AMD K86™ Family BIOS and Software Tools Developers Guide

AMD

Contents

List of Figures

List of Tables

Revision History

Date	Rev	Description
Sept 1996	A	Initial Release
Mar 1997	B	Added write allocation information for K86 family of processors. See "Write Allocate Registers" on page 82 for information about the AMD-K5 processor and "Write Handling Control Register (WHCR)" on page 119 for information about the AMD-K6™ MMX™ enhanced processor.
Mar 1997	B	Added Test and Debug section for the AMD-K6 MMX enhanced processor. See "AMD-K6™ Processor Test and Debug" on page 105 for more information.
Mar 1997	C	Reorganized entire guide
Apr 1997	D	Changed BIOS boot string for the AMD-K6 processor in Table 2, "Summary of AMD-K6™ MMX™ Enhanced Processor CPU IDs and BIOS Boot Strings," on page 4.
June 1997	E	Revised document to comply with MMX trademark.
June 1997	E	Replaced overbar with # to identify active-Low signals.
June 1997	E	Revised information in "Write Handling Control Register (WHCR)" on pages 119 through 121.
June 1997	E	Added (tm) to recommended boot-string for the AMD-K6 MMX enhanced processor on pages 3, 4, and 95.

1

Introduction

This document highlights the BIOS and software modifications required to fully support the K86™ family of processors, which includes the AMD-K5™ processor and the AMD-K6™ MMX™ enhanced processor.

There can be more than one way to implement the functionality detailed in this document, and the information provided is for demonstration purposes.

Audience

It is assumed that the reader possesses the proper knowledge of the K86 processors, the x86 architecture, and programming requirements to understand the information presented in this document.

2

CPU Identification Algorithms

The CPUID instruction provides complete information about the processor (vendor, type, name, etc.) and its capabilities (features). After detecting the processor and its capabilities, software can be accurately tuned to the system for maximum performance and benefit to users. For example, game software can test the performance level available from a particular processor by detecting the type or speed of the processor. If the performance level is high enough, the software can enable additional capabilities or more advanced algorithms. Another example involves testing whether the processor supports MMX™ technology. If the software finds this feature present when it checks the feature bits, it can utilize these more powerful instructions for better performance on new multimedia software.

For more detailed information refer to the *AMD Processor Recognition Application Note*, order# #20734, located at http://www.amd.com

Tables 1 and 2 outline the family codes and model codes for the AMD K86 processors. Table 1 shows the CPU speed, the 'P-Rating', and the recommended BIOS boot-string associated with each AMD-K5 processor.

Table 2 shows the recommended BIOS boot-string for the AMD-K6 MMX enhanced processor. This recommended boot-string is 'AMD-K6(tm)/XXX'. The value for XXX is

determined by calculating the core frequency of the processor. Use the Time Stamp Counter (TSC) to 'clock' a timed operation and compare the result to the Real Time Clock (RTC) to determine the operating frequency.

Note: *Tables 1 and 2 contain information intended to prepare the infrastructure for potential future products. These products may or may not be announced, but BIOS software should be prepared to support these options.*

Table 1. Summary of AMD-K5™ Processor CPU IDs and BIOS Boot Strings

Instruction Family Code	Model Code	CPU Speed (MHz)	CPU Bus Speed (MHz)	Recommended BIOS Boot-String	CPUID Functions 8000_0002, 3, 4 Return Values
5 (AMD-K5™ Processor)	0	75	50	AMD-K5-PR75	undefined
		90	60	AMD-K5-PR90	undefined
		100	66	AMD-K5-PR100	undefined
	1	90	60	AMD-K5-PR120	AMD-K5(tm) Processor
		100	66	AMD-K5-PR133	AMD-K5(tm) Processor
	2	105	60	AMD-K5-PR150	AMD-K5(tm) Processor
		116.7	66	AMD-K5-PR166	AMD-K5(tm) Processor
	3	133	66	AMD-K5-PR200	AMD-K5(tm) Processor

Table 2. Summary of AMD-K6™ MMX™ Enhanced Processor CPU IDs and BIOS Boot Strings

Instruction Family Code	Model Code	CPU Speed (MHz)	CPU Bus Speed (MHz)	Recommended BIOS Boot-String Display
5 (AMD-K6™ MMX™ Enhanced Processor)	6	TBD	60	AMD-K6(tm)/XXX
		TBD	66	AMD-K6(tm)/XXX

3

AMD-K5™ Processor

The AMD-K5 processor is socket 7-compatible and software-compatible with the Pentium® processor. Compatible in this sense means the devices are pin-for-pin compatible and that the same software can be executed on both processors with no software modifications.

The BIOS for the AMD-K5 processor requires minimal changes to fully support the AMD-K5 processor family.

BIOS Consideration Checklist

CPUID

- Use the CPUID instruction to properly identify the AMD-K5 processor.
- Determine the processor type, stepping and features using functions 0000_0001h and 8000_0001h of the CPUID instruction.
- Boot-up display: The processor name is retrieved using CPUID extended functions 8000_0002h, 8000_0003h, and 8000_0004h. See “CPU Identification Algorithms” on page 3 for more information.

CPU Speed Detection

- Use speed detection algorithms that do not rely on repetitive instruction sequences.
- Use the Time Stamp Counter (TSC) to 'clock' a timed operation and compare the result to the Real Time Clock (RTC) to determine the operating frequency. See the example of frequency-determination assembler code available on the AMD website at http://www.amd.com.
- Display the P-Rating shown in Table 1, "Summary of AMD-K5™ Processor CPU IDs and BIOS Boot Strings," on page 4.

Model-Specific Registers (MSRs)

- Access only MSRs implemented in the AMD-K5 processor.
- Program the write allocate registers—Hardware Configuration Register (HWCR), Write Allocate Top-of-Memory and Control Register (WATMCR), and Write Allocate Programmable Memory Range Register (WAPMRR). See "Write Allocate Registers" on page 82 and the *Implementation of Write Allocate in the K86™ Processors Application Note*, order# 21326 for more information.

Cache Testing

- Perform cache testing on the AMD-K5 processor using the Array Access Register MSR. See "Array Access Register (AAR)" on page 28 for more information.

SMM Issues

- The System Management Mode (SMM) functionality of the AMD-K5 processor is identical to Pentium.
- Implement the AMD-K5 processor SMM state-save area in the same manner as Pentium except for the IDT Base and possibly Pentium processor-reserved areas. See "AMD-K5™ Processor System Management Mode (SMM)" on page 7 for more information.

AMD-K5™ Processor System Management Mode (SMM)

System Management Mode (SMM) is an alternate operating mode entered by way of a system management interrupt (SMI) and handled by an interrupt service routine. SMM is designed for system control activities such as power management. These activities appear transparent to conventional operating systems like DOS and Windows. SMM is primarily targeted for use by the Basic Input Output System (BIOS) and specialized low-level device drivers. The code and data for SMM are stored in the SMM memory area, which is isolated from main memory.

The processor enters SMM by the system logic's assertion of the SMI# interrupt and the processor's acknowledgment by the assertion of SMIACT#. At this point the processor saves its state into the SMM memory state-save area and jumps to the SMM service routine. The processor returns from SMM when it executes the RSM (resume) instruction from within the SMM service routine. Subsequently, the processor restores its state from the SMM save area, de-asserts SMIACT#, and resumes execution with the instruction following the point where it entered SMM.

The following sections summarize the SMM state-save area, entry into and exit from SMM, exceptions and interrupts in SMM, memory allocation and addressing in SMM, and the SMI# and SMIACT# signals.

Operating Mode and Default Register Values

The software environment within SMM has the following characteristics:

- Addressing and operation in Real mode
- 4-Gbyte segment limits
- Default 16-bit operand, address, and stack sizes, although instruction prefixes can override these defaults
- Control transfers that do not override the default operand size truncate the EIP to 16 bits
- Far jumps or calls cannot transfer control to a segment with a base address requiring more than 20 bits, as in Real mode segment-base addressing

- A20M# is masked
- Interrupt vectors use the Real-mode interrupt vector table
- The IF flag in EFLAGS is cleared (INTR not recognized)
- The TF flag in EFLAGS is cleared
- The NMI and INIT interrupts are disabled
- Debug register DR7 is cleared (debug traps disabled)

Figure 1 shows the default map of the SMM memory area. It consists of a 64-Kbyte area, between 0003_0000h and 0003_FFFFh, of which the top 32 Kbytes (0003_8000h to 0003_FFFFh) must be populated with RAM. The default code-segment (CS) base address for the area—called the SMM base address—is at 0003_0000h. The top 512 bytes (0003_FE00h to 0003_FFFFh) contain a fill-down SMM state-save area. The default entry point for the SMM service routine is 0003_8000h.

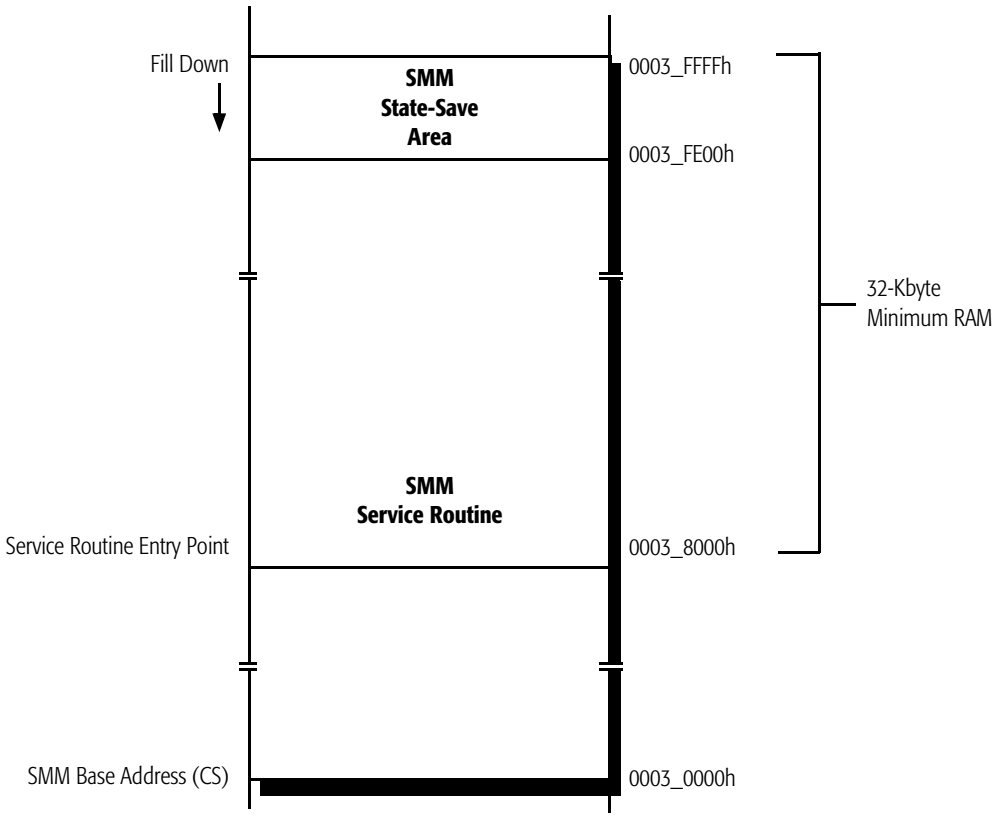


Figure 1. SMM Memory

SMM Initial Register Values

Table 3 shows the initial state of registers when entering SMM.

Table 3. Initial State of Registers in SMM

Register	Initial Contents		
	Selector	Base	Limit
CS	3000h	0003_0000h	4 Gbytes
DS	0000h	0000_0000h	4 Gbytes
ES	0000h	0000_0000h	4 Gbytes
FS	0000h	0000_0000h	4 Gbytes
GS	0000h	0000_0000h	4 Gbytes
SS	0000h	0000_0000h	4 Gbytes
General-Purpose Registers	Unmodified		
EFLAGS	0000_0002h		
EIP	0000_8000h		
CR0	Bits 0, 2, 3, and 31 cleared (PE, EM, TS, and PG); remainder are unmodified		
CR4	0000_0000h		
GDTR	Unmodified		
LDTR	Unmodified		
IDTR	Unmodified		
TR	Unmodified		
DR7	0000_0400h		
DR6	Undefined		

SMM State-Save Area

When the processor acknowledges an SMI interrupt by asserting SMIACT#, it saves its state in the 512-byte SMM state-save area shown in Table 4. The save begins at the top of the SMM memory area (SMM Base Address + FFFFh) and fills down to SMM base address + FE00h.

Table 4 shows the offsets in the SMM state-save area relative to the SMM base address. The SMM service routine can alter any of the read and write values in the state-save area. The contents of any reserved locations in the state-save area are not necessarily the same between the AMD-K5 processor and Pentium or 486 processors.

Table 4. SMM State-Save Area Map

Offset (Hex)	Contents
FFFC	CR0
FFF8	CR3
FFF4	EFLAGS
FFF0	EIP
FFEC	EDI
FFE8	ESI
FFE4	EBP
FFE0	ESP
FFDC	EBX
FFD8	EDX
FFD4	ECX
FFD0	EAX
FFCC	DR6 (FFFF_CFF3h)
FFC8	DR7
FFC4	TR
FFC0	LDTR
FFBC	GS
FFB8	FS
FFB4	DS
FFB0	SS
FFAC	CS
FFA8	ES
FFA4	I/O Trap Dword
FFA0	*reserved*
FF9C	I/O Trap EIP
FF98	*reserved*
FF94	*reserved*
FF90	IDT Base
FF8C	IDT Limit
FF88	GDT Base
FF84	GDT Limit
FF80	TSS Attributes
FF7C	TSS Base
FF78	TSS Limit

Table 4. SMM State-Save Area Map (continued)

Offset (Hex)	Contents
FF74	LDT Attributes
FF70	LDT Base
FF6C	LDT Limit
FF68	GS Attributes
FF64	GS Base
FF60	GS Limit
FF5C	FS Attributes
FF58	FS Base
FF54	FS Limit
FF50	DS Attributes
FF4C	DS Base
FF48	DS Limit
FF44	SS Attributes
FF40	SS Base
FF3C	SS Limit
FF38	CS Attributes
FF34	CS Base
FF30	CS Limit
FF2C	ES Attributes
FF28	ES Base
FF24	ES Limit
FF20	*reserved*
FF1C	*reserved*
FF18	*reserved*
FF14	CR2
FF10	CR4
FF0C	I/O Restart ESI
FF08	I/O Restart ECX
FF04	I/O Restart EDI
FF02	Halt Restart Slot
FF00	I/O Trap Restart Slot
FEFC	SMM Revision Identifier
FEF8	SMM Base Address
FE00–FEF4	*reserved*

SMM Revision Identifier

The SMM revision identifier at offset FEFCh in the SMM state-save area specifies the version of SMM and the extensions available on the processor. The SMM revision identifier fields, shown in Table 5, are as follows:

- *Bits 31–18—reserved*
- *Bit 17*—SMM base address relocation (always 1 = enabled)
- *Bit 16*—I/O trap restart (always 1 = enabled)
- *Bits 15–0*—SMM revision level = 0000

Table 5. SMM Revision Identifier Fields

Bits 31–18	Bit 17	Bit 16	Bits 15–0
Reserved	SMM Base Relocation	I/O Trap Extension	SMM Revision Level
0	1	1	0000

Note: *The I/O trap restart and the SMM base address relocation functions are always enabled in the AMD-K5 processor and do not need to be specifically enabled.*

SMM Base Address

During RESET, the processor sets the code-segment (CS) base address for the SMM memory area—the SMM base address—to its default, 0003_0000h. The SMM base address at offset FEF8h in the SMM state-save area can be changed by the SMM service routine to any address aligned to a 32-Kbyte boundary. (Locations not aligned to a 32-Kbyte boundary cause the processor to enter the Shutdown state when executing the RSM instruction.)

In some operating environments it may be desirable to relocate the 64-Kbyte SMM memory area to a high memory area to provide more low memory for legacy software. During system initialization, the base of the 64-Kbyte SMM memory area is relocated by the BIOS. To relocate the SMM base address, the system enters the SMM handler at the default address. This handler changes the SMM base address location in the SMM state-save area, copies the SMM handler to the new location, and exits SMM.

The next time SMM is entered, the processor saves its state at the new base address. This new address is used for every SMM until the SMM base address in the SMM state-save area is changed or a hardware reset occurs.

Auto Halt Restart Slot

During entry into SMM, the halt restart slot at offset FF02h in the SMM state-save area indicates whether SMM was entered from the Halt state. Before returning from SMM, the halt restart slot can be written to by the SMM service routine to specify whether the return from SMM should take the processor back to the Halt state or to the instruction-execution state specified by the SMM state-save area.

On entry into SMM, the halt restart slot is configured as follows:

- *Bits 15–1*—Undefined
- *Bit 0*—Point of entry to SMM:

  1 = entered from Halt state

  0 = not entered from Halt state

After entry into the SMI handler and before returning from SMM, the halt restart slot can be written using the following definition:

- *Bits 15–1*—Undefined
- *Bit 0*—Point of return from SMM

  1 = return to Halt state

  0 = return to state specified by SMM state-save area

If the return from SMM takes the processor back to the Halt state, the HLT instruction is not re-executed, but the Halt special bus cycle is driven on the bus after the return.

I/O Trap Dword

If the assertion of SMI is recognized on the boundary of an I/O instruction, the I/O trap dword at offset FFA4h in the SMM state-save area contains information about the instruction. The fields of the I/O trap dword, shown in Table 6, are configured as follows:

- *Bits 31–16*—I/O port address
- *Bit 15*—I/O string operation (1 = string, 0 = non-string)
- *Bits 14–2—reserved*
- *Bit 1*—Valid I/O instruction (1 = valid, 0 = invalid)
- *Bit 0*—Input or output instruction (1 = INx, 0 = OUTx)

Table 6. I/O Trap Dword Fields

Bits 31–16	Bit 15	Bit 14–2	Bit 1	Bit 0
I/O Port Address	I/O String Operation	Reserved	Valid I/O Instruction	Input or Output

The I/O trap dword is related to the I/O trap restart slot, described below. Bit 1 of the I/O trap dword (the valid bit) should be tested if the I/O trap restart slot is to be changed.

I/O Trap Restart Slot

The I/O trap restart slot at offset FF00h in the SMM state-save area specifies whether the trapped I/O instruction should be re-executed on return from SMM. This slot in the state-save area is called the I/O instruction restart function. Re-executing a trapped I/O instruction is useful, for example, if an I/O write occurs to a disk that is powered down. The system logic monitoring such an access can assert SMI#. Then the SMM service routine can query the system logic, detect a failed I/O write, take action to power-up the I/O device, enable the I/O trap restart slot feature, and return from SMM.

The fields of the I/O trap restart slot are defined as follows:

- *Bits 31–16—reserved*
- *Bits 15–0*—I/O instruction restart on return from SMM:

  0000h = execute the next instruction after the trapped I/O instruction

  00FFh = re-execute the trapped I/O instruction

Table 7 shows the format of the I/O trap restart slot.

Table 7. I/O Trap Restart Slot

31–16	15–0
Reserved	I/O Instruction restart on return from SMM: ■ 0000h = execute the next instruction after the trapped I/O instruction ■ 00FFh = re-execute the trapped I/O instruction

The processor initializes the I/O trap restart slot to 0000h upon entry into SMM. If SMM is entered as a result of a trapped I/O instruction, the processor indicates the validity of the I/O instruction by setting or clearing bit 1 of the I/O trap dword at offset FFA4h in the SMM state-save area. The SMM service routine should test bit 1 of the I/O trap dword to determine if a valid I/O instruction was being executed when entering SMM and before writing the I/O trap restart slot. If the I/O instruction is valid, the SMM service routine can safely rewrite the I/O trap restart slot with the value 00FFh, causing the processor to re-execute the trapped I/O instruction when the RSM instruction is executed. If the I/O instruction is invalid, writing the I/O trap restart slot has undefined results.

If a second SMI# is asserted and a valid I/O instruction was trapped by the first SMM handler, the CPU services the second SMI# prior to re-executing the trapped I/O instruction. The second entry into SMM never has bit 1 of the I/O trap dword set, and the second SMM service routine must not rewrite the I/O trap restart slot.

During a simultaneous SMI# I/O instruction trap and debug breakpoint trap, the AMD-K5 processor first responds to the SMI# and postpones recognizing the debug exception until after returning from SMM via the RSM instruction. If the debug registers DR3–DR0 are used while in SMM, they must be saved

and restored by the SMM handler. The processor automatically saves and restores DR7–DR6. If the I/O trap restart slot in the SMM state-save area contains the value 00FFh when the RSM instruction is executed, the debug trap does not occur until after the I/O instruction is re-executed.

Exceptions and Interrupts in SMM

When SMM is entered, the processor disables both INTR and NMI interrupts. The processor disables INTR interrupts by clearing the IF flag in the EFLAGS register. To enable INTR interrupts within SMM, the SMM handler must set the IF flag to 1.

Generating an INTR interrupt is a method for unmasking NMI interrupts in SMM. The processor recognizes the assertion of NMI within SMM immediately after the completion of an IRET. The NMI can thus be enabled by using a dummy INTR interrupt. Once NMI is recognized within SMM, NMI recognition remains enabled until SMM is exited, at which point NMI masking is restored to the state it was in before entering SMM.

Because the IF flag is cleared when entering SMM, the HLT instruction should not be executed in SMM without first setting the IF bit to 1. Setting this bit to 1 enables the processor to exit the Halt state by means of an INTR interrupt.

Table 8 summarizes the behavior of all interrupts in SMM.

Table 8. Summary of Interrupts and Exceptions

Priority	Description	Type	Sampling[5]	Vector[1]	Acknowledgment	Point of Interruptibility[6]
1	INTn instructions and all other software exceptions	exceptions	internal	0–255	none	Entry to service routine
2	BUSCHK#	interrupt	level-sensitive	18[2]	none	Entry to service routine[2]
3	R/S#	interrupt	level-sensitive	none	PRDY	Negation of PRDY
4	FLUSH#	interrupt	edge-triggered[4]	none	FLUSH#-Acknowledge special bus cycle	BRDY# of FLUSH# Acknowledge bus cycle
5	SMI#	interrupt	edge-triggered[4]	SMM[3]	SMIACT#	Entry to SMM service routine[7]
6	INIT	interrupt	edge-triggered[4]	BIOS	none	Completion of initialization
7	NMI	interrupt	edge-triggered[4]	2	none	NMI interrupts: IRET from service routine. All others: Entry to service routine.
8	INTR	interrupt	level-sensitive	0–255	Interrupt acknowledge special bus cycle	Entry to service routine
9	STPCLK#	interrupt	level-sensitive	none	Stop-Grant special bus cycle	Negation of STPCLK#

Notes:

1. *For interrupts with vectors, the processor saves its state prior to accessing the service routine and changing the program flow. Interrupts without vectors do not change program flow; instead, they simply pause program flow for the duration of the interrupt function and return to where they left off.*
2. *If the Machine Check Enable (MCE) bit in CR4 is set to 1.*
3. *The entry point for the SMI interrupt handler is at offset 8000h from the SMM Base Address.*
4. *Only the edge-triggered interrupts are latched when asserted. All interrupts are recognized at the next instruction retirement boundary.*
5. *If a bus cycle is in progress, EWBE must be asserted before the interrupt is recognized.*
6. *For external interrupts (most exceptions, by contrast, are recognized when they occur). External interrupts are recognized at instruction boundaries. When MOV or POP instructions load SS, interruptibility is delayed until after the next instruction, thus allowing both SS and the corresponding SP to load.*
7. *After assertion of SMI, subsequent assertions of SMI are masked to prevent recursive entry into SMM. However, other exceptions or interrupts (except INIT and NMI) are taken in the SMM service routine.*

AMD-K5™ Processor RESET State

The state of all architecture registers and Model-Specific Registers (MSRs) after the AMD-K5 processor has completed its initialization due to the recognition of the assertion of RESET are shown in Table 9.

Table 9. State of the AMD-K5™ Processor After RESET

Register	RESET State	Notes
GDTR	base:0000_0000 limit:0000h	
IDTR	base:0000_0000 limit:0000h	
TR	0000h	
LDTR	0000h	
EIP	FFFF_FFF0h	
EFLAGS	0000_0002h	
EAX	0000_0000h	1
EBX	0000_0000h	
ECX	0000_0000h	
EDX	0000_05XXh	2
ESI	0000_0000h	
EDI	0000_0000h	
EBP	0000_0000h	
ESP	0000_0000h	
CS	F000h	
SS	0000h	
DS	0000h	
ES	0000h	
FS	0000h	
GS	0000h	
FPU Stack R7–R0	0000_0000_0000_0000_0000h	

Notes:

1. *The contents of EAX indicate if BIST was successful. If EAX = 0000_0000h, then BIST was successful. If EAX is non-zero, BIST failed.*
2. *EDX contains the AMD-K5 processor signature, which is comprised of the instruction family, model, and stepping.*
3. *These MSRs are described in "AMD-K5™ Processor x86 Architecture Extensions" on page 57.*
4. *The AMD-K5 processor supports write allocate only on Models 1, 2, and 3, with a Stepping of 4 or greater.*

Table 9. State of the AMD-K5™ Processor After RESET (continued)

Register	RESET State	Notes
FPU Control Word	0040h	
FPU Status Word	0000h	
FPU Tag Word	5555h	
FPU Instruction Pointer	0000_0000_0000h	
FPU Data Pointer	0000_0000_0000h	
FPU Opcode Register	000_0000_0000b	
CR0	6000_0010h	
CR2	0000_0000h	
CR3	0000_0000h	
CR4	0000_0000h	
DR7	0000_0400h	
DR6	FFFF_0FF0h	
DR3	0000_0000h	
DR2	0000_0000h	
DR1	0000_0000h	
DR0	0000_0000h	
MCAR	0000_0000_0000_0000h	
MCTR	0000_0000_0000_0000h	
TR12	0000_0000_0000_0000h	
TSC	0000_0000_0000_0000h	
AAR	0000_0000_0000_0000h	3
HWCR	0000_0000_0000_0000h	3
WATMCR	0000_0000_0000_0000h	3, 4
WAPMRR	0000_0000_000F_000Ah	3, 4

Notes:

1. *The contents of EAX indicate if BIST was successful. If EAX = 0000_0000h, then BIST was successful. If EAX is non-zero, BIST failed.*
2. *EDX contains the AMD-K5 processor signature, which is comprised of the instruction family, model, and stepping.*
3. *These MSRs are described in "AMD-K5™ Processor x86 Architecture Extensions" on page 57.*
4. *The AMD-K5 processor supports write allocate only on Models 1, 2, and 3, with a Stepping of 4 or greater.*

Segment Register Attributes

The selector portion of all segment registers is cleared. The access rights and attribute fields are set up as shown in Table 10.

Table 10. Segment Register Attribute Fields Initial Values

Attribute Field	Value	Description
G	0	Byte granularity
D/B	0	16-bit
P	1	Present
DPL	0	Privilege level
S	1	Application segment (except LDTR)
Type	2	Data, read-write

The limit fields are set to FFFFh. For CS, the base address is set to FFFF_0000h; for all others the base address is 0. Note that IDTR and GDTR consist of the just base and limit values, which are initialized to 0 and FFFFh, respectively.

State of the AMD-K5™ Processor After INIT

The assertion of INIT causes the processor to empty its pipelines, initialize most of its internal state, and branch to address FFFF_FFF0h—the same instruction execution starting point used after RESET. Unlike RESET, the processor preserves the contents of its caches, the floating-point state, the SMM base, MSRs, and the CD and NW bits of the CR0 register.

