in response to your configuration change. You want the L1MP and L1.5P values to increase, and the L2LP and MEMP values to decrease.

The example provided in this topic, and other examples in this paper, was extracted from *CPU MF - the “Lucky” 113s - z196 Update and WSC Experiences*. You can find this document at the following website:

<http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/TC000066>

3.4.3 Data in memory

Consider the case of a program that is executing, reading through a file of client records. If every required record is in storage before it is needed, when an instruction makes an update to the record, there can be a short delay when the data is retrieved from memory. However, if the required record is not in storage, an I/O is required to retrieve the record from external storage. This results in the program losing control when the read operation takes place. In turn, this increases the possibility that the data and instructions for that program is replaced in the cache before the program gets dispatched again.

To increase the possibility of the required data being found in storage when required, you can use several configuration methods. For example, for sequential data sets, half-track blocking can increase the possibility that the required record is already in memory when needed. For database files, the use of robustly-sized buffer pools can also improve the chance of required records being memory prior to them being needed by the program.

To discover the benefits of change on how the processor caches are exploited, see the CPU MF metrics from a representative set of windows before the change, and a corresponding set of windows after the change. You can also study the RMF I/O reports. It is likely that response times might increase because you are reading or writing more data in a single I/O, but overall device utilization can decrease, because of the more efficient I/Os.

3.4.4 Impact of 1 MB pages

Traditional System z pages are 4 KB. However, if large amounts of data are being buffered in storage, it can be more efficient to use larger pages instead. Managing 1 MB worth of data in a single 1 MB page is obviously more efficient than managing 256 4 KB pages. Of course, the trade off is that the 1 MB page might contain data that your application is not interested in. To optimize the benefits of this capability, you need to understand how effective the large pages are from a database manager perspective, and how the use of these large pages affects the use of the processor cache resources.

System z 10 introduced support for 1 MB pages, with z/OS support delivered back to z/OS 1.9. DB2 V10 and WebSphere® Application Server V7 also announced explicit support for 1 MB pages. To find more information about z/OS support of 1 MB pages, and considerations for their use, see the “Thanks for the memory” article in the August 2009 issue of *Hot Topics* at the following website:

<http://publibz.boulder.ibm.com/epubs/pdf/e0z2n1a0.pdf>

The counter values for a pair of measurements with and without 1 MB pages is shown in Table 3-3. In this case, a 5% decrease in the number of cycles per instruction and a shift in the numbers from the right side of the table to the left side is shown. The decrease is primarily due to a reduction in the number of translation look-aside buffer page table entry misses that resulted in less CPU being used and an improvement in the effectiveness of Level 1.5 cache. This measurement is from a synthetic benchmark and not a real production environment, however, it shows how you can use the counters data to analyze the results of a data in memory or 1 MB pages exercise.

Table 3-3 Using counters to understand impact of 1 MB pages

Page size	CPI	L1MP	L1.5P	L2LP	L2RP	MEMP
4 KB	4.46	7.13	94.72	4.64	0.01	0.63
1 MB	4.26	7.25	96.56	3.03	0.01	0.41



Understanding your application behavior by using CPU MF

This chapter provides information about how you can use the sampling information collected by CPU MF to better understand the performance and behavior of your applications.

This chapter includes the following topics:

- ▶ Working with sampling data
- ▶ Understanding the sampling data
- ▶ Understanding map data
- ▶ Reporting on sampling data

4.1 Working with sampling data

CPU MF produces sampling information to provide you with insight into where each logical CPU is spending time (Chapter 1, “Introduction to CPU MF” on page 1). However, simply knowing the address and the address space ID of the instruction that is being executed is not sufficient. To understand what programs are being executed when each sample is taken, you also need a map that shows which programs are resident in virtual storage, and the location within virtual storage of each program.

This topic provides information about the sampling and MAP files that are created by hardware instrumentation services (HIS), and the layout of the contents of those files. It also provides information about how you can apply the map information to the samples to understand which programs are appearing in the largest numbers of samples.

4.2 Understanding the sampling data

CPU MF sampling data is written by HIS to files in the UNIX file system. One file is created for each logical CPU for each measurement period.

The file names are in the following format:

```
SYSHISyyyymmdd.hhmms.SMP.xx
```

The following definitions explain each segment of the file format:

SYSHIS	Fixed
yyyy	The year that the collection period started
mm	The month when the collection period started
dd	The day when the collection period started
hh	The hour when the collection period started
mm	The minute when the collection period started
ss	The second when the collection period started
SMP	Fixed
xx	The hexadecimal logical CPU number, starting with 00

You get one file for each logical CPU online to the system at the start of the measurement. You also get a file for any specialty processors, such as zIIPs and zAAPs that are online to the system. There is no indication about which files are for CPs, which are for zIIPs, and which are for zAAPs, but the last two digits of the .SMP file name must match the CPU NUM value in a corresponding RMF report.

Figure 4-1 shows a sample RMF CPU report.

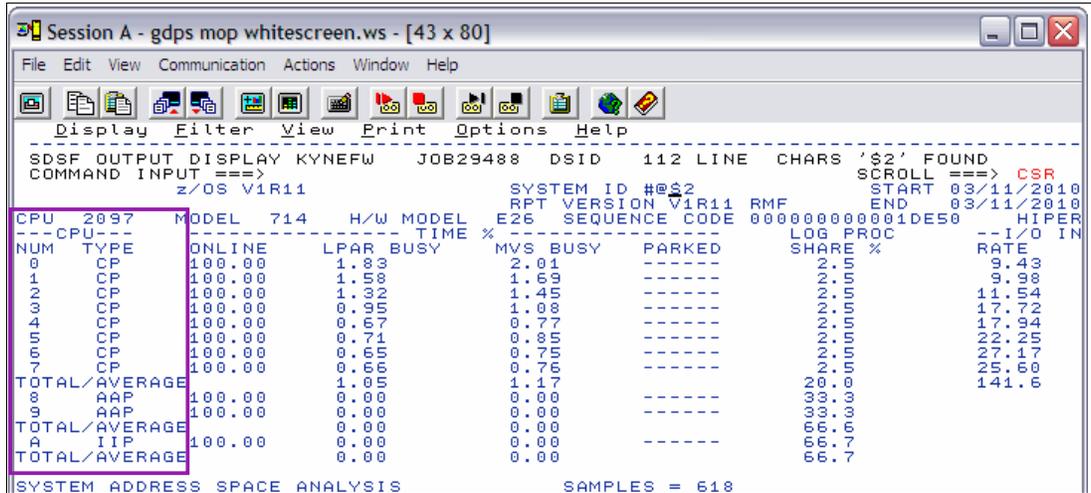


Figure 4-1 RMF CPU report

Figure 4-2 shows the SMP files for a samples collection for the system shown in Figure 4-1. The CPU NUM values in the RMF report in Figure 4-1 run from 0 to A, and the file names in Figure 4-2 also run from 0 to A. In the RMF report (Figure 4-1), there was no use of the zAAPs (CPU NUMs 8 and 9) or the zIIP (CPU NUM A). The browse output shows the file numbers 08, 09, and 0A have a size of 0, meaning that no samples were taken on those logical CPUs over the measurement period.

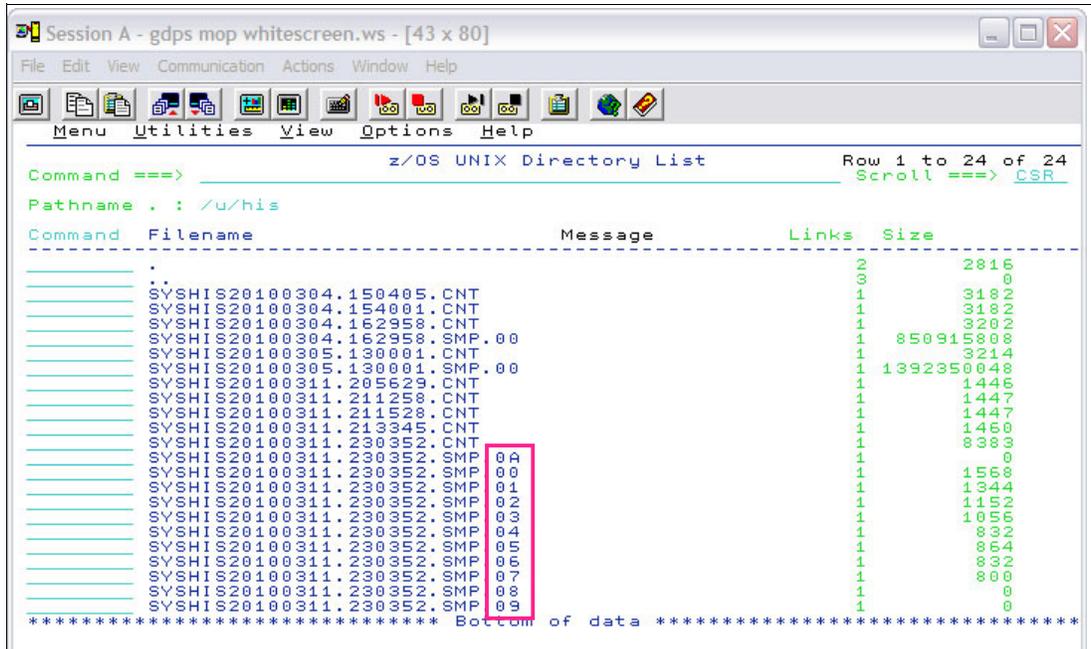


Figure 4-2 List of Sample files

The files are written to the UNIX file system directory identified on the HOME entry for the RACF user ID associated with the HIS started task, or the file system specified on the PATH keyword on the F HIS,B command.