The edge-sensitive interrupts FLUSH# and SMI# are sampled and preserved during the INIT process and are handled accordingly after the initialization is complete. However, the processor resets any pending NMI interrupt upon sampling INIT asserted.

INIT can be used as an accelerator for 80286 code that requires a reset to exit from Protected mode back to Real mode.

AMD-K5™ Processor Test and Debug

The AMD-K5 processor has the following modes in which processor and system operation can be tested or debugged:

- *Hardware Configuration Register (HWCR)*—The HWCR is a MSR that contains configuration bits that enable cache, branch tracing, debug, and clock control functions.
- *Built-In Self-Test (BIST)*—Both normal and test access port (TAP) BIST.
- *Output-Float Test*—A test mode that causes the AMD-K5 processor to float all of its output and bidirectional signals.
- *Cache and TLB Testing*—The Array Access Register (AAR) supports writes and reads to any location in the tag and data arrays of the processor's on-chip caches and TLBs.
- *Debug Registers*—Standard 486 debug functions with an I/O-breakpoint extension.
- *Branch Tracing*—A pair of special bus cycles can be driven immediately after taken branches to specify information about the branch instruction and its target. The Hardware Configuration Register (HWCR) provides support for this and other debug functions.
- *Functional Redundancy Checking*—Support for real-time testing that uses two processors in a master-checker relationship.
- *Test Access Port (TAP) Boundary-Scan Testing*—The JTAG test access functions defined by the *IEEE Standard Test Access Port and Boundary-Scan Architecture (IEEE 1149.1-1990)* specification.
- *Hardware Debug Tool (HDT)*—The hardware debug tool (HDT), sometimes referred to as the debug port or Probe mode, is a collection of signals, registers, and processor microcode enabled when external debug logic drives R/S Low or loads the AMD-K5 processor's Test Access Port (TAP) instruction register with the USEHDT instruction.

The test-related signals are described in Chapter 5 of the *AMD-K5™ Processor Technical Reference Manual*, order# 18524. The signals include the following:

- FLUSH
- FRCMC
- IERR
- INIT
- PRDY
- R/S
- RESET
- TCK
- TDI
- TDO
- TMS
- TRST

The sections that follow provide details on each of the test and debug features.

Hardware Configuration Register (HWCR)

The Hardware Configuration Register (HWCR) is a MSR that contains configuration bits that enable cache, branch tracing, write allocation, debug, and clock control functions. The WRMSR and RDMSR instructions access the HWCR when the ECX register contains the value 83h, as described on page 90. Figure 2 and Table 11 show the format and fields of the HWCR.

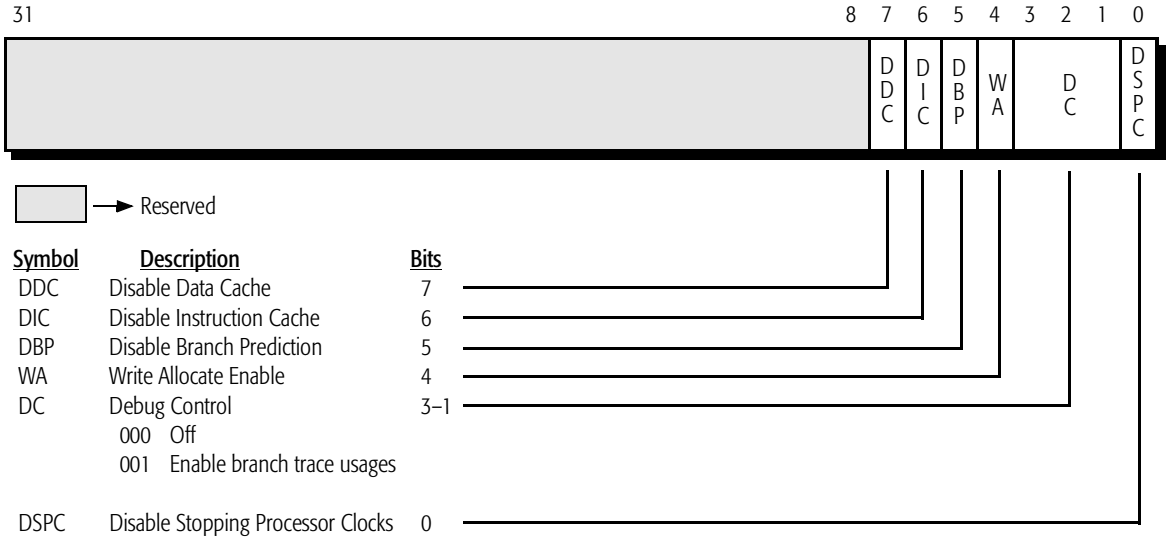


Figure 2. Hardware Configuration Register (HWCR)

Table 11. Hardware Configuration Register (HWCR) Fields

Bit	Mnemonic	Description	Function
31–8	–	–	*reserved*
7	DDC	Disable Data Cache	Disables data cache 0 = enabled, 1 = disabled
6	DIC	Disable Instruction Cache	Disables instruction cache 0 = enabled, 1 = disabled
5	DBP	Disable Branch Prediction	Disables branch prediction 0 = enabled, 1 = disabled
4	WA*	Enable Write Allocate	Enables write allocation 0 = disabled, 1 = enabled

Note:
* *The AMD-K5 processor supports write allocate only on Models 1, 2, and 3, with a Stepping of 4 or greater.*

Table 11. Hardware Configuration Register (HWCR) Fields (continued)

Bit	Mnemonic	Description	Function
3–1	DC	Debug Control	Debug control bits: 000 Off (disable HWCR debug control) 001 Enable branch-tracing messages. See "Branch Tracing" on page 39. 010 *reserved* 011 *reserved* 100 *reserved* 101 *reserved* 110 *reserved* 111 *reserved*
0	DSPC	Disable Stopping Processor Clocks	Disables stopping of internal processor clocks in the Halt and Stop Grant states 0 = enabled, 1 = disabled

Note:
* *The AMD-K5 processor supports write allocate only on Models 1, 2, and 3, with a Stepping of 4 or greater.*

Built-In Self-Test (BIST)

The processor supports the following types of built-in self-test:

- *Normal BIST*—A built-in self-test mode typically used to test system functions after RESET
- *Test Access Port (TAP) BIST*—A self-test mode started by the TAP instruction, RUNBIST

All internal arrays except the TLB are tested in parallel by hardware. The TLB is tested by microcode. The AMD-K5 processor does not report parity errors on IERR for every cache or TLB access. Instead, the AMD-K5 fully tests its caches during the BIST. EADS should not be asserted during a BIST. The AMD-K5 accesses the physical tag array during BISTs, and these accesses can conflict with inquire cycles.

Normal BIST

The normal BIST is invoked if INIT is asserted at the falling edge of RESET. The BIST runs tests on the internal hardware that exercise the following resources:

- Instruction cache:
  - Linear tag directory
  - Instruction array
  - Physical tag directory
- Data cache:
  - Linear tag directory
  - Data array
  - Physical tag directory
- Entry-point and instruction-decode PLAs
- Microcode ROM
- TLB

The BIST runs a linear feedback shift register (LFSR) signature test on the microcode ROM in parallel with a March C test on the instruction cache, data cache, and physical tags. This is followed by the March C test on the TLB arrays and an LFSR signature test on the PLA, in that order. Upon completion of the PLA test, the processor transfers the test result from an internal Hardware Debug Test (HDT) data register to the EAX register for external access, resets the internal microcode, and begins normal code fetching.

The result of the BIST can be accessed by reading the lower 9 bits of the EAX register. If the EAX register value is 0000_0000h, the test completed successfully. If the value is not zero, the non-zero bits indicate where the failure occurred, as shown in Table 12. The processor continues with its normal boot process after the BIST is completed, whether the BIST passed or failed.

Table 12. BIST Error Bit Definition in EAX Register

Bit Number	Bit Value	
	0	1
31–9	No Error	Always 0
8	No Error	Data path
7	No Error	Instruction-cache instructions

Table 12. BIST Error Bit Definition in EAX Register (continued)

Bit Number	Bit Value	
	0	1
6	No Error	Instruction-cache linear tags
5	No Error	Data-cache linear tags
4	No Error	PLA
3	No Error	Microcode ROM
2	No Error	Data-cache data
1	No Error	Instruction cache physical tags
0	No Error	Data-cache physical tags

Test Access Port (TAP) BIST

The TAP BIST performs all the functions of the normal BIST, up to and including the PLA signature test, in the exact manner as the normal BIST. However, after the PLA test, the test result is not transferred to the EAX register.

The TAP BIST is started by loading and executing the RUNBIST instruction in the test access port, as described in "Boundary Scan Architecture Support" on page 41. When the RUNBIST instruction is executed, the processor enters into a reset mode that is identical to that entered when the RESET signal is asserted. Upon completion of the TAP BIST, the result remains in the BIST result register for shifting out through the TDO signal. The TRST signal must be asserted, or the TAP instruction must be changed, to exit TAP BIST and return to normal operation.

Output-Float Test

The Output-Float Test mode is entered if FLUSH is asserted before the falling edge of RESET. This causes the processor to place all of its output and bidirectional signals in the high-impedance state. In this isolated state, system board traces and connections can be tested for integrity and driveability. The Output-Float Test mode can only be exited by asserting RESET again.

On the AMD-K5 processor and Pentium, FLUSH# is an edge-triggered interrupt. On the 486 processor, however, the signal is a level-sensitive input.

Cache and TLB Testing

The internal cache for the AMD-K5 processor is divided into two caches—a 16-Kbyte, 4-way, set-associative instruction cache and an 8-Kbyte, 4-way, set-associative data cache. Cache and TLB testing is often done by the BIOS or operating system during power-up.

Note: *The AMD-K6 MMX enhanced processor does not contain these features. It contains a built-in self-test for all internal memories.*

The individual locations of all SRAM arrays on the AMD-K5 processor are accessible with the RDMSR and WRMSR instructions. To access an array location, set up the Array Access MSR code (82h) in ECX, and the array pointer (see page 28) in EDX. EAX holds the data to be read or written. Tests can be performed on the following arrays:

- *Data Cache*—8-Kbyte, 4-way, set-associative
  - Data array
  - Linear-tag array
  - Physical-tag array
- *Instruction Cache*—16-Kbyte, 4-way, set-associative
  - Instruction array
  - Linear-tag array
  - Physical-tag array
  - Valid-bit array
  - Branch-prediction bit array
- *4-Kbyte TLB*—128-entry, 4-way, set-associative
  - Linear-tag array
  - Page array
- *4-Mbyte TLB*—4-entry, fully associative
  - Linear-tag array
  - Page array

Array Access Register (AAR)

The 64-bit Array Access Register (AAR) is a MSR that contains a 32-bit *array pointer* that identifies the array location to be tested and 32 bits of *array test data* to be read or written. The WRMSR and RDMSR instructions access the AAR when the ECX register contains the value 82h, as described on page 90. Figure 3 shows the format of the AAR.

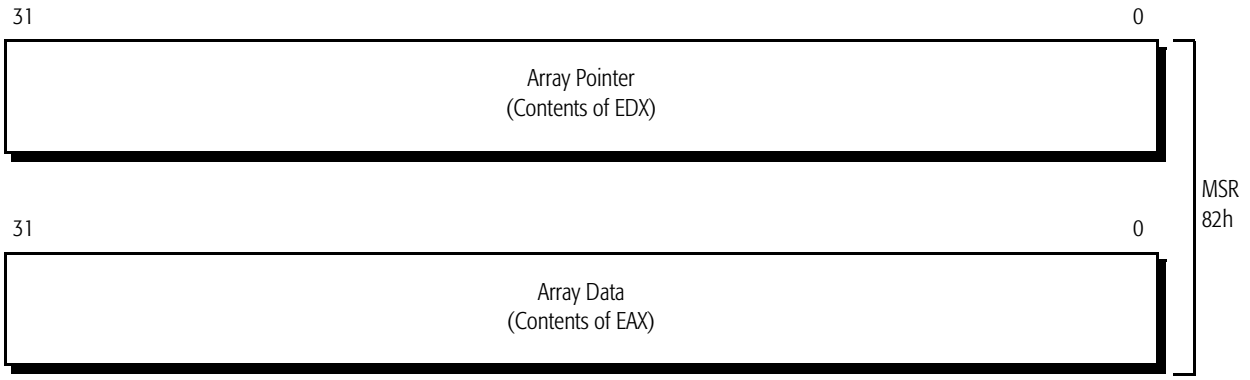


Figure 3. Array Access Register (AAR)

To read or write an array location, perform the following steps:

1. *ECX*—Enter 82h into ECX to access the 64-bit AAR.
2. *EDX*—Enter a 32-bit *array pointer* into EDX, as shown in Figures 4 through 12 (top).
3. *EAX*—Read or write 32 bits of *array test data* to or from EAX, as shown in Figures 4 through 12 (bottom).

Array Pointer

The array pointers entered in EDX (Figures 4 through 12, top) specify particular array locations. For example, in the data- and instruction-cache arrays, the way (or column) and set (or index) in the array pointer specify a cache line in the 4-way, set-associative array. The array pointers for data-cache data and instruction-cache instructions also specify a dword location within that cache line. In the data cache, this dword is 32 bits of data; in the instruction cache, this dword is two instruction bytes plus their associated pre-decode bits. For the 4-Kbyte TLB, the way and set specify one of the 128 TLB entries. In 4-Mbyte TLB, one of only four entries is specified.

Bits 7–0 of every array pointer encode the *array ID*, which identifies the array to be accessed, as shown in Table 13. To simplify multiple accesses to an array, the contents of EDX are

retained after the RDMSR instruction executes (EDX is normally cleared after a RDMSR instruction).

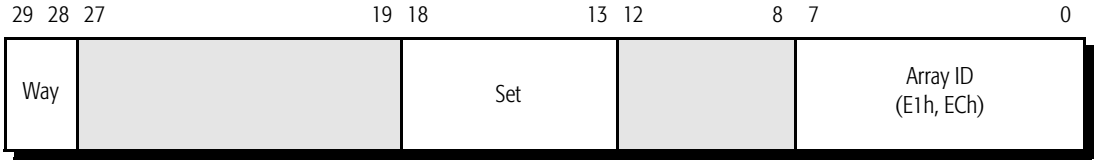
Table 13. Array IDs in Array Pointers

Array Pointer Bits 7–0	Accessed Array
E0h	Data Cache: Data
E1h	Data Cache: Linear Tag
ECh	Data Cache: Physical Tag
E4h	Instruction Cache: Instructions
E5h	Instruction Cache: Linear Tag
EDh	Instruction Cache: Physical Tag
E6h	Instruction Cache: Valid Bits
E7h	Instruction Cache: Branch-Prediction Bits
E8h	4-Kbyte TLB: Page
E9h	4-Kbyte TLB: Virtual Tag
EAh	4-Mbyte TLB: Page
EBh	4-Mbyte TLB: Virtual Tag

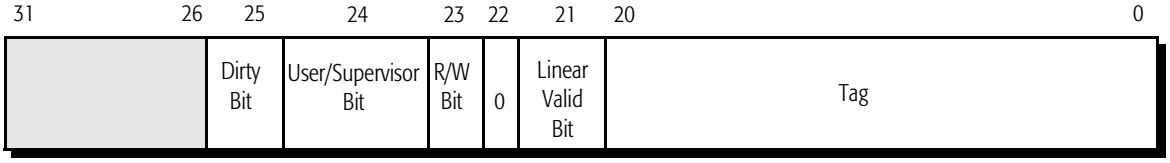
Array Test Data

EAX specifies the test data to be read or written with the RDMSR or WRMSR instruction (see Figures 4 through 12). For example, in Figure 4 (top) the array pointer in EDX specifies a way and set within the data-cache linear tag array (E1h in bits 7–0 of the array pointer) or the physical tag array (ECh in bits 7–0 of the array pointer). If the linear tag array (E1h) is accessed, the data read or written includes the tag and the status bits. The details of the valid fields in EAX are proprietary.

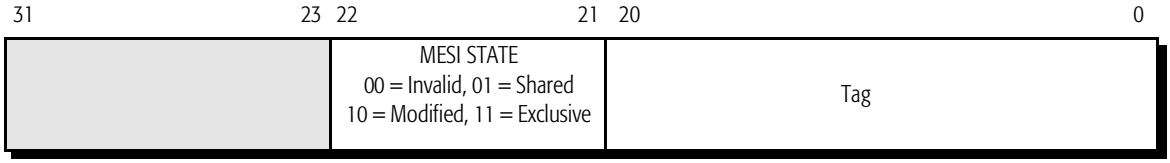
EDX: Array Pointer



EAX: Test Data



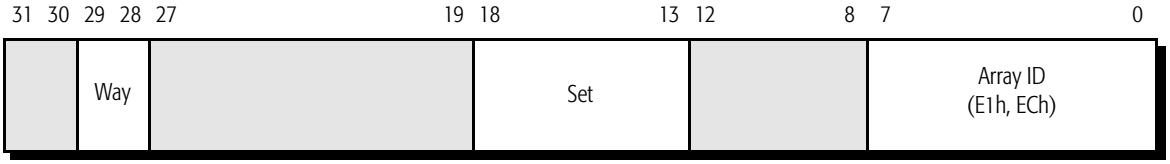
(E1h) Linear Tag



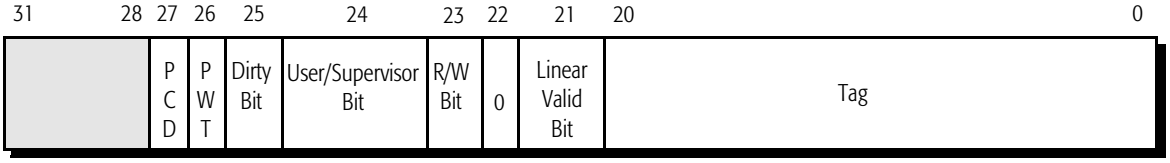
(ECh) Physical Tag

Figure 4. Test Formats: Dcache Tags for the AMD-K5™ Processor Model 0

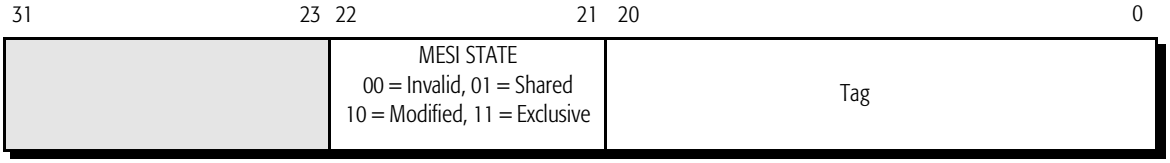
EDX: Array Pointer



EAX: Test Data



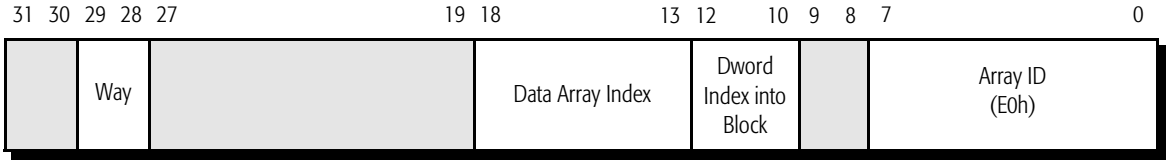
(E1h) Linear Tag



(ECh) Physical Tag

Figure 5. Test Formats: Dcache Tags for the AMD-K5™ Processor Model 1 and Greater

EDX: Array Pointer



EAX: Test Data

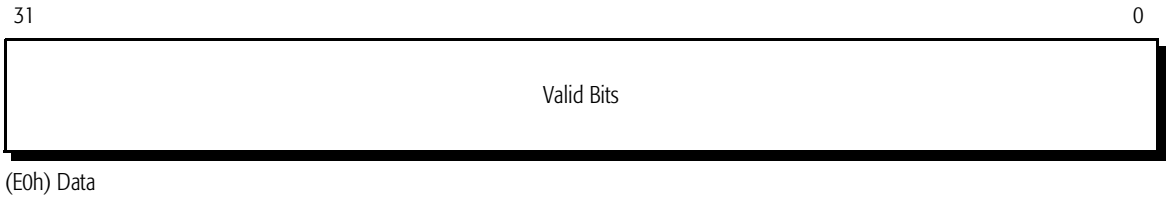
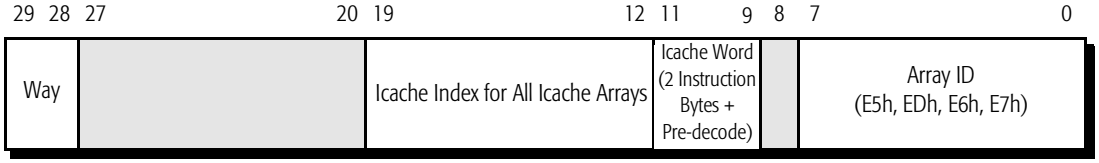
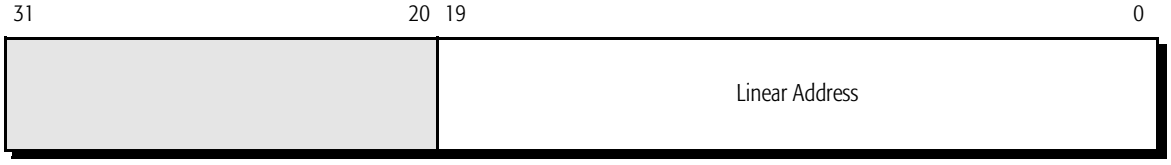


Figure 6. Test Formats: Dcache Data for All Models of the AMD-K5™ Processor

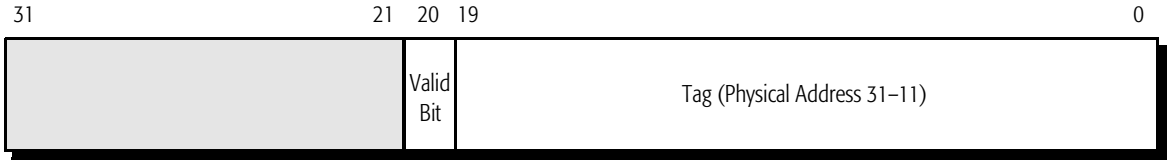
EDX: Array Pointer



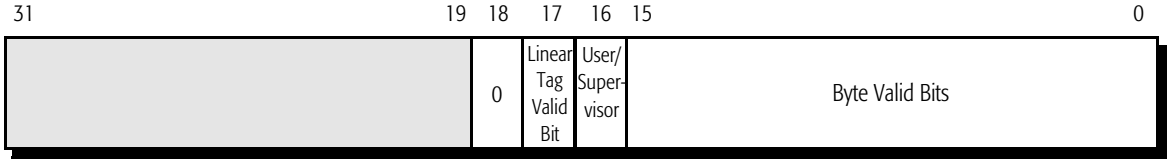
EAX: Test Data



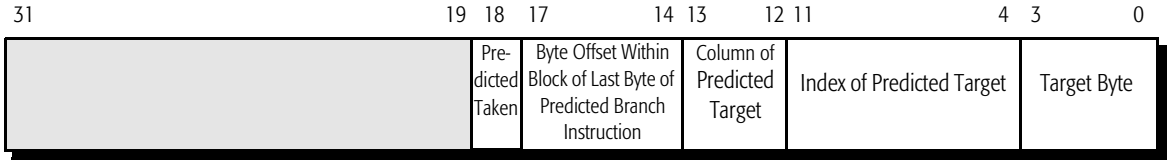
(E5h) Linear Tag



(EDh) Physical Tag



(E6h) Valid Bits



(E7h) Branch-Prediction Bits

Figure 7. Test Formats: Icache Tags for the AMD-K5™ Processor Model 0

EDX: Array Pointer

EAX: Test Data

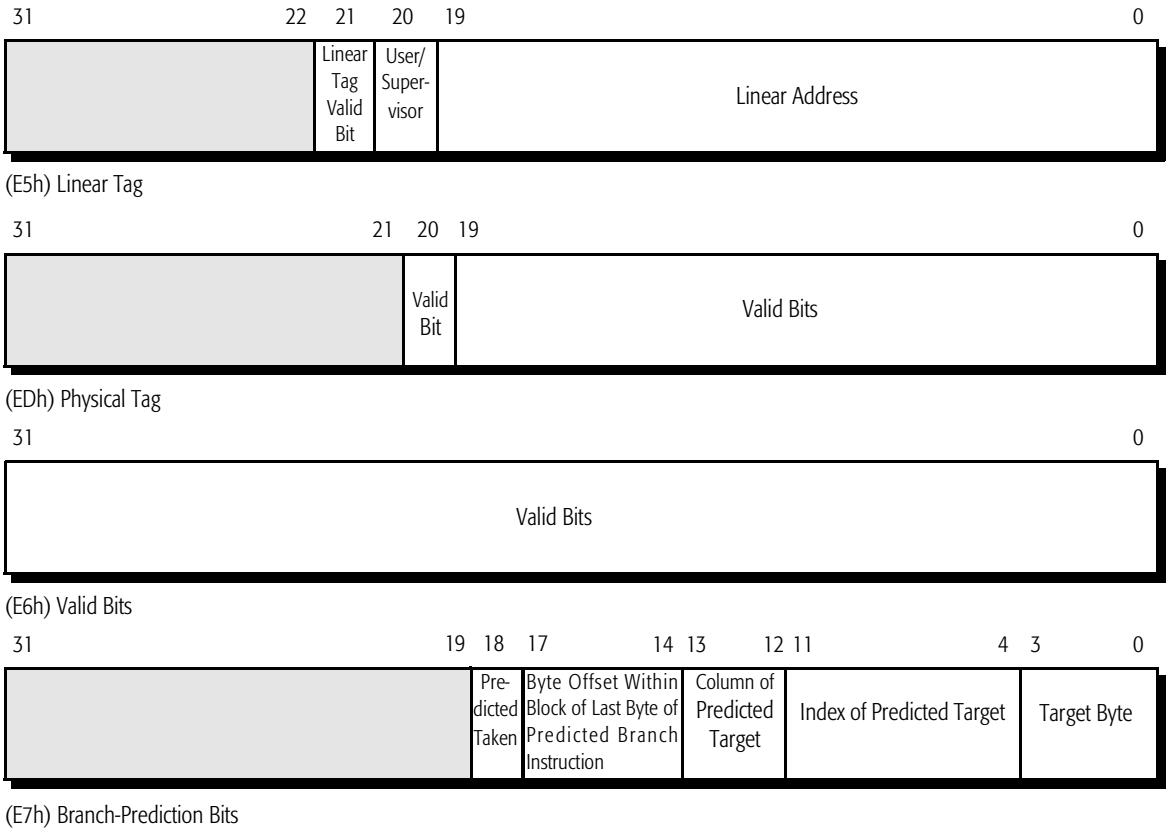
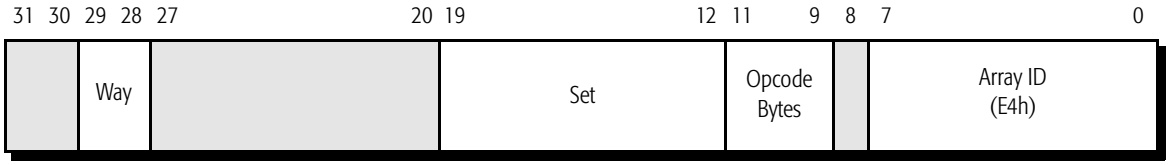


Figure 8. Test Formats: Icache Tags for the AMD-K5™ Processor Model 1 and Greater