Depending on the sampling frequency and the collection period duration, these files can be large. The default sampling frequency can result in 25.6 MB of sampling data being collected each minute.

4.2.1 Format of the sampling data

The format of the sampling file contains many 32-byte sample records, interspersed with certain 64-byte trailer records. To get the format of the records, see the ResourceLink for *Set-Program-Parameter and CPU-Measurement Facilities, SA23-2260*, and “Sampling function output in a .SMP file” in *z/OS MVS System Commands, SA22-7627*.

Figure 4-3 shows the layout of a basic sampling record.

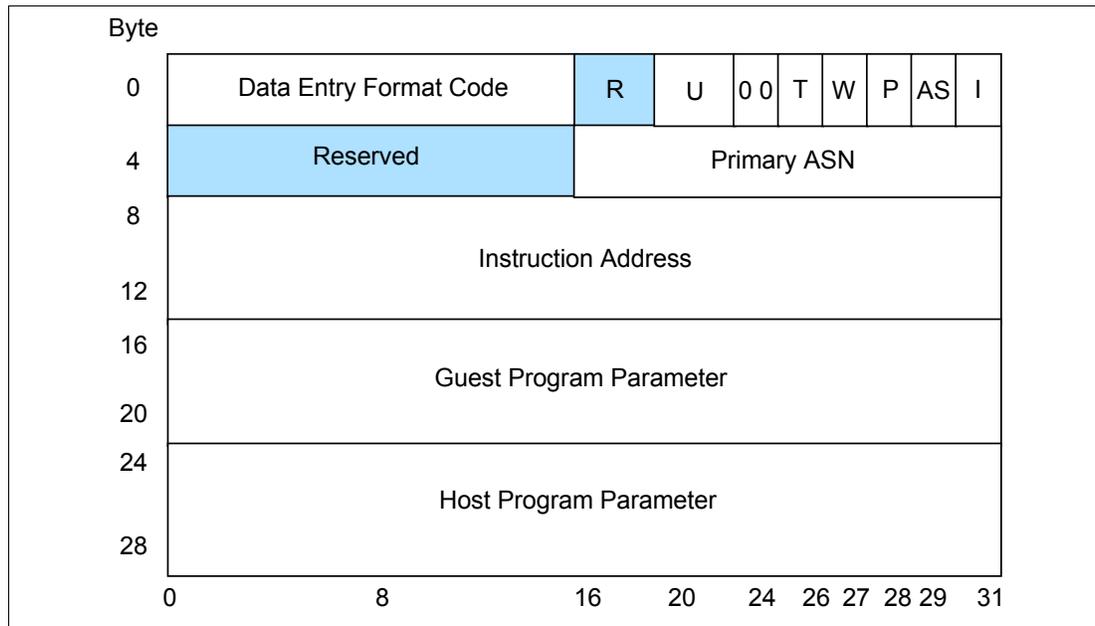


Figure 4-3 Format of a basic sampling record

Diagnostic sampling records: Diagnostic sampling records are intended for IBM problem diagnosis. Therefore, their contents are not addressed in this paper.

To make it easier to visualize the contents of a basic sampling record, Example 4-1 shows various sampling records and a description of the fields in those records.

Example 4-1 Description of sampling fields

```

DE R TW PR INSTRUCT GUEST PG HOST PGM WITH A MODULE MAP (.MAP FILE), WE'LL KNOW IN WHICH MODULE
FC U PASI AS ADDRESS PARAMETR PARAMETR THIS PARTICULAR CPU WAS EXECUTING WHEN THE SAMPLE WAS TAKEN
00 0 2 00 02 000008B8 07650300 00000000 @=18FB88A THIS MODULE IS IN THE COMMON AREA (.MAP SHOWS LIMITS)
01 0 2 00 04 00001F8A 0E901305 00000000 ASID '0024' IS EXECUTING IN SUPERVISOR STATE, ABOVE 16MB LINE

00 0 2 00 00 00000F16 07690300 00000000 @= FF1964 ALSO IN COMMON AREA, BUT BELOW THE 16 MB LINE (RMODE=24)
01 0 0 00 07 00000F94 0ED01303 00000000 ASID '0007' IS EXECUTING IN SUPERVISOR STATE, BELOW 16MB LINE

00 0 2 00 03 00000135 071A0301 00000000 @=8143650 HERE ASID '013C' EXECUTES IN PROBLEM STATE!
01 0 8 00 1C 00008460 0B081C01 00000000 NOTE IT IS EXECUTING IN PROBLEM STATE, WELL ABOVE 16MB LINE

00 0 3 00 00 00000000 8FB80000 00000000 <=MASTER SCHEDULER ORDERED A CPU WAIT (NO WORK TO DO!)
01 0 0 00 01 00000000 0D380000 00000000
| | | RE | | | | |
| | | SE | | | | | HOST PGM PARAMETER WAS ZERO IN ALL THESE OBSERVATIONS
| | | RV | | | | | GUEST PGM PARAMETER
| | | ED | | | | | INSTRUCTION ADDRESSES ARE ZERO WHEN THE LOGICAL CPU IS IN WAIT STATE (NO WORK TO DO)
| | | | | | | | | PRIMARY ASID '0001'=MASTER SCHEDULER, OTHER ACTIVE: '0007' '000B' '0024' & '013C' NEED A .MAP FILE
| | | | | | | | | '2' MEANS ASID AT WORK WITH DAT ON
| | | | | | | | | '3' MEANS CPU WENT INTO WAIT STATE (LAST MASTER SCHEDULER ORDER!)
| | | | | | | | | '2' MEANS SECONDARY ADDRESS SPACE CONTROL MODE, INSTEAD OF PRIMARY, WHEN IT IS ZERO
| | | | | | | | | '8' MEANS PROBLEM STATE, THE MOST USEFUL TO DRIVE APPLICATION ANALYSIS
| | | | | | | | | WATCH OUT FOR AN ODD FIGURE IN THIS SEMI BYTE BELOW, FOR IT INDICATES AN INVALID SAMPLING RECORD
| | | | | | | | | 'R' MEANS 4 BITS RESERVED FOR PROGRAMMING USE, 'U' BELOW IS NUMBER OF UNIQUE INSTRUCTIOS, TO DERIVE CPAI
ALL THESE OBSERVATIONS WERE DONE WITH DATA ENTRY FORMAT CODE EQUALS TO x'0001'

```

Example 4-1 shows a Hex On display of the records, and that each 32-byte record is displayed over two lines.

The first two bytes, bits 0 to 15, are the *data entry format code* (DEFC). In the first sampling record shown in Example 4-1, the DEFC is x'0001'. The DEFC indicates what type of sampling record this is. The current valid values are x'0001', meaning that this is a basic-sampling record, or x'8001', and also a diagnostic-sampling record. Diagnostic sampling is used only by IBM service personnel to help diagnose and investigate specific situations. Therefore, additional information about diagnostic sampling is not provided.

Sample data: In the future, values larger than x'0001' might be used. When either the size, the meaning, or the format of a sample data is different from the previous model, this value is incremented.

Diagnostic sampling: Diagnostic sampling is intended to help IBM diagnose investigate specific situations. Diagnostic sampling can only be enabled by IBM Service Personnel, and the layout of the diagnostic sampling records is not published. Therefore, this capability is not discussed further in this paper.

Approximately every 128 records you also get a 64-byte record with a DEFC of x'8000' or x'C000'. These records are created at the end of each buffer and can be ignored from the perspective of using the sampling data to analyze application performance. All the sampling records shown in Example 4-1 are basic-sampling records.

The first four bits (16-19) of the next byte are currently reserved for future use. The next four bits (20-23) are the number of instructions that were executing simultaneously at the instant the sample was taken. This is related to the use of pipe-lined instructions. The contents of this field can be used to estimate the number of cycles per instruction when a sufficiently small sampling interval and an adequately larger number of samples are used.