EDX: Array Pointer



EAX: Test Data

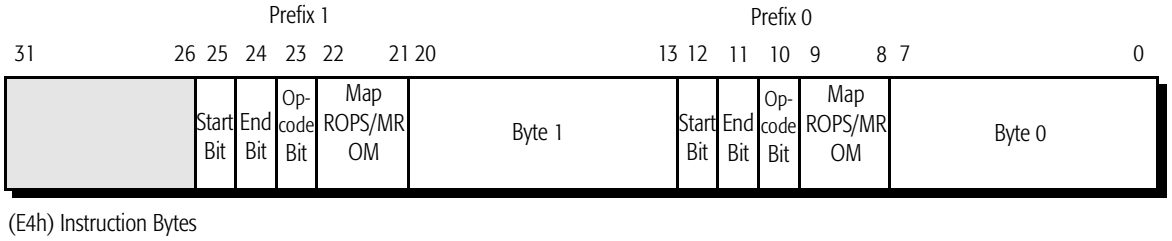
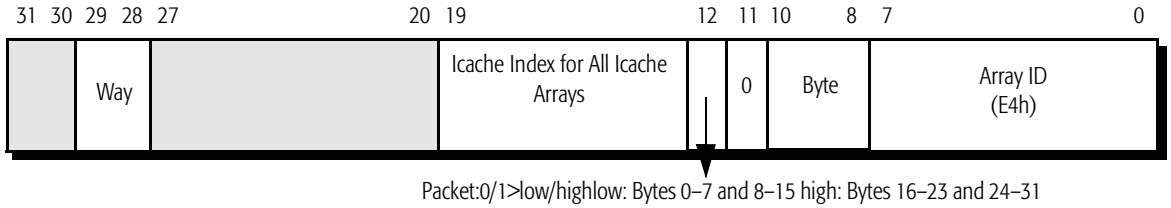


Figure 9. Test Formats: Icache Instructions for the AMD-K5™ Processor Model 0

EDX: Array Pointer



EAX: Test Data

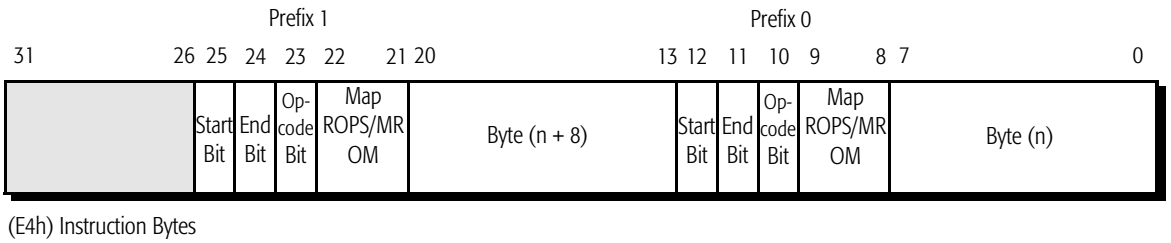
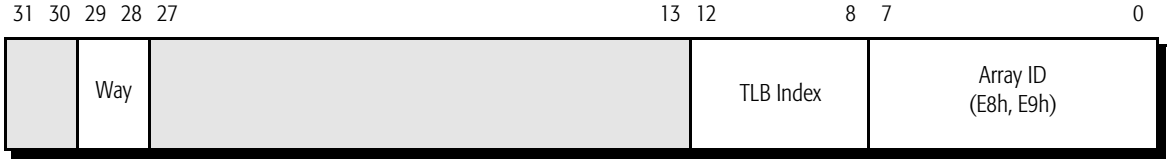
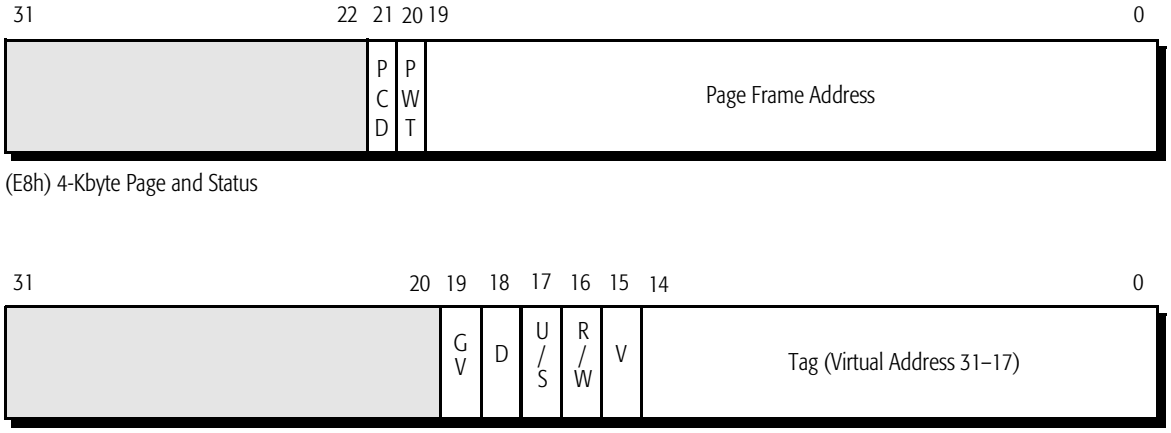


Figure 10. Test Formats: Icache Instructions for the AMD-K5™ Processor Model 1 and Greater

EDX: Array Pointer



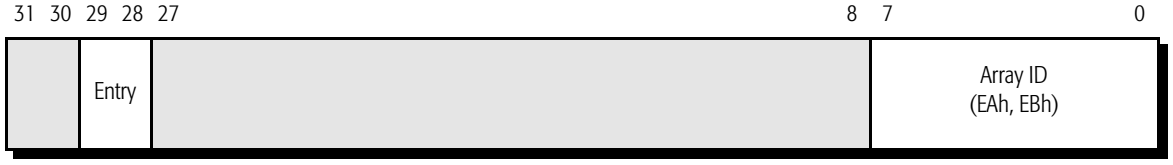
EAX: Test Data



Symbol	Description	Bits
GV	Global Valid Bit	19
D	Dirty Bit	18
U/S	User Supervisor Bit	17
R/W	Read or Write Bit	16
V	Valid Bit	15

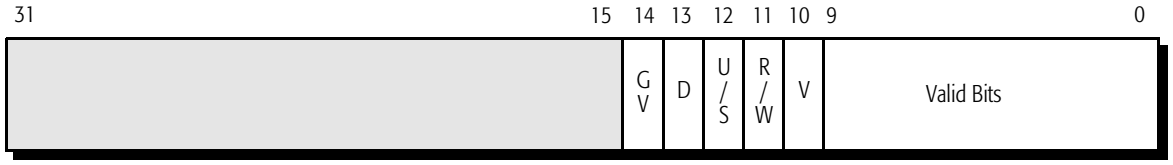
Figure 11. Test Formats: 4-Kbyte TLB for All Models of the AMD-K5™ Processor

EDX: Array Pointer



EAX: Test Data

(EAh) 4-Mbyte Page and Status



(EBh) 4-Mbyte Virtual Tag

Symbol	Description	Bits
GV	Global Valid Bit	14
D	Dirty Bit	13
U/S	User Supervisor Bit	12
R/W	Read or Write Bit	11
V	Valid Bit	10

Figure 12. Test Formats: 4-Mbyte TLB for All Models of the AMD-K5™ Processor

Debug Registers

The processor implements the standard debug functions and registers—DR7–DR6 and DR3–DR0 (often called DR7–DR0)—available on the 486 processor, plus an I/O breakpoint extension.

Standard Debug Functions

The debug functions make the processor's state visible to debug software through four debug registers (DR3–DR0) that are accessed by MOV instructions. Accesses to memory addresses can be set as breakpoints in the instruction flow by invoking one of two debug exceptions (interrupt vectors 1 or 3) during instruction or data accesses to the addresses. The debug functions eliminate the need to embed breakpoints in code and allow debugging of ROM as well as RAM.

For details on the standard 486 debug functions and registers, see the AMD documentation on the Am486® processor or other commercial x86 literature.

I/O Breakpoint Extension

The processor supports an I/O breakpoint extension for breakpoints on I/O reads and writes. This function is enabled by setting bit 3 of CR4, as described in "Control Register 4 (CR4) Extensions" on page 58. When enabled, the I/O breakpoint function is invoked by the following:

- Entering the I/O port number as a breakpoint address (zero-extended to 32 bits) in one of the breakpoint registers, DR3–DR0
- Entering the bit pattern, 10b, in the corresponding 2-bit read-write (R/W) field in DR7

All data breakpoints on the AMD-K5 processor are precise, including those encountered in repeated string operations. The trap is taken after completing the iteration on which the breakpoint match occurs.

Enabled breakpoints slow the processor somewhat. When a data breakpoint is enabled, the processor disables its dual-issue load/store operations and performs only single-issue load/store operations. When an instruction breakpoint is enabled, instruction issue is completely serialized.

Debug Compatibility with the Pentium® Processor

The differences in debug functions between the AMD-K5 processor and Pentium are described in Appendix A of the *AMD-K5™ Processor Technical Reference Manual*, order# 18524.

Branch Tracing

Branch tracing is enabled by writing bits 3–1 with 001b and setting bit 5 to 1 (disabling branch prediction) in the Hardware Configuration Register (HWCR), as described on page 22. When thus enabled, the processor drives two branch-trace message special bus cycles immediately after each taken branch instruction is executed. Both special bus cycles have a BE7–BE0 encoding of DFh (1101_1111b). The first special bus cycle identifies the branch source, the second identifies the branch target. The contents of the address and data bus during these special bus cycles are shown in Table 14.

The branch-trace message special bus cycles are different for the AMD-K5 processor and Pentium, although their BE7–BE0 encodings are the same.

Table 14. Branch-Trace Message Special Bus Cycle Fields

Signals	First Special Bus Cycle	Second Special Bus Cycle
A31	0 = First special bus cycle (source)	1 = Second special bus cycle (target)
A30–A29	Not valid	Operating Mode of Target: 11 = Virtual-8086 Mode 10 = Protected Mode 01 = Not valid 00 = Real Mode
A28	Not valid	Default Operand Size of Target Segment: 1 = 32-bit 0 = 16-bit
A27–A20	0	0
A19–A4	Code Segment (CS) selector of Branch Source	Code Segment (CS) Selector of Branch Target
A3	0	0
D31–D0	EIP of Branch Source	EIP of Branch Target

Functional-Redundancy Checking

When FRCMC is asserted at RESET, the processor enters Functional-Redundancy Checking mode as the checker and reports checking errors on the IERR output. If FRCMC is negated at RESET, the processor operates normally, although it also behaves as the master in a functional-redundancy checking arrangement with a checker.

In the Functional-Redundancy Checking mode, two processors have their signals tied together. One processor (the master) operates normally. The other processor (the checker) has its output and bidirectional signals (except for TDO and IERR) floated to detect the state of the master's signals. The master controls instruction fetching and the checker mimics its behavior by sampling the fetched instructions as they appear on the bus. Both processors execute the instructions in lock step. The checker compares the state of the master's output and bidirectional signals with the state that the checker itself would have driven for the same instruction stream.

Errors detected by the checker are reported on the IERR output of the checker. If a mismatch occurs on such a comparison, the checker asserts IERR for one clock, two clocks after the detection of the error. Both the master and the checker continue running the checking program after an error occurs. No action other than the assertion of IERR is taken by the processor. On the AMD-K5 processor, the IERR output is reserved solely for functional-redundancy checking. No other errors are reported on that output.

Functional-redundancy checking is typically implemented on single-processor, fault-monitoring systems (which have two processors). The master processor runs the operational programs and the checker processor is dedicated entirely to constant checking. In this arrangement, the accurate operation test consists solely of reporting one or more errors. The particular error type or the instruction causing an error is not reported. The arrangement works because the processor is entirely deterministic. Speculative prefetching, speculative execution, and cache replacement all occur in identical ways and at identical times on both processors if their signals are tied together so that they run the same program.

The Functional-Redundancy Checking mode can only be exited by the assertion of RESET. Functional-redundancy checking cannot be performed in the Hardware Debug Tool (HDT) mode. The assertion of FRCMC is not recognized while PRDY is asserted.

Boundary Scan Architecture Support

The AMD-K5 processor provides test features compatible with the Standard Test Access Port (TAP) and Boundary Scan Test Architecture as defined in the IEEE 1149.1-1990 JTAG Specification. The subsections in this topic include:

- Boundary Scan Test Functional Description
- Boundary Scan Architecture
- Registers
- The Test Access Port (TAP) Controller
- JTAG Register Organization
- JTAG Instructions

The external TAP interface consists of five pins:

- TCK: The Test Clock input provides the clock for the JTAG test logic.
- TMS: The Test Mode Select input enables TAP controller operations.
- TDI: The Test Data Input provides serial input to registers.
- TDO: The Test Data Output provides serial output from the registers; the signal is tri-stated except when in the Shift-DR or Shift-IR controller states.
- TRST: The TAP Controller Reset input initializes the TAP controller when asserted Low.

The internal JTAG logic contains the elements listed below:

- The Test Access Port (TAP) Controller—Decodes the inputs on the Test Mode Select (TMS) line to control test operations. The TAP is a general-purpose port that provides access to the test support functions built into the AMD-K5.
- Instruction Register—Accepts instructions from the Test Data Input (TDI) pin. The instruction codes select the specific test or debug operation to be performed or the test data register to be accessed.

- Implemented Test Data Registers—Boundary Scan Register, Device Identification Register, and Bypass Register. See "JTAG Register Organization" on page 44 for more information.

  Note: *See Table 18 on page 49 for more information.*

Boundary Scan Test Functional Description

The boundary scan testing uses a shift register, contained in a boundary scan cell, located between the core logic and the I/O buffers adjacent to each component pin. Signals at each input and output pin are controlled and observed using scan testing techniques. The boundary scan cells are interconnected to form a shift register chain. This register chain, called a Boundary Scan Register (BSR), constructs a serial path surrounding the core logic, enabling test data to be shifted through the boundary scan path. When the system enters the Boundary Scan Test mode, the BSR chain is directed by a test program to pass data along the shift register path.

If all the components used to construct a circuit or PCB contain a boundary scan cell architecture, the resulting serial path can be used to perform component interconnect testing.

Boundary Scan Architecture

Boundary Scan architecture has four basic elements:

- Test Access Port (TAP)
- TAP Controller
- Instruction Register (IR). See "Instruction Register" on page 44 for more information.
- Test Data Registers. See "Registers" on page 43 for more information.

The Instruction and Test Data Registers have separate shift register access paths connected in parallel between the Test Data In (TDI) and Test Data Out (TDO) pins. Path selection and boundary scan cell operation is controlled by the TAP Controller. The controller initializes at start-up, but the Test Reset (TRST) input can asynchronously reset the test logic, if required.

All system integrated circuit (IC) I/O signals are shifted in and out through the serial Test Data In (TDI) and Test Data Out (TDO) path. The TAP Controller is enabled by the Test Mode Select (TMS) input. The Test Clock (TCK), obtained from a system level bus or Automatic Test Equipment (ATE), supplies

the timing signal for data transfer and system architecture operation.

The dedicated TCK input enables the serial test data path between components to be used independently of component-specific system clocks. TCK also ensures that test data can be moved to or from a chip without changing the state of the on-chip system logic.

The TCK signal is driven by an independent 50% duty cycle clock (generated by the Automatic Test Equipment). If the TCK must be stopped (for example, if the ATE must retrieve data from external memory and is unable to keep the clock running), it can be stopped at 0 or 1 indefinitely, without causing any change to the test logic state.

To ensure race-free operation, changes on the TAP's TMS input are clocked into the test logic. Changes on the TAP's TDI input are clocked into the selected register (Instruction or Test Data Register) on the rising edge of TCK. The contents of the selected register are shifted out onto the TAP output (TDO) on the falling edge of TCK.

Registers

Boundary scan architectural elements include an Instruction Register (IR) and a group of Test Data Registers (TDRs). These registers have separate shift-register-based serial access paths connected in parallel between the TDI and TDO pins.

The TDRs are internal registers used by the Boundary Scan Architecture to process the test data. Each Test Data Register is addressed by an instruction scanned into the Instruction Register. The AMD-K5 processor includes the following TDRs:

- Bypass Register (BR). See "Bypass Register" on page 45.
- Boundary Scan Register (BSR). See "Boundary Scan Register" on page 44.
- Device Identification Register (DIR). See "Device Identification Register" on page 45.
- Built-In Self-Test Result Register (BISTRR). See "RUNBIST" on page 48.

Instruction Register. The 5-bit Instruction Register (IR) is a serial-in parallel-out register that includes five shift register-based cells for holding instruction data. The instruction determines which test to run, which data register to access, or both. When the TAP controller enters the Capture-IR state, the processor loads the IDCODE instruction in the IR. Executing Shift-IR starts instructions shifting into the instruction register on the rising edge of TCK. Executing Update-IR loads the instruction from the serial shift register to the parallel register.

The TAP controller is a synchronous, finite-state machine that controls the test and debug logic sequence of operations. The TAP controller changes state in response to the rising edge of TCK and defaults to the test logic reset state at power-up. Reinitialization to the test logic reset state is accomplished by holding the TMS pin High for five TCK periods.

JTAG Register Organization

All registers in the JTAG logic consist of the following two register ranks:

- Shift register
- Parallel output register fed by the shift register

Parallel input data is loaded into the shift register when the TAP controller exits the Capture state (Capture-DR or Capture-IR). The shift register then shifts data from TDI to TDO when in the Shift state (Shift-DR or Shift-IR). The output register holds the current data while new data is shifted into the shift register. The contents of the output register are updated when the TAP controller exits the Update state (Update-DR or Update-IR). The following three registers are described in this section:

- Boundary Scan Register
- Device Identification Register
- Bypass Register

Boundary Scan Register. The Boundary Scan Register (BSR) is a 261-bit shift register with cells connected to all input and output pins and containing cells for tri-state I/O control. This arrangement enables serial data to be loaded into or read from the processor boundary scan area.

Output cells determine the value of the signal driven on the corresponding pin. Input cells only capture data. The EXTEST and SAMPLE/PRELOAD instructions can operate the BSR.

Device Identification Register. The format of the Device Identification Register (DIR) is shown in Table 15. The fields include the following values:

- *Version Number*—This field is incremented by AMD manufacturing for each major revision of silicon.
- *Bond Option*—The two bits of the bond option depend on how the part is bonded at the factory.
- *Part Number*—This field identifies the specific processor model.
- *Manufacturer*—This field is actually only 11 bits (11–1). The least-significant bit, bit 0, is always set to 1, as specified by the IEEE standard.

Table 15. AMD-K5™ Processor Device Identification Register

Version (Bits 31–28)	Bond Option (Bits 27–26)	Part Number (Bits 25–12)	Manufacturer (Bits 11–1)	LSB (Bit 0)
0h	X0b	051Xh	00000000001b	1b

Bypass Register. The Bypass Register, a 1-bit shift register, provides the shortest path between TDI and TDO. When the component is not performing a test operation, this path is selected to allow transfer of test data to and from other components on the board. The Bypass Register is also selected during the HIGHZ, ALL1, ALL0, and BYPASS tests and for any unused instruction codes.

Public Instructions

The processor supports all three IEEE-mandatory instructions (BYPASS, SAMPLE/PRELOAD, EXTEST), three IEEE-optional instructions (IDCODE, HIGHZ, RUNBIST), and three instructions unique to the AMD-K5 processor (ALL1, ALL0, USEHDT). Table 16 shows the complete set of public TAP instructions supported by the processor. The AMD-K5 also implements several private manufacturing test instructions.

The IEEE standard describes the mandatory and optional instructions. The ALL1 and ALL0 instructions simply force all outputs and bidirectionals High or Low. The USEHDT instruction is described on page 57. Any instruction encodings not shown in Table 16 select the BYPASS instruction.

Table 16. Public TAP Instructions

Instruction	Encoding	Register	Description
EXTEST	00000	BSR	As defined by the IEEE standard
SAMPLE/PRELOAD	00001	BSR	As defined by the IEEE standard
IDCODE	00010	DIR	As defined by the IEEE standard
HIGHZ	00011	BR	As defined by the IEEE standard
ALL1	00100	BR	Forces all outputs and bidirectionals High
ALL0	00101	BR	Forces all outputs and bidirectionals Low
USEHDT	00110	HDTR	Accesses the Hardware Debug Tool (HDT) See page 57
RUNBIST	00111	BISTRR	As defined by the IEEE standard
BYPASS	11111	BR	As defined by the IEEE standard
BYPASS	undefined	BR	Undefined instruction encodings select the BYPASS instruction

EXTEST. The EXTEST instruction permits circuits outside the component package to be tested. A common use of the EXTEST instruction is the testing of board interconnects. Boundary scan register cells at output pins are used to apply test stimuli, while those at input pins capture test results. Depending on the value loaded into their control cells in the boundary scan register, the I/O pins are established as input or output. Inputs to the core logic retain the logic value set prior to execution of the EXTEST instruction. Upon exiting EXTEST, input pins are reconnected to the package pins.

SAMPLE/PRELOAD. There are two functions performed by the SAMPLE/PRELOAD instruction, as follows:

- Capturing an instantaneous picture of the normal operation of the device being tested. This function occurs if the instruction is executed while the TAP controller is in the Capture-DR state and causes the Boundary Scan Register to sample the values present at the device pins.
- Preloading data to the device pins to be driven to the board by the EXTEST instruction. This function occurs if the instruction is executed while the TAP controller is in the Update-DR state and causes data to be preloaded to the device pins from the Boundary Scan Register.

IDCODE. The execution of the IDCODE instruction connects the device identification register between TDI and TDO. Upon such connection, the device identification code can be shifted out of the register.

HIGHZ. This instruction forces all output and bidirectional pins into a tri-state condition. When this instruction is selected, the bypass register is selected for shifting between TDI and TDO. A signal called HIZEXT is responsible for forcing the tri-state to occur. This signal is generated in the TAP block, underneath JTAG_BIST, and goes to the PAD_TOP block.

ALL1. This instruction forces all output and bidirectional pins to a High logic level.

The ALL1 instruction, like the HIGHZ instruction, selects the bypass register for shifting between TDI and TDO. A signal called ALL1 is responsible for forcing the pins to a High state. This signal is generated in the TAP block underneath JTAG_BIST and goes to the PAD_TOP block. In the PAD_TOP block, this signal goes to boundary scan cells called BSLCD_OUT. The DOUT pins of the BSLCD_OUT cells are forced High when ALL1 is High. The SELPDR signal selects the boundary scan cells as the source for driving the outputs if the SELPDR signal is High. The SELPDR signal is also generated in the TAP block underneath JTAG_BIST and goes to the PAD_TOP block.

ALL0. This instruction forces all output and bidirectional pins to a Low logic level.

The ALL0 instruction, like the HIGHZ instruction, selects the bypass register for shifting between TDI and TDO. A signal called ALL0 is responsible for forcing the pins to a Low state. This signal is generated in the TAP block underneath JTAG_BIST and goes to the PAD_TOP block. In the PAD_TOP block, this signal goes to boundary scan cells called BSLCD_OUT. The DOUT pins of the BSLCD_OUT cells are forced Low when ALL0 is High. The SELPDR signal selects the boundary scan cells as the source for driving the outputs if the SELPDR signal is High. The SELPDR signal is also generated in the TAP block underneath JTAG_BIST and goes to the PAD_TOP block.

RUNBIST. This version of BIST is similar to the normal BIST mode, except RUNBIST is started by shifting in a TAP instruction. This instruction should behave according to the rules of the IEEE 1149.1 definition of RUNBIST.

When the RUNBIST instruction is updated into the instruction register, a signal from the TAP_RTL block called JTGBIST is asserted High. This signal goes to the PAD_TOP and TESTCTRL blocks. In PAD_TOP, this signal goes to the BRNBIST block and causes both INIT_SAMP and RUNBIST to be asserted. To the rest of the processor, it looks like a normal BIST operation is taking place. The JTGBIST signal also goes to the TESTCTRL block so the BIST controller knows the BIST operation was initiated from the TAP controller. This operation is necessary because the BIST results do not get transferred to the EAX register in this mode of operation. The JTAG_BIST block also asserts the RESET_TAP pin to the CLOCKS block for 15 system clock cycles in order to fake an external reset.

The pattern that is shifted into the boundary scan ring prior to the selection of the RUNBIST instruction is driven at output and bidirectional cells during the duration of the instruction. The results of the execution of RUNBIST are saved in the BIST results register, which is 9 bits long and looks like the least significant 9 bits in the EAX register. This register is selected for shifting between TDI and TDO and can be shifted out after the completion of BIST. Bit 0 (ICACHE data status) is shifted out first. The BIST results should be independent of signals received at non-clock input pins (except for RESET).

BYPASS. The execution of the BYPASS instruction connects the bypass register between TDI and TDO, bypassing the test logic. Because of the pull-up resistor on the TDI input, the bypass register is selected if there is an open circuit in the board-level test data path following an instruction scan cycle. Any unused instruction bit patterns cause the bypass register to be selected for shifting between TDI and TDO.

The control bits listed in Table 18 have the characteristics described in Table 17.

Table 17. Control Bit Definitions

Bit	Definition
144	Controls the direction of the Data bus (D63–D0). If the bit is set to 1, the bus acts as an input. If the bit is set to 0, the bus acts as an output.
213	Controls the direction of the Address bus (A31–A3) and Address Parity (AP). If the bit is set to 1, the bus acts as an input. If the bit is set to 0, the bus acts as an output.
257	Controls pins that can be tri-stated, but these pins never act as inputs. If the bit is set to 1, the pin is tri-stated. If the bit is set to 0, the pin acts as an output.

Table 18. Boundary Scan Register Bit Definitions

Bit	Pin Name	Comments
0	DP7	Output Cell: Controlled by bit 144
1	DP7	Input Cell
2	D63	Output Cell: Controlled by bit 144
3	D63	Input Cell
4	D62	Output Cell: Controlled by bit 144
5	D62	Input Cell
6	D61	Output Cell: Controlled by bit 144
7	D61	Input Cell
8	D60	Output Cell: Controlled by bit 144
9	D60	Input Cell
10	D59	Output Cell: Controlled by bit 144
11	D59	Input Cell
12	D58	Output Cell: Controlled by bit 144
13	D58	Input Cell
14	D57	Output Cell: Controlled by bit 144
15	D57	Input Cell
16	D56	Output Cell: Controlled by bit 144
17	D56	Input Cell
18	DP6	Output Cell: Controlled by bit 144
19	DP6	Input Cell

Table 18. Boundary Scan Register Bit Definitions (continued)

Bit	Pin Name	Comments
20	D55	Output Cell: Controlled by bit 144
21	D55	Input Cell
22	D54	Output Cell: Controlled by bit 144
23	D54	Input Cell
24	D53	Output Cell: Controlled by bit 144
25	D53	Input Cell
26	D52	Output Cell: Controlled by bit 144
27	D52	Input Cell
28	D51	Output Cell: Controlled by bit 144
29	D51	Input Cell
30	D50	Output Cell: Controlled by bit 144
31	D50	Input Cell
32	D49	Output Cell: Controlled by bit 144
33	D49	Input Cell
34	D48	Output Cell: Controlled by bit 144
35	D48	Input Cell
36	DP5	Output Cell: Controlled by bit 144
37	DP5	Input Cell
38	D47	Output Cell: Controlled by bit 144
39	D47	Input Cell
40	D46	Output Cell: Controlled by bit 144
41	D46	Input Cell
42	D45	Output Cell: Controlled by bit 144
43	D45	Input Cell
44	D44	Output Cell: Controlled by bit 144
45	D44	Input Cell
46	D43	Output Cell: Controlled by bit 144
47	D43	Input Cell
48	D42	Output Cell: Controlled by bit 144
49	D42	Input Cell
50	D41	Output Cell: Controlled by bit 144
51	D41	Input Cell
52	D40	Output Cell: Controlled by bit 144
53	D40	Input Cell

Table 18. Boundary Scan Register Bit Definitions (continued)

Bit	Pin Name	Comments
54	DP4	Output Cell: Controlled by bit 144
55	DP4	Input Cell
56	D39	Output Cell: Controlled by bit 144
57	D39	Input Cell
58	D38	Output Cell: Controlled by bit 144
59	D38	Input Cell
60	D37	Output Cell: Controlled by bit 144
61	D37	Input Cell
62	D36	Output Cell: Controlled by bit 144
63	D36	Input Cell
64	D35	Output Cell: Controlled by bit 144
65	D35	Input Cell
66	D34	Output Cell: Controlled by bit 144
67	D34	Input Cell
68	D33	Output Cell: Controlled by bit 144
69	D33	Input Cell
70	D32	Output Cell: Controlled by bit 144
71	D32	Input Cell
72	DP3	Output Cell: Controlled by bit 144
73	DP3	Input Cell
74	D31	Output Cell: Controlled by bit 144
75	D31	Input Cell
76	D30	Output Cell: Controlled by bit 144
77	D30	Input Cell
78	D29	Output Cell: Controlled by bit 144
79	D29	Input Cell
80	D28	Output Cell: Controlled by bit 144
81	D28	Input Cell
82	D27	Output Cell: Controlled by bit 144
83	D27	Input Cell
84	D26	Output Cell: Controlled by bit 144
85	D26	Input Cell
86	D25	Output Cell: Controlled by bit 144
87	D25	Input Cell

Table 18. Boundary Scan Register Bit Definitions (continued)

Bit	Pin Name	Comments
88	D24	Output Cell: Controlled by bit 144
89	D24	Input Cell
90	DP2	Output Cell: Controlled by bit 144
91	DP2	Input Cell
92	D23	Output Cell: Controlled by bit 144
93	D23	Input Cell
94	D22	Output Cell: Controlled by bit 144
95	D22	Input Cell
96	D21	Output Cell: Controlled by bit 144
97	D21	Input Cell
98	D20	Output Cell: Controlled by bit 144
99	D20	Input Cell
100	D19	Output Cell: Controlled by bit 144
101	D19	Input Cell
102	D18	Output Cell: Controlled by bit 144
103	D18	Input Cell
104	D17	Output Cell: Controlled by bit 144
105	D17	Input Cell
106	D16	Output Cell: Controlled by bit 144
107	D16	Input Cell
108	DP1	Output Cell: Controlled by bit 144
109	DP1	Input Cell
110	D15	Output Cell: Controlled by bit 144
111	D15	Input Cell
112	D14	Output Cell: Controlled by bit 144
113	D14	Input Cell
114	D13	Output Cell: Controlled by bit 144
115	D13	Input Cell
116	D12	Output Cell: Controlled by bit 144
117	D12	Input Cell
118	D11	Output Cell: Controlled by bit 144
119	D11	Input Cell
120	D10	Output Cell: Controlled by bit 144
121	D10	Input Cell

Table 18. Boundary Scan Register Bit Definitions (continued)

Bit	Pin Name	Comments
122	D9	Output Cell: Controlled by bit 144
123	D9	Input Cell
124	D8	Output Cell: Controlled by bit 144
125	D8	Input Cell
126	DP	Output Cell: Controlled by bit 144
127	DP	Input Cell
128	D7	Output Cell: Controlled by bit 144
129	D7	Input Cell
130	D6	Output Cell: Controlled by bit 144
131	D6	Input Cell
132	D5	Output Cell: Controlled by bit 144
133	D5	Input Cell
134	D4	Output Cell: Controlled by bit 144
135	D4	Input Cell
136	D3	Output Cell: Controlled by bit 144
137	D3	Input Cell
138	D2	Output Cell: Controlled by bit 144
139	D2	Input Cell
140	D1	Output Cell: Controlled by bit 144
141	D1	Input Cell
142	D0	Output Cell: Controlled by bit 144
143	D0	Input Cell
144	Control	Direction Control. See Table 17.
145	STPLK#	Input Cell
146	FRCMC#	Input Cell
147	PEN#	Input Cell
148	IGNNE#	Input Cell
149	BF	Input Cell
150	INIT	Input Cell
151	SMI#	Input Cell
152	R/S#	Input Cell
153	NMI	Input Cell
154	INTR	Input Cell
155	A21	Output Cell: Controlled by bit 213

Table 18. Boundary Scan Register Bit Definitions (continued)

Bit	Pin Name	Comments
156	A21	Input Cell
157	A22	Output Cell: Controlled by bit 213
158	A22	Input Cell
159	A23	Output Cell: Controlled by bit 213
160	A23	Input Cell
161	A24	Output Cell: Controlled by bit 213
162	A24	Input Cell
163	A25	Output Cell: Controlled by bit 213
164	A25	Input Cell
165	A26	Output Cell: Controlled by bit 213
166	A26	Input Cell
167	A27	Output Cell: Controlled by bit 213
168	A27	Input Cell
169	A28	Output Cell: Controlled by bit 213
170	A28	Input Cell
171	A29	Output Cell: Controlled by bit 213
172	A29	Input Cell
173	A30	Output Cell: Controlled by bit 213
174	A30	Input Cell
175	A31	Output Cell: Controlled by bit 213
176	A31	Input Cell
177	A3	Output Cell: Controlled by bit 213
178	A3	Input Cell
179	A4	Output Cell: Controlled by bit 213
180	A4	Input Cell
181	A5	Output Cell: Controlled by bit 213
182	A5	Input Cell
183	A6	Output Cell: Controlled by bit 213
184	A6	Input Cell
185	A7	Output Cell: Controlled by bit 213
186	A7	Input Cell
187	A8	Output Cell: Controlled by bit 213
188	A8	Input Cell
189	A9	Output Cell: Controlled by bit 213

Table 18. Boundary Scan Register Bit Definitions (continued)

Bit	Pin Name	Comments
190	A9	Input Cell
191	A10	Output Cell: Controlled by bit 213
192	A10	Input Cell
193	A11	Output Cell: Controlled by bit 213
194	A11	Input Cell
195	A12	Output Cell: Controlled by bit 213
196	A12	Input Cell
197	A13	Output Cell: Controlled by bit 213
198	A13	Input Cell
199	A14	Output Cell: Controlled by bit 213
200	A14	Input Cell
201	A15	Output Cell: Controlled by bit 213
202	A15	Input Cell
203	A16	Output Cell: Controlled by bit 213
204	A16	Input Cell
205	A17	Output Cell: Controlled by bit 213
206	A17	Input Cell
207	A18	Output Cell: Controlled by bit 213
208	A18	Input Cell
209	A19	Output Cell: Controlled by bit 213
210	A19	Input Cell
211	A20	Output Cell: Controlled by bit 213
212	A20	Input Cell
213	Control	Direction Control. See Table 17.
214	SCYC	Output Cell: Controlled by bit 257
215	RESET	Input Cell
216	BE7#	Output Cell: Controlled by bit 257
217	BE6#	Output Cell: Controlled by bit 257
218	BE5#	Output Cell: Controlled by bit 257
219	BE4#	Output Cell: Controlled by bit 257
220	BE3#	Output Cell: Controlled by bit 257
221	BE2#	Output Cell: Controlled by bit 257
222	BE1#	Output Cell: Controlled by bit 257
223	BE0#	Output Cell: Controlled by bit 257

Table 18. Boundary Scan Register Bit Definitions (continued)

Bit	Pin Name	Comments
224	W/R#	Output Cell: Controlled by bit 257
225	HIT#	Output Cell
226	CLK	Clock
227	ADSC#	Output Cell: Controlled by bit 257
228	ADS#	Output Cell: Controlled by bit 257
229	CACHE#	Output Cell: Controlled by bit 257
230	BRDYC#	Input Cell
231	BRDY#	Input Cell
232	EADS#	Input Cell
233	PWT	Output Cell: Controlled by bit 257
234	LOCK#	Output Cell: Controlled by bit 257
235	PCD	Output Cell: Controlled by bit 257
236	WB/WT#	Input Cell
237	HITM#	Output Cell
238	KEN#	Input Cell
239	AHOLD	Input Cell
240	BOFF#	Input Cell
241	HLDA	Output Cell
242	HOLD	Input Cell
243	NA#	Input Cell
244	EWBE#	Input Cell
245	M/IO#	Output Cell: Controlled by bit 257
246	FLUSH#	Input Cell
247	A20M#	Input Cell
248	BUSCHK#	Input Cell
249	AP	Output Cell: Controlled by bit 213
250	AP	Input Cell
251	D/C#	Output Cell: Controlled by bit 257
252	BREQ	Output Cell
253	SMIACT#	Output Cell
254	PCHK#	Output Cell
255	APCHK#	Output Cell
256	PRDY	Output Cell
257	Control	Direction Control. See Table 17.

Table 18. Boundary Scan Register Bit Definitions (continued)

Bit	Pin Name	Comments
258	INV	Input Cell
259	FERR#	Output Cell
260	IERR#	Output Cell

Hardware Debug Tool (HDT)

The Hardware Debug Tool (HDT)—sometimes referred to as the debug port or Probe Mode—is a collection of signals, registers, and processor microcode that is enabled when external debug logic drives R/S Low or loads the processor's Test Access Port (TAP) instruction register with the USEHDT instruction.

AMD-K5™ Processor x86 Architecture Extensions

The AMD-K5 processor is compatible with the instruction set, programming model, memory management mechanisms, and other software infrastructure supported by the 486 and Pentium (735\90, 815\100) processors. Operating system and application software that runs on Pentium can be executed on the AMD-K5. Because the AMD-K5 processor takes a significantly different approach to implementing the x86 architecture, some subtle differences from Pentium may be visible to system and code developers. These differences are described in Appendix A of the *AMD-K5™ Processor Technical Reference Manual,* order# 18524.

Call AMD at 1-800-222-9323 to order AMD-K5 support documents.

Before implementing the AMD-K5 processor model-specific features, check CPUID for supported feature flags. See "CPUID" on page 86 for more information.

Additions to the EFLAGS Register

The EFLAGS register on the AMD-K5 processor defines new bits in the upper 16 bits of the register to support extensions to the operating modes. See "Virtual-8086 Mode Extensions (VME)" on page 67 and "CPUID" on page 86 for additional information.

Control Register 4 (CR4) Extensions

Control Register 4 (CR4) was added on the AMD-K5. The bits in this register control the various architectural extensions. The majority of the bits are reserved. The default state of CR4 is all zeros. Figure 13 shows the register and describes the bits. The architectural extensions are described in Table 19.

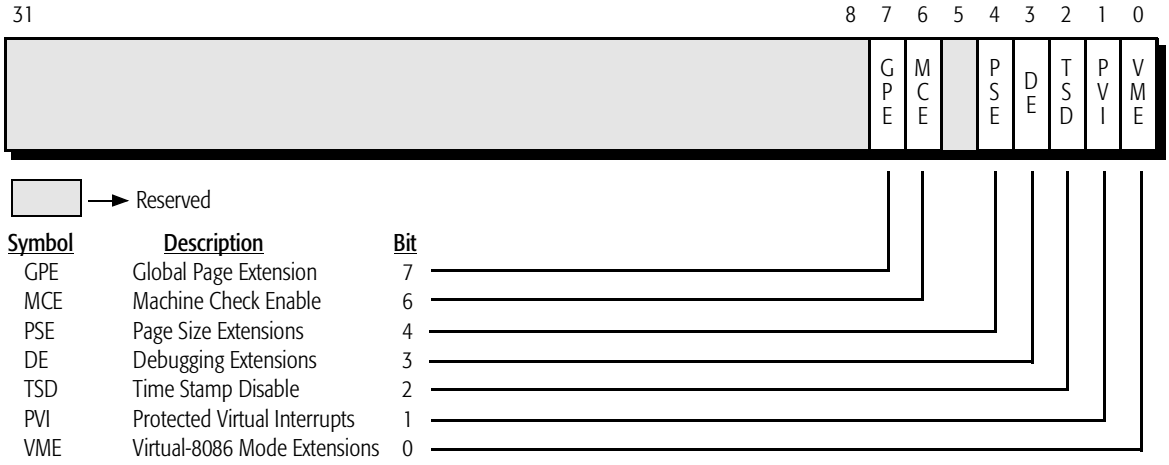


Figure 13. Control Register 4 (CR4)

Table 19. Control Register 4 (CR4) Fields

Bit	Mnemonic	Description	Function
7	GPE	Global Page Extension*	Enables retention of designated entries in the 4-Kbyte TLB or 4-Mbyte TLB during invalidations. 1 = enabled, 0 = disabled. See "Global Pages" on page 65 for details.
6	MCE	Machine-Check Enable	Enables machine-check exceptions. 1 = enabled, 0 = disabled. See "Machine-Check Exceptions" on page 60 for details.
4	PSE	Page Size Extension	Enables 4-Mbyte pages. 1 = enabled, 0 = disabled. See "4-Mbyte Pages" on page 60 for details.
3	DE	Debugging Extensions	Enables I/O breakpoints in the DR7–DR0 registers. 1 = enabled, 0 = disabled. See "Debug Registers" on page 38 for details.
2	TSD	Time Stamp Disable	Selects privileged (CPL=0) or non-privileged (CPL>0) use of the RDTSC instruction, which reads the Time Stamp Counter (TSC). 1 = CPL must be 0, 0 =any CPL. See "Time Stamp Counter (TSC)" on page 81 for details.
1	PVI	Protected Virtual Interrupts	Enables hardware support for interrupt virtualization in Protected mode. 1 = enabled, 0 = disabled. See "Protected Virtual Interrupt (PVI) Extensions" on page 79 for details.
0	VME	Virtual-8086 Mode Extensions	Enables hardware support for interrupt virtualization in Virtual-8086 mode. 1 = enabled, 0 = disabled. See "Virtual-8086 Mode Extensions (VME)" on page 67 for details.

Note:
* *The AMD-K5 processor supports global paging only on Models 1, 2, and 3, with a Stepping of 4 or greater.*

Machine-Check Exceptions

Bit 6 in CR4, the machine-check enable (MCE) bit, controls generation of machine-check exceptions (12h). If enabled by the MCE bit, these exceptions are generated when either of the following occurs:

- System logic asserts BUSCHK to identify a parity or other type of bus-cycle error
- The processor asserts PCHK while system logic asserts PEN to identify an enabled parity error on the D63–D0 data bus

Whether or not machine-check exceptions are enabled, the processor performs the following functions when either type of bus error occurs:

- Latches the physical address of the failed cycle in its 64-bit machine-check address register (MCAR)
- Latches the cycle definition of the failed cycle in its 64-bit machine-check type register (MCTR)

Software can read the MCAR and MCTR registers in the exception handling routine with the RDMSR instruction, as described on page 90. The format of the registers is shown in Figures 20 and 21.

If system software has cleared the MCE bit in CR4 to 0 before a bus-cycle error, the processor attempts to continue execution without generating a machine-check exception. The processor still latches the address and cycle type in MCAR and MCTR as described in this section.

4-Mbyte Pages

The TLBs in the 486 and 386 processors support only 4-Kbyte pages. However, large data structures, such as a video frame buffer or non-paged operating system code, can consume many pages and easily overrun the TLB. The AMD-K5 processor accommodates large data structures by allowing the operating system to specify 4-Mbyte pages as well as 4-Kbyte pages, and by implementing a four-entry, fully-associative 4-Mbyte TLB that is separate from the 128-entry, 4-Kbyte TLB. From a given page directory, the processor can access both 4-Kbyte pages and 4-Mbyte pages, and the page sizes can be intermixed within a page directory. When the Page Size Extension (PSE) bit in CR4 is set, the processor translates linear addresses using either the 4-Kbyte TLB or the 4-Mbyte TLB, depending on the state of the page size (PS) bit in the page-directory entry. Figures 14 and 15 show how 4-Kbyte and 4-Mbyte page translations work.

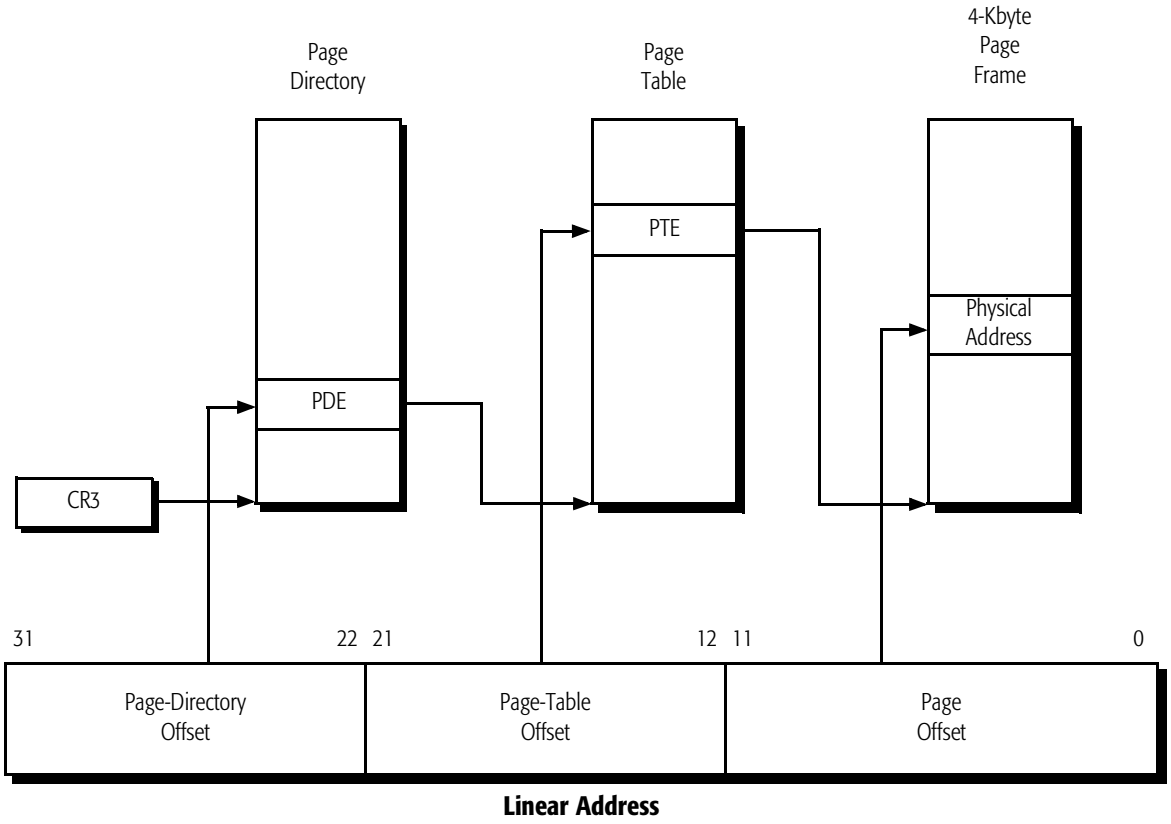


Figure 14. 4-Kbyte Paging Mechanism

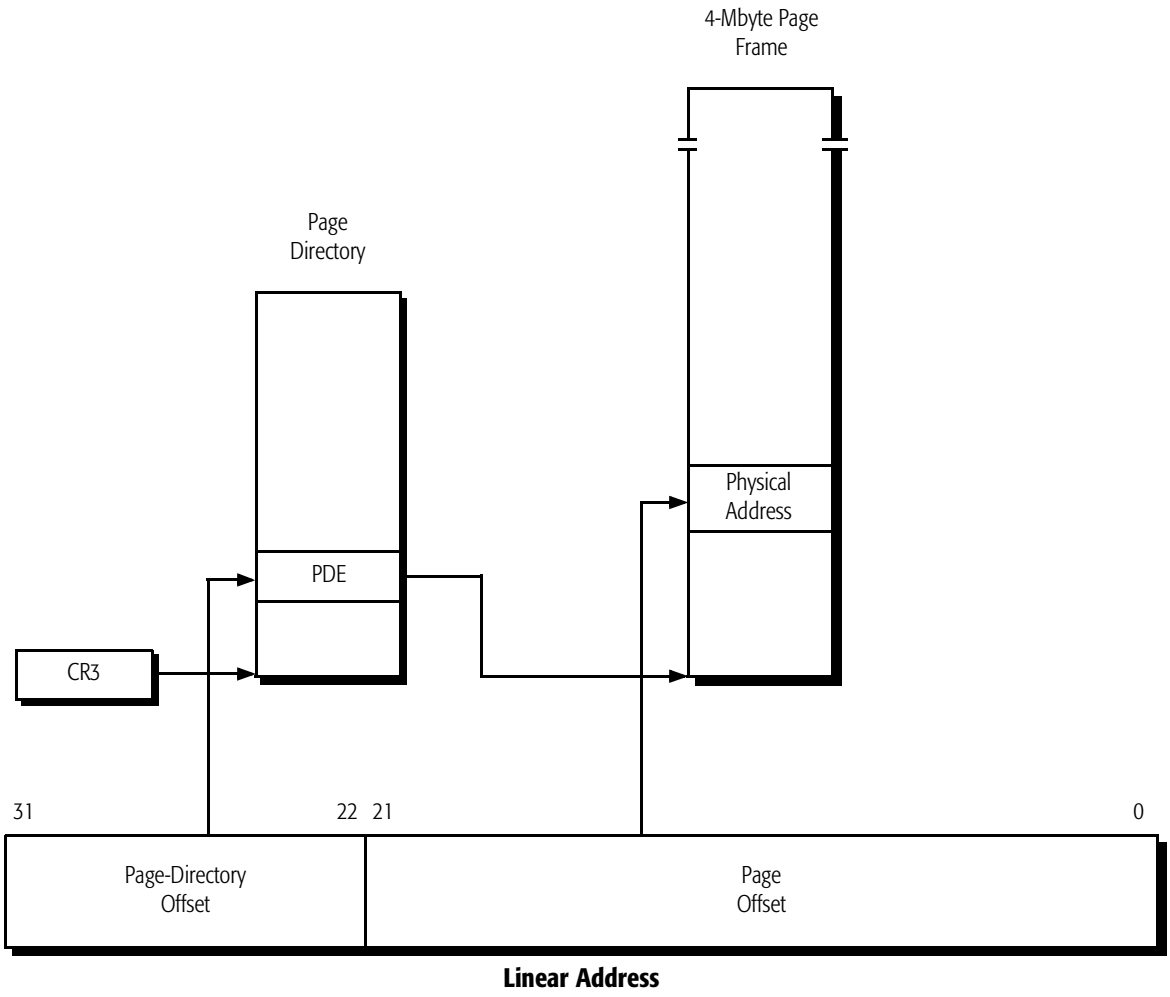


Figure 15. 4-Mbyte Paging Mechanism

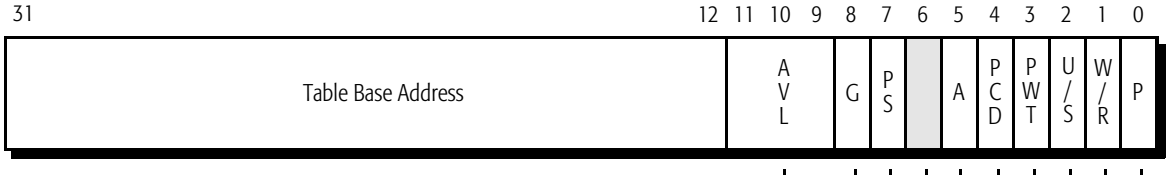
To enable the 4-Mbyte paging option:

1. Set the Page Size Extension (PSE) bit in CR4 to 1.
2. Set the Page Size (PS) bit in the page-directory entry to 1.
3. Write the physical base addresses of 4-Mbyte pages in bits 31–22 of page-directory entries. (Bits 21–12 of these entries must be cleared to 0 or the processor generates a page fault.)
4. Load CR3 with the base address of the page directory that contains these page-directory entries.

Figure 13 and Table 19 show the fields in CR4. Figure 16 and Table 20 show the fields in a page-directory entry.

4-Kbyte page translation differs from 4-Mbyte page translation in the following ways:

- *4-Kbyte Paging (Figure 14)*—Bits 31–22 of the linear address select an entry in a 4-Kbyte page directory in memory, whose physical base address is stored in CR3. Bits 21–12 of the linear address select an entry in a 4-Kbyte page table in memory, whose physical base address is specified by bits 31–22 of the page-directory entry. Bits 11–0 of the linear address select a byte in a 4-Kbyte page, whose physical base address is specified by the page-table entry.
- *4-Mbyte Paging (Figure 15)*—Bits 31–22 of the linear address select an entry in a 4-Mbyte page directory in memory, whose physical base address is stored in CR3. Bits 21–0 of the linear address select a byte in a 4-Mbyte page in memory, whose physical base address is specified by bits 31–22 of the page-directory entry. Bits 21–12 of the page-directory entry must be cleared to 0.



Symbol	Description	Bits
AVL	Available to Software	11–9
G	Global	8
PS	Page Size 0 = 4 Kbytes	7
	Reserved = 0	6
A	Accessed	5
PCD	Page Cache Disable	4
PWT	Page Writethrough	3
U/S	User/Supervisor	2
W/R	Write/Read	1
P	Present (valid)	0

Figure 16. Page-Directory Entry (PDE)

Table 20. Page-Directory Entry (PDE) Fields

Bit	Mnemonic	Description	Function
31–12	BASE	Physical Base Address	For 4-Kbyte pages, bits 31–12 contain the physical base address of a 4-Kbyte page table. For 4-Mbyte pages, bits 31–22 contain the physical base address of a 4-Mbyte page and bits 21–12 must be cleared to 0. (The processor generates a page fault if bits 21–12 are not cleared to 0.)
11–9	AVL	Available to Software	Software may use this field to store any type of information. When the page-directory entry is not present (P bit cleared), bits 31–1 become available to software.
8	G	Global*	0 = local, 1 = global.
7	PS	Page Size	0 = 4-Kbyte, 1 = 4-Mbyte.
6	D	Dirty	For 4-Kbyte pages, this bit is undefined and ignored. The processor does not change it. 0 = not written, 1 = written. For 4-Mbyte pages, the processor sets this bit to 1 during a write to the page that is mapped by this page-directory entry. 0 = not written, 1 = written.
5	A	Accessed	The processor sets this bit to 1 during a read or write to any page that is mapped by this page-directory entry. 0 = not read or written, 1 = read or written.
4	PCD	Page Cache Disable	Specifies cacheability for all pages mapped by this page-directory entry. Whether a location in a mapped page is actually cached also depends on several other factors. 0 = cacheable page, 1 = non-cacheable.
3	PWT	Page Writethrough	Specifies writeback or writethrough cache protocol for all pages mapped by this page-directory entry. Whether a location in a mapped page is actually cached in a writeback or writethrough state also depends on several other factors. 0 = writeback page, 1 = writethrough page.
2	U/S	User/Supervisor	0 = user (any CPL), 1 = supervisor (CPL < 3).
1	W/R	Write/Read	0 = read or execute, 1 = write, read, or execute.
0	P	Present	0 = not valid, 1 = valid.

Note:
* *The AMD-K5 processor supports global paging only on Models 1, 2, and 3, with a Stepping of 4 or greater.*

Global Pages

The AMD-K5 processor supports global paging only on Models 1, 2, and 3, with a Stepping of 4 or greater.

The processor's performance can sometimes be improved by making some pages *global* to all tasks and procedures. This can be done for both 4-Kbyte pages and 4-Mbyte pages.