The CPI for a particular measurement can be estimated by dividing the number of busy samples by the total number of unique instructions in all busy samples. Busy samples have the wait-state bit set to zero.

Bits 24 and 25 of the next byte are currently reserved for future use. Bit 26 indicates the DAT mode at the time the sample was taken. In the first sampling record in Example 4-1 on page 55, the value was 2, with bit 26 set to 1, indicating that DAT was on.

Bit 27 indicates if the logical CPU was in a wait state at the time the sample was taken, indicating that the scheduler had no work to do, so it placed the logical CPU into a wait state. The fourth sampling record in Example 4-1 on page 55 has a 3 in this field, indicating that DAT was on (a 1 in bit 26), and that the logical PU was in a wait state (a 1 in bit 27).

Bit 28 indicates if the logical CPU was in problem state when the sample was taken. Bits 29 and 30 contain the address space control bits from the PSW. If the next bit (31) is turned on, that is an indicator that this is an invalid sampling record and can be ignored. This can happen if the processor detects that the information that is placed in the record is not consistent.

The next two bytes, bits 32 to 47, are currently reserved for future use.

The next two bytes, bits 48 to 63, contain the primary ASN in bits 48-63 from control register 4 of the CPU. This allows you to identify which address space is executing. The MAP file, introduced in 4.3, "Understanding map data" on page 61, contains the address space ID and the name of the job or started task executing in that address space.

Eight bytes after that, bits 64 to 127, contain the address of the instruction that was executing at the time the sample was taken. By combining the address space ID with the virtual address, you can identify the program that was being executed when the sample was taken.

The next eight bytes contain the `guest program` parameter. The `guest program` parameter can be used to identify the individual task that contributed to the sample data.

When sampling occurs, if the CPU is running at the virtual machine level, then consider the following results:

- ▶ If the host indicator is 1 at the logical partition level, the `guest program` parameter contains the program parameter most recently set by the PU at the virtual machine level.
- ▶ If the host indicator is 0, the contents of the `guest program` parameter are unpredictable.

When sampling occurs, if the PU is running at the logical partition level, then consider the following results:

- ▶ If the host indicator is set to 0, the `guest program` parameter contains the program parameter most recently set by the CPU at the logical partition level.
- ▶ If the host indicator is set to 1, the contents of the `guest program` parameter are set to zeros.

The last 8 bytes contain the `host program` parameter. When sampling occurs, if the host indicator is set to 1 at the LPAR level, this field contains the program parameter most recently set by the CPU at the LPAR. If the host indicator is set to 0, this field contains zeroes. In all sampling data shown in the examples in this paper, this field is 0.

Currently, the `guest program` parameter and the `host program` parameter fields are reserved for future use and therefore always contain zeroes.

The address space ID and the instruction address can help you identify the program that was running, and the bit that indicates that the logical CPU was in problem state, as you are most likely interested in the time that the CPU is executing your application code.

Each 4 KB buffer contains 126 sample records, 32 bytes each, and one 64-byte trailer record. The trailer record can be useful in certain cases because it contains a time stamp indicating when the 4 KB buffer became full with the last 126 samples. Also, the last buffer might not contain a trailer record, so the last record in the SMP file might be a sampling record or it might be a trailer record.

Important: The definitive source of information about the contents of the sampling records and trailer records is *Set-Program-Parameter and CPU-Measurement Facilities*, SA23-2260. Always check the version of that document that corresponds to your hardware and software levels to ensure you are using the latest information.

At this stage, you might want to see sampling data and the settings of the individual fields.

4.2.2 Working with the sampling data

The sampling data is written to a UNIX file as binary data. The directory that the file is written to defaults to the directory specified in the `HOME` parameter for the HIS user ID, but you can specify another directory using the `PATH="pathname"` keyword on the `F HIS,BEGIN` command. For example, if you use a different UNIX file for each system (see 2.2.6, “UNIX System Services file allocation” on page 20), but want to use the same HIS JCL on all systems, use the `PATH=` keyword to indicate which file must be used to hold the sampling data.

Depending on your preference, you can work with the UNIX file or you can copy the contents over to a traditional MVS data set. In either case, if you want to look at the data, issue the `HEX ON` command to see the data.

As an alternative, you can copy the file contents to a traditional MVS data set, as either a sequential data set, or a member of a PDS. In this case, the MVS data set must be pre-allocated with an Logical REcord Length (LRECL) of 32, and you can use an OGET command to copy the data to the target data set, as shown in Figure 4-5. You must specify the BINARY keyword on the OGET command.

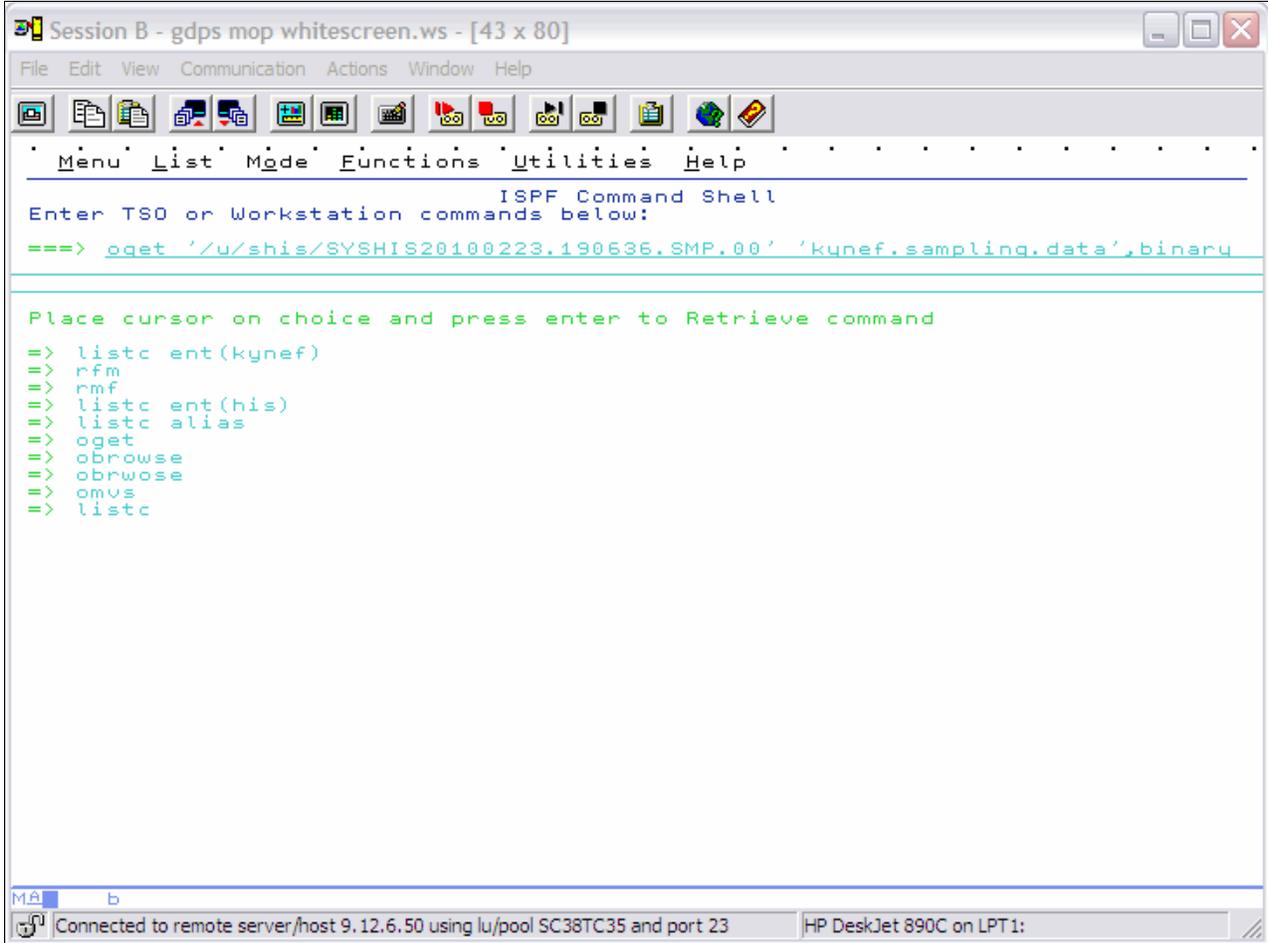


Figure 4-5 Using OGET command to move sampling data

After you have the data in your LRECL 32 data set, you can browse the data set and view the sampling data in a slightly more human-friendly format, as shown in Figure 4-6.

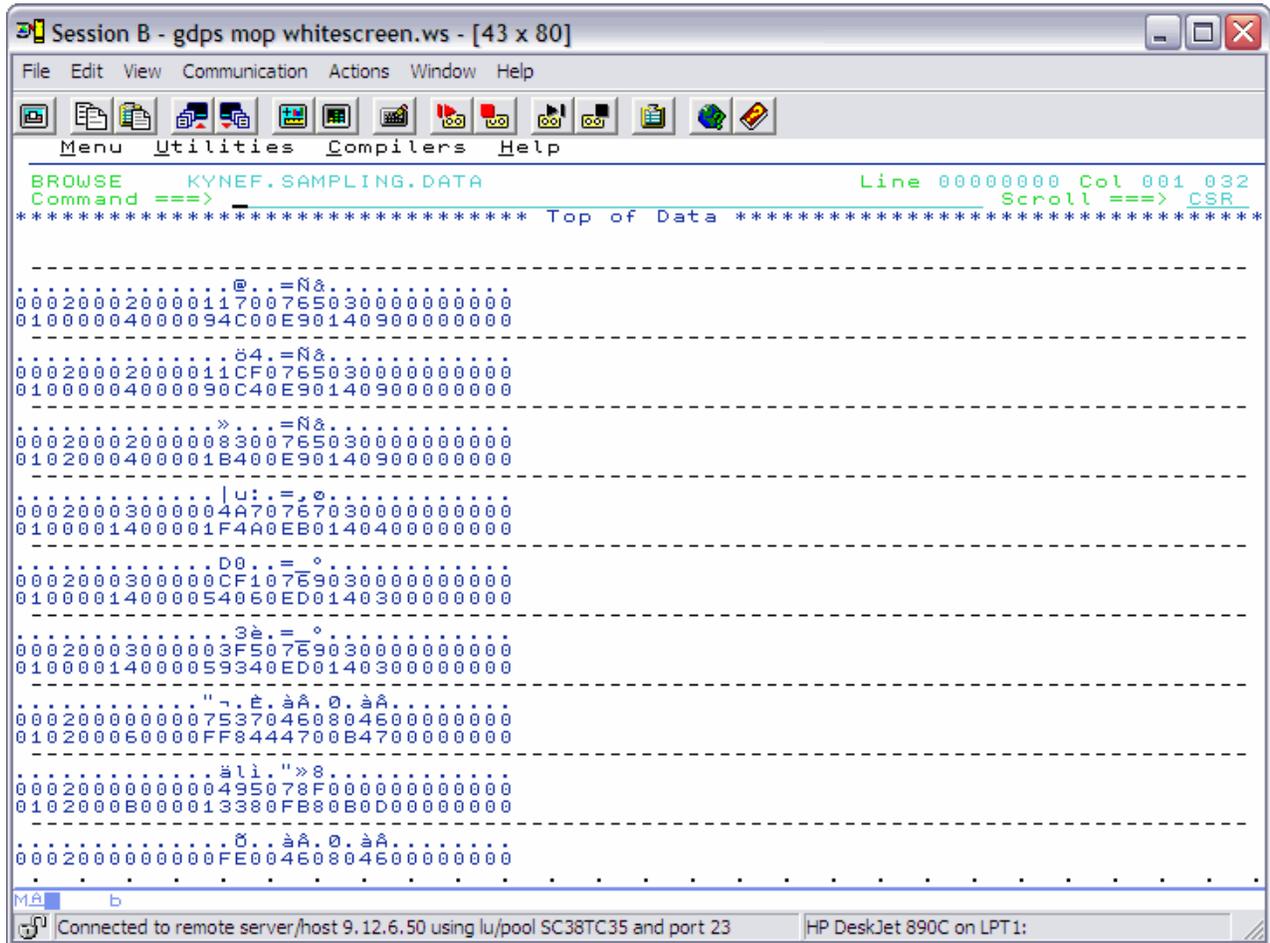


Figure 4-6 Browsing sampling data in MVS data set

Because the records are lined up underneath each other, it is easier to find the particular fields you might be interested in, or to sort, or exclude records based on particular fields.

Now that you understand the contents and the layout of the sampling data, you must use that information to understand what the CPU was running by combining the sampling data with the map data.

4.3 Understanding map data

The HIS started task includes the ability to create maps of virtual storage for one or multiple address spaces. The information contained in the MAP files can then be combined to the sampling data to get insight into what modules or even what CSECTS are being executed during the sampling interval.

To have a MAP file created at the end of the measurement period, you *must* specify MAPJOB or MAPASID on the F HIS,BEGIN command. To see any errors that might be encountered when the program is gathering the map information, you must specify the MAPVERBOSE keyword. Diagnostic information is presented on the console for any modules that the program has a problem gathering information for.

Example 4-2 contains certain samples of the messages you might see. In this case, the AIR571I message is presented because the map program was not able to obtain CSECT information for certain programs. The AIR520I message means that no programs were loaded in the private area of the named address space, ASID 0017. If you are encountering messages other than these, ensure that APAR OA30429 is applied to your system, because it addresses a number of issues in the MAP program.

Example 4-2 Map console messages

```
HIS022I HIS DATA COLLECTION IS ENDING. 751
  OUTPUT FILE PREFIX: SYSHIS20100312.152236.
  TITLE= ''
AIR571I MAPREQ UNABLE TO EXTRACT CSECT DATA (AOS LOADER) 752
MEMBER NAME =IXCI2PVX
AIR571I MAPREQ UNABLE TO EXTRACT CSECT DATA (AOS LOADER) 753
MEMBER NAME =ISGNPVX
IEF196I IEF237I DEOC ALLOCATED TO SYS00067
IEF196I IEF285I   SYS1.MIGLIB                               KEPT
IEF196I IEF285I   VOL SER NOS= Z1BRC1.
AIR571I MAPREQ UNABLE TO EXTRACT CSECT DATA (AOS LOADER) 757
MEMBER NAME =IWMI2PVX
AIR520I MAPREQ WAS UNABLE TO PROCESS A REQUEST 758
TO EXTRACT INFORMATION FOR ADDRESS SPACE, ASID=0017
BECAUSE CSVINFO RETURNED NO ENTRIES FOR IT
AIR571I MAPREQ UNABLE TO EXTRACT CSECT DATA (AOS LOADER) 759
MEMBER NAME =HOSCNVT
AIR571I MAPREQ UNABLE TO EXTRACT CSECT DATA (AOS LOADER) 760
MEMBER NAME =HA$PSUBS
```

4.3.1 Exceptions

The map process obtains information about which programs are resident in each address space by reading control blocks that are created by a part of MVS known as Contents Supervisor. In general, Contents Supervisor is responsible for loading and deleting programs.

However, in certain cases, programs are loaded into storage in a manner that Contents Supervisor is unable to manage. The following IBM products can be used to load programs into storage:

- ▶ CICS. CICS loads user programs into an area of storage that it allocates. It tells Contents Supervisor when programs are loaded, but it does not inform Contents Supervisor when it removes a program from virtual storage.

However, there are APARs available for CICS TS V3 and CICS TS V4 whereby CICS creates information about its user programs in a format that is compatible with the output from the map program. If you create a program to process the MAP files, it must also be able to process the equivalent files from CICS. For more information, see APARs PM08568 and PM08573.

- ▶ DB2. DB2 and IMS Resource Lock Manager (IRLM) load certain modules into CSA in a way that Contents Supervisor is not aware of. DB2 also dynamically generates executable code in storage, again, in a way that Contents Supervisor is not aware of.

The DB2 Diagnose utility can be used to print out the Module Entry Point List (MEPL). The MEPL does not report the IRLM load module information. DB2 V8 and V9 might not provide accurate information about all of the CSECTs contained in a load module, however this information is scheduled to be complete in DB2 V10. If you want to use the information from the MEPL together with sampling data, you must provide a program to reformat the MEPL report into a format that is compatible with the output from the map program.

- ▶ Java. Java programs are also loaded in a manner that Contents Supervisor is unable to track. However, the Java profiling tool (JPROF) can be used to create a file containing information about Java programs. You can write a program that can reformat the JPROF output into a layout that is compatible with the map program output.