The processor invalidates (flushes) both the 4-Kbyte TLB and the 4-Mbyte TLB whenever CR3 is loaded with the base address of the new task's page directory. The processor loads CR3 automatically during task switches, and the operating system can load CR3 at any other time. Unnecessary invalidation of certain TLB entries can be avoided by specifying those entries as *global* (a global TLB entry references a *global page)*. This improves performance after TLB flushes. Global entries remain in the TLB and need not be reloaded. For example, entries may reference operating system code and data pages that are always required. The processor operates faster if these entries are retained across task switches and procedure calls.

To specify individual pages as *global:*

1. Set the Global Page Extension (GPE) bit in CR4.
2. (Optional) Set the Page Size Extension (PSE) bit in CR4.
3. Set the relevant Global (G) bit for that page:

   For 4-Kbyte pages—Set the G bit in both the page-directory entry (shown in Figure 16 and Table 20) and the page-table entry (shown in Figure 17 and Table 21).

   For 4-Mbyte pages—(Optional) After the PSE bit in CR4 is set, set the G bit in the page-directory entry (shown in Figure 16 and Table 20).
4. Load CR3 with the base address of the page directory.

The INVLPG instruction clears both the V and G bits for the referenced entry. To invalidate all entries in both TLBs, including global-page entries:

1. Clear the Global Page Extension (GPE) bit in CR4.
2. Load CR3 with the base address of another (or same) page directory.

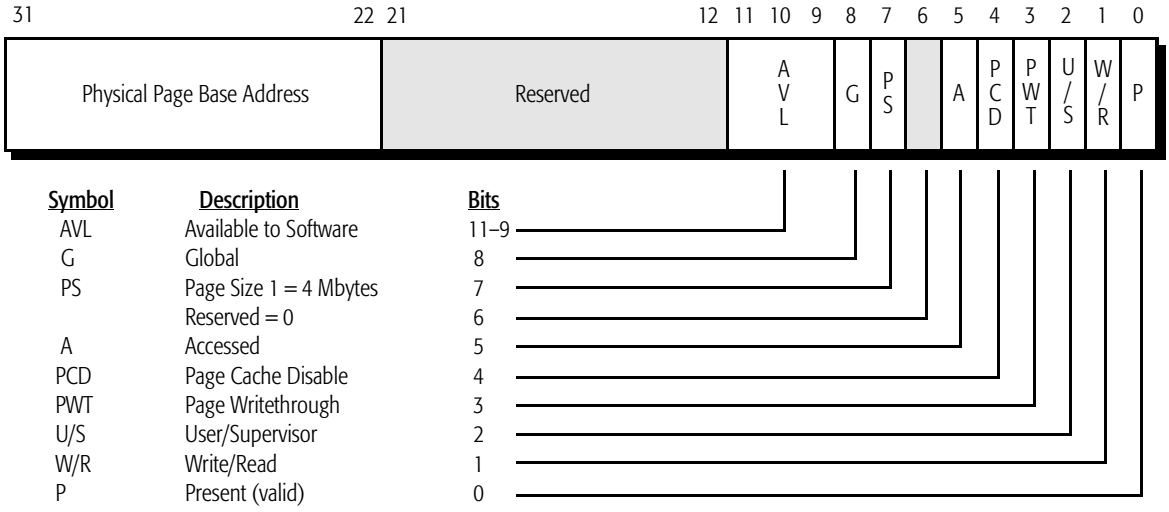


Figure 17. Page-Table Entry (PTE)

Table 21. Page-Table Entry (PTE) Fields

Bit	Mnemonic	Description	Function
31–12	BASE	Physical Base Address	The physical base address of a 4-Kbyte page.
11–9	AVL	Available to Software	Software may use the field to store any type of information. When the page-table entry is not present (P bit cleared), bits 31–1 become available to software.
8	G	Global*	0 = local, 1 = global.
7	PS	Page Size	This bit is ignored in page-table entries, although clearing it to 0 preserves consistent usage of this bit between page-table and page-directory entries.
6	D	Dirty	The processor sets this bit to 1 during a write to the page that is mapped by this page-table entry. 0 = not written, 1 = written.
5	A	Accessed	The processor sets this bit to 1 during a read or write to any page that is mapped by this page-table entry. 0 = not read or written, 1 = read or written.
4	PCD	Page Cache Disable	Specifies cacheability for all locations in the page mapped by this page-table entry. Whether a location is actually cached also depends on several other factors. 0 = cacheable page, 1 = non-cacheable.

Note:
* *The AMD-K5 processor supports global paging only on Models 1, 2, and 3, with a Stepping of 4 or greater.*

Table 21. Page-Table Entry (PTE) Fields (continued)

Bit	Mnemonic	Description	Function
3	PWT	Page Writethrough	Specifies writeback or writethrough cache protocol for all locations in the page mapped by this page-table entry. Whether a location is actually cached in a writeback or writethrough state also depends on several other factors. 0 = writeback, 1 = writethrough.
2	U/S	User/Supervisor	0 = user (any CPL), 1 = supervisor (CPL < 3).
1	W/R	Write/Read	0 = read or execute, 1 = write, read, or execute.
0	P	Present	0 = not valid, 1 = valid.

Note:
* *The AMD-K5 processor supports global paging only on Models 1, 2, and 3, with a Stepping of 4 or greater.*

Virtual-8086 Mode Extensions (VME)

The Virtual-8086 Mode Extensions (VME) bit in CR4 (bit 0) enable performance enhancements for 8086 programs running as protected tasks in Virtual-8086 mode. These extensions include:

- Virtualizing maskable external interrupt control and notification via the VIF and VIP bits in EFLAGS
- Selectively intercepting software interrupts (INT*n* instructions) via the Interrupt Redirection Bitmap (IRB) in the Task State Segment (TSS)

Interrupt Redirection in Virtual-8086 Mode Without VME Extensions. 8086 programs expect to have full access to the interrupt flag (IF) in the EFLAGS register, which enables maskable external interrupts via the INTR signal. When 8086 programs run in Virtual-8086 mode on a 386 or 486 processor, they run as protected tasks and access to the IF flag must be controlled by the operating system on a task-by-task basis to prevent corruption of system resources.

Without the VME extensions available on the AMD-K5 processor, the operating system controls Virtual-8086 mode access to the IF flag by trapping instructions that can read or write this flag. These instructions include STI, CLI, PUSHF, POPF, INT*n*, and IRET. This method prevents changes to the real IF when the I/O privilege level (IOPL) in EFLAGS is less than 3, the privilege level at which all Virtual-8086 tasks run. The operating system maintains an image of the IF flag for each Virtual-8086 program by emulating the instructions that read or write IF. When an external maskable interrupt occurs, the

operating system checks the state of the IF image for the current Virtual-8086 program to determine whether the program is allowing interrupts. If the program has disabled interrupts, the operating system saves the interrupt information until the program attempts to re-enable interrupts.

The overhead for trapping and emulating the instructions that enable and disable interrupts and the maintenance of virtual interrupt flags for each Virtual-8086 program can degrade the processor's performance. This performance can be regained by running Virtual-8086 programs with IOPL set to 3, thus allowing changes to the real IF flag from any privilege level, but with a loss in protection.

In addition to the performance overhead caused by virtualization of the IF flag in Virtual-8086 mode, software interrupts (those caused by INT*n* instructions that vector through interrupt gates) cannot be masked by the IF flag or virtual copies of the IF flag. These flags only affect hardware interrupts. Software interrupts in Virtual-8086 mode are normally directed to the Real mode interrupt vector table (IVT), but it may be desirable to redirect interrupts for certain vectors to the Protected mode interrupt descriptor table (IDT).

The processor's Virtual-8086 mode extensions support both of these cases—hardware (external) interrupts and software interrupts—with mechanisms that preserve high performance without compromising protection. Virtualization of hardware interrupts is supported via the Virtual Interrupt Flag (VIF) and Virtual Interrupt Pending (VIP) flag in the EFLAGS register. Redirection of software interrupts is supported with the Interrupt Redirection Bitmap (IRB) in the TSS of each Virtual-8086 program.

Hardware Interrupts and the VIF and VIP Extensions.
When VME extensions are enabled, the IF-modifying instructions that are normally trapped by the operating system are allowed to execute, but they write and read the VIF bit rather than the IF bit in EFLAGS. This operation leaves maskable interrupts enabled for detection by the operating system. It also indicates to the operating system whether the Virtual-8086 program is able to, or expecting to, receive interrupts.

When an external interrupt occurs, the processor switches from the Virtual-8086 program to the operating system, in the same manner as on a 386 or 486 processor. If the operating system determines the interrupt is for the Virtual-8086 program, it checks the state of the VIF bit in the program's EFLAGS image on the stack. If VIF has been set by the processor (during an attempt by the program to set the IF bit), the operating system permits access to the appropriate Virtual-8086 handler via the interrupt vector table (IVT). If VIF has been cleared, the operating system holds the interrupt pending. The operating system can do this by saving appropriate information (such as the interrupt vector), setting the program's VIP flag in the EFLAGS image on the stack, and returning to the interrupted program. When the program subsequently attempts to set IF, the set VIP flag causes the processor to inhibit the instruction and generate a general-protection exception with error code zero, thereby notifying the operating system that the program is now prepared to accept the interrupt.

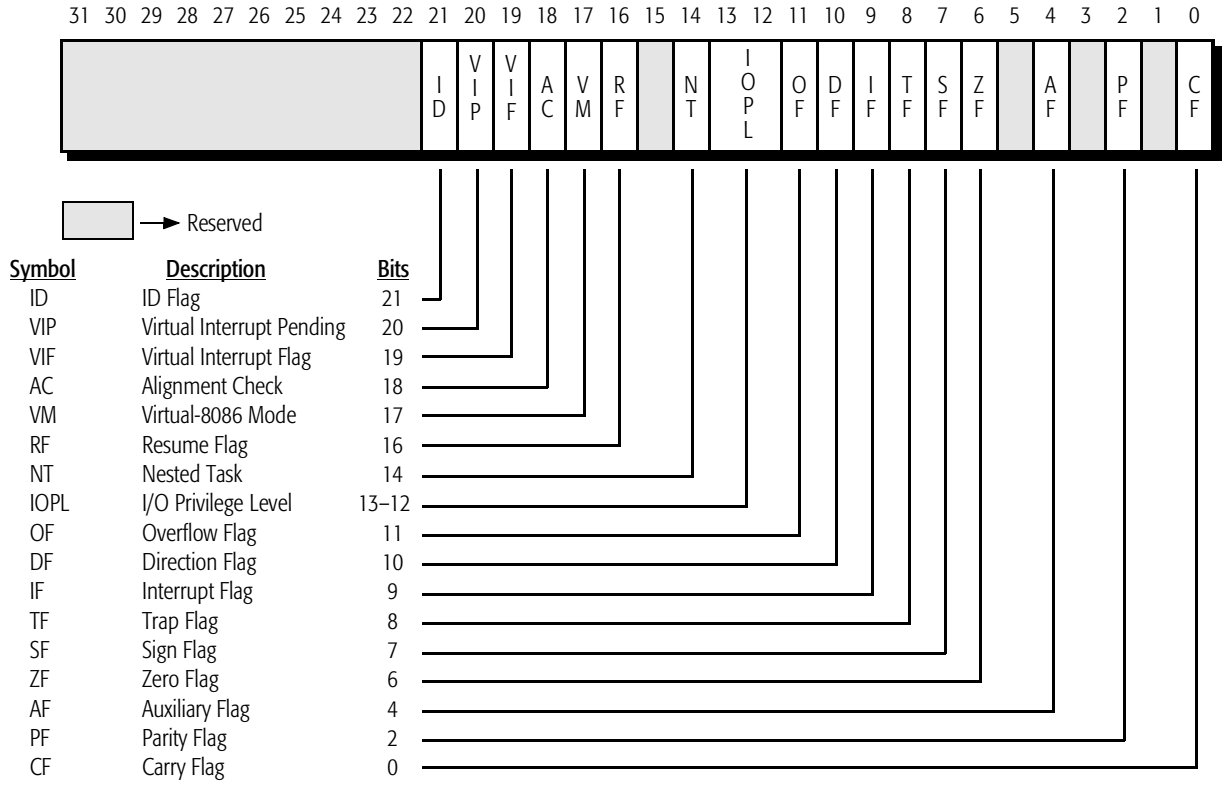
Thus, when VME extensions are enabled, the VIF and VIP bits are set and cleared as follows:

- *VIF*—This bit is controlled by the processor and used by the operating system to determine whether an external maskable interrupt should be passed on to the program or held pending. VIF is set and cleared for instructions that can modify IF, and it is cleared during software interrupts through interrupt gates. The original IF value is preserved in the EFLAGS image on the stack.
- *VIP*—This bit is set and cleared by the operating system via the EFLAGS image on the stack. It is set when an interrupt occurs for a Virtual-8086 program whose VIF bit is cleared. The bit is checked by the processor when the program subsequently attempts to set VIF.

Figure 18 and Table 22 show the VIF and VIP bits in the EFLAGS register. The VME extensions support conventional emulation methods for passing interrupts to Virtual-8086 programs, but they make it possible for the operating system to avoid time-consuming emulation of most instructions that write or read the IF.

The VIF and IF flags only affect the way the operating system deals with hardware interrupts (the INTR signal). Software interrupts are handled like machine-generated exceptions and

cannot be masked by real or virtual copies of IF (See "Software Interrupts and the Interrupt Redirection Bitmap (IRB) Extension" on page 75). The VIF and VIP flags only ease the software overhead associated with managing interrupts so that virtual copies of the IF flag do not have to be maintained by the operating system. Instead, each task's TSS holds its own copy of these flags in its EFLAGS image.



Symbol	Description	Bits
ID	ID Flag	21
VIP	Virtual Interrupt Pending	20
VIF	Virtual Interrupt Flag	19
AC	Alignment Check	18
VM	Virtual-8086 Mode	17
RF	Resume Flag	16
NT	Nested Task	14
IOPL	I/O Privilege Level	13–12
OF	Overflow Flag	11
DF	Direction Flag	10
IF	Interrupt Flag	9
TF	Trap Flag	8
SF	Sign Flag	7
ZF	Zero Flag	6
AF	Auxiliary Flag	4
PF	Parity Flag	2
CF	Carry Flag	0

Figure 18. EFLAGS Register

Table 22. Virtual-Interrupt Additions to EFLAGS Register

Bit	Mnemonic	Description	Function
20	VIP	Virtual Interrupt Pending	Set by the operating system (via the EFLAGS image on the stack) when an external maskable interrupt (INTR) occurs for a Virtual-8086 program whose VIF bit is cleared. The bit is checked by the processor when the program subsequently attempts to set VIF.
19	VIF	Virtual Interrupt Flag	When the VME bit in CR4 is set, the VIF bit is modified by the processor when a Virtual-8086 program running at less privilege than the IOPL attempts to modify the IF bit. The VIF bit is used by the operating system to determine whether a maskable interrupt should be passed on to the program or held pending.

Tables 23 through 27 show the effects, in various x86-processor modes, of instructions that read or write the IF and VIF flag. The column headings in this table include the following values:

- *PE*—Protection Enable bit in CR0 (bit 0)
- *VM*—Virtual-8086 Mode bit in EFLAGS (bit 17)
- *VME*—Virtual Mode Extensions bit in CR4 (bit 0)
- *PVI*—Protected-mode Virtual Interrupts bit in CR4 (bit 1)
- *IOPL*—I/O Privilege Level bits in EFLAGS (bits 13–12)
- *Handler CPL*—Code Privilege Level of the interrupt handler
- *GP(0)*—General-protection exception, with error code = 0
- *IF*—Interrupt Flag bit in EFLAGS (bit 9)
- *VIF*—Virtual Interrupt Flag bit in EFLAGS (bit 19)

Table 23. Instructions that Modify the IF or VIF Flags—Real Mode

TYPE	PE	VM	VME	PVI	IOPL	GP(0)	IF	VIF
CLI	0	0	0	0	–	No	IF ← 0	–
STI	0	0	0	0	–	No	IF ← 1	–
PUSHF	0	0	0	0	–	No	Pushed	–
POPF	0	0	0	0	–	No	Popped	–
IRET	0	0	0	0	–	No	Popped	–

Note:
"–" Not applicable.

Table 24. Instructions that Modify the IF or VIF Flags—Protected Mode

TYPE	PE	VM	VME	PVI	IOPL	Handler CPL	GP(0)	IF	VIF
CLI	1	0	–	0	≥ CPL	–	No	IF ← 0	–
CLI	1	0	–	0	< CPL	–	Yes	–	–
STI	1	0	–	0	≥ CPL	–	No	IF ← 1	–
STI	1	0	–	0	< CPL	–	Yes	–	–
PUSHF	1	0	–	0	≥ CPL	–	No	Pushed	–
PUSHF	1	0	–	0	< CPL	–	No	Pushed	–
PUSHFD	1	0	–	0	≥ CPL	–	No	Pushed	Pushed
PUSHFD	1	0	–	0	< CPL	–	No	Pushed	Pushed
POPF	1	0	–	0	≥ CPL	–	No	Popped	–
POPF	1	0	–	0	< CPL	–	No	Not Popped	–
POPFD	1	0	–	0	≥ CPL	–	No	Popped	Not Popped
POPFD	1	0	–	0	< CPL	–	No	Not Popped	Not Popped
IRET	1	0	–	0	–	= 0	No	Popped	–
IRET	1	0	–	0	≥ CPL	> 0	No*	Popped	–
IRET	1	0	–	0	< CPL	> 0	No*	Not Popped	–
IRETD	1	0	–	0	–	= 0	No	Popped	Popped
IRETD	1	0	–	0	≥ CPL	> 0	No*	Popped	Not Popped
IRETD	1	0	–	0	< CPL	> 0	No*	Not Popped	Not Popped

Notes:

** GP(0), if the CPL of the task executing IRETD is greater than the CPL of the task to which it is returning.*

"–" Not applicable.

Table 25. Instructions that Modify the IF or VIF Flags—Virtual-8086 Mode

TYPE	PE	VM	VME	PVI	IOPL	GP(0)	IF	VIF
CLI	1	1	0	–	3	No	IF ← 0	No Change
CLI	1	1	0	–	< 3	Yes	–	–
STI	1	1	0	–	3	No	IF ← 1	No Change
STI	1	1	0	–	< 3	Yes	–	–
PUSHF	1	1	0	–	3	No	Pushed	–
PUSHF	1	1	0	–	< 3	Yes	–	–
PUSHFD	1	1	0	–	3	No	Pushed	Pushed
PUSHFD	1	1	0	–	< 3	Yes	–	–
POPF	1	1	0	–	3	No	Popped	–
POPF	1	1	0	–	< 3	Yes	–	–
POPFD	1	1	0	–	3	No	Popped	Not Popped
POPFD	1	1	0	–	< 3	Yes	–	–
IRETD[2]	1	1	0	–	–	No	Popped	Popped

Notes:

1. *All Virtual-8086 mode tasks run at CPL = 3.*
2. *All protected virtual interrupt handlers run at CPL = 0.*

"–" Not applicable.

Table 26. Instructions that Modify the IF or VIF Flags—Virtual-8086 Mode Interrupt Extensions (VME)[1]

TYPE	PE	VM	VME	PVI	IOPL	GP(0)	IF	VIF
CLI	1	1	1	–	3	No	IF ← 0	No Change
CLI	1	1	1	–	< 3	No	No Change	VIF ← 0
STI	1	1	1	–	3	No	IF ← 1	No Change
STI	1	1	1	–	< 3	No[3]	No Change	VIF ← 1
PUSHF	1	1	1	–	3	No	Pushed	Not Pushed
PUSHF	1	1	1	–	< 3	No	Not Pushed	Pushed into IF
PUSHFD	1	1	1	–	3	No	Pushed	Pushed
PUSHFD	1	1	1	–	< 3	Yes	–	–
POPF	1	1	1	–	3	No	Popped	Not Popped
POPF	1	1	1	–	< 3	No	Not Popped	Popped from IF
POPFD	1	1	1	–	3	No	Popped	Not Popped
POPFD	1	1	1	–	< 3	Yes	–	–
IRET from V86 Mode	1	1	1	–	3	No	Popped	Not Popped
IRET from V86 Mode	1	1	1	–	< 3	No[3]	Not Popped	Popped from IF
IRETD from V86 Mode	1	1	1	–	3	No	Popped	Not Popped
IRETD from V86 Mode	1	1	1	–	< 3	Yes	–	–
IRETD from Protected Mode[2]	1	1	1	–	–	No[3]	Popped	Popped

Notes:

1. *All Virtual-8086 mode tasks run at CPL = 3.*
2. *All protected virtual interrupt handlers run at CPL = 0.*
3. *GP(0) if an attempt is made to set VIF when VIP = 1.*

"–" Not applicable.

Table 27. Instructions that Modify the IF or VIF Flags—Protected Mode Virtual Interrupt Extensions (PVI)[1]

TYPE	PE	VM	VME	PVI	IOPL	GP(0)	IF	VIF
CLI	1	0	–	1	3	No	IF ← 0	No Change
CLI	1	0	–	1	< 3	No	No Change	VIF ← 0
STI	1	0	–	1	3	No	IF ← 1	No Change
STI	1	0	–	1	< 3	No[3]	No Change	VIF ← 1
PUSHF	1	0	–	1	3	No	Pushed	Not Pushed
PUSHF	1	0	–	1	< 3	No	Pushed	Not Pushed
PUSHFD	1	0	–	1	3	No	Pushed	Pushed
PUSHFD	1	0	–	1	< 3	No	Pushed	Pushed
POPF	1	0	–	1	3	No	Popped	Not Popped
POPF	1	0	–	1	< 3	No	Not Popped	Not Popped
POPFD	1	0	–	1	3	No	Popped	Not Popped
POPFD	1	0	–	1	< 3	No	Not Popped	Not Popped
IRETD[2]	1	0	–	1	–	No[3]	Popped	Popped

Notes:

1. *All Protected mode virtual interrupt tasks run at CPL = 3.*
2. *All protected mode virtual interrupt handlers run at CPL = 0.*
3. *GP(0) if an attempt is made to set VIF when VIP = 1.*

"–" Not applicable.

Software Interrupts and the Interrupt Redirection Bitmap (IRB) Extension.

In Virtual-8086 mode, software interrupts (INT*n* exceptions that vector through interrupt gates) are trapped by the operating system for emulation because they would otherwise clear the real IF. When VME extensions are enabled, these INT*n* instructions are allowed to execute normally, vectoring directly to a Virtual-8086 service routine via the Virtual-8086 interrupt vector table (IVT) at address 0 of the task address space. However, it may still be desirable for security or performance reasons to intercept INT*n* instructions on a vector-specific basis to allow servicing by Protected-mode routines accessed through the interrupt descriptor table (IDT). This is accomplished by an Interrupt Redirection Bitmap (IRB) in the TSS, which is created by the operating system in a manner similar to the IO Permission Bitmap (IOPB) in the TSS.

Figure 19 shows the format of the TSS with the Interrupt Redirection Bitmap near the top. The IRB contains 256 bits, one for each possible software-interrupt vector. The most-significant bit of the IRB is located immediately below the base of the IOPB. This bit controls interrupt vector 255. The least-significant bit of the IRB controls interrupt vector 0.

The bits in the IRB work as follows:

- *Set*—If set to 1, the INT*n* instruction behaves as if the VME extensions are not enabled. The interrupt vectors to a Protected-mode routine if IOPL = 3, or it causes a general-protection exception with error code zero if IOPL<3.
- *Cleared*—If cleared to 0, the INT*n* instruction vectors directly to the corresponding Virtual-8086 service routine via the Virtual-8086 program's IVT.

Only software interrupts can be redirected via the IRB to a Real mode IVT—hardware interrupts cannot. Hardware interrupts are asynchronous events and do not belong to any current virtual task. The processor thus has no way of deciding which IVT (for which Virtual-8086 program) to direct a hardware interrupt to. Hardware interrupts, therefore, always require operating system intervention. The VIF and VIP bits described in "Hardware Interrupts and the VIF and VIP Extensions" on page 68 are provided to assist the operating system in this intervention.

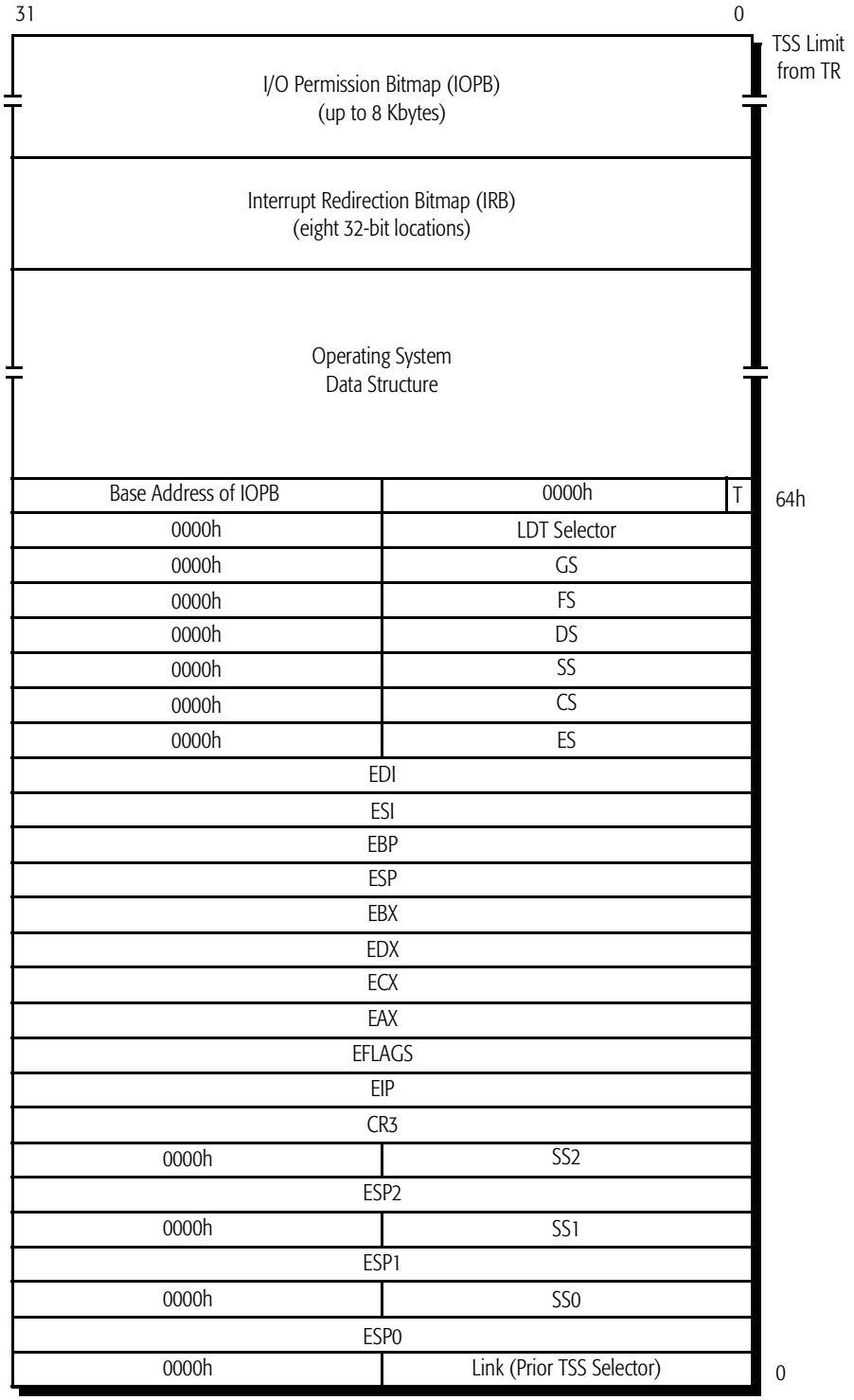


Figure 19. Task State Segment (TSS)

Table 28 compares the behavior of hardware and software interrupts in various x86-processor operating modes. It also shows which interrupt table is accessed: the Protected-mode IDT or the Real- and Virtual-8086-mode IVT. The column headings in this table include:

- *PE*—Protection Enable bit in CR0 (bit 0)
- *VM*—Virtual-8086 Mode bit in EFLAGS (bit 17)
- *VME*—Virtual Mode Extensions bit in CR4 (bit 0)
- *PVI*—Protected-Mode Virtual Interrupts bit in CR4 (bit 1)
- *IOPL*—I/O Privilege Level bits in EFLAGS (bits 13–12)
- *IRB*—Interrupt Redirection Bit for a task, from the Interrupt Redirection Bitmap (IRB) in the tasks TSS
- *GP(0)*—General-protection exception, with error code = 0
- *IDT*—Protected-Mode Interrupt Descriptor Table
- *IVT*—Real- and Virtual-8086 Mode Interrupt Vector Table

Table 28. Interrupt Behavior and Interrupt-Table Access

Mode	Interrupt Type	PE	VM	VME	PVI	IOPL	IRB	GP(0)	IDT	IVT
Real mode	Software	0	0	0	–	0	–	–	–	✓
	Hardware	0	0	0	–	0	–	–	–	✓
Protected mode	Software	1	0	0	–	–	–	–	✓	–
	Hardware	1	0	0	–	–	–	–	✓	–
Virtual-8086 mode*	Software	1	1	0	–	= 3	–	No	✓	–
	Software	1	1	0	–	< 3	–	Yes	✓	–
	Hardware	1	1	0	–	–	–	No	✓	–
Virtual-8086 Mode Extensions (VME)*	Software	1	1	1	0	–	0	No	–	✓
	Software	1	1	1	0	= 3	1	No	✓	–
	Software	1	1	1	0	< 3	1	Yes	✓	–
	Hardware	1	1	1	0	–	–	No	✓	–
Protected Virtual Extensions (PVI)	Software	1	0	1	1	–	–	No	✓	–
	Hardware	1	0	1	1	–	–	No	✓	–

Notes:
** All Virtual-8086 tasks run at CPL = 3.*
"–" Not applicable.

Protected Virtual Interrupt (PVI) Extensions

The Protected Virtual Interrupts (PVI) bit in CR4 enables support for interrupt virtualization in Protected mode. In this virtualization, the processor maintains program-specific VIF and VIP flags in a manner similar to those in Virtual-8086 Mode Extensions (VME). When a program is executed at CPL = 3, it can set and clear its copy of the VIF flag without causing general-protection exceptions.

The only differences between the VME and PVI extensions are that, in PVI, selective INT*n* interception using the Interrupt Redirection Bitmap in the TSS does not apply, and only the STI and CLI instructions are affected by the extension.

Tables 23 through 28 show, among other things, the behavior of hardware and software interrupts as well as instructions that affect interrupts in Protected mode with the PVI extensions enabled.

Model-Specific Registers (MSRs)

The processor supports MSRs that can be accessed with the RDMSR and WRMSR instructions when CPL = 0. The following index values in the ECX register access specific MSRs:

- Machine-Check Address Register (MCAR)—ECX = 00h
- Machine-Check Type Register (MCTR)—ECX = 01h
- Time Stamp Counter (TSC)—ECX = 10h
- Array Access Register (AAR)—ECX = 82h
- Hardware Configuration Register (HWCR)—ECX = 83h
- Write Allocate Top-of-Memory and Control Register (WATMCR)—ECX = 85h
- Write Allocate Programmable Memory Range Register (WAPMRR)—ECX = 86h

  Note: *The AMD-K5 processor supports write allocate only on Models 1, 2, and 3, with a Stepping of 4 or greater.*

The RDMSR and WRMSR instructions are described on page 90. The following sections describe the format of the registers.

Machine-Check Address Register (MCAR)

The processor latches the address of the current bus cycle in its 64-bit Machine-Check Address Register (MCAR) when a bus-cycle error occurs. These errors are indicated either by (a) system logic asserting BUSCHK, or (b) the processor asserting PCHK while system logic asserts PEN.

The MCAR can be read with the RDMSR instruction when the ECX register contains the value 00h. Figure 20 shows the format of the MCAR register. The contents of the register can be read with the RDMSR instruction.

If system software has set the MCE bit in CR4 before the bus-cycle error, the processor also generates a machine-check exception as described on page 60.

Figure 20. Machine-Check Address Register (MCAR)

Machine-Check Type Register (MCTR)

The processor latches the cycle definition and other information about the current bus cycle in its 64-bit Machine-Check Type Register (MCAR) at the same times that the Machine-Check Address Register (MCAR) latches the cycle address—when a bus-cycle error occurs. These errors are indicated either by (a) system logic asserting BUSCHK, or (b) the processor asserting PCHK while system logic asserts PEN.

The MCTR can be read with the RDMSR instruction when the ECX register contains the value 01h. Figure 21 and Table 29 show the formats of the MCTR register. The contents of the register can be read with the RDMSR instruction. The processor clears the CHK bit (bit 0) in MCTR when the register is read with the RDMSR instruction.

If system software has set the MCE bit in CR4 before the bus-cycle error, the processor also generates a machine-check exception as described on page 60.

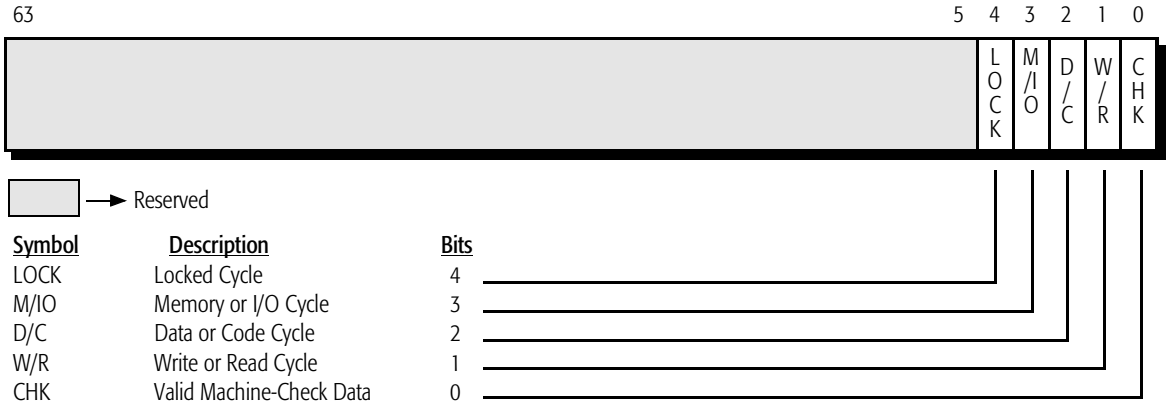


Figure 21. Machine-Check Type Register (MCTR)

Table 29. Machine-Check Type Register (MCTR) Fields

Bit	Mnemonic	Description	Function
4	LOCK	Locked Cycle	Set to 1 if the processor was asserting LOCK# during the bus cycle.
3	M/IO#	Memory or I/O	1 = memory cycle, 0 = I/O cycle
2	D/C#	Data or Code	1 = data cycle, 0 = code cycle
1	W/R#	Write or Read	1 = write cycle, 0 = read cycle
0	CHK	Valid Machine-Check Data	The processor sets the CHK bit to 1 when both the MCTR and MCAR registers contain valid information. The processor clears the CHK bit to 0 when software reads the MCTR with the RDMSR instruction.

Time Stamp Counter (TSC)

With each processor clock cycle, the processor increments a 64-bit time stamp counter (TSC) MSR. The counter can be written or read using the WRMSR or RDMSR instructions when the ECX register contains the value 10h and CPL = 0. The counter can also be read using the RDTSC instruction (see page 89), but the required privilege level for this instruction is determined by the Time Stamp Disable (TSD) bit in CR4. With any of these instructions, the EDX and EAX registers hold the upper and lower doublewords (dwords) of the 64-bit value to be written to or read from the TSC, as
follows:

- *EDX*—Upper 32 bits of TSC
- *EAX*—Lower 32 bits of TSC

The TSC can be loaded with any arbitrary value.

Array Access Register (AAR)

The Array Access Register (AAR) contains pointers for testing the tag and data arrays for the instruction cache, data cache, 4-Kbyte TLB, and 4-Mbyte TLB. The AAR can be written or read with the WRMSR or RDMSR instruction when the ECX register contains the value 82h.

For details on the AAR, see "Cache and TLB Testing" on page 27.

Hardware Configuration Register (HWCR)

The Hardware Configuration Register (HWCR) contains configuration bits that control miscellaneous debugging functions. The HWCR can be written or read with the WRMSR or RDMSR instruction when the ECX register contains the value 83h.

For details on the HWCR, see "Hardware Configuration Register (HWCR)" on page 22.

Write Allocate Registers

The AMD-K5 processor supports write allocate only on Models 1, 2, and 3, with a Stepping of 4 or greater. Use the CPUID instruction to determine if the proper revision of the processor is present (See the *AMD Processor Recognition Application Note*, order# 20734, located at http://www.amd.com.).

Two MSRs are defined to support write allocate. The MSRs are accessed using the RDMSR and WRMSR instructions (see "RDMSR and WRMSR" of the *AMD-K5™ Processor Software Development Guide*, order# 20007). The following index values in the ECX register access the MSRs:

- Write Allocate Top-of-Memory and Control Register (WATMCR)—ECX = 85h
- Write Allocate Programmable Memory Range Register (WAPMRR)—ECX = 86h

For more information about write allocate, see the *Implementation of Write Allocate in the K86™ Processors Application Note*, order# 21326.

Three non-write-allocatable memory ranges are defined for use with the write allocate feature—one fixed range and two programmable ranges.

Fixed Range. The fixed memory range is 000A_0000h–000F_FFFFh and can be enabled or disabled. When enabled, write allocate can not be performed in this range.

This region of memory, which includes standard VGA and other peripheral and BIOS access, is considered non-cacheable. Performing a write allocate in this area can cause compatibility problems. It is recommended that this bit be enabled (set to 1) to prevent write allocate to this range. Set bit 16 of WATMCR to enable protection of this range.

Programmable Range. One programmable memory range is xxxx_0000h–yyyy_FFFFh, where xxxx and yyyy are defined using bits 15–0 and bits 31–16 of WAPMRR, respectively. Set bit 17 of WATMCR to enable protection of this range. When enabled, write allocate can not be performed in this range.

This programmable memory range exists because a small number of uncommon memory-mapped I/O adapters are mapped to physical RAM locations. If a card like this exists in the system configuration, it is recommended that the BIOS program the 'memory hole' for the adapter into this non-write-allocatable range.

Top of Memory. The other programmable memory range is defined by the 'top-of-memory' field. The top of memory is equal to zzzz_0000h, where zzzz is defined using bits 15–0 of WATMCR. Addresses above zzzz_0000h are protected from write allocate when bit 18 of WATMCR is enabled.

Once the BIOS determines the size of RAM installed in the system, this size should also be used to program the top of memory. For example, a system with 32 Mbytes of RAM requires that the top-of-memory field be programmed with a value of 0200h, which enables protection from write allocate for memory above that value. Set bit 18 of WATMCR to enable protection of this range.

Caching and write allocate are generally not performed for the memory above the amount of physical RAM in the system. Video frame buffers are usually mapped above physical RAM. If write allocate were attempted in that memory area, there could be performance degradation or compatibility problems.

Bits 18–16 of WATMCR control the enabling or disabling of the three memory ranges as follows:

- Bit 18: Top-of-Memory Enable bit

  0 = disabled (default)
  1 = enabled (write allocate can not be performed above Top of Memory)

- Bit 17: Programmable Range Enable bit

  0 = disabled (default)

  1 = enabled (write allocate can not be performed in this range)

- Bit 16: Fixed Range Enable bit

  0 = disabled (default)

  1 = enabled (write allocate can not be performed in this range)

Figures 22 and 23 show the bit positions for these two new registers.

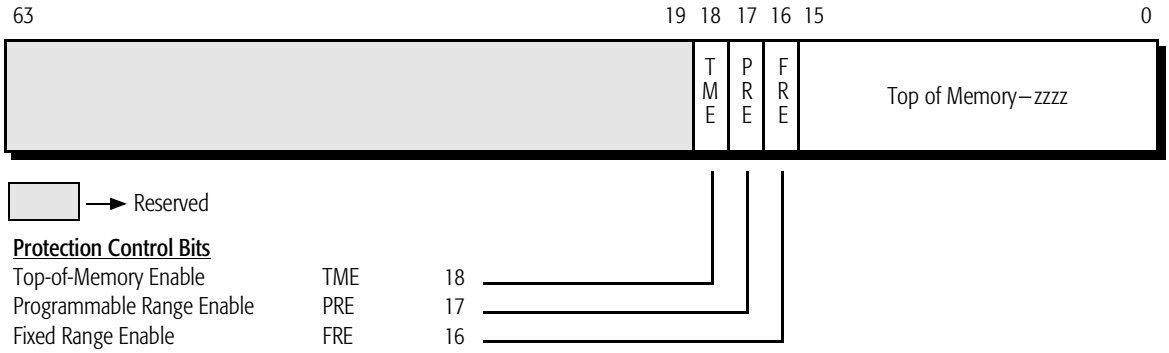


Figure 22. Write Allocate Top-of-Memory and Control Register (WATMCR)—MSR 85h

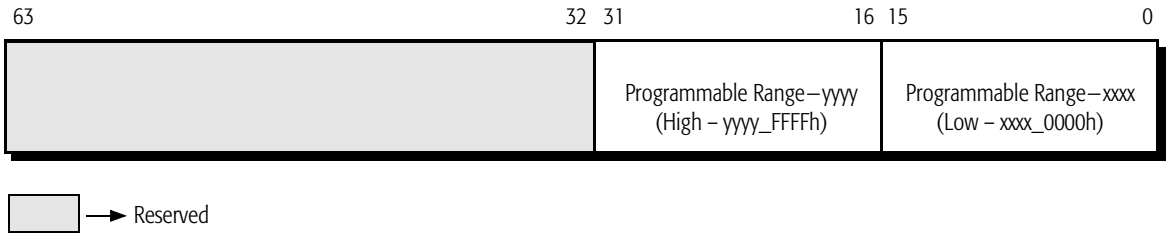


Figure 23. Write Allocate Programmable Memory Range Register (WAPMRR)—MSR 86h

Enable Write Allocate

Write allocate is enabled by setting bit 4 (WA) of the HWCR to 1. For more information on the HWCR, see "Hardware Configuration Register (HWCR)" on page 22. Figure 2 on page 23 shows the revised definition of the Hardware Configuration Register.

New AMD-K5™ Processor Instructions

In addition to supporting all the 486 processor instructions, the AMD-K5 processor implements the following instructions:

- CPUID
- CMPXCHG8B
- MOV to and from CR4
- RDTSC
- RDMSR
- WRMSR
- RSM
- Illegal instruction (reserved opcode)

CPUID

mnemonic	*opcode*	*description*
CPUID	0F A2h	Identify processor and its feature set

Privilege: Any level
Registers Affected: EAX, EBX, ECX, EDX
Flags Affected: None
Exceptions Generated: None

The CPUID instruction is an application-level instruction that software executes to identify the processor and its feature set. This instruction offers multiple functions, each providing a different set of information about the processor. The CPUID instruction can be executed from any privilege level. Software can use the information returned by this instruction to tune its functionality for the specific processor and its features.

Not all processors implement the CPUID instruction. Therefore, software must test to determine if the instruction is present on the processor. If the ID bit (21) in the EFLAGS register is writeable, the CPUID instruction is implemented.

The CPUID instruction supports multiple functions. The information associated with each function is obtained by executing the CPUID instruction with the function number in the EAX register. Functions are divided into two types: standard functions and extended functions. Standard functions are found in the low function space, 0000_0000h–7FFF_FFFFh. In general, all x86 processors have the same standard function definitions.

Extended functions are defined specifically for processors supplied by the vendor listed in the vendor identification string. Extended functions are found in the high function space, 8000_0000h–8FFF_FFFFh. Because not all vendors have defined extended functions, software must test for their presence on the processor.

For more detailed information refer to the *AMD Processor Recognition Application Note*, order# 20734, located at http://www.amd.com.

CMPXCHG8B

mnemonic	*opcode*	*description*
CMPXCHG8B *r/m64*	0F C7h	Compare and exchange 8-byte operand

Privilege: Any level
Registers Affected: EAX, EBX, ECX, EDX
Flags Affected: ZF
Exceptions Generated:

Exception	Real	Virtual 8086	Protected	Description
Invalid opcode (6)	X	X	X	Invalid opcode if destination is a register.
Stack exception (12)			X	During instruction execution, the stack segment limit was exceeded.
General protection (13)	X	X	X	During instruction execution, the effective address of one of the segment registers used for the operand points to an illegal memory location.
Page fault (14)		X	X	A page fault resulted from the execution of the instruction.
Alignment check (17)		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment mask bit (AM) of the control register (CR0) is set to 1. (In Protected Mode, CPL = 3.)

The CMPXCHG8B instruction is an 8-byte version of the 4-byte CMPXCHG instruction supported by the 486 processor. CMPXCHG8B compares a value from memory with a value in the EDX and EAX register, as follows:

- *EDX* — Upper 32 bits of compare value
- *EAX* — Lower 32 bits of compare value

If the memory value matches the value in EDX and EAX, the ZF flag is set to 1 and the 8-byte value in ECX and EBX is written to the memory location, as follows:

- *ECX* — Upper 32 bits of exchange value
- *EBX* — Lower 32 bits of exchange value

MOV to and from CR4

mnemonic	*opcode*	*description*
MOV CR4,*r32*	0F 22h	Move to CR4 from register
MOV *r32*,CR4	0F 20h	Move to register from CR4

Privilege: CPL = 0
Registers Affected: CR4, 32-bit general-purpose register
Flags Affected: OF, SF, ZF, AF, PF, and CF are undefined
Exceptions Generated:

Exception	Real	Virtual 8086	Protected	Description
General protection (13)	X		X	If 1 is written to any reserved bits.
		X		Executing this instruction in Virtual 8086 mode.
			X	If CPL not = 0.

These instructions read and write control register 4 (CR4).

RDTSC

mnemonic	*opcode*	*description*
RDTSC	0F 31h	Read time stamp counter

Privilege: Selectable by TSD bit in CR4
Registers Affected: EAX, EDX
Flags Affected: none
Exceptions Generated:

Exception	Real	Virtual 8086	Protected	Description
General protection (13)		X		Executing this instruction in Virtual 8086 mode.
			X	If CPL not = 0 when TSD bit of CR4 = 1.

The AMD-K5 processor's 64-bit time stamp counter (TSC) increments on each processor clock. In Real or Protected mode, the counter can be read with the RDMSR instruction and written with the WRMSR instruction when CPL = 0. However, in Protected mode, the RDTSC instruction can be used to read the counter at privilege levels higher than CPL = 0.

The required privilege level for using the RDTSC instruction is determined by the Time Stamp Disable (TSD) bit in CR4, as follows:

- *CPL = 0* — Set the TSD bit in CR4 to 1
- *Any CPL* — Clear the TSD bit in CR4 to 0

The RDTSC instruction reads the counter value into the EDX and EAX registers as follows:

- *EDX* — Upper 32 bits of TSC
- *EAX* — Lower 32 bits of TSC

The following example shows how the RDTSC instruction can be used. After this code is executed, EAX and EDX contain the time required to execute the RDTSC instruction.

```
mov   ecx,10h           ;Time Stamp Counter Access via MSRs
mov   eax,00000000h     ;Initialize the eax part of the Counter to zero
mov   edx,00000000h     ;Initialize the edx part of the Counter to zero
db    0Fh, 30h          ;WRMSR
db    0Fh, 31h          ;RDTSC
db    0Fh, 31h          ;RDTSC
```

RDMSR and WRMSR

mnemonic	*opcode*	*description*
RDMSR	0F 32h	Read model-specific register (MSR)
WRMSR	0F 30h	Write model-specific register (MSR)

Privilege: CPL=0
Registers Affected: EAX, ECX, EDX
Flags Affected: none
Exceptions Generated:

Exception	Real	Virtual 8086	Protected	Description
General protection (13)	X			For unimplemented MSR address.
		X		Executing this instruction in Virtual 8086 mode.
			X	For unimplemented MSR address OR if CPL not = 0.

The RDMSR or WRMSR instructions can be used in Real or Protected mode to access several 64-bit MSRs. These registers are addressed by the value in ECX, as follows:

- *00h:* Machine-Check Address Register (MCAR). This may contain the physical address of the last bus cycle for which the BUSCHK or PCHK signal was asserted. For details, see "Machine-Check Address Register (MCAR)" on page 80.
- *01h:* Machine-Check Type Register (MCTR). This contains the cycle definition of the last bus cycle for which the BUSCHK or PCHK signal was asserted. For details, see "Machine-Check Type Register (MCTR)" on page 80. The processor clears the CHK bit (bit 0) in MCTR when the register is read with the RDMSR instruction.
- *10h*: Time Stamp Counter (TSC). This contains a time value. The TSC can be initialized to any value with the WRMSR instruction, and it can be read with either the RDMSR or RDTSC instruction. For details, see "Time Stamp Counter (TSC)" on page 81.
- *82h*: Array Access Register (AAR). This contains an array pointer and test data for testing the processor's cache and TLB arrays. For details on the AAR, see "Cache and TLB Testing" on page 27.
- *83h*: Hardware Configuration Register (HWCR). This contains configuration bits that control miscellaneous debugging functions. For details, see "Hardware Configuration Register (HWCR)" on page 22.
- *85h*: Write Allocate Top-of-Memory and Control Register (WATMCR)
- *86h*: Write Allocate Programmable Memory Range Register (WAPMRR)

Note: *The AMD-K5 processor supports write allocate only on Models 1, 2, and 3, with a Stepping of 4 or greater.*

The above values in ECX identify the register to be read or written. The EDX and EAX registers contain the MSR values to be read or written, as follows:

- *EDX*—Upper 32 bits of MSR. For the AAR, this contains the array pointer and (in contrast to all other MSRs) its contents are not altered by a RDMSR instruction.
- *EAX*—Lower 32 bits of MSR. For the AAR, this contains the data to be read/written.

All MSRs are 64 bits wide. However, the upper 32 bits of the AAR are write-only and are not returned on a read. EDX remains unaltered, making it more convenient to maintain the array pointer.

If an attempt is made to execute either the RDMSR or WRMSR instruction when CPL is greater than 0, or to access an undefined MSR, the processor generates a general-protection exception with error code zero.

Model-Specific Registers, as their name implies, may or may not be implemented by later models of the AMD-K5 processor.

RSM

mnemonic	*opcode*	*description*
RSM	0F AAh	Resume execution (exit System Management Mode)

Privilege: CPL = 0
Registers Affected: CS, DS, ES, FS, GS, SS, EIP, EFLAGS, LDTR, CR3, EAX, EBX, ECX, EDX, ESP, EBP, EDI, ESI
Flags Affected: none
Exceptions Generated:

Exception	Real	Virtual 8086	Protected	Description
Invalid opcode (6)	X	X	X	Invalid opcode if not in SMM Mode.

The RSM instruction should be the last instruction in an System Management Mode (SMM) service routine. It restores the processor state that was saved when the SMI interrupt was asserted. This instruction is only valid when the processor is in SMM. It generates an invalid opcode exception at all other times.

The processor enters the Shutdown state if any of the following illegal conditions are encountered during the execution of the RSM instruction:

- the SMM base value is not aligned on a 32-Kbyte boundary
- Any reserved bit of CR4 is set to 1
- The PG bit is set while the PE is cleared in CR0
- The NW bit is set while the CD bit is cleared in CR0

Illegal Instruction (Reserved Opcode)

mnemonic	*opcode*	*description*
(none)	0F FFh	Illegal instruction (reserved opcode)

Privilege: Any level
Registers Affected: none
Flags Affected: none
Exceptions Generated:

Exception	Real	Virtual 8086	Protected	Description
Invalid opcode (6)	X	X	X	Invalid opcode if executed.

This opcode always generates an invalid opcode exception. The opcode will not be used in future AMD K86 processors.

4

AMD-K6™ MMX™ Enhanced Processor

The following sections describe additional information required by BIOS developers to properly incorporate the AMD-K6 MMX enhanced processor into a system. The BIOS for the AMD-K6 needs minimal changes in order to fully support the AMD-K6 processor family.

BIOS Consideration Checklist

CPUID

- Use the CPUID instruction to properly identify the AMD-K6 processor.
- Determine the processor type, stepping and features using functions 0000_0001h and 8000_0001h of the CPUID instruction.
- Boot-up display: The processor name should be displayed as 'AMD-K6(tm)/XXX'. See "CPU Identification Algorithms" on page 3 for more information.

CPU Speed Detection

- Use speed detection algorithms that do not rely on repetitive instruction sequences.
- Use the Time Stamp Counter (TSC) to 'clock' a timed operation and compare the result to the Real Time Clock (RTC) to determine the operating frequency. See the example of frequency-determination assembler code available on the AMD website at http://www.amd.com.
- Display the P-Rating shown in Table 2, "Summary of AMD-K6™ MMX™ Enhanced Processor CPU IDs and BIOS Boot Strings," on page 4.

Model-Specific Registers (MSRs)

- Only access MSRs implemented in the AMD-K6 processor.
- Enable Write Allocation by programming the Write Handling Control Register (WHCR). See "Write Handling Control Register (WHCR)" on page 119 and the *Implementation of Write Allocate in the K86™ Processors Application Note*, order# 21326 for more information.

Cache Testing

- Use the AMD-K6 processor's BIST function to test internal memories. See "Built-In Self-Test (BIST)" on page 106 for more information. The AMD-K6 does not contain MSRs to allow for cache testing.

SMM Issues

- The System Management Mode (SMM) functionality of the AMD-K6 processor is identical to Pentium.
- Implement the AMD-K6 processor SMM state-save area in the same manner as Pentium except for the IDT Base and possibly Pentium-reserved areas. See "AMD-K6™ Processor System Management Mode" on page 97 for more information.

AMD-K6™ Processor System Management Mode

The System Management Mode (SMM) in the AMD-K6 MMX enhanced processor is similar to the AMD-K5 processor. This section points out the differences. See "AMD-K5™ Processor System Management Mode (SMM)" on page 7 for details on the AMD-K5 processor implementation of SMM.

Initial Register Values

The general purpose registers and DR6 are unmodified when entering SMM. Table 30 shows the default register values when entering SMM.

Table 30. Initial State of Registers in SMM

Register	Initial Contents		
	Selector	Base	Limit
CS	3000h	0003_0000h	4 Gbytes
DS	0000h	0000_0000h	4 Gbytes
ES	0000h	0000_0000h	4 Gbytes
FS	0000h	0000_0000h	4 Gbytes
GS	0000h	0000_0000h	4 Gbytes
SS	0000h	0000_0000h	4 Gbytes
General-Purpose Registers	Unmodified		
EFLAGS	0000_0002h		
EIP	0000_8000h		
CR0	Bits 0, 2, 3, and 31 cleared (PE, EM, TS, and PG); remainder are unmodified.		
CR4	0000_0000h		
GDTR	Unmodified		
LDTR	Unmodified		
IDTR	Unmodified		
TR	Unmodified		
DR7	0000_0400h		
DR6	Unmodified		

SMM State-Save Area

When the SMI# is recognized the AMD-K6 processor saves its state to the state-save area shown in Table 31. If the SMI# has been relocated, the state dump begins at CS Base + 7FFFh (8000 + 7FFFh). The default CS Base is 30000h.

Table 31. AMD-K6™ Processor State-Save Map

Address Offset	AMD-K5™	AMD-K6™
FFFCh	CR0	CR0
FFF8h	CR3	CR3
FFF4h	EFLAGS	EFLAGS
FFF0h	EIP	EIP
FFECh	EDI	EDI
FFE8h	ESI	ESI
FFE4h	EBP	EBP
FFE0h	ESP	ESP
FFDCh	EBX	EBX
FFD8h	EDX	EDX
FFD4h	ECX	ECX
FFD0h	EAX	EAX
FFCCh	DR6	DR6
FFC8h	DR7	DR7
FFC4h	TR	TR
FFC0h	LDTR Base	LDTR Base
FFBCh	GS	GS
FFB8h	FS	FS
FFB4h	DS	DS
FFB0h	SS	SS
FFACh	CS	CS
FFA8h	ES	ES
FFA4h	I/O Trap Dword	I/O Trap Dword
FFA0h	–	–
FF9Ch	I/O Trap EIP *	I/O Trap EIP *
FF98h	–	–

Notes:
– *No dump at that address.*
* *Only contains information if SMI# was asserted on a valid corresponding I/O.*

Table 31. AMD-K6™ Processor State-Save Map (continued)

Address Offset	AMD-K5™	AMD-K6™
FF94h	–	–
FF90h	IDT Base	IDT Base
FF8Ch	IDT Limit	IDT Limit
FF88h	GDT Base	GDT Base
FF84h	GDT Limit	GDT Limit
FF80h	TSS Attr	TSS Attr
FF7Ch	TSS Base	TSS Base
FF78h	TSS Limit	TSS Limit
FF74h	LDT Attr	–
FF70h	LDT Base	LDT Low
FF6Ch	LDT Limit	LDT High
FF68h	GS Attr	GS Attr
FF64h	GS Base	GS Base
FF60h	GS Limit	GS Limit
FF5Ch	FS Attr	FS Attr
FF58h	FS Base	FS Base
FF54h	FS Limit	FS Limit
FF50h	DS Attr	DS Attr
FF4Ch	DS Base	DS Base
FF48h	DS Limit	DS Limit
FF44h	SS Attr	SS Attr
FF40h	SS Base	SS Base
FF3Ch	SS Limit	SS Limit
FF38h	CS Attr	CS Attr
FF34h	CS Base	CS Base
FF30h	CS Limit	CS Limit
FF2Ch	ES Attr	ES Attr
FF28h	ES Base	ES Base
FF24h	ES Limit	ES Limit
FF20h	–	–
FF1Ch	–	–

Notes:

– *No dump at that address.*

* *Only contains information if SMI# was asserted on a valid corresponding I/O.*

Table 31. AMD-K6™ Processor State-Save Map (continued)

Address Offset	AMD-K5™	AMD-K6™
FF18h	–	–
FF14h	CR2	CR2
FF10h	CR4	CR4
FF0Ch	I/O restart ESI*	I/O restart ESI*
FF08h	I/O restart ECX*	I/O restart ECX*
FF04h	I/O restart EDI*	I/O restart EDI*
FF02h	HALT Restart Slot	HALT Restart Slot
FF00h	I/O Restart Slot	I/O Restart Slot
FEFCh	SMM RevID	SMM RevID
FEF8h	SMM BASE	SMM BASE
FEF7–FE00h	–	–

Notes:
– *No dump at that address.*
* *Only contains information if SMI# was asserted on a valid corresponding I/O.*

SMM Revision Identifier

The SMM Revision Identifier specifies the version of SMM and the extensions available on the processor. Table 32 defines the bits associated with this register. A 1 present in either the I/O Trap Extension or the SMM Base Relocation indicates this feature is available for use.

Table 32. SMM Revision Identifier

31–18	17	16	15–0
Reserved	SMM Base Relocation	I/O Trap Extension	SMM Revision Level
0	1	1	0002h

SMM Base Address

This feature is compatible with the AMD-K5 processor and Pentium. See “SMM Base Address” on page 12.

Auto Halt Restart

This feature is compatible with the AMD-K5 processor and the Pentium processor. See "Auto Halt Restart Slot" on page 13.

I/O Trap Dword

If the assertion of SMI# is recognized on the boundary of an I/O bus cycle, the I/O trap doubleword at offset FFA4h in the SMM state-save area contains information about the associated I/O instruction. The AMD-K6 processor provides additional information at this offset when compared to the AMD-K5 processor. The AMD-K6 processor provides a bit to determine if the I/O string operand is a REP string operation. The fields of the I/O Trap Dword are configured as shown in Table 33.

Table 33. AMD-K6™ Processor I/O Trap Dword Configuration

31–16	15–4	3	2	1	0
I/O Port Address	Reserved	Rep String Operation	I/O String Operation	Valid I/O Instruction	Input or Output

I/O Trap Restart

This feature is compatible with the AMD-K5 processor. See "I/O Trap Restart Slot" on page 14.

Exceptions and Interrupts Within SMM

This feature is compatible with the AMD-K5. See "Exceptions and Interrupts in SMM" on page 16.

AMD-K6™ Processor Reset State

Table 34 shows the state of all architecture registers and MSRs after the processor has completed its initialization resulting from the recognition of the assertion of RESET.

Table 34. State of the AMD-K6™ Processor After RESET

Register	RESET State	Notes
GDTR	base:0000_0000 limit:0FFFFh	
IDTR	base:0000_0000 limit:0FFFFh	
TR	0000h	
LDTR	0000h	
EIP	FFFF_FFF0h	
EFLAGS	0000_0002h	
EAX	0000_0000h	1
EBX	0000_0000h	
ECX	0000_0000h	
EDX	0000_056xh	2
ESI	0000_0000h	
EDI	0000_0000h	
EBP	0000_0000h	
ESP	0000_0000h	
CS	F000h	
SS	0000h	
DS	0000h	
ES	0000h	
FS	0000h	
GS	0000h	
FPU Stack R7–R0	0000_0000_0000_0000_0000h	
FPU Control Word	0040h	
FPU Status Word	0000h	
FPU Tag Word	5555h	

Notes:

1. *The contents of EAX indicate if BIST was successful. If EAX = 0000_0000h, then BIST was successful. If EAX is non-zero, BIST failed.*
2. *EDX contains the AMD-K6 processor signature.*
3. *These Model-Specific Registers are described in "AMD-K6™ Processor x86 Architecture Extensions" on page 117.*

Table 34. State of the AMD-K6™ Processor After RESET (continued)

Register	RESET State	Notes
FPU Instruction Pointer	0000_0000_0000h	
FPU Data Pointer	0000_0000_0000h	
FPU Opcode Register	000_0000_0000b	
CR0	6000_0010h	
CR2	0000_0000h	
CR3	0000_0000h	
CR4	0000_0000h	
DR7	0000_0400h	
DR6	FFFF_0FF0h	
DR3	0000_0000h	
DR2	0000_0000h	
DR1	0000_0000h	
DR0	0000_0000h	
MCAR	0000_0000_0000_0000h	
MCTR	0000_0000_0000_0000h	
TR12	0000_0000_0000_0000h	
TSC	0000_0000_0000_0000h	
EFER	0000_0000_0000_0000h	3
STAR	0000_0000_0000_0000h	3
WHCR	0000_0000_0000_0000h	3

Notes:

1. *The contents of EAX indicate if BIST was successful. If EAX = 0000_0000h, then BIST was successful. If EAX is non-zero, BIST failed.*
2. *EDX contains the AMD-K6 processor signature.*
3. *These Model-Specific Registers are described in "AMD-K6™ Processor x86 Architecture Extensions" on page 117.*

Segment Register Attributes

See Table 10 on page 20 for segment register attribute initial values.

State of the AMD-K6™ Processor After INIT

The assertion of INIT causes the processor to empty its pipelines, initialize most of its internal state, and branch to address FFFF_FFF0h—the same instruction execution starting point used after RESET. Unlike RESET, the processor preserves the contents of its caches, the floating-point state, the SMM base, the MMX state, MSRs, and the CD and NW bits of the CR0 register.

The edge-sensitive interrupts FLUSH# and SMI# are sampled and preserved during the INIT process and are handled accordingly after the initialization is complete. However, the processor resets any pending NMI interrupt upon sampling INIT asserted.

INIT can be used as an accelerator for 80286 code that requires a reset to exit from Protected mode back to Real mode.

AMD-K6™ Processor Cache

The internal L1 cache of the AMD-K6 MMX enhanced processor consists of two separate caches—a 32-Kbyte instruction cache and a 32-Kbyte data cache. The instruction cache also incorporates a 20-Kbyte pre-decode cache in addition to a 64-entry TLB. The data cache utilizes a 128-entry TLB. The cache line is 32 bytes wide. Two adjacent cache lines are associated with each tag (a 64-byte sector with two 32-byte cache lines).

The AMD-K5 processor uses the Array Access Register (AAR), a MSR that allows for testing of the processor caches. The AMD-K6 processor does not contain these features. The AMD-K6 contains a built-in self-test (BIST) for all internal memories. However, cache information can be provided by utilizing the CPUID instruction. For more detailed information refer to the *AMD Processor Recognition Application Note*, order# 20734, located at http://www.amd.com.

Function 8000_0005h of the CPUID instruction returns processor cache information. Table 35 shows the information returned by the CPUID instruction when EAX = 8000_0005h.

Table 35. Data Returned by the CPUID Instruction

Register	Field Bits	Field Description
EBX	31–24	Data TLB—Associativity
	23–16	Data TLB—Number of entries
	15–8	Instruction TLB—Associativity
	7–0	Instruction TLB—Number of entries
ECX	31–24	L1 data cache—Size (Kbytes)
	23–16	L1 data cache—Associativity
	15–8	L1 data cache—Lines per tag
	7–0	L1 data cache—Line size (bytes)
EDX	31–24	L1 instruction cache—Size (Kbytes)
	23–16	L1 instruction cache—Associativity
	15–8	L1 instruction cache—Lines per tag
	7–0	L1 instruction cache—Line size (bytes)

Note:
Full associativity is indicated by a value of FFh.

AMD-K6™ Processor Test and Debug

The AMD-K6 MMX enhanced processor implements various test and debug modes to enable the functional and manufacturing testing of systems and boards that use the processor. In addition, the debug features of the processor allow designers to debug the instruction execution of software components. This section describes the following test and debug features:

- *Built-In Self-Test (BIST)*—The BIST, which is invoked after the falling transition of RESET, runs internal tests that exercise most on-chip RAM and ROM structures.
- *Tri-State Test Mode*—A test mode that causes the processor to float its output and bidirectional pins.
- *Boundary-Scan Test Access Port (TAP)*—The Joint Test Action Group (JTAG) test access function defined by the *IEEE Standard Test Access Port and Boundary-Scan Architecture (IEEE 1149.1-1990)* specification.

- *Level-One (L1) Cache Inhibit*—A feature that disables the processor's internal L1 instruction and data caches.
- *Debug Support*—Consists of all x86-compatible software debug features, including the debug extensions.

Built-In Self-Test (BIST)

Following the falling transition of RESET, the processor unconditionally runs its BIST. The internal resources tested during BIST include the following:

- L1 instruction and data caches
- Instruction and Data Translation Lookaside Buffers (TLBs)
- Microcode Read-Only Memory (ROM)
- Programmable Logic Arrays

The contents of the EAX general-purpose register after the completion of RESET indicate if the BIST was successful. If EAX contains 0000_0000h, then BIST was successful. If EAX is non-zero, the BIST failed. Following the completion of the BIST, the processor jumps to address FFFF_FFF0h to start instruction execution, regardless of the outcome of the BIST.

The BIST takes approximately 295,000 processor clocks to complete.

Tri-State Test Mode

The Tri-State Test mode causes the processor to float its output and bidirectional pins, which is useful for board-level manufacturing testing. In this mode, the processor is electrically isolated from other components on a system board, allowing automated test equipment (ATE) to test those components that drive the same signals as those the processor floats.

If the FLUSH# signal is sampled Low during the falling transition of RESET, the processor enters the Tri-State Test mode. See the *AMD-K6™ MMX™ Enhanced Processor Data Sheet*, order# 20695, for more information.

Boundary-Scan Test Access Port (TAP)

The boundary-scan Test Access Port (TAP) is an IEEE standard that defines synchronous scanning test methods for complex logic circuits, such as boards containing a processor. The AMD-K6 processor supports the TAP standard defined in the *IEEE Standard Test Access Port and Boundary-Scan Architecture (IEEE 1149.1-1990)* specification.

Boundary scan testing uses a shift register consisting of the serial interconnection of boundary-scan cells that correspond to each I/O buffer of the processor. This non-inverting register chain, called a Boundary Scan Register (BSR), is used to capture the state of every processor pin and to drive every processor output and bidirectional pin to a known state.

Each BSR of every component on a board that implements the boundary-scan architecture can be serially interconnected to enable component interconnect testing.

TAP Registers

The AMD-K6 processor provides an Instruction Register (IR) and three Test Data Registers (TDR) to support the boundary-scan architecture. The IR and one of the TDRs—the Boundary-Scan Register (BSR)—consist of a shift register and an output register. The shift register is loaded in parallel in the Capture states (See the *IEEE Standard Test Access Port and Boundary-Scan Architecture (IEEE 1149.1-1990)* specification for more information). In addition, the shift register is loaded and shifted serially in the Shift states. The output register is loaded in parallel from its corresponding shift register in the Update states.

Instruction Register (IR). The IR is a 5-bit register, without parity, that determines which instruction to run and which test data register to select. When the TAP controller enters the Capture-IR state, the processor loads the following bits into the IR shift register:

- *01b*—Loaded into the two least significant bits, as specified by the IEEE 1149.1 standard
- *000b*—Loaded into the three most significant bits

Loading 00001b into the IR shift register during the Capture-IR state results in loading the SAMPLE/PRELOAD instruction.

For each entry into the Shift-IR state, the IR shift register is serially shifted by one bit toward the TDO pin. During the shift, the most significant bit of the IR shift register is loaded from the TDI pin.

The IR output register is loaded from the IR shift register in the Update-IR state, and the current instruction is defined by the IR output register. See "TAP Instructions" on page 111 for a list and definition of the instructions supported by the AMD-K6.

Boundary Scan Register (BSR). The BSR is a Test Data Register consisting of the interconnection of 152 boundary-scan cells. Each output and bidirectional pin of the processor requires a two-bit cell, where one bit corresponds to the pin and the other bit is the output enable for the pin. When a 0 is shifted into the enable bit of a cell, the corresponding pin is floated, and when a 1 is shifted into the enable bit, the pin is driven valid. Each input pin requires a one-bit cell that corresponds to the pin. The last cell of the BSR is reserved and does not correspond to any processor pin.

The total number of bits that comprise the BSR is 281. Table 36 on page 109 lists the order of these bits, where TDI is the input to bit 280, and TDO is driven from the output of bit 0. The entries listed as *pin*_E (where *pin* is an output or bidirectional signal) are the enable bits.

If the BSR is the register selected by the current instruction and the TAP controller is in the Capture-DR state, the processor loads the BSR shift register as follows:

- If the current instruction is SAMPLE/PRELOAD, then the current state of each input, output, and bidirectional pin is loaded. A bidirectional pin is treated as an output if its enable bit equals 1, and it is treated as an input if its enable bit equals 0.
- If the current instruction is EXTEST, then the current state of each input pin is loaded. A bidirectional pin is treated as an input, regardless of the state of its enable.

While in the Shift-DR state, the BSR shift register is serially shifted toward the TDO pin. During the shift, bit 280 of the BSR is loaded from the TDI pin.

The BSR output register is loaded with the contents of the BSR shift register in the Update-DR state. If the current instruction is EXTEST, the processor's output pins, as well as those bidirectional pins that are enabled as outputs, are driven with their corresponding values from the BSR output register.

Table 36. Boundary Scan Register Bit Definitions

Bit	Pin/Enable	Bit	Pin/Enable	Bit	Pin/Enable	Bit	Pin/Enable	Bit	Pin/Enable	Bit	Pin/Enable
280	D35_E	247	D21	214	D4_E	181	A3	148	A20	115	A16
279	D35	246	D18_E	213	D4	180	A31_E	147	A13_E	114	FERR_E
278	D29_E	245	D18	212	DP0_E	179	A31	146	A13	113	FERR#
277	D29	244	D19_E	211	DP0	178	A21_E	145	DP7_E	112	HIT_E
276	D33_E	243	D19	210	HOLD	177	A21	144	DP7	111	HIT#
275	D33	242	D16_E	209	BOFF#	176	A30_E	143	BE6_E	110	BE7_E
274	D27_E	241	D16	208	AHOLD	175	A30	142	BE6	109	BE7
273	D27	240	D17_E	207	STPCLK#	174	A7_E	141	A12_E	108	NA#
272	DP3_E	239	D17	206	INIT	173	A7	140	A12	107	ADSC_E
271	DP3	238	D15_E	205	IGNNE#	172	A24_E	139	CLK	106	ADSC#
270	D25_E	237	D15	204	BF1	171	A24	138	BE4_E	105	BE5_E
269	D25	236	DP1_E	203	BF2	170	A18_E	137	BE4	104	BE5#
268	D0_E	235	DP1	202	RESET	169	A18	136	A10_E	103	WB/WT#
267	D0	234	D13_E	201	BF0	168	A5_E	135	A10	102	PWT_E
266	D30_E	233	D13	200	FLUSH#	167	A5	134	D63_E	101	PWT
265	D30	232	D6_E	199	INTR	166	A22_E	133	D63	100	BE3_E
264	DP2_E	231	D6	198	NMI	165	A22	132	BE2_E	99	BE3#
263	DP2	230	D14_E	197	SMI#	164	EADS#	131	BE2	98	BREQ_E
262	D2_E	229	D14	196	A25_E	163	A4_E	130	A15_E	97	BREQ
261	D2	228	D11_E	195	A25	162	A4	129	A15	96	PCD_E
260	D28_E	227	D11	194	A23_E	161	HITM_E	128	BRDY#	95	PCD
259	D28	226	D1_E	193	A23	160	HITM#	127	BE1_E	94	W_E
258	D24_E	225	D1	192	A26_E	159	A9_E	126	BE1	93	W/R#
257	D24	224	D12_E	191	A26	158	A9	125	A14_E	92	SMIACT_E
256	D26_E	223	D12	190	A29_E	157	SCYC_E	124	A14	91	SMIACT#
255	D26	222	D10_E	189	A29	156	SCYC	123	BRDYC#	90	EWBE#
254	D22_E	221	D10	188	A28_E	155	A8_E	122	BE0_E	89	DC_E
253	D22	220	D7_E	187	A28	154	A8	121	BE0	88	D/C#
252	D23_E	219	D7	186	A27_E	153	A19_E	120	A17_E	87	APCHK_E
251	D23	218	D8_E	185	A27	152	A19	119	A17	86	APCHK#
250	D20_E	217	D8	184	A11_E	151	A6_E	118	KEN#	85	CACHE_E
249	D20	216	D9_E	183	A11	150	A6	117	A20M#	84	CACHE#
248	D21_E	215	D9	182	A3_E	149	A20_E	116	A16_E	83	ADS_E

Table 36. Boundary Scan Register Bit Definitions (continued)

Bit	Pin/Enable	Bit	Pin/Enable	Bit	Pin/Enable	Bit	Pin/Enable	Bit	Pin/Enable	Bit	Pin/Enable
82	ADS#	68	DP6_E	54	D53_E	40	D43_E	26	D38_E	12	D3_E
81	AP_E	67	DP6	53	D53	39	D43	25	D38	11	D3
80	AP	66	D54_E	52	D47_E	38	D62_E	24	D58_E	10	D39_E
79	INV	65	D54	51	D47	37	D62	23	D58	9	D39
78	HLDA_E	64	D50_E	50	D59_E	36	D49_E	22	D42_E	8	D32_E
77	HLDA	63	D50	49	D59	35	D49	21	D42	7	D32
76	PCHK_E	62	D56_E	48	D51_E	34	DP4_E	20	D36_E	6	D5_E
75	PCHK#	61	D56	47	D51	33	DP4	19	D36	5	D5
74	LOCK_E	60	D55_E	46	D45_E	32	D46_E	18	D60_E	4	D37_E
73	LOCK#	59	D55	45	D45	31	D46	17	D60	3	D37
72	M_E	58	D48_E	44	D61_E	30	D41_E	16	D40_E	2	D31_E
71	M/IO#	57	D48	43	D61	29	D41	15	D40	1	D31
70	D52_E	56	D57_E	42	DP5_E	28	D44_E	14	D34_E	0	Reserved
69	D52	55	D57	41	DP5	27	D44	13	D34		

Device Identification Register (DIR). The DIR is a 32-bit Test Data Register selected during the execution of the IDCODE instruction. The fields of the DIR and their values are shown in Table 37 and are defined as follows:

- *Version Code*—This 4-bit field is incremented by AMD manufacturing for each major revision of silicon.
- *Part Number*—This 16-bit field identifies the specific processor model.
- *Manufacturer*—This 11-bit field identifies the manufacturer of the component (AMD).
- *LSB*—The least significant bit (LSB) of the DIR is always set to 1, as specified by the IEEE 1149.1 standard.

Table 37. AMD-K6™ Processor Device Identification Register

Version Code (Bits 31–28)	Part Number (Bits 27–12)	Manufacturer (Bits 11–1)	LSB (Bit 0)
Xh	0560h	00000000001b	1b

Bypass Register (BR). The BR is a Test Data Register consisting of a 1-bit shift register that provides the shortest path between TDI and TDO. When the processor is not involved in a test

operation, the BR can be selected by an instruction to allow the transfer of test data through the processor without having to serially scan the test data through the BSR. This functionality preserves the state of the BSR and significantly reduces test time.

The BR register is selected by the BYPASS and HIGHZ instructions as well as by any instructions not supported by the AMD-K6.

TAP Instructions

The processor supports the three instructions required by the IEEE 1149.1 standard—EXTEST, SAMPLE/PRELOAD, and BYPASS—as well as two additional optional instructions—IDCODE and HIGHZ.

Table 38 shows the complete set of TAP instructions supported by the processor along with the 5-bit Instruction Register encoding and the register selected by each instruction.

Table 38. Supported TAP Instructions

Instruction	Encoding	Register	Description
EXTEST[1]	00000b	BSR	Sample inputs and drive outputs
SAMPLE / PRELOAD	00001b	BSR	Sample inputs and outputs, then load the BSR
IDCODE	00010b	DIR	Read DIR
HIGHZ	00011b	BR	Float outputs and bidirectional pins
BYPASS[2]	00100b–11110b	BR	Undefined instruction, execute the BYPASS instruction
BYPASS[3]	11111b	BR	Connect TDI to TDO to bypass the BSR

Notes:

1. *Following the execution of the EXTEST instruction, the processor must be reset in order to return to normal, non-test operation.*
2. *These instruction encodings are undefined on the AMD-K6 processor and default to the BYPASS instruction.*
3. *Because the TDI input contains an internal pullup, the BYPASS instruction is executed if the TDI input is not connected or open during an instruction scan operation. The BYPASS instruction does not affect the normal operational state of the processor.*

EXTEST. When the EXTEST instruction is executed, the processor loads the BSR shift register with the current state of the input and bidirectional pins in the Capture-DR state and drives the output and bidirectional pins with the corresponding values from the BSR output register in the Update-DR state.

SAMPLE/PRELOAD. The SAMPLE/PRELOAD instruction performs two functions. These functions are as follows:

- During the Capture-DR state, the processor loads the BSR shift register with the current state of every input, output, and bidirectional pin.
- During the Update-DR state, the BSR output register is loaded from the BSR shift register in preparation for the next EXTEST instruction.

The SAMPLE/PRELOAD instruction does not affect the normal operational state of the processor.

BYPASS. The BYPASS instruction selects the BR register, which reduces the boundary-scan length through the processor from 281 to one (TDI to BR to TDO). The BYPASS instruction does not affect the normal operational state of the processor.

IDCODE. The IDCODE instruction selects the DIR register, allowing the device identification code to be shifted out of the processor. This instruction is loaded into the IR when the TAP controller is reset. The IDCODE instruction does not affect the normal operational state of the processor.

HIGHZ. The HIGHZ instruction forces all output and bidirectional pins to be floated. During this instruction, the BR is selected and the normal operational state of the processor is not affected.

L1 Cache Inhibit

Purpose

The AMD-K6 MMX enhanced processor provides a means for inhibiting the normal operation of its L1 instruction and data caches while still supporting an external level-two (L2) cache. This capability allows system designers to disable the L1 cache during the testing and debug of an L2 cache.

If the Cache Inhibit bit (bit 3) of Test Register 12 (TR12) is set to 0, the processor's L1 cache is enabled and operates as described in the Cache Organization section of the *AMD-K6™ MMX™ Enhanced Processor Data Sheet*, order# 20695. If the Cache Inhibit bit is set to 1, the L1 cache is disabled and no new cache lines are allocated. Even though new allocations do not occur, valid L1 cache lines remain valid and are read by the

processor when a requested address hits a cache line. In addition, the processor continues to support inquire cycles initiated by the system logic, including the execution of writeback cycles when a modified cache line is hit.

While the L1 is inhibited, the processor continues to drive the PCD output signal appropriately, which system logic can use to control external L2 caching.

In order to completely disable the L1 cache so no valid lines exist in the cache, the Cache Inhibit bit must be set to 1 and the cache must be flushed in one of the following ways:

- By asserting the FLUSH# input signal
- By executing the WBINVD instruction
- By executing the INVD instruction (modified cache lines are not written back to memory)

Debug

The AMD-K6 processor implements the standard x86 debug functions, registers, and exceptions. In addition, the processor supports the I/O breakpoint debug extension. The debug feature assists programmers and system designers during software execution tracing by generating exceptions when one or more events occur during processor execution. The exception handler, or debugger, can be written to perform various tasks, such as displaying the conditions that caused the breakpoint to occur, displaying and modifying register or memory contents, or single-stepping through program execution.

The following sections describe the debug registers and the various types of breakpoints and exceptions supported by the processor.

For more details on the register definitions see the Test and Debug chapter in the *AMD-K6™ MMX™ Enhanced Processor Data Sheet*, order# 20695.

Debug Registers

Figures 24 through 27 show the 32-bit debug registers supported by the processor. Table 39 provides LEN and RW information for DR7 as displayed in Figure 24.

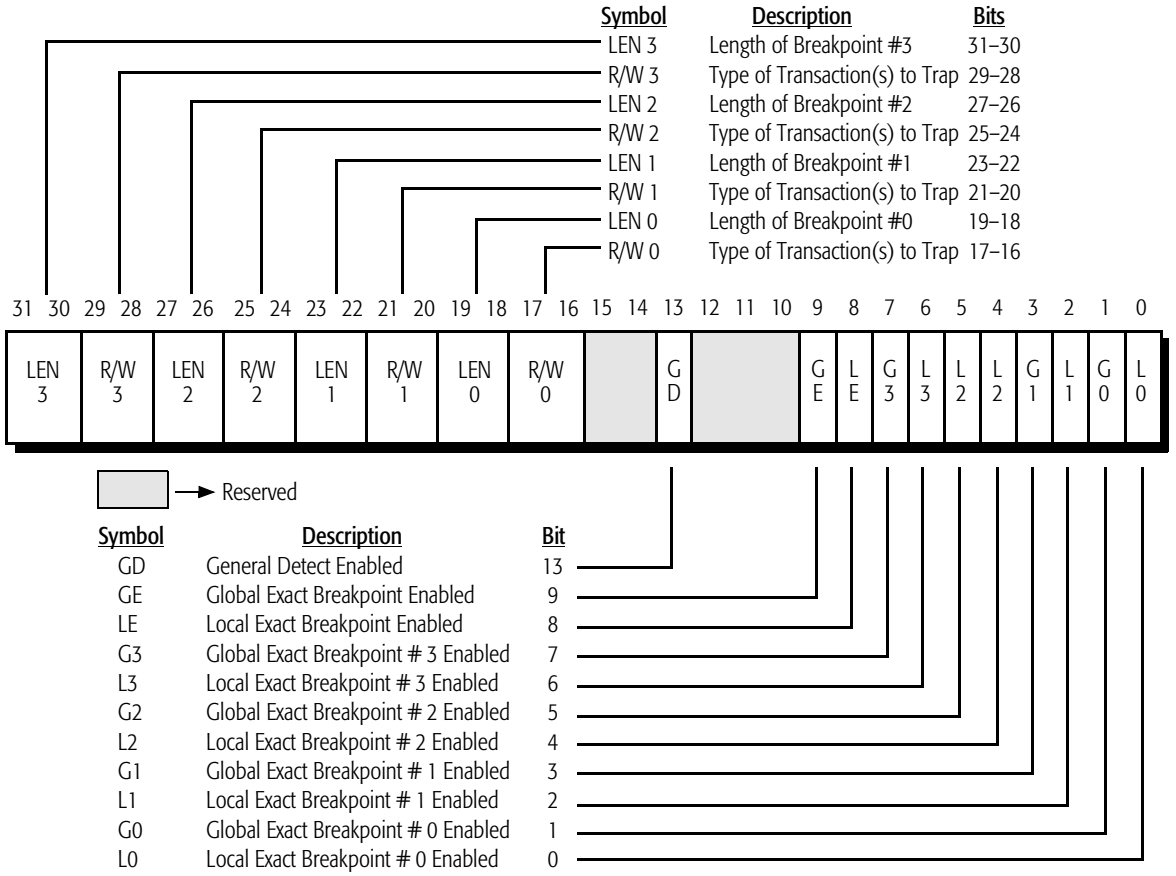


Figure 24. Debug Register DR7

Table 39. DR7 LEN and RW Definitions

LEN Bits[1]	RW Bits	Breakpoint
00b	00b[2]	Instruction Execution
00b	01b	One-byte Data Write
01b		Two-byte Data Write
11b		Four-byte Data Write
00b	10b[3]	One-byte I/O Read or Write
01b		Two-byte I/O Read or Write
11b		Four-byte I/O Read or Write
00b	11b	One-byte Data Read or Write
01b		Two-byte Data Read or Write
11b		Four-byte Data Read or Write

Notes:
1. *LEN bits equal to 10b is undefined.*
2. *When RW equals 00b, LEN must be equal to 00b.*
3. *When RW equals 10b, debugging extensions (DE) must be enabled (bit 3 of CR4 must be set to 1). If DE is set to 0, RW equal to 10b is undefined.*

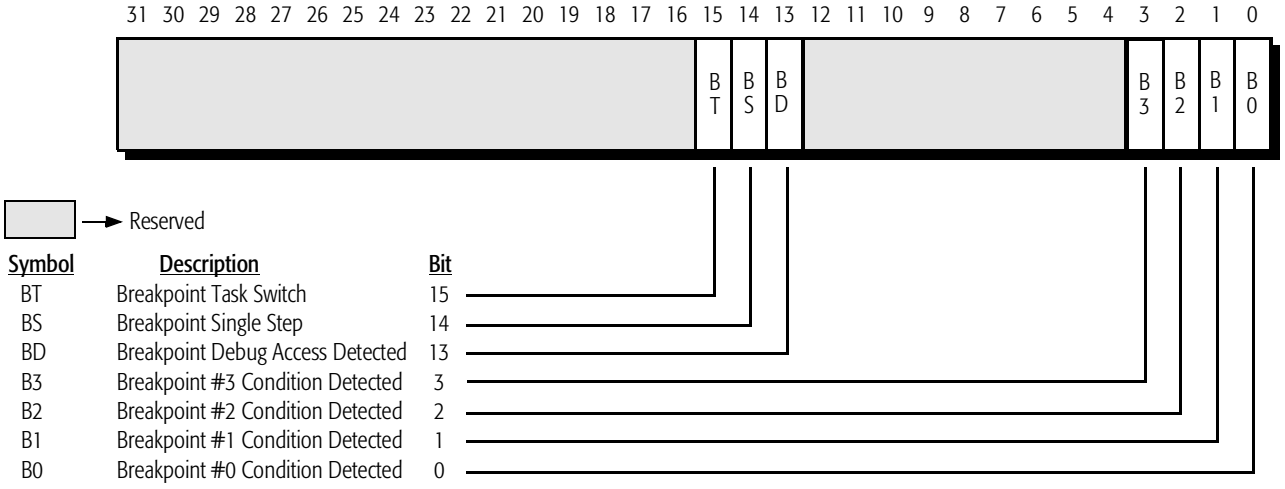


Figure 25. Debug Register DR6

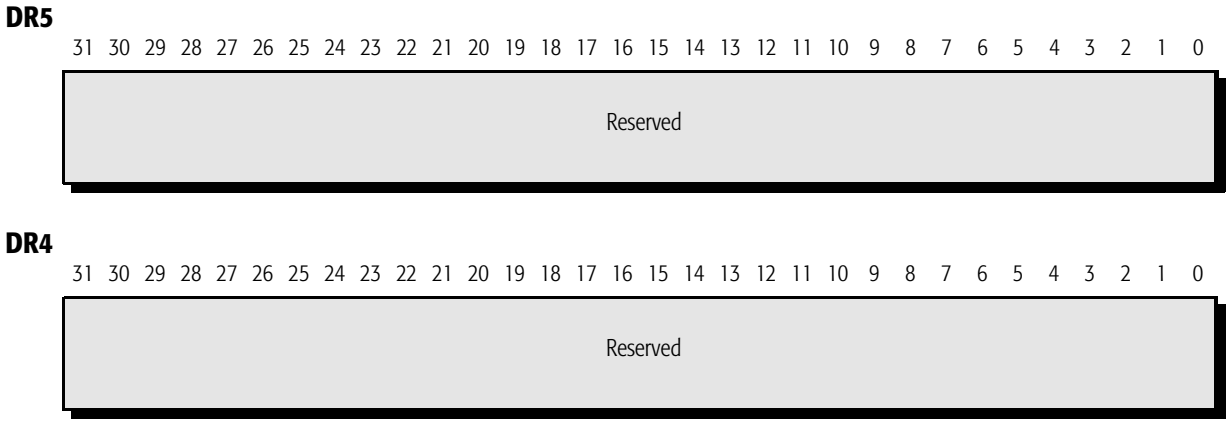


Figure 26. Debug Registers DR5 and DR4

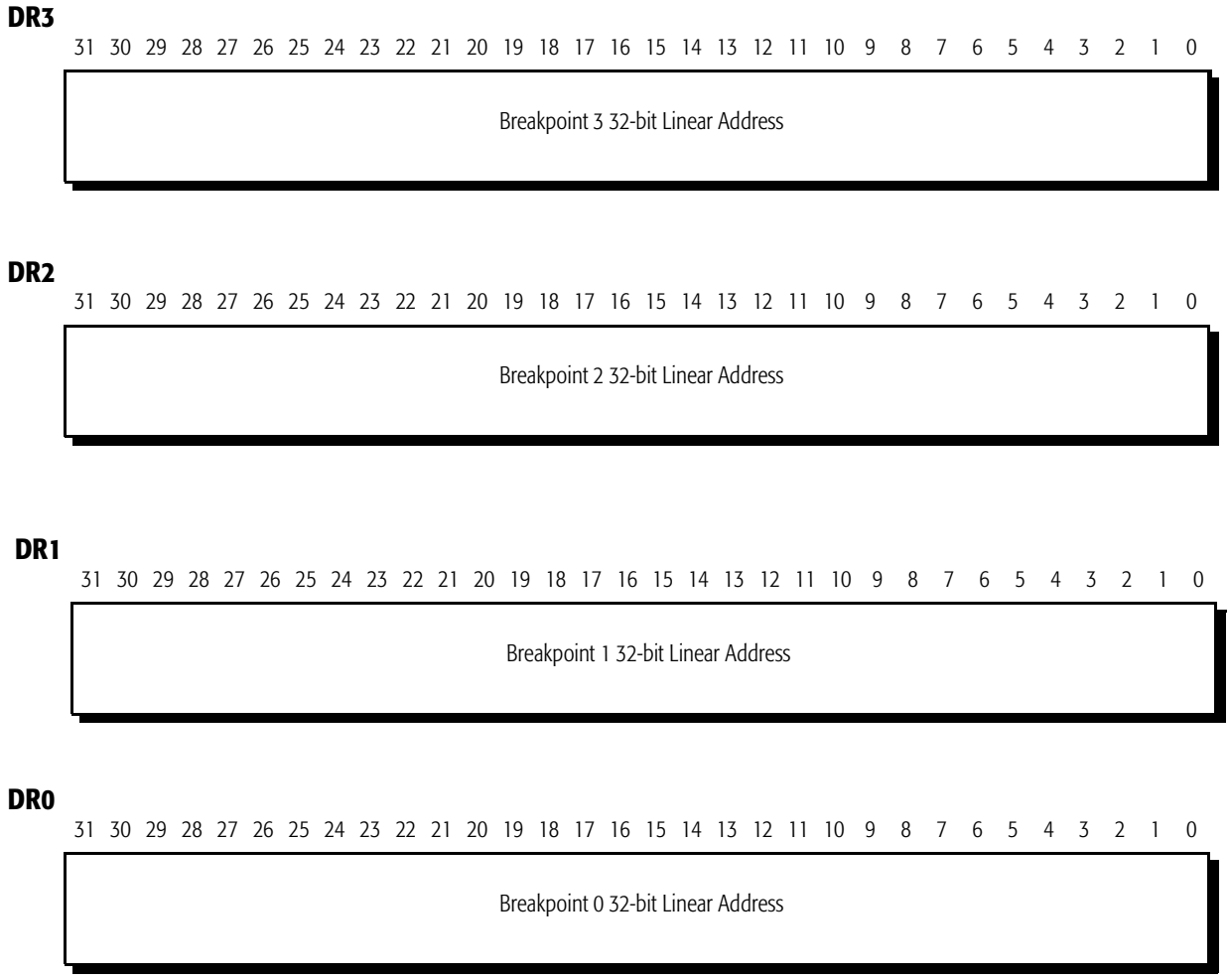


Figure 27. Debug Registers DR3, DR2, DR1, and DR0

AMD-K6™ Processor x86 Architecture Extensions

This section documents the extensions that have been added to the AMD-K6 MMX enhanced processor.

Model-Specific Registers (MSR)

The AMD-K6 processor provides the following six MSRs. The contents of ECX selects the MSR to be addressed by the RDMSR and WRMSR instruction.

- Machine-Check Address Register (MCAR)—ECX = 00h
- Machine-Check Type Register (MCTR)—ECX = 01h
- Test Register 12 (TR12)—ECX = 0Eh
- Time Stamp Counter (TSC)—ECX = 10h
- Extended Feature Enable Register (EFER)—ECX = C000_0080h
- SYSCALL Target Address Register (STAR)—ECX = C000_0081h
- Write Handling Control Register (WHCR)—ECX = C000_0082h

These six MSRs are read and written by the RDMSR and WRMSR instructions. (The TSC can also be read by the RDTSC instruction.) The target register for the RDMSR and WRMSR instructions is addressed by the contents of ECX. The only values allowed in ECX by the AMD-K6 processor are 00h, 01h, 0Eh, 10h, C000_0080h, C000_0081h, and C000_0082h for the MCAR, MCTR, TR12, TSC, EFER, STAR and WHCR registers respectively. The usage of any other reserved value in ECX results in a general protection exception.

Machine-Check Address Register (MCAR)

See Figure 20 on page 80 and “Machine Check Exception” on page 122.

Machine-Check Type Register (MCTR)

See Figure 21 on page 81 and “Machine Check Exception” on page 122.

Test Register 12 (TR12)

The AMD-K6 processor also provides the 64-bit Test Register 12 (TR12), but only the function of the Cache Inhibit (CI) bit (bit 3 of TR12) is supported. All other bits in TR12 have no effect on the processor's operation. The I/O Trap Restart function (bit 9 of TR12) is always enabled on the AMD-K6.

Time Stamp Counter (TSC)

See "Time Stamp Counter (TSC)" on page 81.

Extended Feature Enable Register (EFER)

The Extended Feature Enable Register (EFER) contains the control bits that enable the extended features of the AMD-K6 processor. Figure 28 shows the format of the EFER register, and Table 40 defines the function of each bit of the EFER register. The EFER register is MSR C000_0080h.

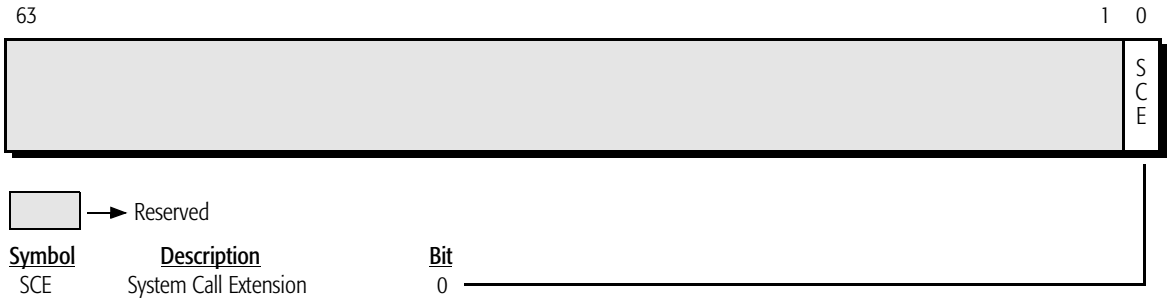


Figure 28. Extended Feature Enable Register (EFER)

Table 40. Extended Feature Enable Register (EFER) Definition

Bit	Description	R/W	Function
63–1	Reserved	R	Writing a 1 to any reserved bit causes a general protection fault to occur. All reserved bits are always read as 0.
0	System Call Extension (SCE)	R/W	SCE must be set to 1 to enable the usage of the SYSCALL and SYSRET instructions.

SYSCALL Target Address Register (STAR)

The SYSCALL Target Address Register (STAR) contains the target EIP address used by the SYSCALL instruction, and contains the 16-bit selector base used by the SYSCALL and SYSRET instructions. Figure 29 shows the format of the STAR register, and Table 41 defines the fields of the STAR register. The STAR register is MSR C000_0081h.

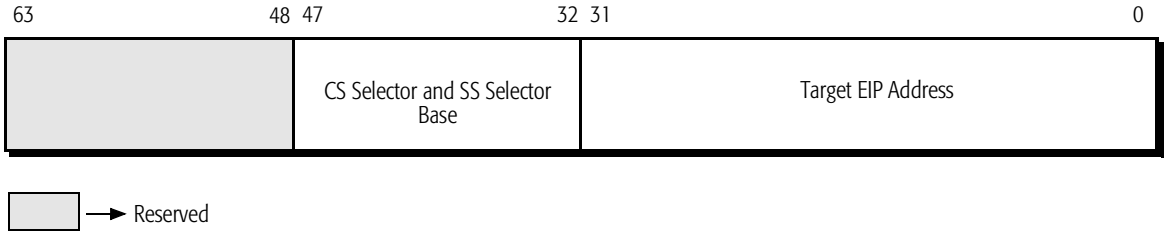


Figure 29. SYSCALL Target Address Register (STAR)

Table 41. SYSCALL Target Address Register (STAR) Definition

Bit	Description	R/W	Function
31–0	Target EIP Address	R/W	This address is copied into the EIP and points to the new starting address.
47–32	CS and SS Selector Base	R/W	During the SYSCALL instruction, this field is copied into the CS register and the contents of this field, plus 8, are copied into the SS register. During the SYSRET instruction, this field, plus 16, is copied into the SS register, and bits 1–0 of the SS register are set to 11b.
63–48	Reserved	R	Writing a 1 to any reserved bit causes a general protection fault to occur. All reserved bits are always read as 0.

Write Handling Control Register (WHCR)

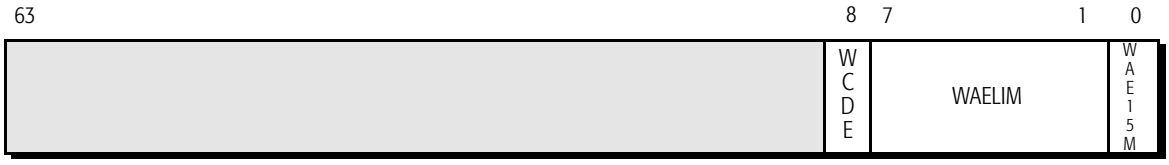
The AMD-K6 processor contains a split level-one (L1) 64-Kbyte writeback cache organized as a separate 32-Kbyte instruction cache and a 32-Kbyte data cache with two-way set associativity. The cache line size is 32 bytes, and lines are read from memory using an efficient pipelined burst read cycle. Further performance gains are achieved by the implementation of a write allocation scheme.

For more information about write allocate, see the *Implementation of Write Allocate in the K86™ Processors Application Note*, order# 21326.

Write allocate, if enabled, occurs when the processor has a pending memory write cycle to a cacheable line and the line does not currently reside in the L1 cache. In this case, the processor performs a burst read cycle to fetch the cache line addressed by the pending write cycle. The data associated with the pending write cycle is merged with the recently-allocated data-cache line and stored in the processor's L1 data cache. The

final MESI state of the cache line depends on the state of the WB/WT# and PWT signals during the burst read cycle and the subsequent cache write hit.

Write Handling Control Register (WHCR). The Write Handling Control Register (WHCR) is an MSR that contains three fields—the WCDE bit, the Write Allocate Enable Limit (WAELIM) field, and the Write Allocate Enable 15-to-16-Mbyte (WAE15M) bit (See Figure 30).



Symbol	Description	Bits
WCDE	Write Cacheability Detection Enable	8
WAELIM	Write Allocate Enable Limit	7–1
WAE15M	Write Allocate Enable 15-to-16-Mbyte	0

***Note**: Hardware RESET initializes this MSR to all zeros.*

Figure 30. Write Handling Control Register (WHCR)—MSR C000_0082h

Write Cacheability Detection Enable. When the Write Cacheability Detection Enable (WCDE) bit (bit 8) of the Write Handling Control Register (WHCR) MSR is set to 1, this write allocate mechanism is enabled. For more details on the Write Cacheability Detection Mechanism, see the Cache Organization chapter in the *AMD-K6™ MMX™ Enhanced Processor Data Sheet*, order# 20695.

If the address is cacheable, support of the Write Cacheability Detection mechanism requires the system logic to assert KEN# during a write cycle. Some chipsets assert KEN# during a write cycle and some chipsets do not assert KEN# during a write cycle. (Triton chipsets eventually generate a correct value for KEN#, but not during the sample point. Therefore do not enable WCDE in systems that use the Triton chipset.) If Write Cacheability Detection is enabled, KEN# is sampled during write cycles in the same manner it is sampled during read cycles (KEN# is sampled on the clock edge on which the first BRDY# or NA# of a cycle is sampled asserted). Future chipsets may take advantage of this mechanism, but currently AMD recommends setting this bit to zero (disabled).

Write Allocate Enable Limit. The WAELIM field is 7 bits wide. This field, multiplied by 4 Mbytes, defines an upper memory limit. Any pending write cycle that addresses memory below this limit causes the processor to perform a write allocate. Write allocate is disabled for memory accesses at and above this limit unless the processor determines a pending write cycle is cacheable by means of one of the other Write Cacheability Detection mechanisms. The maximum value of this limit is $((2^7-1) \cdot 4$ Mbytes) = 508 Mbytes. When all the bits in this field are set to 0, all memory is above this limit and the write allocate mechanism is disabled.

Write Allocate Enable 15-to-16-Mbyte. The WAE15M bit is used to enable write allocations for the memory write cycles that address the 1 Mbyte of memory between 15 Mbytes and 16 Mbytes. This bit must be set to 0 to prevent write allocates in this memory area. This sub-mechanism of the WAELIM provides a memory hole to prevent write allocates. This memory hole is provided to account for a small number of uncommon memory-mapped I/O adapters that use this particular memory address space. If the system contains one of these peripherals, the bit should be set to 0. The WAE15M bit is ignored if the value in the WAELIM field is set to less than 16 Mbytes.

By definition, write allocations in the AMD-K6 processor are never performed in the memory area between 640 Kbytes and 1 Mbyte. It is not safe to perform write allocations between 640 Kbytes and 1 Mbyte (000A_0000h to 000F_FFFFh) because it is considered a non-cacheable region of memory.

See the Software Environment section of the *AMD-K6™ MMX™ Enhanced Processor Data Sheet*, order# 20695, for more information.

Note: *The BIOS should enable the write allocate mechanisms only after performing any memory sizing and typing algorithms.*

Machine Check Exception

The AMD-K6 processor does not support the generation of a machine check exception.

The processor provides a 64-bit Machine Check Address Register (MCAR) and a 64-bit Machine Check Type Register (MCTR), but because the processor does not support machine check exceptions, the contents of the MCAR and MCTR are only affected by the WRMSR instruction and by RESET being sampled asserted (where all bits in each register are reset to 0).

The processor also provides the Machine Check Exception (MCE) bit in Control Register 4 (CR4, bit 6) as a read-write bit. However, the state of this bit has no effect on the operation of the processor.

The processor does not provide the BUSCHK and PEN signals provided by Pentium.

New AMD-K6™ Processor Instructions

This section documents and explains the new instructions added to the AMD-K6 processor above and beyond the AMD-K5 processor.

- SYSCALL
- SYSRET
- MMX™ Instructions—57 new instructions for multimedia software. See “MMX™ Instructions” on page 127.

System Call Extensions

Setting bit 0 (SCE) in the Extended Feature Enable Register (See “Extended Feature Enable Register (EFER)” on page 118) enables the system call extensions. The system call extensions consist of two new instructions, SYSCALL and SYSRET, that allow OS vendors fast protection-level switching to and from CPL0.

SYSCALL

mnemonic	*opcode*	*description*
SYSCALL	0F05h	Call operating system

Privilege: none
Registers Affected: ECX, EIP, CS, SS
Flags Affected: IF, VM
Machine State Affected: CPL, CS (base, limit, attr), SS (base, limit, attr)
Exceptions Generated:

Exception	Real	Virtual 8086	Protected	Description
Invalid opcode (6)	X	X	X	The System Call Extension bit (SCE) of the Extended Feature Enable Register (EFER) is set to 0. (The EFER register is MSR C000_0080h.)

The SYSCALL instruction provides a fast method for transferring control to a fixed entry point in an operating system.

The EIP register is copied into the ECX register. Bits 31–0 of the 64-bit SYSCALL Target Address Register (See "SYSCALL Target Address Register (STAR)" on page 118) are copied into the EIP register. (The STAR register is Model-Specific Register C000_0081h.)

The IF and VM flags are set to 0 to disable interrupts and force the processor out of Virtual-8086 mode.

New selectors are loaded with no checking performed as follows:

- Bits 47–32 of the STAR register are copied into the CS register
- (Bits 47–32 of the STAR register) + 8 are copied into the SS register

The CS and SS registers must not be modified by the operating system between the execution of the SYSCALL instruction and its corresponding SYSRET instruction.

The processor's CPL is set to 0 regardless of the value of bits 33–32 of the STAR register. There are no permission checks of the CPL, Real mode, or Virtual-8086 mode.

The following descriptors are loaded to specify fixed 4-Gbyte flat segments as follows:

- The CS_base and the SS_base are both set to zero
- The CS_limit and the SS_limit are both set to 4-Gbyte
- The CS segment attributes are set to Read-only
- The SS segment attributes are set to Read-Write and Expand-Up

The operating system must set the STAR register and the appropriate descriptor table entries to reflect the values loaded by the processor during the SYSCALL instruction.

Related Instructions See the SYSRET instruction.

SYSRET

mnemonic	*opcode*	*description*
SYSRET	0F07h	Return from operating system

Privilege: CPL = 0
Registers Affected: EIP, CS, SS
Flags Affected: IF
Machine State Affected: CPL, CS (base, limit, attr)
Exceptions Generated:

Exception	Real	Virtual 8086	Protected	Description
Invalid opcode (6)	X	X	X	The System Call Extension bit (SCE) of the Extended Feature Enable Register (EFER) is set to 0. (The EFER register is MSR C000_0080h.)
General protection (13)	X	X	X	The CPL is not equal to 0.

The SYSRET instruction is the return instruction used in conjunction with the SYSCALL instruction to provide fast entry/exit to an operating system.

The ECX register, which points to the next sequential instruction after the corresponding SYSCALL instruction, is copied into the EIP register.

The IF flag is set to 1 in order to enable interrupts.

New selectors are loaded without any checking as follows:

- Bits 47–32 of the STAR register are copied into the CS register
- Bits 1–0 of the CS register are set to 11b (CPL of 3), regardless of the value of bits 33–32 of the STAR register
- (Bits 47–32 of the STAR register) + 16 are copied into the SS register
- Bits 1–0 of the SS register are set to 11b (RPL of 3), regardless of the value of bits 33–32 of the STAR register

The CS and SS registers must not be modified by the operating system between the execution of the SYSCALL instruction and its corresponding SYSRET instruction.

If the CPL is not equal to 0 when the SYSRET instruction is executed, a general protection fault exception is generated with an error code of 0.

A new descriptor is loaded for CS to specify a fixed 4-Gbyte flat segment as follows:

- The CS_base is set to zero
- The CS_limit is set to 4-Gbyte
- The CS segment attributes are set to Read-only

The operating system must set the STAR register and the appropriate descriptor table entries to reflect the values loaded by the processor during the SYSCALL instruction.

Related Instructions See the SYSCALL instruction.

MMX™ Instructions

The AMD-K6 MMX enhanced processor implements the complete MMX instruction set. For a detailed description refer to *AMD-K6™ MMX™ Enhanced Processor Multimedia Technology*, order# 20726, located at http://www.amd.com. Table 42 lists the MMX instructions.

Table 42. MMX™ Instructions and Descriptions

Instruction	Description
EMMS	Empty MMX State
MOVD	Move 32 Bits
MOVQ	Move 64 Bits
PACKSSWB /PACKSSDW	Pack with Signed Saturation
PACKUSWB	Pack with Unsigned Saturation
PADDB/PADDW/PADDD	Packed Add
PADDSB/PADDSW	Packed Add with Saturation
PADDUSB/PADDUSW	Packed Add Unsigned with Saturation
PAND	Bitwise Logical And
PANDN	Bitwise Logical And Not
PXOR	Bitwise Logical Exclusive OR
POR	Bitwise Logical OR
PCMPEQB/PCMPEQW/PCMPEQD	Packed Compare for Equal
PCMPGTB/PCMPGTW/PCMPGTD	Packed Compare for Greater Than
PMADDWD	Packed Multiply and Add
PMULLW	Packed Multiply Low
PMULHW	Packed Multiply High
PSLLW/PSLLD/PSLLQ	Packed Shift Left Logical
PSRAW/PSRAD	Packed Shift Right Arithmetic
PSRLW/PSRLD/PSRLQ	Packed Shift Right Logical
PSUBB/PSUBW/PSUBD	Packed Subtract
PSUBSB/PSUBSW	Packed Subtract with Saturation
PSUBUSB/PSUBSW	Packed Subtract Unsigned with Saturation
PUNPCKHBW/PUNPCKHWD/PUNPCKHDQ	Unpack High Packed Data
PUNPCKLBW/PUNPCKLWD/PUNPCKLDQ	Unpack Low Packed Data

Index

Numerics

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W