4.3.2 Formatting map records

Before you can use the map data, you need to understand its layout. Figure 4-7 shows the first few lines of a MAP file in the UNIX file system. Unlike the sampling data file, this file is human-readable.

```

BROWSE /u/his/SYSHIS20100222.233735.MAP Line 00000000 Col 001 080
Command ==> Scroll ==> CSR_
***** Top of Data *****
I SYS #@$2
I SMFI#@$2
I OS z/OS
I FMIDHBB7760
I DATE10053
I TIME23392786
I MAP V1R1
I LPID000000011
I MACH00002097
B BDY PRIVATE 00000000007FFFFFFF
B BDY CSA 0000000000CE3FFF
B BDY CSAALLOC0004C14802EEB328
B BDY CSACONVT0000000000000000
B BDY MLPA 00C6400000C64FFF
B BDY FLPA 0000000000000000
B BDY PLPA 00C6500000E35FFF
B BDY SQA 00E3600000FD5FFF
B BDY SQAALLOC00E9DA801706878
B BDY RWNVC 00FD60000FE386F
B BDY RON 00FE40000FFFFFFF
B BDY ERON 010000000199945F
B BDY ERWN 0199A00001A12FFF
B BDY ESQA 01A1300004FC6FFF
B BDY EPLPA 04FC700009351FFF
B BDY EFLPA 0000000000000000
B BDY EMLPA 0935200009355FFF
B BDY ECSA 09356000018DFFFFFF
B BDY EPRV 18E000007FFFFFFF
B BDY DONUC 7F9AF0007F9B2FFF
C NNVC IEAVFX00000000000001FFF
E NNVC IEATCBP 00000218
M NNVC IGC203 00FD60000FD6BF8
M NNVC IECVPRNT00FD6C000FD70F7
E NNVC PRDSE 00FD6C06
E NNVC PRTSIO 00FD6C0C
E NNVC PRTEOS 00FD6C12
E NNVC PRTRAP 00FD6C18
MA b

```

Figure 4-7 Contents of MAP file

To understand the layout of the records, consider the following descriptions:

- ▶ The first byte includes an indicator of which type of record this one is. The following options are available:
 - I** This record provides identifying information about the system.
 - B** This record shows the layout of virtual storage and its boundaries.
 - C** This record is for a CSECT.
 - E** This record identifies an ENTRY within a module.
 - M** This record is for a module.
- ▶ The second byte indicates the memory area where the module resides and have the following possible values:
 - C** Common
 - F** Fixed link pack area
 - M** Modified link pack area
 - N** Nucleus
 - P** Pageable link pack area

- X** This record contains information about a program that is running in the private area of the address space identified by the four bytes following this character.
- ▶ The third through sixth bytes identify where this module is running and have the following possible values:
 - BDY** Contains the start and end address of one of the system areas.
 - FLPA** This record contains information about a module that is resident in the fixed link pack area.
 - MLPA** Contains information about a module that is resident in the modified link pack area.
 - NUC** Contains information about a module or CSECT that is part of the MVS nucleus.
 - PLPA** Contains information about a module that is resident in the pageable link pack area.
 - xxxx** Four hexadecimal digits, which are the number of the address space for when the private area contains the module being reported on in this record.
 - ▶ The next eight bytes contain the name of the module or CSECT.
 - ▶ The next eight bytes contain the starting address in virtual storage for this module or CSECT.
 - ▶ The next eight bytes contain the ending address in virtual storage for the module.
 - ▶ For a record that is identifying a module in the private area of an address space, the next 8 bytes contain the address space name.
 - ▶ Finally, the last 48 bytes contain the following information:
 - VOLSER=xxxxxxDSN=yyyyyyy.zzzzz.www (if this line is for an MVS load module)
 - <pathname>: /aaa/bb/ccc/ddd (if this line is for a z/OS UNIX executable)
 - LPA LIST CONCATENATION (if this line is for an MVS load module in PLPA)
 - cccccccccc (if this line is for a CSECT with a name longer than 8 bytes)

To make it a little easier to understand the layout of the MAP file, Figure 4-8 contains a marked-up version of a file. It does not contain all the possible record types. However, it can help to make it a little easier to understand how the records are laid out.

```

BROWSE   ORSONI.CPUMF.MAP(LASTMAP)                               Line 00000000 Col 001 080
Command ==>                                                    Scroll ==> CSR
***** Top of Data *****
HEADING INFORMATION
I SYS #@#2              <= SYSTEM IDENTIFICATION
I SMFI#@#2             <= SMF IDENTIFICATION
I OS z/OS               <= SYSTEM NAME
I FMIDHBB7760          <= FMID
I DATE10048            <= JULIAN DATE FROM CVT
I TIME14186512         <= TIME
I MAP V1R1             <= RELEASE/VERSION
I LPID00000011         <= LPAR ID
I MACH00002097         <= MACHINE TYPE = Z10
                        <= VIRTUAL STORAGE AREAS MAP, DESCRIBED BY GDA AND CVT
BNDRY NAME   STARTING   ENDING
B BDY PRIVATE 0000000007FFFFFF <= BELOW 16 MB LINE: PSA, THE FIRST 8K OF VIRTUAL STORAGE (COMMON)
B BDY CSA      0080000000C63FFF <= BELOW 16 MB LINE: COMMON STORAGE AREA
B BDY CSAALLOC0004C64802F16210 <= Allocated CSA
B BDY CSAACONVT0000000000000000 <= BELOW 16 MB LINE: NO CSA CONVERTED TO HOST SQA
B BDY MLPA     00C6400000C64FFF <= BELOW 16 MB LINE: MODIFIED LINK PACK AREA
B BDY FLPA     0000000000000000 <= BELOW 16 MB LINE: NO FIXED LINK PACK AREA
B BDY PLPA     00C6S00000E3SFFF <= BELOW 16 MB LINE: PAGEABLE LINK PACK AREA
B BDY SQA      00E3600000FD5FFF <= BELOW 16 MB LINE: SYSTEM QUEUE AREA
B BDY SQAALLOC000E9168017C2338 <= Allocated SQA
B BDY RWNNUC   00FD600000FE386F <= BELOW 16 MB LINE: READ AND WRITE NUCLEUS
B BDY RON      00FE400000FF5FFF <= BELOW 16 MB LINE: READ ONLY NUCLEUS

BNDRY NAME   STARTING   ENDING
B BDY ERON     010000000199945F <= ABOVE 16 MB LINE AREAS
B BDY ERWN     0199A00001A12FFF <= ABOVE 16 MB LINE: READ AND WRITE NUCLEUS
B BDY ESQA     01A1300004FC6FFF <= ABOVE 16 MB LINE: SYSTEM QUEUE AREA
B BDY EPLPA    04FC7000093851FFF <= ABOVE 16 MB LINE: PAGEABLE LINK PACK AREA
B BDY EFLPA    0000000000000000 <= ABOVE 16 MB LINE: NO FIXED LINK PACK AREA EITHER
B BDY EMLPA    09385200009385FFF <= ABOVE 16 MB LINE: MODIFIED LINK PACK AREA
B BDY ECSA     09386000018DFFFFF <= ABOVE 16 MB LINE: COMMON STORAGE AREA
B BDY EPRV     18E000007FFFFFFF <= ABOVE 16 MB LINE: PRV USER REGION (PVT) AREA, FROM 398 MB TO 2 GB
B BDY DONUC    7F9AF0007F9B2FFF <= ABOVE 16 MB LINE: DAT Off Nucleus

```

Figure 4-8 Annotated MAP file

For the latest information about the layout of the map records, always see *z/OS MVS System Commands*, SA22-7627.

4.3.3 MAP file usage considerations

If you use the map data to identify the program that was resident at the addresses listed in the sampling data, remember that the map data is a point-in-time snapshot, taken at the end of the collection period.

Given that the default sampling interval is ten minutes, there might not be many changes in which programs are loaded over that time. However, it is *possible* that a program that the MAP file identifies as being resident at a given address was not the one that was there when certain samples were taken. The map records do not contain a time stamp to identify when the program was loaded, and the sampling records do not contain a time stamp to identify what precise time each one is reporting on. Address spaces such as CICS, DB2, and IMS tend to load programs and keep them loaded at the same location. Therefore, it is not likely that the point-in-time nature of the map records results in issues, if you review the activity of these subsystems. Alternatively, the programs used by TSO users, short-running batch jobs, and interactive UNIX System Services users tend to be loaded for a short time, so the map information for those address spaces might not be accurate. Any environment where modules are frequently loaded and deleted can be an issue.

If you want to collect sampling data for more than ten minutes, consider doing several separate ten minute runs, and collecting map information at the end of each ten minutes. This method can be more accurate than letting sampling run for an hour, for example, and then only having one MAP file at the end.

4.4 Reporting on sampling data

Depending on your objective in enabling the collection of sampling data, you might like to generate certain types of reports. The CPU MF and HIS services help you collect valuable data, but they are not intended to provide tools to analyze that data.

For ideas about the type of information that you can extract from the data, you can use the sample Java reporting program provided on the IBM Alpha Works website. To download the Java reporting program, see “Report Generator for Hardware Instrumentation Sample Data” at:

<http://www.alphaworks.ibm.com/tech/rghisd>

This Java reporting program analyzes the sampling and map data to provide a report similar to that shown in Figure 4-9.

```

HISreport 1.0.0 - 20100804
*** © Copyright IBM Corp. 2010 ***

Report for all Home ASIDs

```

SAMPLES	ISAMPLES	CPU %	CPI	PASN	JOBNAME	MODULE	CSECT
=====	=====	=====	=====	=====	=====	=====	=====
131346	17224	60.900	7.63	0136	ABC2DBM1	DSNBBM	<NoCSECT>
10853	2533	5.032	4.28	0000	<COMMON>	Nucleus	IEAVEDS0
7211	531	3.343	13.58	0000	<COMMON>	Nucleus	IEAVEEXT
5834	1528	2.705	3.82	0000	<COMMON>	Nucleus	IOSVUSER
2506	985	1.162	2.54	011C	BBBBB	BBB9XY10	BBB10@
1931	63	0.895	30.65	0125	ABC2IRLM	DXRRLS20	DXRRL246
1535	164	0.712	9.36	0000	<COMMON>	Nucleus	IEAVECMS
1424	19	0.660	74.95	0000	<COMMON>	Nucleus	IXLM2SR
1419	158	0.658	8.98	0000	<COMMON>	Nucleus	IEAVELKX
1391	197	0.645	7.06	011F	CDEFXCMS	<NoModule>	<NoCSECT>
1376	114	0.638	12.07	0000	<COMMON>	ERBMFEDV	ERBMFEDV
1330	115	0.617	11.57	011C	BBBBB	BBXYTRAN	BBXYXA98
1132	963	0.525	1.18	0000	<COMMON>	IEEMB836	IEFTB728
1117	12	0.518	93.08	0125	ABC2IRLM	DXRRLS20	DXRRL247
1059	241	0.491	4.39	0153	XYZMYTM	XYZTLOG	<NoCSECT>
1050	104	0.487	10.10	0006	XCFAS	IXCI2PVX	<NoCSECT>

Figure 4-9 Sample report

In this report, the first column, SAMPLES, is a count of the number of samples where the named program was being executed. Given that samples are taken at reasonably consistent intervals, the number of samples for a program is a good indicator of the percent of used cycles that were consumed by that program.

ISAMPLES are the total number of samples that fall within a specific module which also have a special bit, instruction unique cycle, turned on. The instruction unique cycle is contained in bits 20 to 23 of the sample. As the CPU is running, this bit is only on for one cycle per instruction. If you have an instruction that takes an average of three cycles to execute, and you have 900 samples at this specific instruction, you can expect the ISAMPLES value to be 300 for this instruction. Of the 900 samples of this three-cycle instruction, this ISAMPLE indicator is only on for one of those three cycles. Therefore, chances are that one third of the samples have this bit turned on.

The next column, CPU %, represents the percent of all observed samples that are taken when this program is executing. It is calculated by dividing the total number of observed samples by the number in the SAMPLES column.

The next column, CPI, is also a calculated column. It is arrived at by dividing the total number of samples for this program by the ISAMPLES number. The result is an indication of the number of cycles per instruction for this program.

The PASN column contains the address space number, in hex.

The JOBNAME column contains either the name of the address space that contained the executing program, or <COMMON>, if the module was contained in common storage.

The MODULE column contains the name of the load module that was executing.

If that column contains <NoModule>, one possible cause is that the sample was taken early in the sampling interval, but that program ended and was deleted before the map of virtual storage was taken. Another possible reason is that the program was loaded in a manner that Contents Supervisor is unable to track it, so the MAP file created by HIS does not contain any information about that module.

The CSECT column contains the name of the CSECT within the module. In order to get this information, the HIS address space must have RACF authority to read the data set containing the load module. If HIS does not have the required access, the report contains <NoCSECT>.

If a load module contains multiple CSECTs, it is likely that certain samples are taken when one CSECT is being executed, and other samples are taken when another CSECT is being executed. The report program uses the virtual start and end address of each CSECT to map the samples that it encounters to the correct CSECT.

To download and run the report program, see “Report Generator for Hardware Instrumentation Sample Data” at:

<http://www.alphaworks.ibm.com/tech/rghisd/download>

Due to the volume of data being processed, certain customers have encountered out-of-storage problems with the program. If this is an issue for you, use the 64-bit Java, with Java heap sizes of -Xmx2000m -Xms1500m.

4.4.1 IBM service offering

Another option is to engage the IBM Lab Services organization to help you analyze the sampling data. Lab Services has the skills and powerful PC-based tools to analyze and visualize the HIS data.

The offering can help you to identify potential tuning opportunities to make your system run more efficiently and to improve the performance on your system. Figure 4-10 shows the information provided by the analysis tool.

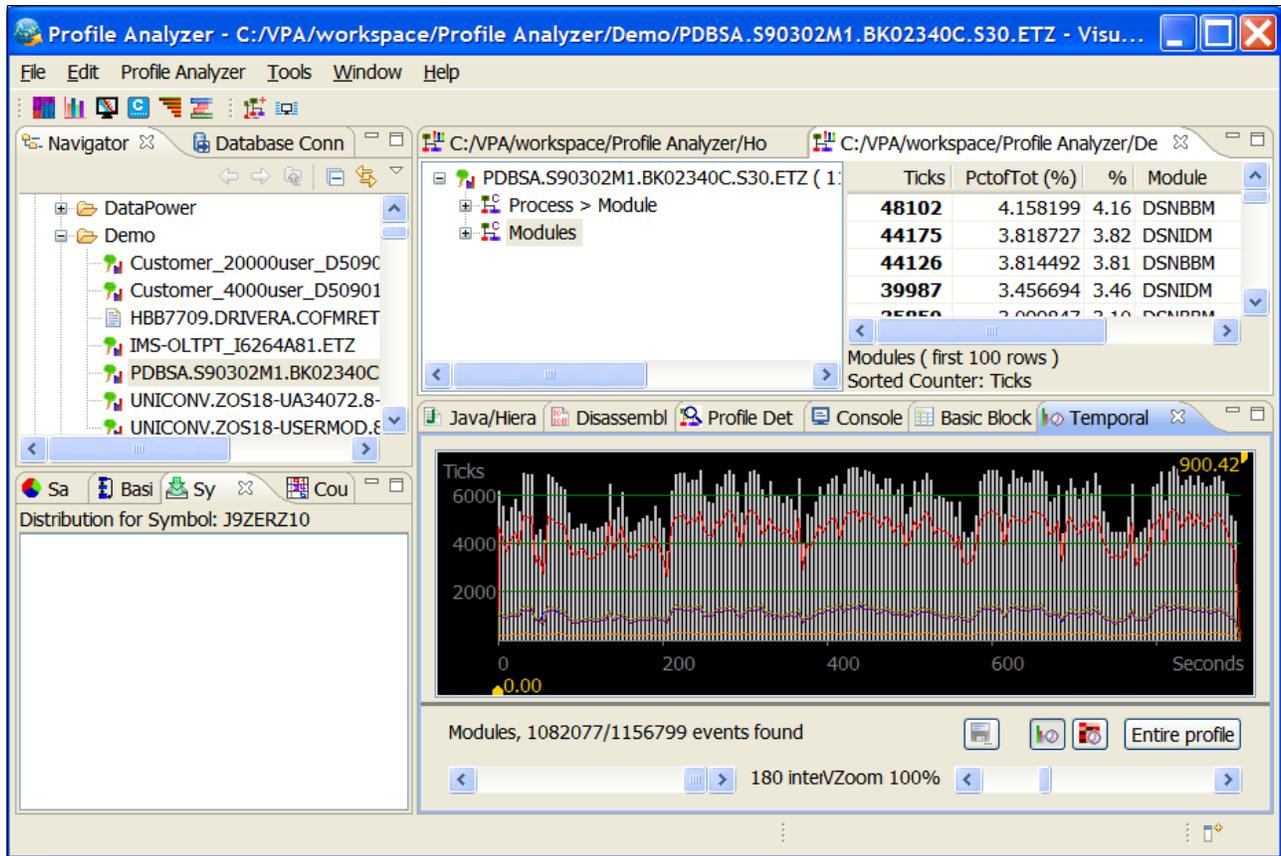


Figure 4-10 Sample report from analysis tool

For more information about the Lab Services offering, send an email to stgls@us.ibm.com.

4.4.2 Creating your own reports

If you want to create your own program to analyze the sampling data, the layout of the sample data can be found in the following documents:

- ▶ *Set-Program-Parameter and CPU-Measurement Facilities*, SA23-2260
- ▶ *IBM The CPU-Measurement Facility Extended Counters Definition for z10 and z196*, SA23-2261

You can do the following tasks, for example, in your program to extract the most *interesting* information:

- ▶ Read in sample entries, trim the right-most bits of the PSW address to a 64-byte or larger boundary, and sort them.

- ▶ Count consecutive duplicate entries, discard those with low counts, and determine the load module name and the offset into the program using the MAP files.
- ▶ Sort the remaining data by count to find program *hot spots*.
- ▶ Create either a text-based report or a comma-separated value file to be used as input to a spreadsheet program on a PC.

As you gain more experience with using the sampling data and analyzing the behavior of your programs, you can develop other ways to use this data to provide the perspectives that you need to help you fully understand your applications behavior.



A

Extended counters for z10 and z196

Of the various counter sets, all except the extended counters have the same meaning for z10 and z196 processors. The extended counter meanings for z10 and z196 are described in this appendix.

This appendix includes the following topics:

- ▶ Extended counters for z10
- ▶ Extended counters for z196

Extended counters for z10

The extended counters have a numbered identification rather than a textual description. The extended counter set contains the following information about the use of the processor cache and memory.

Extended counters 128 to 135 provide information about events in the cache hierarchy beyond the Level 1 caches that are described in the basic counter set. These counters provide information about where a Level 1 cache directory write found its associated cache line. These counters provide a break out of where the data that was returned to Level 1 cache was sourced from:

- | | |
|--------------------|--|
| Counter 128 | The number of directory writes to the Level-1 instruction cache directory where the returned cache line was sourced from the Level 1.5 cache. |
| Counter 129 | The number of directory writes to the Level 1 data cache directory where the installed cache line was sourced from the Level 1.5 cache. |
| Counter 130 | The number of directory writes to the Level 1 instruction cache directory where the installed cache lined was sourced from the Level 2 cache that is on the same book as the instruction cache. |
| Counter 131 | The number of directory writes to the Level 1 data cache directory where the installed cache line was sourced from the Level 2 cache that is on the same book as the data cache. |
| Counter 132 | The number of directory writes to the Level 1 instruction cache directory where the installed cache line was sourced from a Level 2 cache that is not on the same book as the instruction cache. |
| Counter 133 | The number of directory writes to the Level 1 data cache directory where the installed cache line was sourced from a Level 2 cache that is not on the same book as the data cache. |
| Counter 134 | The number of directory writes to the Level 1 data cache where the installed cache line was sourced from memory that is attached to the same book as the data cache. |
| Counter 135 | The number of directory writes to the Level 1 instruction cache where the installed cache line was sourced from memory that is attached to the same book as the instruction cache. |

There is no counter for the number of directory writes that are sourced from memory that is attached to a *separate* book than the book where the Level 1 cache is, for example, remote memory. However, the number of directory writes can be calculated using information in a number of counters.

To calculate the number of directory writes to the Level 1 instruction cache that were sourced from memory in a separate book, subtract counters 128, 130, 132, and 135 from counter 2.

To calculate the number of directory writes to the Level 1 data cache that were sourced from memory in a separate book, subtract counters 129, 131, 133, and 134 from counter 4.

The extended counters 136 to 142 and 145 to 147 contain additional counters related to the cache hierarchy and processor:

Counter 136	The number of directory writes to the Level 1 data cache where the line was originally in a shared state in the cache but has been updated to be in the exclusive state that allows stores to the cache line.
Counter 137	The number of times a cache line in the Level 1 instruction cache was invalidated by a store on the same CPU as the Level 1 instruction cache.
Counter 138	The number of times a translation entry was written into the Level 1 instruction translation look-aside buffer (ITLB1).
Counter 139	The number of times a translation entry was written to the Level 1 data translation look-aside buffer (DTLB1).
Counter 140	The number of times a translation entry was written to the Level 1.5 translation look-aside buffer (TLB) page table entry arrays.
Counter 141	The number of times a translation entry was written to the Level 1.5 TLB common region segment table entry arrays.
Counter 142	The number of times a translation entry was written to the Level 1.5 TLB common region segment table entry arrays for a one-megabyte large page translation.
Counter 143	Undefined
Counter 144	Undefined
Counter 145	Level 1 instruction TLB miss in progress. In increments of one for every cycle when an ITLB1 miss is in progress.
Counter 146	Level 1 data TLB miss in progress. In increments of one for every cycle when a DTLB1 miss is in progress.
Counter 147	The number of times a store was sent to the Level 1.5 cache.
Counter 148	Undefined
Counter 149	Undefined
Counter 150	Undefined
Counter 151	Undefined

Extended counters for z196

The extended counters provide information about hardware facilities and structures that are specific to a machine family. Even though much of the information provided in the extended counters for z196 is similar to z10, details of the machines' microarchitectures differ, meaning that the z196 extended counter definitions are not the same as z10. Also, considering the same type of information can be stored for z196, the counter used for a specific value is not always the same, so care must be taken if you write a program to process Type 113 records from more than one processor generation.

z196 terminology: The z196 packages four processors on a single chip. Each processor has a private Level 1 instruction cache (64 KB), a private Level 1 data cache (128 KB) and a private Level 2 cache that caches both instructions and data (1.5 MB). These four processors on a chip share a Level 3 cache (24 MB). When the Level 3 cache is described as *On Chip* it physically resides on the same chip as the processor.

The z196 packages six processor chips on a single book. The z196 can contain up to four physical books in a system. Each book contains a Level 4 cache (192 MB). When the Level 3 cache is described as *Off Chip/On Book*, the cache is on a separate chip from the processor, but in the same book. When a Level 4 cache is described as *On Book*, it is in the same book as the processor. *Off Book* means the Level 4 cache is in a separate book than the processor. Local memory is memory that is physically attached to the book the processor is on. Remote memory is memory that is physically attached to a book separate from the processor.

The contents of the extended counters for z196 are as follows:

Counter 128	The number of directory writes to the Level 1 data cache directory where the returned cache line was sourced from the Level 2 cache.
Counter 129	The number of directory writes to the Level 1 instruction cache directory where the installed cache line was sourced from the Level 2 cache.
Counter 130	Count of cycles consumed when a Level 1 data TLB miss was in progress.
Counter 131	Count of cycles consumed when a Level 1 instruction TLB miss was in progress.
Counter 132	Undefined
Counter 133	The number of stores that were sent to the Level 2 cache.
Counter 134	The number of directory writes to the Level 1 data cache directory where the installed cache line was sourced from a Level 3 cache that is not on the same book as the data cache.
Counter 135	The number of directory writes to the Level 1 data cache directory where the installed cache line was sourced from the Level 4 cache that is on the same book as the data cache.
Counter 136	The number of directory writes to the Level 1 instruction cache directory where the installed cache lined was sourced from the Level 4 cache that is on the same book as the instruction cache.
Counter 137	The number of directory writes to the Level 1 data cache where the line was originally in a read-only state but was updated to be in the exclusive state that allows stores to the cache line.

Counter 138	The number of directory writes to the Level 1 data cache directory where the installed cache line was sourced from a Level 4 cache that is not on the same book as the data cache.
Counter 139	The number of directory writes to the Level 1 Instruction cache directory where the installed cache line was sourced from a Level 4 cache that is not on the same book as the instruction cache.
Counter 140	The number of times a translation entry was written to the Level 1 data TLB for a one-megabyte page.
Counter 141	The number of directory writes to the Level 1 data cache where the installed cache line was sourced from memory that is attached to the same book as the data cache.
Counter 142	The number of directory writes to the Level 1 instruction cache where the installed cache line was sourced from memory that is attached to the same book as the instruction cache.
Counter 143	The number of directory writes to the Level 1 instruction cache directory where the installed cache line was sourced from a Level 3 cache that is not on the same book as the instruction cache.
Counter 144	The number of times a translation entry was written to the Level 1 DTLB1.
Counter 145	The number of times a translation entry was written into the Level 1 ITLB1.
Counter 146	The number of times a translation entry was written to the Level 2 TLB page table entry arrays.
Counter 147	The number of times a translation entry was written to the Level 2 TLB common region segment table entry arrays for a one-megabyte page translation.
Counter 148	The number of times a translation entry was written to the Level 2 TLB common region segment table entry arrays.
Counter 149	Undefined
Counter 150	The number of directory writes to the Level 1 data cache directory where the installed cache line was sourced from a Level 3 cache that is on the same book as the data cache.
Counter 151	Undefined
Counter 152	The number of directory writes to the Level 1 data cache directory where the installed cache line was sourced from a Level 3 cache that is on a separate chip in the same book as the data cache.
Counter 153	The number of directory writes to the Level 1 instruction cache directory where the installed cache line was sourced from a Level 3 cache that is on the same chip in the same book as the data cache.
Counter 154	Undefined
Counter 155	The number of directory writes to the Level 1 instruction cache directory where the installed cache line was sourced from a Level 3 cache that is on a separate chip in the same book as the instruction cache.
Counter 156	Undefined
	There is no counter for the number of directory writes that are sourced from memory that is attached to a <i>separate</i> book than the book where

the Level 1 cache is (remote memory). However, those numbers can be calculated using information in a number of counters.

To calculate the number of directory writes to the Level 1 instruction cache that were sourced from memory in a separate book, subtract counters 129, 136, 139, 142, 143, 153, and 155 from counter 2.

To calculate the number of directory writes to the Level 1 data cache that were sourced from memory in a separate book, subtract counters 128, 134, 135, 138, 141, 150 and 152 from counter 4.

The detailed descriptions for the extended counter fields for z196 are contained in the section titled "Extended Counters Definitions for z196" in *IBM The CPU-Measurement Facility Extended Counters Definition for z10 and z196*, SA23-2261.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this paper.

IBM Redbooks

The following IBM Redbooks publications provide additional information about the topic in this document. Note that some publications referenced in this list might be available in softcopy only.

- ▶ *IBM zEnterprise System Technical Guide*, SG24-7833
- ▶ *z/OS Version 1 Release 10 Implementation*, SG24-7605
- ▶ *z/OS Version 1 Release 12 Implementation*, SG24-7853

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks publications, at this Web site:

ibm.com/redbooks

Other publications

These publications are also relevant as further information sources:

- ▶ *IBM The CPU-Measurement Facility Extended Counters Definition for z10 and z196*, SA23-2261
- ▶ *Set-Program-Parameter and CPU-Measurement Facilities*, SA23-2260
- ▶ *z/OS MVS System Commands*, SA22-7627
- ▶ *z/OS MVS System Management Facilities (SMF)*, SA22-7630

Online resources

The following website is also relevant as a further information source:

- ▶ CPU MF - the "Lucky" 113s - z196 Update and WSC Experiences
<http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/TC000066>

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

Numerics

- 1 MB pages 50
 - page table entry 50
- WebSphere Application Server 50

A

- address space identifier number (ASID) 8
- APAR (authorized program analysis report) 9
- ASID (address space identifier number) 8
- authorized program analysis report (APAR) 9

B

- basic counters 35
- BC (business class) 1
- business class (BC) 1

C

- central processor assist for cryptographic function (CPACF) 6
- control sections (CSECTs) 8
- counter sets 6
 - basic counter set 6, 35
 - cycle count 35
 - instruction count 35
 - L1 data-cache directory write count 36
 - L1 data-cache penalty cycle count 36
 - L1 instruction-cache directory write count 35
 - L1 instruction-cache penalty cycle count 36
 - coprocessor-group counter set 6
 - crypto-activity 6, 35, 38
 - extended 6, 35, 40
 - group 6, 17, 35
 - problem-state 6, 35, 37
- counters collection 2
- counters control information 10
- counters data 10
 - CPU Measurement Facility (CPU MF) 6, 24
 - file names 24
 - IBM capacity planning tools 6
 - Java 7
 - Unix file 34
- counters function 2
- CPACF (central processor assist for cryptographic function) 6
- CPU Measurement Facility (CPU MF) 1, 11, 47
 - activating authorities 16
 - basic sampling data layout 55
 - controlling counters collection 25
 - cost 28
 - counters data 6, 24
 - data collection duration 26
 - enablement 12

- file system mount 20
- hardware requirements 12
- HMC and SE required changes 13
- Level 1 cache 4, 72
- Level 1.5 cache 72
- passing data to HIS 8
- prerequisite 12
 - hardware 2, 12
 - System z10 12
 - software 2, 13
- RACF setup 17
- samples collection 2
- samples compiler register allocation 7
- sampling 7–8
 - control information 10
 - data 8, 24
 - data file location 57
 - data file names 24
 - frequency 8
 - default value 8
 - function 2
 - multiple logical CPUs 8
 - multiple logical CPUs and shared engines 26
 - overhead 26
 - records 8, 55
- setup checklist 12
- SMF records 34
- SMF setup 19
- starting data collection 11, 22
- System z 2
 - Application Assist Processor (zAAP) 2
 - Integrated Information Processor (zIIP) 2
- traditional systems management tools 2
- cryptographic-related functions 38
 - Advanced Encryption Standard 38
 - AES blocked cycle count 39
 - AES blocked function count 39
 - AES cycle count 39
 - AES function count 39
 - Data Encryption Algorithm (DEA) 38
 - blocked cycle count 39
 - cycle count 39
 - function count 39
 - PRNG blocked cycle count 39
 - PRNG blocked function count 38
 - PRNG cycle count 38
 - PRNG function count 38
 - pseudo random number generator 38
 - Secure Hash Algorithm 38
 - SHA blocked cycle count 39
 - SHA blocked function count 39
 - SHA cycle count 39
 - SHA function count 39
- CSECTs (control sections) 8
- cycles per instruction metrics 42

D

data definition name (DDNAME), control information 10
Data Encryption Algorithm (DEA) 39
 blocked cycle count 39
 cycle count 39
data in memory (counters) 47, 50
data translation look-aside buffer (DTLB) 73
DB2 6
 Performance Expert 6
 V10 50
DDNAME (data definition name) 10
DEA (Data Encryption Algorithm) 39
direct access storage device (DASD) 7
 volume 7
DTLB (data translation look-aside buffer) 73

E

EC (enterprise class) 1
enterprise class (EC) 1
extended counters, common region segment table entry 73

H

hardware instrumentation services (HIS) 8, 48
 APARs 13
 commands 23
 commands for HIS started task 22
 control information 10
 data definition name (DDNAME) 10
 DURATION 10
 PATH 10
 sample HIS started task JCL 21
 samples per minute 10
 starting HIS started tasks 21
 stopping HIS started tasks 22
 TITLE 10
 Workload Manager 19
 z/OS 9
Hardware Management Console (HMC) 13
 sampling and counters 13
 user ID requirement 14
hardware prerequisites (CPU MF) 2, 12
hardware prerequisites (CPU MF), System z10 12
HiperDispatch 6–7, 47, 49
HMC (Hardware Management Console) 13

J

job control language (JCL), HIS 21

L

logical partition (LPAR) 2, 49
 changing security 15
 CICS 3, 6, 13
 IMS 3
 input/output (I/O) 4
 LoIO-mix 3
 OLTP-T 3

TI-mix 3
workloads 3, 47

M

map considerations 65
map data 10
map file names 25
map function 8, 25
map records format 63
map records layout 63
MODIFY command 9

O

OGET command 59

P

percentage of problem state metrics 43
processing unit (PU) 6, 8
program status word (PSW) 6
PSW (program status word) 6
PU (processing unit) 6, 8

R

Redbooks Web site 77
Relative Nest Intensity 48
 defined 5
 the nest 5
Resource Access Control Facility (RACF) 8
 application identify mapping 18
 authorizations for CPU MF 17
 STARTED profile 18
 user ID for HIS 17
Resource Measurement Facility (RMF) 4, 6
 I/O reports 50

S

sampling 24, 55
 CPU Measurement Facility (CPU MF) 7–8, 55
 data 24
 data file location 57
 data file names 24
 multiple logical CPUs and shared engines 26
 overhead 26
 records 55
SE (Support Element) 9, 15
software prerequisites
 CPU Measurement Facility (CPU MF) 2, 13
Support Element (SE) 9, 15
System Management Facilities (SMF) 9
 CPU MF setup 19
 IDCAMS PRINT 42
 records 40, 42, 47
 CPU Measurement Facility (CPU MF) 34
Type 113 record 42
 layout 41
Type 113 records 23, 40, 48
using counters 23

System z 2
 Application Assist Processor (zAAP) 2
 CPU Measurement Facility (CPU MF) 2
 Integrated Information Processor (zIIP) 2
 Large Systems Performance Reference 2

T

translation look-aside buffer (TLB) 6, 50, 73

U

UNIX file system 8, 10
UNIX file system considerations 20
UNIX sample counters 24
UNIX sampling file sizes 21

W

Workload Manager (WLM) 8
 sampling considerations 19

Z

z/OS 50
 hardware instrumentation services (HIS) 9
z10 71
 extended counter 72
z196 71
 extended counter 74
 Off Book 74
 Off Chip 74
 On Book 74
 On Chip 74
zAAPs (sampling data) 52
zCP3000 4, 7
zIIPs (sampling data) 52
zPCR 4, 7
 Processor Capacity Reference 48



Setting Up and Using the IBM System z CPU Measurement Facility with z/OS



Understand CPU MF and discover its capabilities with data collection

See how to set up and manage CPU MF for your enterprise

Learn how to benefit from the data collected by CPU MF

This IBM Redpaper publication can help you install and manage the IBM System z CPU Measurement Facility (CPU MF) capability. In this paper, you can learn how CPU MF gathers data and how this data can be used to more effectively manage your mainframe environment. You can also learn how this data can be used by IBM to help you more accurately characterize your workloads in preparation for capacity upgrades.

The topics covered in this paper target system programmers, capacity planners, or other technical personnel with working knowledge of systems and capacity.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks