MIPS64™ 20Kc™ Processor Core User's Manual

Document Number: MD00126
Revision 01.10
Sept 28, 2002

MIPS Technologies, Inc.
1225 Charleston Road
Mountain View, CA 94043-1353

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Copyright © 2002 MIPS Technologies, Inc. All rights reserved.

Unpublished rights (if any) are reserved under the Copyright Laws of the United States of America.

If this document is provided in source format (i.e., in a modifiable form such as in FrameMaker or Microsoft Word format), then its use and distribution is subject to a written agreement with MIPS Technologies, Inc. ("MIPS Technologies"). UNDER NO CIRCUMSTANCES MAY A DOCUMENT PROVIDED IN SOURCE FORMAT BE DISTRIBUTED TO A THIRD PARTY WITHOUT THE EXPRESS WRITTEN CONSENT OF MIPS TECHNOLOGIES.

This document contains information that is proprietary to MIPS Technologies. Any copying, reproducing, modifying, or use of this information (in whole or in part) which is not expressly permitted in writing by MIPS Technologies or a contractually-authorized third party is strictly prohibited. At a minimum, this information is protected under unfair competition and copyright laws. Violations thereof may result in criminal penalties and fines.

MIPS Technologies or any contractually-authorized third party reserves the right to change the information contained in this document to improve function, design or otherwise. MIPS Technologies does not assume any liability arising out of the application or use of this information, or of any error of omission in such information. Any warranties, whether express, statutory, implied or otherwise, including but not limited to the implied warranties of merchantability or fitness for a particular purpose, are excluded. Any license under patent rights or any other intellectual property rights owned by MIPS Technologies or third parties shall be conveyed by MIPS Technologies or any contractually-authorized third party in a separate license agreement between the parties.

The information contained in this document shall not be exported or transferred for the purpose of reexporting in violation of any U.S. or non-U.S. regulation, treaty, Executive Order, law, statute, amendment or supplement thereto.

The information contained in this document constitutes one or more of the following: commercial computer software, commercial computer software documentation or other commercial items. If the user of this information, or any related documentation of any kind, including related technical data or manuals, is an agency, department, or other entity of the United States government (“Government”), the use, duplication, reproduction, release, modification, disclosure, or transfer of this information, or any related documentation of any kind, is restricted in accordance with Federal Acquisition Regulation 12.212 for civilian agencies and Defense Federal Acquisition Regulation Supplement 227.7202 for military agencies. The use of this information by the Government is further restricted in accordance with the terms of the license agreement(s) and/or applicable contract terms and conditions covering this information from MIPS Technologies or any contractually-authorized third party.

MIPS®, R3000®, R4000®, R5000® and R10000® are among the registered trademarks of MIPS Technologies, Inc. in the United States and certain other countries, and MIPS16™, MIPS16e™, MIPS32™, MIPS64™, MIPS-3D™, MIPS-based™, MIPS I™, MIPS II™, MIPS III™, MIPS IV™, MIPS V™, MDMX™, MIPSsim™, MIPSsimCA™, MIPSsimIA™, QuickMIPS™, SmartMIPS™, MIPS Technologies logo, 4K™, 4Kc™, 4Km™, 4Kp™, 4KE™, 4KEc™, 4KEm™, 4KEp™, 4KS™, 4KSc™, M4K™, 5K™, 5Kc™, 5Kf™, 20K™, 20Kc™, 25Kf™, R4300™, ASMACRO™, ATLAS™, BusBridge™, CoreFPGA™, CoreLV™, EC™, JALGO™, MALTA™, MGB™, PDtrace™, SEAD™, SEAD-2™, SOC-it™, The Pipeline™, and YAMON™ are among the trademarks of MIPS Technologies, Inc.

All other trademarks referred to herein are the property of their respective owners.

Template: B1.06, Build with Conditional Tags: 2B 20KC

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table of Contents

Chapter 1 Introduction to the 20Kc Processor 1
1.1 Features 1
1.2 Architectural Overview 3
1.2.1 Instruction Fetch Unit 4
1.2.2 Instruction Dispatch Unit 4
1.2.3 Integer Execution Unit 4
1.2.4 Floating-Point Unit 5
1.2.5 Load/Store Unit 5
1.2.6 Memory Management Unit 5
1.2.7 Bus Interface Unit 5
1.2.8 EJTAG Unit 6
1.3 System Overview 6

Chapter 2 Instruction Set Overview 7
2.1 CPU Instruction Formats 7
2.2 Load and Store Instructions 8
2.2.1 Scheduling for Load Use Latencies 8
2.2.2 Defining Access Types 8
2.3 Computational Instructions 10
2.3.1 Multiply and Divide Instructions 10
2.4 Jump and Branch Instructions 10
2.4.1 Overview of Jump Instructions 10
2.4.2 Overview of Branch Instructions 11
2.5 Control Instructions 11
2.6 Coprocessor Instructions 11
2.7 Enhancements to the MIPS Architecture 11

Chapter 3 Pipeline 13
3.1 Pipeline Overview 13
3.2 Fetch Pipeline 14
3.2.1 F Stage: Instruction Fetch 15
3.2.2 V Stage: Validate 15
3.3 Dispatch Pipeline 15
3.3.1 D Stage: Instruction Decode 16
3.3.2 R Stage: Register File Read 16
3.4 Integer and Load/Store Pipelines 16
3.4.1 Integer Pipeline A 16
3.4.2 Integer Pipeline B 18
3.5 Floating-Point Pipeline 20
3.5.1 M Stage: FP Multiplier Array - First Pass 20
3.5.2 N Stage: FP Multiplier Array - Second Pass 21
3.5.3 I Stage: FP Multiply Completion 21
3.5.4 J Stage: FP Add 21
3.5.5 K Stage: FP Normalization 21
3.5.6 Z Stage: FPR Write 21
3.6 Instruction Latencies and Repeat Rates 21
3.7 Instruction Fetch Rules 24
3.8 Instruction Dispatch Rules 28
3.9 Dispatch of Privileged Instructions 32

Chapter 4 Memory Management 33

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

4.1 Operating Modes 33
4.2 Other Modes 34
4.2.1 64-bit Address Enable 34
4.2.2 64-bit Operations Enable 34
4.2.3 64-bit FPR Enable 34
4.3 Processor Mode Selection 35
4.4 Addressing Modes 35
4.5 Address Space 36
4.5.1 Access Control as a Function of Address and Operating Mode 39
4.5.2 Address Translation and Cache Coherency Attributes for the kseg0 and kseg1 Segments 40
4.5.3 Address Translation and Cache Coherency Attributes for the xkphys Segment 41
4.5.4 Address Translation for the kuseg Segment when $Status_{ERL} = 1$ 42
4.5.5 Special Behavior for the kseg3 Segment when $Debug_{DM} = 1$ 42
4.5.6 Special Behavior for Data References in User Mode with $Status_{UX} = 0$ 43
4.6 Address Segments 43
4.6.1 User Mode Segments 43
4.6.2 Supervisor Mode Segments 44
4.6.3 Kernel Mode Segments 46
4.6.4 Debug Mode 49
4.7 Virtual Address Translation 50
4.7.1 Page Size Support 50
4.7.2 Address Space Identifiers and Global Processes 51
4.7.3 Address Translation Mechanism 51
4.8 Translation Lookaside Buffers 52
4.8.1 20Kc TLB Organization 53
4.8.2 TLB Tag and Data Formats 55
4.9 TLB Instructions 57
4.9.1 Hits, Misses, and Multiple Matches 57
4.9.2 Page Sizes and Replacement Algorithm 58

Chapter 5 Exceptions and Interrupts 59
5.1 Exception Conditions 59
5.2 Exception Types 60
5.3 Exception Priority 60
5.4 Exception Vector Locations 62
5.5 General Exception Processing 63
5.6 Debug Exception Processing 64
5.7 Exceptions 64
5.7.1 Reset Exception 64
5.7.2 Soft Reset Exception 65
5.7.3 Debug Single Step Exception 66
5.7.4 Debug Interrupt Exception 66
5.7.5 Debug Instruction Break Exception 66
5.7.6 Non-Maskable Interrupt (NMI) Exception 66
5.7.7 Machine Check Exception 67
5.7.8 Bus Error Exception — Instruction Fetch or Data Access 67
5.7.9 Cache Error Exception 68
5.7.10 Interrupt Exception 69
5.7.11 Debug Software Breakpoint Exception 69
5.7.12 Watch Exception — Instruction Fetch or Data Access 69
5.7.13 Address Error Exception — Instruction Fetch/Data Access 70
5.7.14 TLB Refill and XTLB Refill Exceptions 71
5.7.15 TLB Invalid Exception — Instruction Fetch or Data Access 72
5.7.16 Execution Exception — System Call 72
5.7.17 Execution Exception — Breakpoint 72
5.7.18 Execution Exception — Reserved Instruction 73

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

5.7.19 Execution Exception — Coprocessor Unusable 74
5.7.20 Execution Exception — Integer Overflow 74
5.7.21 Execution Exception — Trap 75
5.7.22 Precise Debug Data Break Exception 75
5.7.23 Imprecise Debug Data Break Exception 75
5.7.24 TLB Modified Exception — Data Access 75
5.8 Exception Handling and Servicing Flowcharts 76
5.9 Interrupts 81

Chapter 6 Coprocessor Registers 83
6.1 CP0 Register Summary 83
6.2 CP0 Registers 84
6.2.1 Index Register (CP0 Register 0, Select 0) 85
6.2.2 Random Register (CP0 Register 1, Select 0) 86
6.2.3 EntryLo0, EntryLo1 (CP0 Registers 2 and 3, Select 0) 87
6.2.4 Context Register (CP0 Register 4, Select 0) 88
6.2.5 PageMask Register (CP0 Register 5, Select 0) 89
6.2.6 Wired Register (CP0 Register 6, Select 0) 89
6.2.7 BadVAddr Register (CP0 Register 8, Select 0) 90
6.2.8 Count Register (CP0 Register 9, Select 0) 91
6.2.9 EntryHi Register (CP0 Register 10, Select 0) 91
6.2.10 Compare Register (CP0 Register 11, Select 0) 92
6.2.11 Status Register (CP0 Register 12, Select 0) 92
6.2.12 Cause Register (CP0 Register 13, Select 0) 96
6.2.13 Exception Program Counter (CP0 Register 14, Select 0) 98
6.2.14 Processor Identification (CP0 Register 15, Select 0) 99
6.2.15 Config Register (CP0 Register 16, Select 0) 99
6.2.16 Config1 Register (CP0 Register 16, Select 1) 102
6.2.17 Load Linked Address (CP0 Register 17, Select 0) 105
6.2.18 WatchLo Register (CP0 Register 18) 105
6.2.19 WatchHi Register (CP0 Register 19) 106
6.2.20 XContext Register (CP0 Register 20) 107
6.2.21 Debug Register (CP0 Register 23) 108
6.2.22 Debug Exception Program Counter Register (CP0 Register 24) 111
6.2.23 Performance Counter Registers (CP0 Register 25, Selects 0, 1) 112
6.2.24 DErrCtl Register (CP0 Register 26, Select 0) 114
6.2.25 IErrCtl Register (CP0 Register 26, Select 1) 115
6.2.26 CacheErr Register (CP0 Register 27, Select 0) 115
6.2.27 ITagLo Register (CP0 Register 28, Select 0) 116
6.2.28 IDataLo Register (CP0 Register 28, Select 1) 117
6.2.29 DTagLo Register (CP0 Register 28, Select 2) 118
6.2.30 DDataLo Register (CP0 Register 28, Select 3) 118
6.2.31 ITagHi Register (CP0 Register 29, Select 0) 119
6.2.32 IDataHi Register (CP0 Register 29, Select 1) 119
6.2.33 DTagHi Register (CP0 Register 29, Select 2) 120
6.2.34 DDataHi Register (CP0 Register 29, Select 3) 120
6.2.35 ErrorEPC (CP0 Register 30, Select 0) 121
6.2.36 DESAVE Register (CP0 Register 31) 121
6.3 CP0 Hazards 121
6.4 CP1 Register Summary 122
6.5 CP1 Registers 122
6.5.1 Floating-Point Implementation Register (CP1 Register 0) 123
6.5.2 Floating-Point Condition Codes Register (CP1 Register 25) 124
6.5.3 Floating-Point Exceptions Register (CP1 Register 26) 125
6.5.4 Floating-Point Enables Register (CP1 Register 28) 125
6.5.5 Floating-Point Control and Status Register (CP1 Register 31) 126

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Chapter 7 Caches 129
7.1 Instruction Cache 129
7.1.1 Memory Management implications of the Virtual I-Cache 130
7.2 Data Cache 130
7.3 Cache Protocol 131
7.4 Cache Attributes 131
7.4.1 Uncached 131
7.4.2 Uncached Accelerated 132
7.4.3 Non-Coherent Write-Back 133
7.4.4 Coherent Exclusive Write-Back 133
7.4.5 Non-Coherent, Write-Through with No Write-Allocate 133
7.4.6 Encoding 134

Chapter 8 Bus Interface Unit 135
8.1 20Kc System Interface Features 135
8.1.1 Processor and External Requests 135
8.1.2 Multiplexed Unidirectional 32-bit Processor Address/Data Bus 136
8.1.3 Multiplexed Unidirectional 64-bit SOC Controller Address/Data Bus 136
8.1.4 Support for Multiple Outstanding Split Transactions 137
8.1.5 Credit-Based Flow Control 137
8.2 Bus Encoding (64-bit EB_SysAD Mode) 139
8.2.1 PrcCmd/SysCmd Bus Encoding (Command Cycles) 139
8.2.2 PrcCmd/SysCmd Bus Encoding (Data Cycles) 144
8.3 Processor and External Request Protocols (64-bit EB_SysAD Mode) 145
8.3.1 Processor Requests 145
8.3.2 External Requests 149
8.3.3 Coherency Conflicts 151
8.3.4 Data Ordering 153
8.3.5 Data Alignment 154
8.3.6 Dual Multiplexed Address and Data Buses 154
8.4 Bus Encoding (32-bit EB_SysAD Mode) 156
8.4.1 PrcCmd/SysCmd Bus Encoding (Command Cycles) 156
8.5 Processor and External Request Protocols (32-bit EB_SysAD Mode) 162
8.5.1 Processor Requests 162
8.6 20Kc Signal Descriptions 167

Chapter 9 Reset and Initialization 175
9.1 Processor Reset Signals 175
9.2 Processor Initialization Signals 175
9.3 Reset Sequences 176
9.3.1 Power-On Reset Sequence 176
9.3.2 ColdReset Sequence 177
9.3.3 Warm Reset Sequence 178

Chapter 10 Power Management 179
10.1 Register-Controlled Power Management 179
10.2 Instruction-Controlled Power Management 179

Chapter 11 EJTAG Debug Support 181
11.1 EJTAG Components and Options 181
11.1.1 EJTAG Extensions to the MIPS Processor Core 182
11.1.2 Debug Control Register 182
11.1.3 Hardware Breakpoint Unit 182
11.1.4 EJTAG Test Access Port 183
11.2 Register and Memory Map Overview 183
11.2.1 Coprocessor 0 Register Overview 183
11.2.2 Memory-Mapped EJTAG Register Overview 183

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

11.2.3 Data Hardware Breakpoint Register Overview 184
11.2.4 Register Field Notations 186
11.3 EJTAG Processor Core Extensions 186
11.3.1 Debug Mode Execution 186
11.3.2 Debug Mode Instruction Set 186
11.3.3 Debug Mode Address Space 187
11.3.4 Debug Mode Handling of Processor Resources 189
11.3.5 Debug Exceptions 189
11.3.6 Debug Mode Exceptions 193
11.3.7 Interrupts and NMIs 196
11.3.8 Reset and Soft Reset of the Processor 197
11.3.9 EJTAG Instructions 197
11.3.10 EJTAG Coprocessor 0 Registers 198
11.4 Debug Control Register 198
11.5 Hardware Breakpoints 199
11.5.1 Introduction 200
11.5.2 Overview of Instruction and Data Breakpoint Registers 201
11.5.3 Conditions for Matching Breakpoints 202
11.5.4 Debug Exceptions from Breakpoints 204
11.5.5 Breakpoints Used as Triggerpoints 206
11.5.6 Instruction Breakpoint Registers 207
11.5.7 Data Breakpoint Registers 210
11.6 EJTAG Test Access Port 214
11.6.1 TAP Signals 215
11.6.2 TAP Controller 216
11.6.3 Test-Logic-Reset State 217
11.6.4 Instruction Register and Special Instructions 218
11.6.5 Data Registers 220

Chapter 12 20Kc Test Features 231
12.1 Cache Test Mode 231
12.1.1 Cache Test Mode Interface Signals 231
12.1.2 System Interface Clock Divisor and Mode 231
12.1.3 Entering Cache Test Mode 232
12.1.4 Exit from Cache Test Mode 234
12.1.5 Cache Test Mode Commands 234
12.1.6 Read/Write Granularity 235
12.1.7 Encodings 235
12.1.8 Protocols 239
12.2 PLL Bypass Mode 240
12.3 BIST (Built-In Self Test) 241
12.3.1 Overview 241
12.3.2 Algorithms for Memory Test 244
12.3.3 BIST Integration on 20Kc Cache Memories 245
12.3.4 Cycles for Memory BIST Testing 246

Chapter 13 Instruction Set Architecture 247
13.1 CPU Architecture 247
13.1.1 CPU Register Overview 247
13.1.2 Endianness 248
13.1.3 CPU Instruction Overview 248
13.2 FPU Architecture 254
13.2.1 FPU Register Overview 254
13.2.2 FPU Instruction Overview 255
13.3 Coprocessor Architecture 258
13.4 Privileged Instruction Set Architecture 258

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

13.4.1 Privileged Register Overview 258
13.4.2 Privileged Instruction Overview 258
13.5 EJTAG Support Instructions 259
13.6 Instruction Bit Encoding 259
13.7 MIPS64 Instruction Descriptions 266
13.7.1 UNPREDICTABLE and UNDEFINED 266
13.7.2 Unprivileged Instructions 268
13.7.3 Privileged Instructions 271

Chapter 14 Floating-Point Unit 285
14.1 Special FCSR Bits 285
14.1.1 Flush-to-Zero (FS) 285
14.1.2 Flush-Override (FO) 286
14.1.3 Flush-to-Nearest (FN) 286
14.1.4 Summary of FS, FO and FN Bits 286
14.2 FCSR Cause Bit Update Flow 287
14.2.1 Exceptions Triggered by CTC1 287
14.2.2 Generic Cause Bit Update Flow 287
14.2.3 Multiply-Add Cause Bit Update Flow 287
14.2.4 Cause Update Flow for Operands 287
14.2.5 Cause Update Flow for Paired Single 288
14.2.6 Cause Update Flow for Unimplemented Operation 288
14.3 Denormal Handling 288
14.4 Reciprocal and Reciprocal Square Root 289
14.4.1 Forced Inexact 289
14.4.2 Forced Round-to-Nearest 289
14.4.3 Forced Flush-to-Nearest 289
14.4.4 Special Results 289
14.4.5 RSQRT2 Implementation 290
14.5 Single-Precision Result or Single-Word Load 290
14.6 QNaN Priority 290
14.7 Convert Ranges 290
14.7.1 Convert Integer to Float 290
14.7.2 Convert Float to Integer 291
14.7.3 Convert Double to Single Precision: CVT.S.D 291

Appendix A Revision History 293

Appendix B References 295

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

List of Figures

Figure 1-1: 20Kc Processor Core Block Diagram....3
Figure 2-1: Instruction Formats....8
Figure 3-1: 20Kc Core Pipeline....14
Figure 3-2: Dispatch Pipeline....15
Figure 3-3: Integer Execution Pipeline A....17
Figure 3-4: Integer Execution Pipeline B....19
Figure 3-5: Floating-Point Pipeline....20
Figure 3-6: Instruction Fetch Groups....24
Figure 3-7: Branch Target Within Instruction Group....24
Figure 3-8: Taken Branch Within Instruction Group....25
Figure 3-9: Taken Branch as Last Instruction in Fetch Group....25
Figure 3-10: Bubble Caused by Taken Branch of Jump Instruction....26
Figure 3-11: Decode Restrictions Based on Fetch Groups....26
Figure 3-12: Instruction Stall on Buffer Full....27
Figure 3-13: Resumption of Instruction Fetch After Stall....27
Figure 3-14: Instructions Dispatched in Program Order....28
Figure 3-15: Two Instruction Dispatch Limit....28
Figure 3-16: Dispatch Restrictions Based on Input Dependencies....29
Figure 3-17: Dispatch Restrictions Based on Output Dependencies....30
Figure 3-18: Dispatch Restrictions Based on Branch Delay Slots....30
Figure 3-19: Dispatch Restrictions Relating to MULT....30
Figure 3-20: Dispatch Restrictions Arising From DMULT....30
Figure 3-21: Dispatch Restrictions Arising from MUL Instruction....31
Figure 3-22: Dispatch Restrictions on MTHI/MTLO Instructions....31
Figure 3-23: Dispatch Restrictions Arising from Floating-Point Double-Precision Multiplies....31
Figure 3-24: Dispatch Restrictions on CTC1 and CFC1....31
Figure 3-25: Dispatch Restrictions on Privileged Instructions....32
Figure 3-26: Dispatch Restrictions Arising from Certain MTC0 Instructions....32
Figure 3-27: Dispatch Restrictions on Certain MTC0 Instructions....32
Figure 4-1: Address Components....36
Figure 4-2: Virtual Address Space....37
Figure 4-3: Address Interpretation for the xkphys Segment....41
Figure 4-4: User Mode Addressing....44
Figure 4-5: Supervisor Mode Addressing....45
Figure 4-6: Kernel Mode Addressing (32-bit)....46
Figure 4-7: Kernel Mode Addressing (64-bit)....48
Figure 4-8: Debug Mode Virtual Address Space....50
Figure 4-9: Overview of a Virtual-to-Physical Address Translation....51
Figure 4-10: 20Kc Virtual Address Translation Example....52
Figure 4-11: Contents of a TLB Entry....53
Figure 4-12: TLB Tag Entry Format....55
Figure 4-13: TLB Data Array Entry Format....56
Figure 5-1: General Exception Handler (HW)....77
Figure 5-2: General Exception Servicing Guidelines (SW)....78
Figure 5-3: TLB Miss Exception Handler (HW)....79
Figure 5-4: TLB Exception Servicing Guidelines (SW)....80
Figure 5-5: Reset, Soft Reset and NMI Exception Handling and Servicing Guidelines....81
Figure 6-1: Wired and Random Entries in the TLB....90
Figure 7-1: Instruction Cache State Diagram....131
Figure 7-2: Data Cache State Diagram....131

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Figure 8-1: Processor and SOC Controller Requests 136
Figure 8-2: Request/Response Combinations 136
Figure 8-3: Transaction Tags........ 137
Figure 8-4: Credit Counter Programming 138
Figure 8-5: SOC Controller Resource Control........ 138
Figure 8-6: Processor Resource Control........ 139
Figure 8-7: Command Bus Formats During a Command Cycle 140
Figure 8-8: Command Bus Formats During a Data Cycle 144
Figure 8-9: 32-bit EB_PrcAD/64-bit EB_SysAD Block Read Protocol........ 147
Figure 8-10: 32-bit EB_PrcAD/64-bit EB_SysAD Double/Single/Partial Word Read Protocol........ 147
Figure 8-11: 32-bit EB_PrcAD Block Write Protocol 148
Figure 8-12: 32-bit EB_PrcAD Double/Single/Partial Word Write Protocol 149
Figure 8-13: External Intervention with Associated Data Response........ 150
Figure 8-14: External Intervention with No Data Response 150
Figure 8-15: External Invalidation 151
Figure 8-16: 64-bit Block Data Ordering 153
Figure 8-17: 32-bit Block Data Ordering 154
Figure 8-18: Data Alignment Example 154
Figure 8-19: Command Bus Formats During a Command Cycle (32-bit EB_SysAD Mode)........ 157
Figure 8-20: Command Bus Formats During a Data Cycle 161
Figure 8-21: 32-bit EB_PrcAD/32-bit EB_SysAD Block Read Protocol........ 163
Figure 8-22: 32-bit EB_PrcAD/32-bit EB_SysAD Double/Single/Partial Word Read Protocol........ 164
Figure 8-23: 32-bit EB_PrcAD Block Write Protocol 164
Figure 8-24: 32-bit EB_PrcAD Double/Single/Partial Word Write Protocol 165
Figure 8-25: External Intervention with Associated Data Response........ 166
Figure 8-26: External Intervention with No Data Response 167
Figure 8-27: External Invalidation 167
Figure 8-28: 20Kc System Interface Signal Groupings 168
Figure 9-1: Power-On Reset Sequence........ 177
Figure 9-2: Cold Reset Sequence 178
Figure 9-3: Warm Reset Sequence 178
Figure 11-1: Simplified Block Diagram of EJTAG Components 182
Figure 11-2: DCR Register Format 198
Figure 11-3: Instruction Hardware Breakpoint Overview........ 200
Figure 11-4: Data Hardware Breakpoint Overview 200
Figure 11-5: IBS Register Format 208
Figure 11-6: IBAn Register Format 208
Figure 11-7: IBMn Register Format........ 209
Figure 11-8: IBASIDn Register Format 209
Figure 11-9: IBCn Register Format........ 209
Figure 11-10: DBS Register Format........ 211
Figure 11-11: DBAn Register Format 212
Figure 11-12: DBMn Register Format 212
Figure 11-13: DBASIDn Register Format 212
Figure 11-14: DBCn Register Format 213
Figure 11-15: DBVn Register Format 214
Figure 11-16: Test Access Port (TAP) Overview 215
Figure 11-17: TAP Controller State Diagram 216
Figure 11-18: Shifting of the Instruction Register During the Shift IR State 217
Figure 11-19: Shifting of the Instruction Register During the Shift DR State 218
Figure 11-20: Selecting Registers Using the ALL Instruction 219
Figure 11-21: EJ_TDI to EJ_TDO Path when in Shift-DR State and FASTDATA Instruction is Selected 219
Figure 11-22: Device ID Register Format 220
Figure 11-23: Implementation Register Format........ 221
Figure 11-24: Data Register Format 222

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Figure 11-25: Address Register Format 224
Figure 11-26: EJTAG Control Register Format 225
Figure 11-27: Fastdata Register Format 228
Figure 11-28: Bypass Register Format 230
Figure 12-1: Entering Cache Test Mode After a Power-On Reset Sequence 233
Figure 12-2: Entry into Cache Test Mode during a ColdReset Sequence 234
Figure 12-3: Normal Read Cycle 239
Figure 12-4: Data/Instruction Cache Tag Normal Write 240
Figure 12-5: Write Same Data Command 240
Figure 12-6: Integration of BIST with Memory 241
Figure 12-7: BIST algorithm selection 242
Figure 12-8: External Signal Behavior for a Memory Test 243
Figure 12-9: Retention Testing Example Waveform 244
Figure 12-10: Example of March Test Written in March Test Notation 245
Figure 12-11: March C+ Algorithm in March Test Notation 245
Figure 12-12: IFA-13 Algorithm in March Test Notation 245
Figure 13-1: CPU Registers in MIPS64 Native Mode 248
Figure 13-2: FPU Registers if StatusFR is 1 254
Figure 13-3: FPU Registers if StatusFR is 0 255
Figure 13-4: Usage of Address Fields to Select Index and Way 272

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

List of Tables

Table 2-1: Byte Access within a Word....9
Table 3-1: Instruction Groups, Latencies, and Repeat Rates 21
Table 3-2: Static Dispatch Rules 29
Table 4-1: Processor Modes....35
Table 4-2: Virtual Memory Address Spaces 38
Table 4-3: Address Space Access and TLB Refill Selection as a Function of Operating Mode 39
Table 4-4: Address Translation and Cache Coherency Attributes for the kseg0 and kseg1 Segments....41
Table 4-5: Address Translation and Cacheability Attributes for the xkphys Segment 41
Table 4-6: xkphys Spaces....49
Table 4-7: Physical Address Generation 55
Table 4-8: TLB Tag Entry Fields 55
Table 4-9: TLB Data Array Entry Fields 56
Table 4-10: TLB Instructions 57
Table 4-11: Mask and Page Size Values 57
Table 5-1: Exception Type Characteristics 60
Table 5-2: Priority of Exceptions 60
Table 5-3: Exception Vector Base Addresses 62
Table 5-4: Exception Vector Offsets....62
Table 5-5: Exception Vectors 62
Table 5-6: CP0 Register States on a Cache Error Exception 68
Table 5-7: Register States an Interrupt Exception....69
Table 5-8: Register States on a Watch Exception 70
Table 5-9: CP0 Register States on an Address Exception Error 71
Table 5-10: CP0 Register States on a TLB/XTLB Refill Exception....71
Table 5-11: CP0 Register States on a TLB Invalid Exception....72
Table 5-12: Register States on a Coprocessor Unusable Exception....74
Table 5-13: Register States on a TLB Modified Exception 76
Table 5-14: Mapping of Interrupts to the Cause and Status Registers....82
Table 6-1: CP0 Registers 83
Table 6-2: CP0 Register Field Types 85
Table 6-3: Index Register Field Descriptions....86
Table 6-4: Random Register Field Descriptions 87
Table 6-5: EntryLo0, EntryLo1 Register Field Descriptions 87
Table 6-6: Cache Coherency Attributes 88
Table 6-7: Context Register Field Descriptions 88
Table 6-8: PageMask Register Field Descriptions 89
Table 6-9: Values for the Mask Field of the PageMask Register....89
Table 6-10: Wired Register Field Descriptions....90
Table 6-11: BadVAddr Register Field Description....91
Table 6-12: Count Register Field Description....91
Table 6-13: EntryHi Register Field Descriptions....92
Table 6-14: Compare Register Field Description....92
Table 6-15: Status Register Field Descriptions 93
Table 6-16: Cause Register Field Descriptions 97
Table 6-17: Cause Register Exc Code Field Descriptions 97
Table 6-18: EPC Register Field Description....99
Table 6-19: PRId Register Field Descriptions....99
Table 6-20: Config Register Field Descriptions 100
Table 6-21: Config1 Register Field Descriptions 102
Table 6-22: LLAddr Register Field Descriptions....105

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 6-23: WatchLo Register Field Descriptions 106
Table 6-24: WatchHi Register Field Descriptions 107
Table 6-25: XContext Register Field Descriptions 108
Table 6-26: Debug Register Field Descriptions 109
Table 6-27: Debug Register Formats 112
Table 6-28: Performance Counter Control Register Field Descriptions 112
Table 6-29: Performance Counter Register Field Descriptions 114
Table 6-30: DErrCtl Register Field Descriptions 115
Table 6-31: IErrCtl Register Field Descriptions 115
Table 6-32: CacheErr Register Field Descriptions 116
Table 6-33: ITagLo Register Field Descriptions 117
Table 6-34: IDataLo Register Field Descriptions 117
Table 6-35: DTagLo Register Field Descriptions 118
Table 6-36: DDataLo Register Field Description 119
Table 6-37: ITagHi Register Field Descriptions 119
Table 6-38: IDataHi Register Field Descriptions 120
Table 6-39: DTagHI Register Field Descriptions 120
Table 6-40: DDataHi Register Field Description 120
Table 6-41: ErrorEPC Register Field Description 121
Table 6-42: DESAVE Register Description 121
Table 6-43: CP0 Hazard Spacing 122
Table 6-44: CP1 Registers 122
Table 6-45: CP1 Register Field Types 123
Table 6-46: FIR Register Field Descriptions 123
Table 6-47: FCCR Register Field Descriptions 125
Table 6-48: FEXR Register Field Descriptions 125
Table 6-49: FENR Register Field Descriptions 125
Table 6-50: FCSR Register Field Descriptions 126
Table 7-1: Instruction and Data Cache Attributes 129
Table 8-1: Request Encoding 140
Table 8-2: Encoding of RQS[4:3] for Processor Read Requests 141
Table 8-3: Encoding of RQS[1:0] for Processor Block Read Requests 141
Table 8-4: Encoding of RQS[2:0] for Processor D/S/P Word Read Requests 141
Table 8-5: Encoding of RQS[4:3] for Processor Write Requests 142
Table 8-6: Encoding of RQS[1:0] for Processor Block Write Requests 142
Table 8-7: Encoding of RQS[2:0] for Processor D/S/P Word Write Requests 142
Table 8-8: Encoding of RQS[2:0] for a Processor Uncached Accelerated Block Write Request 143
Table 8-9: Encoding of RQS[1:0] for External Intervention Requests 143
Table 8-10: Encoding of the C[1:0] Field 144
Table 8-11: Encoding of the DS[4:0] Field 144
Table 8-12: Conflict Resolution 151
Table 8-13: Acceptable Responses to External Requests 152
Table 8-14: 20Kc System Interface Address and Data Transfer Requirements 155
Table 8-15: Processor Address/Data Bus and Corresponding Parity Bit 155
Table 8-16: System Address Bus and Check Bits 156
Table 8-17: Request Encoding 157
Table 8-18: Encoding of RQS[4:3] for Processor Read Requests 157
Table 8-19: Encoding of RQS[1:0] for Processor Block Read Requests 158
Table 8-20: Encoding of RQS[2:0] for Processor D/S/P Word Read Requests 158
Table 8-21: Encoding of RQS[4:3] for Processor Write Requests 159
Table 8-22: Encoding of RQS[1:0] for Processor Block Write Requests 159
Table 8-23: Encoding of RQS[2:0] for Processor D/S/P Word Write Requests 159
Table 8-24: Encoding of RQS[2:0] for Processor Uncached Accelerated Write Requests 160
Table 8-25: Encoding of RQS[1:0] for External Intervention Requests 160
Table 8-26: Encoding of the C[1:0] Field 161

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 8-27: Encoding of the DS[4:0] Field 161
Table 8-28: 20Kc Core Signal Direction Key 169
Table 8-29: Signal Prefix Key 169
Table 8-30: 20Kc Core Signal Descriptions 169
Table 9-1: Clock Multiplier Ratios 175
Table 11-1: Overview of Coprocessor 0 Registers for EJTAG 183
Table 11-2: Overview of Debug Control Register as Memory-Mapped Register for EJTAG 183
Table 11-3: Overview of Instruction Hardware Breakpoint Registers 184
Table 11-4: Overview of Data Hardware Breakpoint Registers 184
Table 11-5: Overview of Test Access Port Registers 185
Table 11-6: Register Field Read/Write Notations 186
Table 11-7: Presence of dseg Segment 187
Table 11-8: Physical Address and Cache Attribute for dseg, dmseg, and drseg 187
Table 11-9: Access to dmseg Address Range 188
Table 11-10: Access to drseg Address Range 188
Table 11-11: Debug Exception Vector Locations 190
Table 11-12: Exception Handling in Debug Mode 194
Table 11-13: Coprocessor 0 Registers for EJTAG 198
Table 11-14: DCR Register Field Descriptions 199
Table 11-15: Instruction Breakpoint Register Summary 201
Table 11-16: Data Breakpoint Register Summary 201
Table 11-17: Instruction Breakpoint Condition Parameters 202
Table 11-18: Data Breakpoint Condition Parameters 203
Table 11-19: Behavior on Precise Exceptions from Data Breakpoints 205
Table 11-20: Rules for Update of BSn Bits on Precise Exceptions from Data Breakpoints 205
Table 11-21: Rules for Update of BSn Bits on Data Triggerpoints 207
Table 11-22: Instruction Breakpoint Register Mapping 207
Table 11-23: IBS Register Field Descriptions 208
Table 11-24: IBAn Register Field Descriptions 208
Table 11-25: IBMn Register Field Descriptions 209
Table 11-26: IBASIDn Register Field Descriptions 209
Table 11-27: IBCn Register Field Descriptions 210
Table 11-28: Data Breakpoint Register Mapping 210
Table 11-29: DBS Register Field Descriptions 211
Table 11-30: DBAn Register Field Descriptions 212
Table 11-31: DBMn Register Field Descriptions 212
Table 11-32: DBASIDn Register Field Descriptions 212
Table 11-33: DBCn Register Field Descriptions 213
Table 11-34: DBVn Register Field Descriptions 214
Table 11-35: TAP Instruction Overview 218
Table 11-36: EJTAG TAP Data Registers 220
Table 11-37: Device ID Register Field Descriptions 220
Table 11-38: Implementation Register Field Descriptions 221
Table 11-39: Data Register Field Descriptions 222
Table 11-40: Data Register Contents 223
Table 11-41: Address Register Field Descriptions 224
Table 11-42: EJTAG Control Register Field Descriptions 225
Table 11-43: Combinations of ProbTrap and ProbEn 228
Table 11-44: Fastdata Register Field Description 228
Table 11-45: Operation of the FASTDATA Access 229
Table 11-46: Bypass Register Field Description 230
Table 12-1: SysAD[31:0] Encoding for Address/Command Cycles 235
Table 12-2: SysAD[31:0] Encoding for Write Data Cycles 237
Table 12-3: EB_SysADP[3:0] Encoding for Write Data Cycles 237
Table 12-4: EB_PrcAD[31:0] Encoding for Read Data Cycles 238

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 12-5: EB_PrcADP[3:0] Encoding for Read Cycles 238
Table 12-6: Cache Test Mode Read Latency 239
Table 12-7: Normal Write Command 239
Table 12-8: Chip-Level Memory BIST Interface for the 20Kc Processor 242
Table 12-9: Status and Progress Indications 244
Table 13-1: CPU Load, Store, and Memory Control Instructions 249
Table 13-2: CPU Arithmetic Instructions 250
Table 13-3: CPU Logical Instructions 250
Table 13-4: CPU Move Instructions 251
Table 13-5: CPU Shift Instructions 251
Table 13-6: CPU Branch and Jump Instructions 252
Table 13-7: CPU Trap Instructions 252
Table 13-8: Obsolete Branch Instructions 253
Table 13-9: Embedded Application Instructions 253
Table 13-10: FPU Load and Store Instructions 255
Table 13-11: FPU Arithmetic Instructions 256
Table 13-12: FPU Move Instructions 256
Table 13-13: FPU Convert Instructions 257
Table 13-14: FPU Branch Instructions 258
Table 13-15: Obsolete FPU Branch Instructions 258
Table 13-16: Privileged Instructions 258
Table 13-17: EJTAG Support Instructions 259
Table 13-18: Symbols Used in the Instruction Encoding Tables 259
Table 13-19: MIPS64 Encoding of the Opcode Field 260
Table 13-20: MIPS64 SPECIAL Opcode Encoding of Function Field 260
Table 13-21: MIPS64 REGIMM Encoding of rt Field 261
Table 13-22: MIPS64 SPECIAL2 Encoding of Function Field 261
Table 13-23: MIPS64 MOVCI Encoding of tf Bit 261
Table 13-24: MIPS64 COPz Encoding of rs Field 262
Table 13-25: MIPS64 COPz Encoding of rt Field When rs=BCz 262
Table 13-26: MIPS64 COP0 Encoding of rs Field 262
Table 13-27: MIPS64 COP0 Encoding of Function Field When rs=CO 263
Table 13-28: MIPS64 COP1 Encoding of rs Field 263
Table 13-29: MIPS64 COP1 Encoding of Function Field When rs=S 264
Table 13-30: MIPS64 COP1 Encoding of Function Field When rs=D 264
Table 13-31: MIPS64 COP1 Encoding of Function Field When rs=W or L 265
Table 13-32: MIPS64 COP1 Encoding of Function Field When rs=PS 265
Table 13-33: MIPS64 COP1 Encoding of tf Bit When rs=S, D, or PS, Function=MOVCF 265
Table 13-34: MIPS64 COP1X Encoding of Function Field 266
Table 13-35: PREF Hint Field Encodings 268
Table 13-36: Usage of Effective Address 271
Table 13-37: Encoding of Bits[17:16] of CACHE Instruction 272
Table 13-38: Encoding of Bits [20:18] of the CACHE Instruction 273
Table 14-1: Flushing of Results 285
Table 14-2: Denorm/Tiny Handling for All Combinations of FS/FO/FN 286
Table 14-3: Denormal Operand Handling 288
Table 14-4: Default Answers for RECIPx, RSQRTx ASE Instructions 289
Table 14-5: Convert Integer to Float: CVT.[DS].[WL] 290
Table 14-6: Convert Float to Int: CVT/ROUND/CEIL/FLOOR/TRUNC/.[WL].[DS] 291

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Introduction to the 20Kc Processor

The MIPS64™ 20Kc™ processor core from MIPS® Technologies, Inc. is a - performance microprocessor core for integration in SOC applications. The 20Kc core implements the MIPS64 Instruction Set Architecture (ISA) and the MIPS-3D™ Application Specific Extension (ASE) instructions that accelerate 3D geometry processing.

The 20Kc core is a pipelined, dual-issue processor core featuring two integer execution units, a 32-KByte 4-way set-associative instruction cache, a 32-KByte non-blocking 4-way set-associative data cache, an MMU with a fully associative 8-entry data TLB (Micro-TLB) and a fully associative 48 dual-entry instruction/data TLB (Joint-TLB), and an IEEE-754-compliant, MIPS-3D-capable floating-point unit. The 20Kc core can issue two integer instructions, one integer and one floating-point instruction, or one floating-point operate instruction and one floating-point load/store instruction per cycle. The pipelined floating-point unit executes single-precision, double-precision, and paired-single (2-way SIMD single precision floating-point) instructions.

The 20Kc core is designed for easy integration into Application Specific Standard Product (ASSP) and Application Specific Integrated Circuit (ASIC) devices. The design of the 20Kc core makes it ideal for applications such as feature-rich digital set-top boxes, 3D game platforms, mid-range to high-end office automation products, and networking devices.

This chapter provides an overview of the 20Kc processor and consists of the following sections:

- Section 1.1, "Features"
- Section 1.2, "Architectural Overview"
- Section 1.3, "System Overview"

For 20Kc performance and process details, refer to the appropriate datasheet (Ref [*2*], Ref [*3*]).

1.1 Features

The key features of the 20Kc processor core are listed below:

- 64-bit address and data paths that support:
  – 32-bit addressing and 32-bit operations
  – 32-bit addressing and 64-bit operations
  – 64-bit addressing and 64-bit operations
- MIPS64 Compatible Instruction Set:
  - Based on MIPS V ISA and backward compatible with MIPS32™ Architecture
  - Includes MIPS-3D Application Specific Extension instructions
  - Supervisor Mode support for backward compatibility
  - Multiply-Accumulate and Multiply-Subtract Instructions
  - Targeted Multiply Instruction
  - Count Leading One and Count Leading Zero Instructions
  - Wait Instruction

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

  - Conditional Move Instruction
  - Prefetch Instruction
- Dual-issue superscalar microarchitecture capable of executing:
  - Two integer instructions (some pair-wise restrictions)
  - One integer and one floating-point instruction
  - One floating-point operate instruction and one floating-point load/store instruction
- Seven-stage pipeline for high-frequency operation with dynamic branch and jump prediction to minimize mispredict penalties
- Four instructions fetched per cycle and up to two branches dynamically predicted per cycle
- 32-KByte 4-way set-associative instruction cache with way prediction:
  - 32-byte cache line
  - Line locking support
  - Instruction prefetch support
  - Way prediction
- 32-KByte 4-way set associative non-blocking data cache:
  - 32-byte cache line
  - Write-back, write-allocate
  - Software programmable to write-through/no-write allocate on a per-page basis
  - Data cache hits under misses with up to four outstanding misses
  - Critical word first return to minimize blocking delay
  - Line locking support
  - Non-blocking prefetches
- Uncached accelerated operation gathers and bursts 32 bytes to the core output
- Pipelined integer multiply-accumulate unit
- Integer execution bypasses, load-to-use bypass, and store-to-load bypass
- MIPS-3D-capable floating-point unit:
  - Fully pipelined IEEE-754-compliant floating-point unit with the Multiply/Add (MADD) instruction
  - 2-way SIMD single-precision (Paired-Single) floating-point instructions
  - 13 new MIPS-3D instructions that accelerate 3D geometry processing
  - MIPS-3D includes support for pipelined reciprocal and reciprocal square root instructions with both reduced and full precision capabilities
- Programmable Memory Management Unit:
  - 40-bit virtual address space
  - 36-bit physical address space
  - Fully associative 8-entry Micro-TLB for data
  - Fully associative 48 dual-entry (even/odd page pair per entry) Joint-TLB for instructions and data
  - Variable pages size from 4 KBytes to 16 MBytes in 4x increments
- High-performance unidirectional core interface:

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

- 64-bit parity protected multiplexed address/data input bus
- 32-bit parity protected multiplexed address/data output bus
- Protocol supports split transactions and out-of-order data return
- External interventions and invalidates supported for coherent I/O
- Block reads/writes (32-byte bursts)
- Sub-block and partial word reads/writes
- Credit-based flow control for bus efficiency and elimination of retry overhead
- Core to SOC clock ratios from 2:1 to 8:1

- EJTAG Debug Support with single stepping, instruction and data breakpoints in virtual address space, and execution from debug probe memory

1.2 Architectural Overview

The 20Kc core is a pipelined, dual-issue microprocessor core featuring two integer execution units, a 32-KByte 4-way set-associative instruction cache, a non-blocking 32-KByte 4-way set-associative data cache, an MMU with a fully associative 8-entry data TLB and a fully associative 48 dual-entry instruction/data TLB, and an IEEE-754-compliant MIPS-3D capable floating-point unit. The 20Kc core can issue two integer instructions, or one integer and one floating-point instruction, or one floating-point operate instruction and one floating-point load/store instruction per cycle. The pipelined floating-point unit executes single precision, double precision, and paired-single (2-way SIMD single precision floating-point) instructions. Figure 1-1 shows a block diagram of the 20Kc processor.

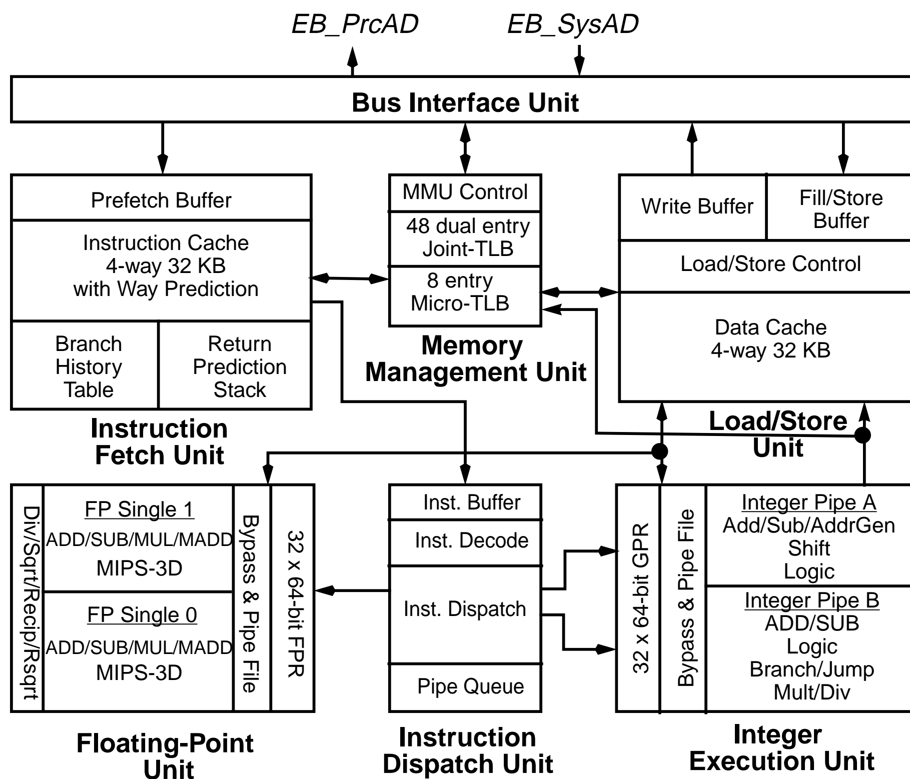


Figure 1-1 20Kc Processor Core Block Diagram

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Dual dispatch is possible except when restricted by input dependencies, output dependencies to loads that miss in the data cache or in the Micro-TLB, output dependencies to ‘long’ instructions, execution unit resource requirements, or other special instruction serializing conditions. The two-way pipeline is symmetric for most instruction types; two instructions of the same type can dispatch in the same dispatch slot. Exceptions to this are loads, stores, control transfer, shift, integer multiply, integer divide, floating-point, and Coprocessor 0 instructions.

The 20Kc core has an internal 32-KByte instruction cache and an internal 32-KByte data cache. The instruction cache can handle up to two outstanding misses, or one miss and one instruction prefetch. The data cache can handle up to four outstanding load/store/prefetch misses before stalling instruction dispatch.

As shown in Figure 1-1, the 20Kc core is divided into the following eight units:

- Instruction Fetch Unit
- Instruction Dispatch Unit
- Integer Execution Unit
- Floating-Point Unit
- Load/Store Unit
- Memory Management Unit
- Bus Interface Unit

1.2.1 Instruction Fetch Unit

The Instruction Fetch Unit supplies a speculative stream of instructions to the Instruction Dispatch Unit. The instruction stream is speculative because the Instruction Fetch Unit predicts the outcome of branches using a 256-entry branch history table, and predicts the address of jump returns using a four-entry return prediction stack. The Instruction Fetch Unit fetches an address aligned group of four instructions per cycle from an internal 32-KByte Instruction Cache. The Instruction Cache is organized as a 4-way set associative cache that utilizes way prediction for read operations. It is virtually indexed and virtually tagged. A two-entry instruction cache fill buffer (called the prefetch buffer) allows two instruction misses, or one instruction miss and one instruction prefetch.

1.2.2 Instruction Dispatch Unit

The Instruction Dispatch Unit controls the two integer execution pipelines and the floating-point execution pipeline. It dispatches up to two instructions per cycle to the execution pipelines. The Instruction Dispatch Unit receives address aligned groups of up to four valid instructions from the Instruction Fetch Unit, and stages them through a two-entry instruction fetch buffer (four instructions per entry) thus allowing the Instruction Fetch Unit to ‘get ahead’ of dispatch. The Instruction Dispatch Unit decodes up to two instructions per cycle from the instruction fetch buffer. It then dispatches the decoded instructions in program order to the execution pipelines (subject to dispatch restrictions, instruction dependencies and availability of resources). The Instruction Dispatch Unit keeps track of the dispatched instructions in a pipe queue and in an outstanding load queue. The pipe queue has one entry for each instruction that has been dispatched to the execution pipelines. The outstanding load queue has one entry for each load transaction that missed in the Data Cache. Finally, the Instruction Dispatch Unit prioritizes exceptions and maintains the architectural state of the processor.

1.2.3 Integer Execution Unit

The Integer Execution Unit contains the integer register file (GPR), *pipefiles* to stage results before GPR write-back, two arithmetic logic units (ALUs), and an integer multiply-divider. Associated with each ALU is a bypass multiplexer that allows results to be forwarded between the ALUs, and from the Load/Store Unit to the ALUs. The Instruction Execution Unit executes up to two instructions in parallel in two asymmetric pipelines. The ALU in Pipeline A executes add/sub,

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

shift, logic, load/store, and Coprocessor 0 instructions. The ALU in Pipeline B executes add/subtract, logic, and branch instructions. Integer multiplies and divides are also dispatched to Pipeline B and executed in the integer multiply-divider.

1.2.4 Floating-Point Unit

The 20Kc on-chip Floating-Point Unit (FPU) implements the MIPS64 ISA (Instruction Set Architecture) for floating-point computation and MIPS-3D ASE (Application Specific Extension). The implementation supports the ANSI/IEEE Standard 754 (IEEE Standard for Binary Floating-Point Arithmetic) and 13 MIPS-3D ASE instructions to enhance geometric computations in 3D-graphics applications. The hardware supports IEEE single- and double-precision data formats, plus the MIPS64 paired single data format. The performance is optimized for single and paired single precision formats. Most instructions have a one cycle throughput and a four cycle latency (see Section 3.5, "Floating-Point Pipeline").

The FPU is tightly coupled to the Instruction Dispatch Unit. Instructions are always dispatched and completed in order. The exception model is precise at all times.

The FPU implements the MIPS64 multiply-add (MADD) and multiply-subtract (MSUB) instructions with intermediate rounding after the intermediate multiply function. The result is guaranteed to be the same as executing a MUL instruction and an ADD instruction separately, but the instruction latency, instruction fetch, dispatch bandwidth, and the total number of register accesses are improved. A fast Flush-to-Zero mode is implemented to optimize performance. IEEE denormalized (denorm) input operands are supported by hardware for some instructions. IEEE denormalized result operands are not supported by hardware.

1.2.5 Load/Store Unit

The Load/Store Unit handles all instructions related to memory transactions and data cache management. It receives load/store instructions from the Instruction Dispatch Unit, load/store addresses from the Instruction Execution Unit, store data from the Instruction Execution Unit or FPU, address translation information from the Memory Management Unit (MMU), and cache refill data from the BIU. The Load/Store Unit has an internal Data Cache which is organized as a 4-way set associative cache. It is physically indexed and physically tagged. The Load/Store Unit is non-blocking, and allows four outstanding *Data Cache* misses to proceed in parallel before stalling instruction dispatch.

1.2.6 Memory Management Unit

The Memory Management Unit (MMU) handles address translation for the Instruction Fetch Unit and the Load/Store Unit. Since the instruction cache within the Instruction Fetch Unit is virtual, address translation is required only on cache misses. The MMU receives instruction cache miss requests from the Instruction Fetch Unit, completes the address translation in an internal 48 dual-entry Joint-TLB and forwards the miss request to the Bus Interface Unit (BIU). The BIU completes the instruction miss transaction and returns the cache line to the Instruction Fetch Unit. The data cache in the Load/Store Unit is physically tagged. Thus, the MMU must complete an address translation for each data cache access. It does this using an 8-entry Micro-TLB in the load-store path. The Micro-TLB is backed up by the 48 dual-entry Joint-TLB that is shared with instruction fetches that miss in the instruction cache.

1.2.7 Bus Interface Unit

The Bus Interface Unit (BIU) controls the bus interface and manages the flow of read and write data onto the external bus.

The BIU is responsible for servicing memory requests from the Load/Store Unit and the Instruction Fetch Unit. The BIU implements a high performance bus interface using a new bus protocol which retains some features of the traditional SysAD bus used in other MIPS processors. The BIU allows the processor to access external resources needed to satisfy

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

cache misses and uncached operations, while permitting the system controller to access some of the processors's internal resources.

1.2.8 EJTAG Unit

In addition to the standard JTAG Boundary Scan, the 20Kc core also supports the EJTAG or Enhanced JTAG. EJTAG enhances JTAG by providing a communication interface to a Probe controller for hardware/software debug. EJTAG uses the five pins of the JTAG interface mentioned above for control and data exchange and an additional pin called DINT for generating debug exceptions. The 20Kc EJTAG interface is fully compliant with *EJTAG Specification, Rev 2.6* (Ref [1]).

To facilitate EJTAG support for a MIPS processor, two instructions have been added to the MIPS ISA: SDBBP (Software Debug Breakpoint) and DERET (Debug Exception Return). SDBBP raises a debug breakpoint exception and passes control to an exception handler. DERET executes a return from a debug exception. Because DERET is a control transfer instruction, it cannot be placed in the delay slot of any other control transfer instruction. A debug exception has higher priority than any other exception except for the Reset exception. Even non-maskable interrupts (NMI) are masked when in Debug Mode.

1.3 System Overview

The 20Kc bus interface supports a unique interconnect protocol that is different from the traditional SysAD bus used in many embedded MIPS processors. One difference includes a 32-bit outbound multiplexed address/data bus.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Instruction Set Overview

This chapter provides a general overview on the three CPU instruction set formats of the MIPS architecture: Immediate, Jump, and Register. Refer to Chapter 13, “Instruction Set Architecture,” for a complete listing and description of instructions.

This chapter discusses the following topics:

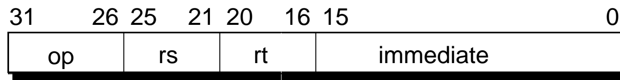
- Section 2.1, "CPU Instruction Formats"
- Section 2.2, "Load and Store Instructions"
- Section 2.3, "Computational Instructions"
- Section 2.4, "Jump and Branch Instructions"
- Section 2.5, "Control Instructions"
- Section 2.6, "Coprocessor Instructions"
- Section 2.7, "Enhancements to the MIPS Architecture"

2.1 CPU Instruction Formats

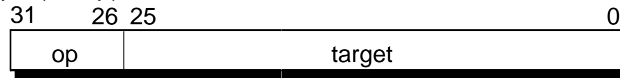
Each CPU instruction consists of a single 32-bit word, aligned on a word boundary. Figure 2-1 shows the three instruction formats: immediate (I-type), jump (J-type), and register (R-type). The use of a small number of instruction formats simplifies instruction decoding, allowing the compiler to synthesize more complicated (and less frequently used) operations and addressing modes from these three formats as needed.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

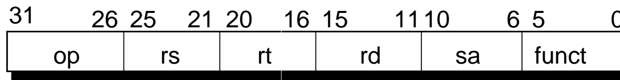
I-Type (Immediate)



J-Type (Jump)



R-Type (Register)



op	6-bit operation code
rs	5-bit source register specifier
rt	5-bit target (source/destination) register or branch condition
immediate	16-bit immediate value, branch displacement or address displacement
target	26-bit jump target address
rd	5-bit destination register specifier
sa	5-bit shift amount
funct	6-bit function field

Figure 2-1 Instruction Formats

2.2 Load and Store Instructions

Load and store are immediate (I-type) instructions that move data between memory and the general registers. The only addressing mode that load and store instructions directly support is base register plus 16-bit signed immediate offset.

2.2.1 Scheduling for Load Use Latencies

In the 20Kc core, the instruction immediately following a load instruction can use the contents of the loaded register. Hardware interlocks insert additional real cycles required in order to compensate for the longer latency of a load instruction. It is recommended, however, that the software schedule for these latencies to achieve higher performance.

2.2.2 Defining Access Types

Access type indicates the size of a core data item to be loaded or stored, set by the load or store instruction opcode.

Regardless of access type or byte ordering (endianness), the address given specifies the low-order byte in the addressed field. For a big-endian configuration, the low-order byte is the most-significant byte; for a little-endian configuration, the low-order byte is the least-significant byte.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

The access type and the three low-order bits of the address define the bytes accessed within the addressed word as shown in Table 2-1. Only the combinations shown in Table 2-1 are permissible; other combinations cause address error exceptions.

Table 2-1 Byte Access within a Word

Access Type Mnemonic (Value)	Low-Order Address Bits			Bytes Accessed															
				Big Endian (Byte)								Little Endian (Byte)							
	2	**1**	**0**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Doubleword (7)	0	0	0	0	1	2	3	4	5	6	7	7	6	5	4	3	2	1	0
Septibyte (6)	0	0	0	0	1	2	3	4	5	6			6	5	4	3	2	1	0
	0	0	1		1	2	3	4	5	6	7	7	6	5	4	3	2	1	
Sextibyte (5)	0	0	0	0	1	2	3	4	5					5	4	3	2	1	0
	0	1	0			2	3	4	5	6	7	7	6	5	4	3	2		
Quintibyte (4)	0	0	0	0	1	2	3	4							4	3	2	1	0
	0	1	1				3	4	5	6	7	7	6	5	4	3			
Word (3)	0	0	0	0	1	2	3									3	2	1	0
	1	0	0					4	5	6	7	7	6	5	4				
Triplebyte (2)	0	0	0	0	1	2											2	1	0
	0	0	1		1	2	3									3	2	1	
	1	0	0					4	5	6			6	5	4				
	1	0	1						5	6	7	7	6	5					
Halfword (1)	0	0	0	0	1													1	0
	0	1	0			2	3									3	2		
	1	0	0					4	5					5	4				
	1	1	0							6	7	7	6						
Byte (0)	0	0	0	0															0
	0	0	1		1													1	
	0	1	0			2											2		
	0	1	1				3									3			
	1	0	0					4							4				
	1	0	1						5					5					
	1	1	0							6			6						
	1	1	1								7	7							

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

2.3 Computational Instructions

Computational instructions can be either in register (R-type) format, in which both operands are registers, or in immediate (I-type) format, in which one operand is a 16-bit immediate.

Computational instructions perform the following operations on register values:

- Arithmetic
- Logical
- Shift
- Multiply
- Divide

These operations fit in the following four categories of computational instructions:

- ALU Immediate instructions
- Three-operand Register-type Instructions
- Shift Instructions
- Multiply and Divide Instructions

2.3.1 Multiply and Divide Instructions

Certain multiply instructions in the integer pipeline are different in that the product of the multiply instruction is saved in the HI and LO registers and not in the general purpose registers. The data can be transferred from the HI and LO registers to the general purpose registers via the MFHI (move from HI) and MFLO (move from LO) instructions. If the multiply instruction is followed by an MFHI or MFLO before the product is available, the pipeline interlocks until this product does become available.

2.4 Jump and Branch Instructions

Jump and branch instructions change the control flow of a program. All jump and branch instructions occur with a delay of one instruction: that is, the instruction immediately following the jump or branch (this is known as the instruction in the *delay slot*) always executes while the target instruction is being fetched from storage.

2.4.1 Overview of Jump Instructions

Subroutine calls in high-level languages are usually implemented with Jump or Jump and Link instructions, both of which are J-type instructions. In J-type format, the 26-bit target address shifts left two bits and combines with the high-order four bits of the current program counter to form an absolute address.

Returns, dispatches, and large cross-page jumps are usually implemented with the Jump Register or Jump and Link Register instructions. Both are R-type instructions that take the 64-bit address contained in one of the general-purpose registers.

For a list of the jump instructions, refer to Table 13-6 on page 252.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

2.4.2 Overview of Branch Instructions

All branch instruction target addresses are computed by adding the address of the instruction in the delay slot to the 16-bit *offset* (shifted left two bits and sign-extended to 64 bits). All branches occur with a delay of one instruction.

If a conditional branch likely is not taken, the instruction in the delay slot is nullified.

Branches, jumps, ERET, and DERET instructions should not be placed in the delay slot of a branch or jump.

2.5 Control Instructions

Control instructions allow the software to initiate traps; they are either R-type or I-type.

2.6 Coprocessor Instructions

CP0 instructions perform operations on the System Control Coprocessor registers to manipulate the memory management and exception handling facilities of the processor. Refer to Chapter 13, “Instruction Set Architecture,” for a listing of CP0 instructions.

2.7 Enhancements to the MIPS Architecture

The 20Kc core implements additional instructions over and above those indicated in the MIPS documents. Refer to Chapter 13, “Instruction Set Architecture,” for a complete list of these instructions.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Chapter 3

Pipeline

This chapter describes the seven-stage pipeline contained within the 20Kc core. This chapter contains the following sections:

- Section 3.1, "Pipeline Overview"
- Section 3.2, "Fetch Pipeline"
- Section 3.3, "Dispatch Pipeline"
- Section 3.4, "Integer and Load/Store Pipelines"
- Section 3.5, "Floating-Point Pipeline"
- Section 3.6, "Instruction Latencies and Repeat Rates"
- Section 3.7, "Instruction Fetch Rules"
- Section 3.8, "Instruction Dispatch Rules"
- Section 3.9, "Dispatch of Privileged Instructions"

3.1 Pipeline Overview

The seven main pipeline stages are listed below. The **bold** capital letter is the stage nickname.

- **F**etch (**F**)
- **V**alidate (**V**)
- **D**ecode (**D**)
- Register-file **R**ead (**R**)
- e**X**ecute (**X**)
- data **C**ache read (**C**)
- register file **W**rite (**W**)

The pipeline is divided into the four pipeline segments shown in Figure 3-1.

- Fetch pipeline
- Dispatch pipeline
- Integer execute and load/store pipelines (2)
- Floating-point pipeline

Instructions are fetched in program order and dispatched to the integer and floating-point pipelines in program order. Instructions may complete out of order, but results written into the integer and floating-point register files are completed in program order. The uniform pipeline flow is modified for loads and stores that miss in the Micro-TLB or in the data cache, for integer multiplies and divides, for double-precision floating-point multiply or multiply-add operations, for floating-point divides and square-roots, and for floating-point special case operations.

Most exceptions and interrupts are precise. Note that the bus error, data parity error, and EJTAG data value break exceptions are imprecise. Exceptions are synchronized at the integer register file write stage (**W** stage). Floating-point

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

exceptions are predicted during the floating-point execution **I** stage (thus lining up with the **W** stage) and finalized during the floating-point register file write stage (**Z** stage).

Figure 3-1 shows a diagram of the 20Kc processor pipeline.

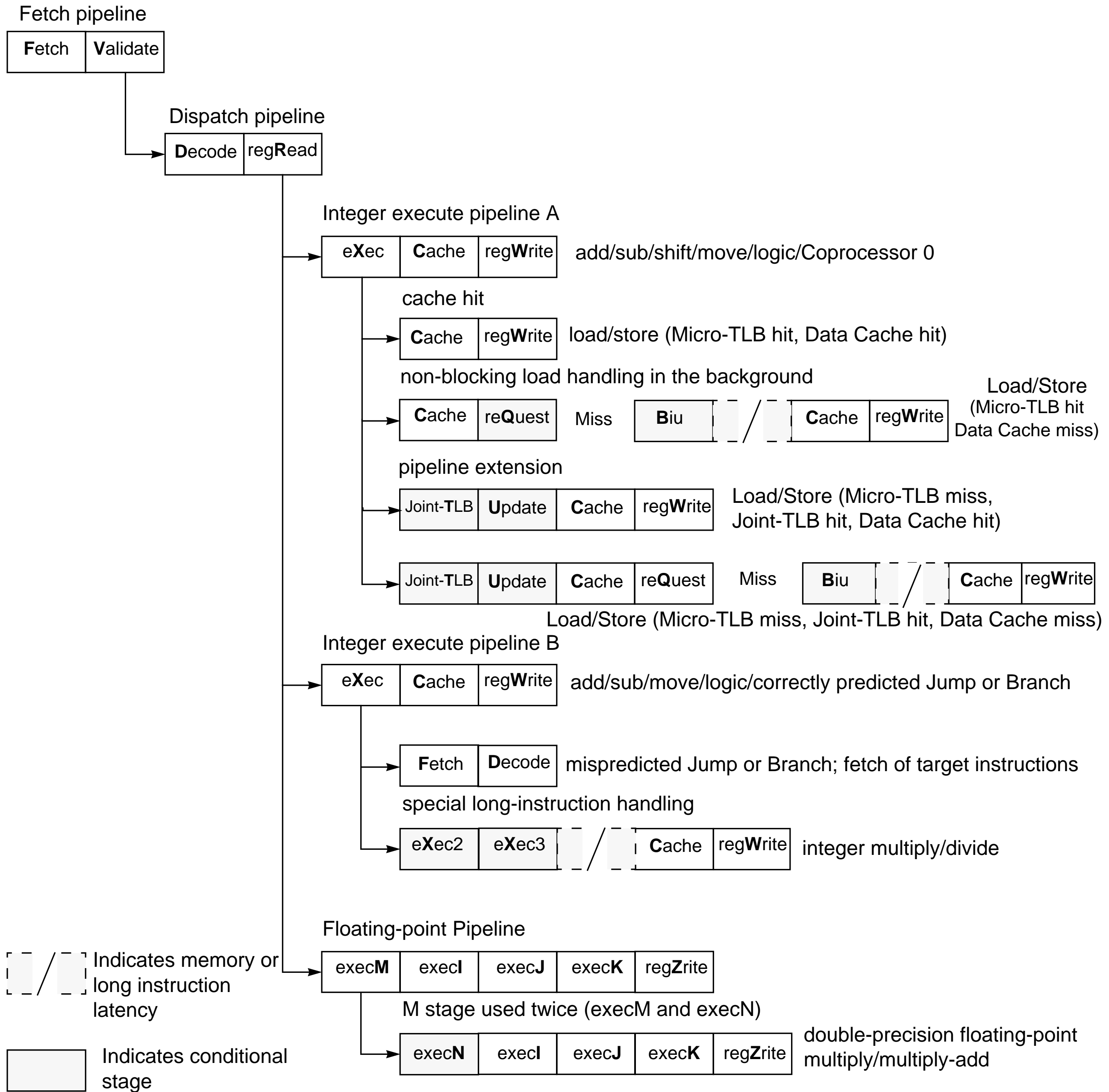


Figure 3-1 20Kc Core Pipeline

3.2 Fetch Pipeline

The Fetch pipeline consists of two stages: **F**etch and **V**alidate.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

3.2.1 F Stage: Instruction Fetch

During the Instruction **F**etch stage, the following events occur:

- Instruction fetch from the instruction cache
- Next fetch address prediction

3.2.2 V Stage: Validate

During the **V**alidate stage, the following events occur:

- Instruction validation
- Next fetch address calculation
- Branch prediction
- Subroutine return address prediction

Instructions are fetched from the instruction cache in a “bundle” of four 32-bit words. Instruction validation determines which instructions in the bundle are valid. For example, for a jump to an instruction in the middle of a bundle, the preceding instructions in the bundle are marked invalid.

3.3 Dispatch Pipeline

The Dispatch pipeline consists of two stages: **D**ecode and Register File **R**ead.

The Instruction Dispatch Unit reads up to two instructions per cycle from the Instruction Fetch Unit during the first half of the **D**ecode stage (**D** stage). It decodes the two instructions and writes the decoded instruction into a decode buffer.

The read of the register file is done during the first half of the register file **R**ead stage (**R** stage).The Instruction Execution Unit and the Floating-Point Unit deliver instruction source operands to the execution units during the second half of the **R** stage. The source operands are forwarded from one of the following:

- The register files
- One of the two “pipefile” registers that pipeline write data for the register file write
- Bypassed result from one of the execution units
- Load data from the Load/Store Unit

Figure 3-2 shows a diagram of the dispatch pipeline.

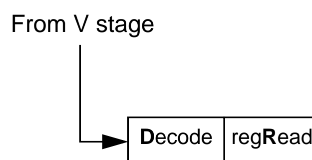


Figure 3-2 Dispatch Pipeline

The 20Kc core dispatches up to two instructions per cycle from the decode buffer. Dispatch is restricted by source operand availability, execution unit resources, and special instruction dispatch restrictions.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

3.3.1 D Stage: Instruction Decode

During the Instruction **D**ecode stage, the following events occur:

- Instruction decode
- Source register input dependency check
- Destination register output dependency check
- Dispatch restriction checks

3.3.2 R Stage: Register File Read

During the Register File **R**ead stage, the following events occur:

- Instruction dispatch
- Register file read (*GPR*/*FPR*)
- Pipefile read
- Source operand bypass

3.4 Integer and Load/Store Pipelines

The following stages control the integer, branch, and load/store operations.

The Instruction Execution Unit initiates the execution of integer, control transfer, and load/store instructions during the first half of the eXecution stage (**X** stage). It completes all integer instructions with the exception of multiplies, divides, loads, and stores during the second half of the **X** stage. It then forwards the results to dependent instructions (in the **R** stage), and writes them into the two pipefiles in the *GPR*. It stages the results in the pipefiles through the data **C**ache read stage (**C** stage) and writes them in program order into the *GPR* during the integer register **W**rite stage (**W** stage).

3.4.1 Integer Pipeline A

Integer instructions are executed in one of two asymmetric pipelines. Integer Pipeline A can execute:

- adds
- subtracts
- shifts
- logic operations
- moves to and from Coprocessor 0 and Coprocessor 1
- address generation for loads and stores

Integer Pipeline A, in conjunction with Integer Pipeline B, can execute conditional moves.

Figure 3-3 shows a diagram of Integer Pipeline A.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

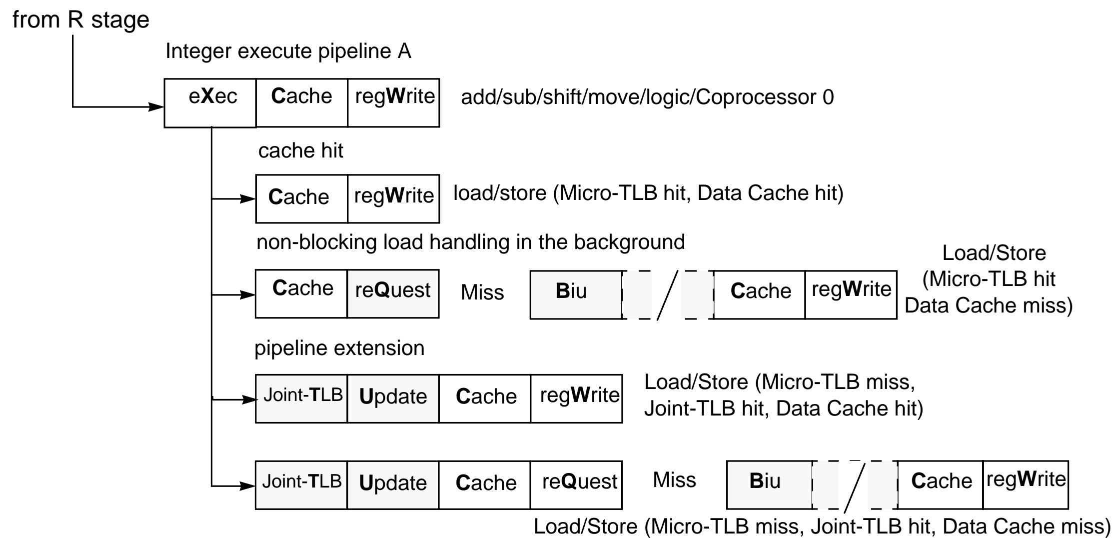


Figure 3-3 Integer Execution Pipeline A

3.4.1.1 X Stage: Execution

During the E**x**ecution stage, the following events occur:

- Execution of integer add, subtract, logic, shift, conditional moves
- Bypass of integer results to dependent instructions
- Address generation for loads and stores
- Micro-TLB lookup for loads and stores

3.4.1.2 C Stage: Cache

During the Data **C**ache stage, the following events occur:

- Data Cache access
- Load data write into pipefiles
- Load data forwarding to dependent instructions

3.4.1.3 W Stage: Register Write

During the Register **W**rite stage, the following events occur:

- Result written to the *GPR*
- Resolution of exceptions

3.4.1.4 Loads and Stores

The 20Kc core can have up to four outstanding loads, stores, or prefetches that miss in the data cache. It can continue instruction dispatch until it encounters an input or an output dependency to a missing load, or until it encounters the fourth missing load, store, or prefetch.

Loads and stores require two or more cycles to complete. The 20Kc core calculates the virtual address for loads and stores and looks up the address translation in the Micro-TLB in the e**X**ecution stage (**X** stage). A hit in the Micro-TLB

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

allows the 20Kc core to access the data cache during the data **C**ache read stage (**C** stage), and to forward the load data to the integer and floating-point pipefiles and to dependent instructions in the integer pipelines. Dependent instructions in the floating-point pipeline receive the forwarded data in the reg**W**rite stage (**W** stage) of the load instruction.

Loads and stores that miss in the Micro-TLB cause a two-cycle pipeline extension. The 20Kc core accesses the Joint-TLB during the **T**LB lookup stage (**T** stage), and updates the Micro-TLB during the Micro-TLB **U**pdate stage (**U** stage). The data cache is then accessed during the **C** stage. The following subsections describe the above conditional stages.

3.4.1.5 T Stage: TLB Lookup

The **T** stage is a conditional stage that only occurs on a Micro-TLB miss. During the **T**LB Lookup stage, a Joint-TLB lookup is performed for loads and stores that miss in the Micro-TLB.

3.4.1.6 U Stage: Update

The **U** stage always follows the **T** stage and only occurs if the **T** stage occurs. During the **U**pdate stage, the following events occur:

- Update from Joint-TLB to Micro-TLB
- Regenerate cache address

3.4.1.7 Q Stage: Memory Request

The **Q** stage is a conditional stage that occurs after the **C** stage and performs a memory request from the Load/Store buffer to the BIU whenever a load or store misses in the Data Cache. If the load or store hits in the Data Cache, this stage is bypassed.

3.4.1.8 B Stage: BIU Refill

The **B** stage always follows the **Q** stage and occurs after the data is retrieved from system memory. The number of cycles between the **Q** and **B** stages is variable and is dependent on memory latency. During the **B**IU Refill stage, the data received from memory is forwarded to the load/store unit. The load/store unit refills the fill/store buffers in a subsequent cycle.

3.4.2 Integer Pipeline B

Integer Pipeline B also uses the **X**, **C**, and **W** pipeline stages and can execute:

- adds
- subtracts
- logic operations
- integer multiplies and divides
- control transfer instructions

Integer Pipeline A, in conjunction with Integer Pipeline B, can execute conditional moves.

Figure 3-4 shows a diagram of Integer Pipeline B.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

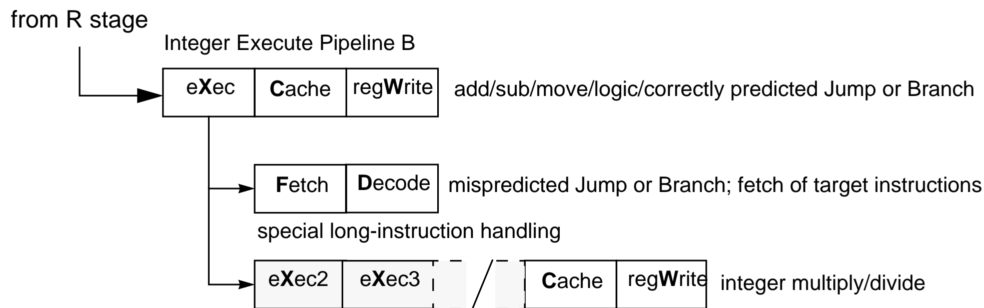


Figure 3-4 Integer Execution Pipeline B

3.4.2.1 X Stage: Execution

During the E**x**ecution stage, the following events occur:

- Execution of integer add, sub, logic, conditional move
- Resolution of control transfers
- Bypass of integer results to dependent instructions
- First cycle of integer multiply and divide

3.4.2.2 C Stage: Cache

During the Data **C**ache stage, the following event occurs:

- Result of **X** stage calculation written into pipefiles

3.4.2.3 W Stage: Register Write

During the Register **W**rite stage, the following events occur:

- Result written to the *GPR*
- Resolution of exceptions

3.4.2.4 Integer Multiply and Divide

Integer multiply and divide operations begin in the e**X**ec stage (**X** stage) and require more than two cycles to complete. Integer word multiplies are pipelined, and can be dispatched every other cycle. Integer double multiplies and all integer divides are not pipelined, and thus the 20Kc core stalls dispatch if it encounters any of the following operations:

- integer multiply
- double multiply
- divide while executing any multiply or divide
- integer double multiply or divide while executing any multiply or divide

3.4.2.5 Control Transfer Instructions

For control transfer instructions, a mispredict is signalled if the prediction is incorrect, and the 20Kc core restarts the instruction fetch from the correct instruction address after the execution of the control transfer instruction. The control transfer mispredict latency (the minimum delay between fetching the control transfer and when the target instructions are fetched if there is a mispredict) is five cycles.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

3.4.2.6 Conditional Moves

Conditional moves are single-dispatch instructions. The register condition is always tested in the Arithmetic Logic Unit (ALU) in Pipeline B, the actual move is done in the ALU in Pipeline A. The condition is either determined from an integer register or from one or several of eight floating-point condition code bits (CCs).

3.5 Floating-Point Pipeline

The Floating-Point Unit (FPU) initiates the execution of floating-point instructions during the execM stage (M stage). Execution takes four cycles (**M** stage, **I** stage, **J** Stage, and **K** stage) for all instructions except double-precision multiply, double precision multiply-add, divide, square root, and instructions with special case operands or results. An extra exec**N** stage (**N** stage) is required for all double-precision multiplies and multiply-adds and for some of the geometry instructions. Divide, Square Root, Reciprocal, and Reciprocal Square Root instructions take more than five cycles to complete.

Figure 3-5 shows a diagram of the floating-point pipeline.

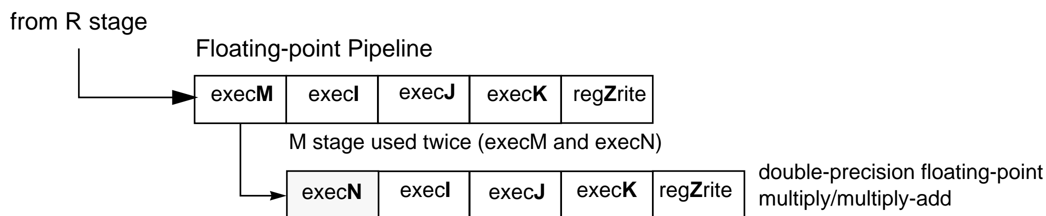


Figure 3-5 Floating-Point Pipeline

The floating-point unit is pipelined, and a new single-precision instruction or double-precision add can be dispatched every cycle. A double-precision multiply or multiply-add can be dispatched every other cycle. Floating-point exceptions are predicted at the end of the second floating-point execute stage (**I** stage). This aligns them with exceptions detected within the integer pipelines. If a floating-point exception is predicted, the 20Kc core stalls dispatch until the instruction that caused the predicted exception has completed.

Floating-point loads differ from integer loads only in that the load data returned from the data cache is written into the pipefile in the *FPR* rather than into the pipefile in the *GPR*. Load data is written into the pipefile during the end of the **I** stage (which lines up with the end of the **C** stage). It is then staged in the pipefile through the **J** stage and **K** stage, and written into the *FPR* during the **Z** stage. Floating-point load data cannot be directly bypassed to dependent instructions. It must be written into the pipefile, and bypassed from the pipefile, thus adding one extra stage to the load to use latency.

Moves between the *GPR* and *FPR* are handled using the integer and floating-point load/store paths. A move from the *GPR* to the *FPR* has a latency of three cycles. Similarly, a move from the *FPR* to the *GPR* has a latency of two cycles.

Conditional moves between floating-point registers that are based on integer register conditions are dispatched both to the floating-point unit and the integer unit. The integer unit determines the integer register condition, and the floating-point unit completes the actual move.

3.5.1 M Stage: FP Multiplier Array - First Pass

During the Multiplier Array - First Pass stage, the following events occur:

- First cycle through the multiplier array (**M**)
- Exponent calculation

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

3.5.2 N Stage: FP Multiplier Array - Second Pass

During the Multiplier Array - Second Pass stage, a second pass occurs through the multiplier array (**N**).

3.5.3 I Stage: FP Multiply Completion

During the **I** stage, the following events occur:

- Calculation of rounded multiply result
- Prediction of possible exceptions

3.5.4 J Stage: FP Add

During the **J** stage, the following events occur:

- Alignment shift
- Result add
- Leading zero prediction

3.5.5 K Stage: FP Normalization

During the **K** stage, the following events occur:

- Final Rounding
- Normalization
- Final resolution of exceptions

3.5.6 Z Stage: FPR Write

During the FPR Write (**regZrite**) stage, the result is written to the *FPR*.

3.6 Instruction Latencies and Repeat Rates

Table 3-1 contains the instruction latencies and repeat rates for the 20Kc core. In this table, ‘Latency’ refers to the number of cycles necessary for the first instruction to produce the result needed by the second instruction. The ‘repeat rate’ refers to the maximum rate at which an instruction can be executed per processor core cycle.

Table 3-1 Instruction Groups, Latencies, and Repeat Rates

Instruction Group	Included Instructions	Latency	Repeat Rate
Integer	AND[I], NOR, OR[I], XOR[I], LUI, [D]ADD[I][U], [D]SUB[U], SLT[I][U], [D]CLO, [D]CLZ, SSNOP	1	1
cMove	MOVN, MOVZ, MOVF, MOVT	1	1
Shift	[D]SLL, DSLL32, [D]SLLV, [D]SRL, DSRL32, [D]SRLV	1	1
	[D]SRA, DSRA32, [D]SRAV	1	1

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 3-1 Instruction Groups, Latencies, and Repeat Rates (Continued)

Instruction Group	Included Instructions	Latency	Repeat Rate
MulDiv	MTHI, MTLO	3	3
	MFHI, MFLO	1	3
	MULT[U], MADD[U], MSUB[U],	4	2
	DMULT[U]	7	7
	MUL,	4	2
	DIV[U]	13..42	13..42
	DDIV[U]	13..72	13..72
Branch	BEQ[L], BGEZ[AL][L], BGTZ[L], BLEZ[L], BLTZ[AL][L], BNE[L],	0, 4*	1
	J[AL][R],	0, 4*	1
	BC1F[L], BC1T[L]	0, 4*	1
Load/Store	LB[U], LH[U], LW[U], LD,	2	1
	LWL, LWR, LDL, LDR,	4	1
	SB, SH, SW, SD, SWL, SWR, SDL, SDR,	2	1
	SC[D], LL[D],	2	1
	LD[U][X]C1, LW[X]C1,	3	1
	SD[U][X]C1, SW[X]C1,	3	1
	PREF[X], SYNC	1	1
Trap	BREAK, SYSCALL,	3	1
	TEQ[I], TGE[I][U], TLT[I][U], TNE[I]	1, 4*	1

* Latency = 0 or 1 if branch/jump/conditional trap predicted correctly. Latency = 4 if branch/jump/conditional trap predicted incorrectly.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 3-1 Instruction Groups, Latencies, and Repeat Rates (Continued)

Instruction Group	Included Instructions	Latency	Repeat Rate
Fp	ABS.[S,D,PS], MOV.[S,D,PS], NEG.[S,D,PS], ADD.[S,D,PS], SUB.[S,D,PS], ADDR.PS, C.cond.[S,D,PS], MUL.[S,PS]	4	1
	MULR.[S,PS], MADD.[S,PS], MSUB.[S,PS], NMADD.[S,PS], NMSUB.[S,PS], P[UL][UL].PS, CABS.cond.[S,D,PS]	4	1
	CVT.D.S, CVT.PS.PW, CVT.[S,D].[W,L]	4	1
	CVT.S.D, CVT.PW.PS, CVT.[W,L].[S,D], CEIL.[W,L].[S,D], FLOOR.[W,L].[S,D], ROUND.[W,L].[S,D], TRUNC.[W,L].[S,D]	5	1
	MUL.D	5	2
	MADD.D, MSUB.D, NMADD.D, NMSUB.D, RECIP1.[S,D,PS], RSQRT1.[S,D,PS], RECIP2.D, RSQRT2.D	5	2
	RECIP2.[S,PS], RSQRT2.[S,PS]	4	1
	RECIP.S	13	13
	RECIP.D	25	25
	RSQRT.S	17	17
	RSQRT.D	35	35
	DIV.S, SQRT.S	17	17
	DIV.D, SQRT.D	32	32
FpMove	MTC1	3	1
	MFC1	2	1
	CTC1	4	3
	CFC1	2	1
Priv	CACHE,	min 3	min 3
	ERET, DERET, SDBBP,	3	1
	TLBP, TLBR, TLBWI, TLBWR,	4	4
	WAIT,	-	-
	[D]MTC0	3	1
	[D]MFC0	2	1

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

3.7 Instruction Fetch Rules

Instructions are fetched from the instruction cache in groups of four instructions as shown in Figure 3-6. The group of four instructions is aligned to one of two groups of four instructions within a 32-byte cache line.

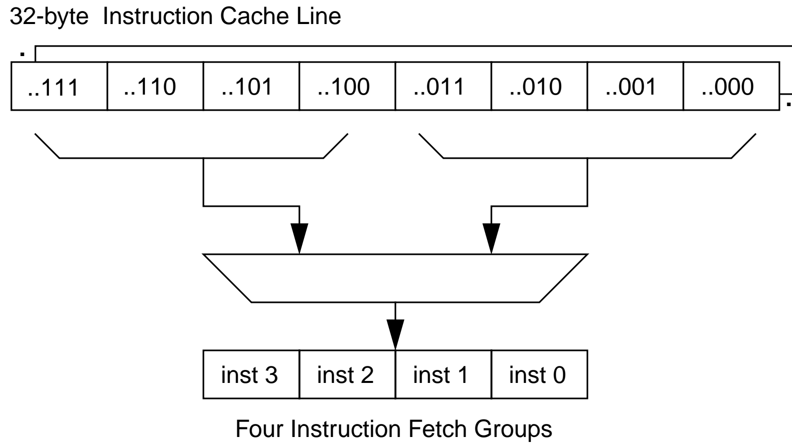


Figure 3-6 Instruction Fetch Groups

A branch target within an instruction group causes invalid instructions to be fetched up to the branch target as shown in Figure 3-7.

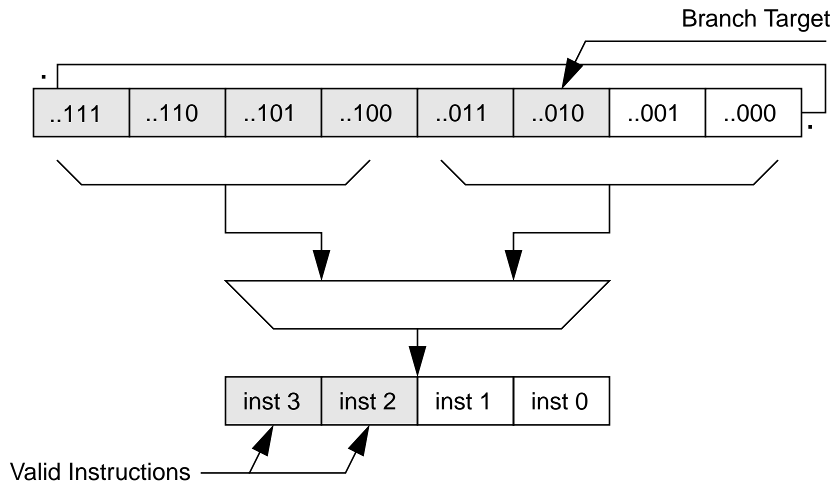


Figure 3-7 Branch Target Within Instruction Group

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

A taken branch within the instruction group causes invalid instructions to be fetched following the delay slot of the branch as shown in Figure 3-8.

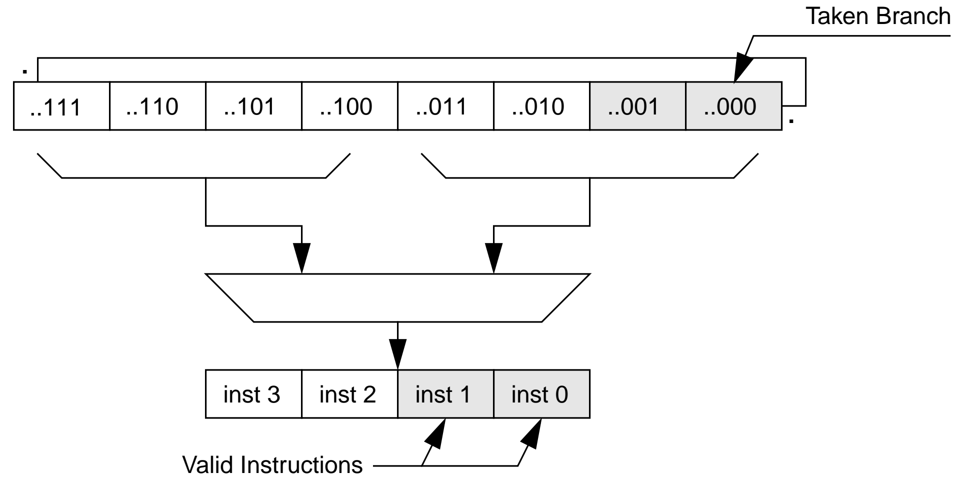


Figure 3-8 Taken Branch Within Instruction Group

A taken branch as last instruction in a group causes the following group to include the delay slot as the only valid instruction as shown in Figure 3-9.

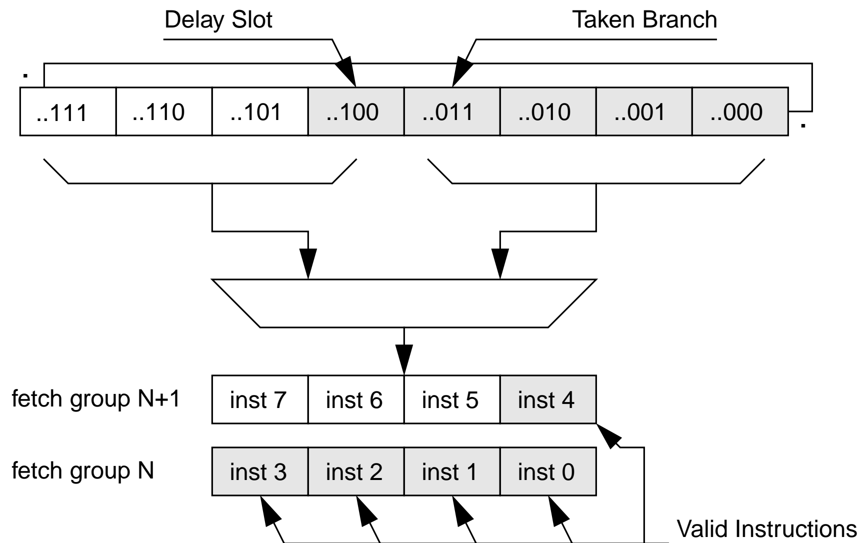


Figure 3-9 Taken Branch as Last Instruction in Fetch Group

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

A taken branch or jump causes a one cycle bubble in instruction fetch as shown in Figure 3-10.

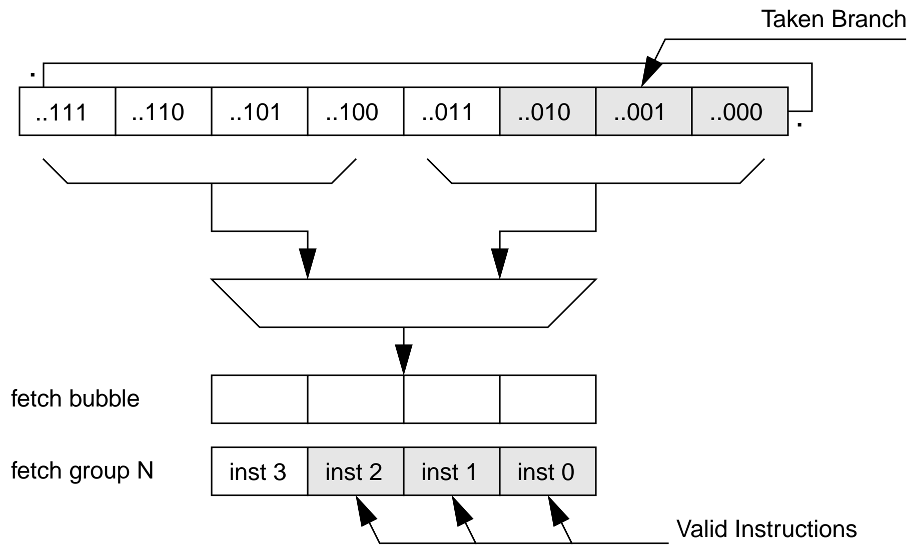


Figure 3-10 Bubble Caused by Taken Branch of Jump Instruction

Up to two instructions are decoded per cycle. Both instructions have to be part of the same fetch group. An attempt is made to always have two decoded instructions available for instruction dispatch. Figure 3-11 shows the decode restrictions.

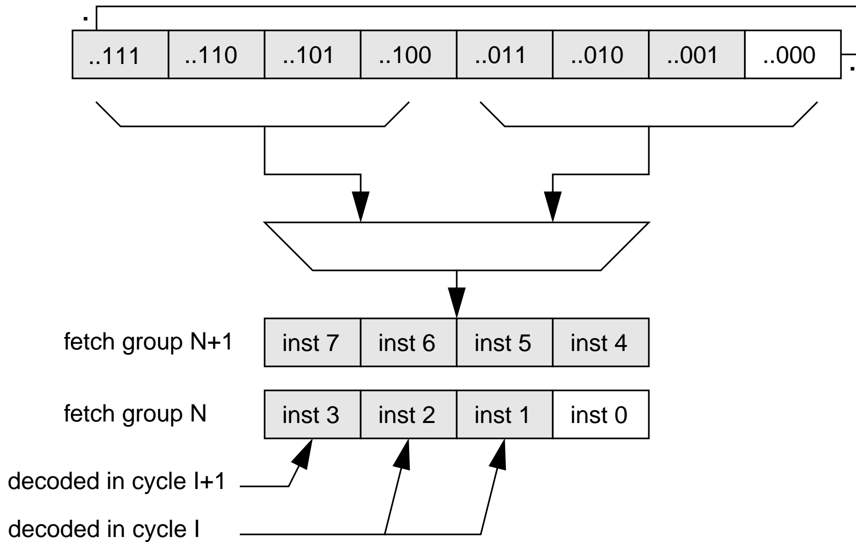


Figure 3-11 Decode Restrictions Based on Fetch Groups

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

An instruction fetch stalls after two instruction groups have been fetched as shown in Figure 3-12.

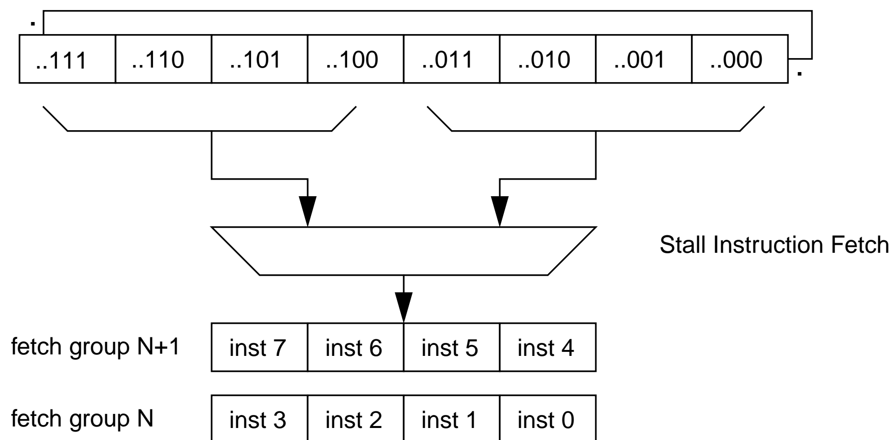


Figure 3-12 Instruction Stall on Buffer Full

Instruction fetching resumes after all instructions in one of the groups have been decoded as shown in Figure 3-13.

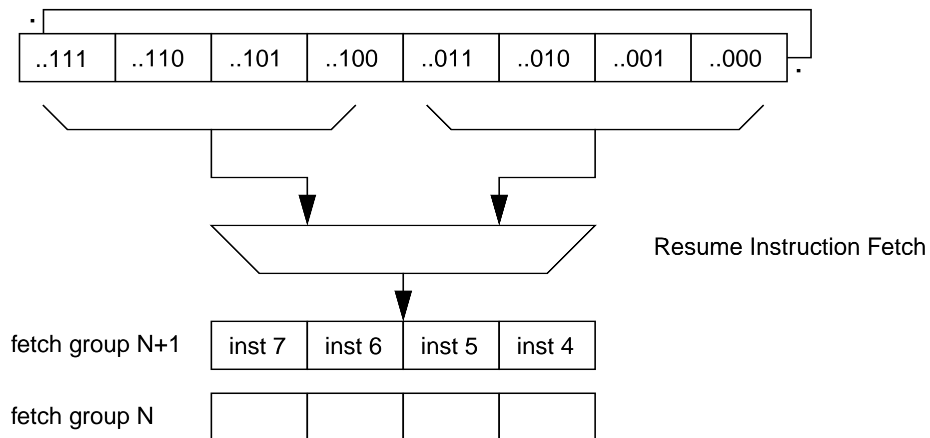


Figure 3-13 Resumption of Instruction Fetch After Stall

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

3.8 Instruction Dispatch Rules

Instructions are dispatched in program order from the group of decoded instructions as shown in Figure 3-14.

Instruction										
ADD r1, r2, r3	F	V	D	R	X	C	W			
ADD r4, r5, r6	F	V	D	R	X	C	W			
ADD r7, r8, r9	F	V		D	R	X	C	W		
ADD r10, r11, r12	F	V		D	R	X	C	W		
ADD r13, r14, r15		F	V		D	R	X	C	W	
ADD r16, r17, r18		F	V		D	R	X	C	W	
ADD r19, r20, r21		F	V			D	R	X	C	W

Figure 3-14 Instructions Dispatched in Program Order

Up to two instructions can be dispatched per cycle as shown in Figure 3-15.

Instruction							
ADD r1, r2, r3	D	R	X	C	W		
ADD r6, r7, r8	D	R	X	C	W		
ADD r1, r2, r3		D	R	X	C	W	
ADD r6, r7, r8		D	R	X	C	W	
ADD r1, r2, r3			D	R	X	C	W
ADD r6, r7, r8			D	R	X	C	W

Figure 3-15 Two Instruction Dispatch Limit

Instructions are dispatched according to the static dispatch rules in Table 3-2 unless other dispatch restrictions apply. **Note:** 1 and 2 in the table refer to the first and second instructions in the pair being considered for dispatch. The notation "1, 2" in the matrix means both instructions are dispatched, whereas “1” means just the first instruction is dispatched. For example, two integers are dispatched together (provided no other restrictions), but if an Integer is followed by a cMove, only the Integer is dispatched.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 3-2 Static Dispatch Rules

Inst[2] / Inst[1]	Integer[2]	cMove[2]	Shift[2]	MulDiv[2]	Branch[2]	LdSt[2]	Trap[2]	Fp[2]	FpMove[2]	Priv[2]
Integer[1]	1, 2	1	1, 2	1, 2	1, 2	1, 2	1, 2	1, 2	1, 2	1
cMove[1]	1	1	1	1	1	1	1	1	1	1
Shift[1]	1, 2	1	1	1, 2	1, 2	1	1, 2	1, 2	1	1
MulDiv[1]	1, 2	1	1, 2	1	1	1, 2	1	1, 2	1, 2	1
Branch[1]	1, 2	1	1, 2	1	1	1, 2	1	1, 2	1, 2	1
LdSt[1]	1, 2	1	1	1, 2	1, 2	1	1, 2	1, 2	1	1
Trap[1]	1, 2	1	1, 2	1	1	1, 2	1	1, 2	1, 2	1
Fp[1]	1, 2	1	1, 2	1, 2	1, 2	1, 2	1, 2	1	1, 2	1
FpMove[1]	1, 2	1	1	1, 2	1, 2	1	1, 2	1, 2	1	1
Priv[1]	1	1	1	1	1	1	1	1	1	1

Instructions are dispatched only if their input dependencies are expected to be met during the **R** stage of the pipeline as shown in Figure 3-16.

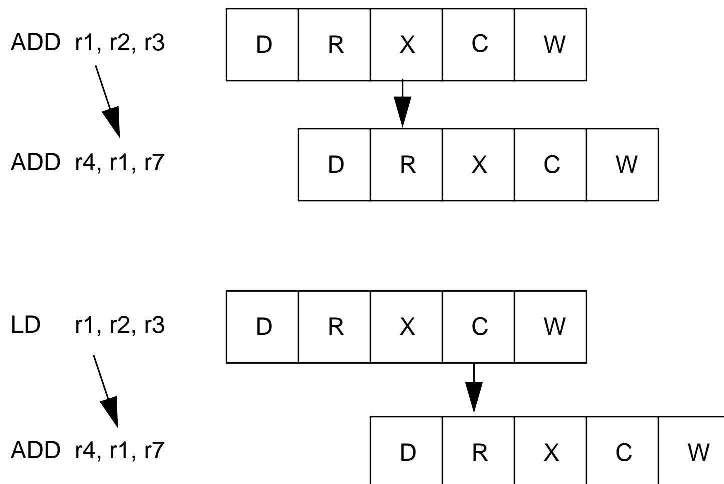
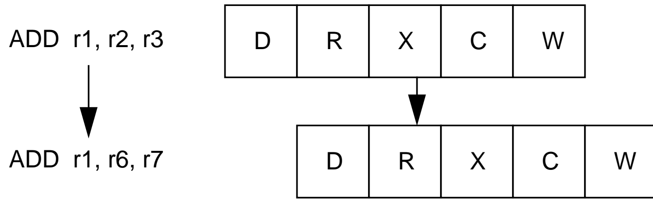


Figure 3-16 Dispatch Restrictions Based on Input Dependencies

Figure 3-17 shows the dispatch restrictions based on output dependencies. Instructions are dispatched only if their output dependencies are expected to be met during the **R** stage of the pipeline. Output dependencies are thus handled the same way as input dependencies; that is, an instruction that writes into a given register is dispatched following the same dispatch rules as an instructions that reads the given register.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.



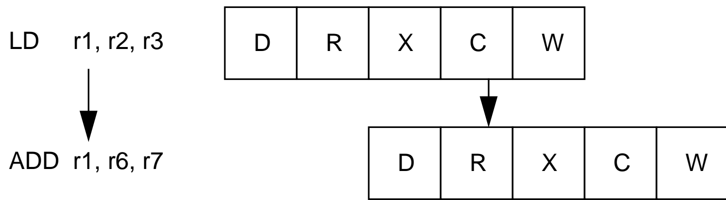


Figure 3-17 Dispatch Restrictions Based on Output Dependencies

Branch and Jump instructions cannot be dispatched until their delay slot has been fetched as shown in Figure 3-18.

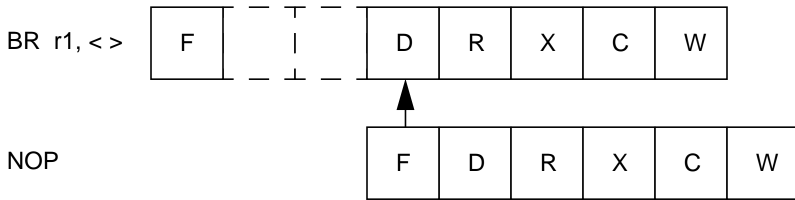


Figure 3-18 Dispatch Restrictions Based on Branch Delay Slots

Figure 3-19 shows the dispatch restrictions for multiply instructions. Integer MUL, MULT[U], MADD[U], and MSUB[U] instructions can be pipelined. MULT[U], MADD[U], and MSUB[U] have a latency of four cycles and a repetition rate of two cycles.

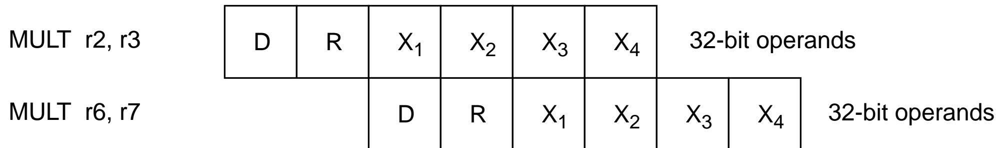


Figure 3-19 Dispatch Restrictions Relating to MULT

Integer DMULT[U] and [D]DIV[U] cannot be pipelined with any other multiply/divide instructions as shown in Figure 3-20.

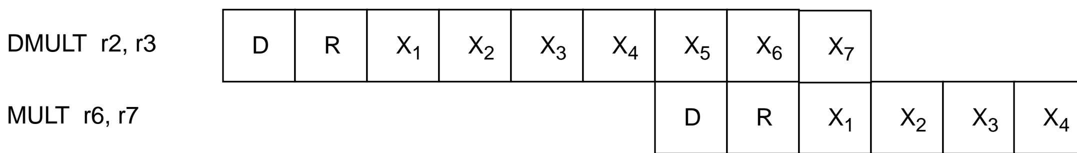


Figure 3-20 Dispatch Restrictions Arising From DMULT

Only one instruction having a GPR register result can be dispatched in the second cycle following the dispatch of a MUL instruction as shown in Figure 3-21. A branch and link, or a jump and link cannot be dispatched in the second cycle following the dispatch of a MUL.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

MUL r1, r2, r3	D	R	X_1	X_2	X_3	X_4	W
ADD r6, r7, r8			D	R	X	C	W
ADD r9, r10, r11			~~D~~	~~R~~	~~X~~	~~C~~	~~W~~

Figure 3-21 Dispatch Restrictions Arising from MUL Instruction

Figure 3-22 shows the dispatch restrictions on MTHI/MTLO instructions. A move to HI/LO cannot be pipelined with a move from HI/LO. Move to/from HI/LO cannot be pipelined with any integer multiply/divide instructions.

MTHI r1	D	R	X	C	W			
MFHI r6				D	R	X	C	W

MTHI r1	D	R	X	C	W				
MADD r6, r7				D	R	X_1	X_2	X_3	X_4

MADD r6, r7	D	R	X_1	X_2	X_3	X_4		
MTHI r1				D	R	X	C	W

Figure 3-22 Dispatch Restrictions on MTHI/MTLO Instructions

Figure 3-23 shows the dispatch restrictions arising from floating-point double-precision multiply instructions. Floating-point double-precision multiply and multiply-add instructions cause a one cycle dispatch bubble following the dispatch of the instruction; they can only be dispatched as the younger (later in program order) of up to two instructions.

MUL.D f1, f2, f3	D	R	M	N	I	J	K	Z
ADD r1, r6, r7			D	R	X	C	W	

Figure 3-23 Dispatch Restrictions Arising from Floating-Point Double-Precision Multiplies

Figure 3-24 shows the dispatch restrictions on CTC1 and CFC1 instructions. Moves to and from the floating-point control registers cannot be done until all outstanding floating-point instructions have reached the **K** stage.

Cop1/Cop1X	D	R	M	I	J	K	Z			
CTC1/CFC1						D	R	X	C	W

Figure 3-24 Dispatch Restrictions on CTC1 and CFC1

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

3.9 Dispatch of Privileged Instructions

Figure 3-25 shows the dispatch restrictions for privileged instructions. All privileged instructions are single issue.

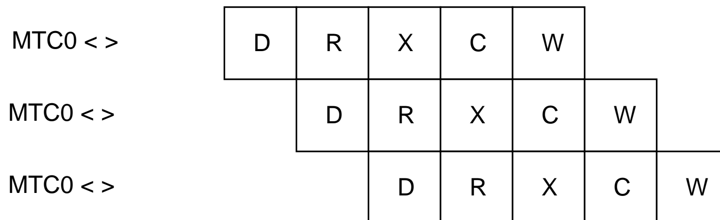


Figure 3-25 Dispatch Restrictions on Privileged Instructions

Figure 3-26 shows the dispatch restrictions for some MTC0 instructions. Moves to the *Status*, *Cause*, *EntryHi*, and *Debug* registers stall dispatch of all subsequent instructions until the **W** stage of the move.

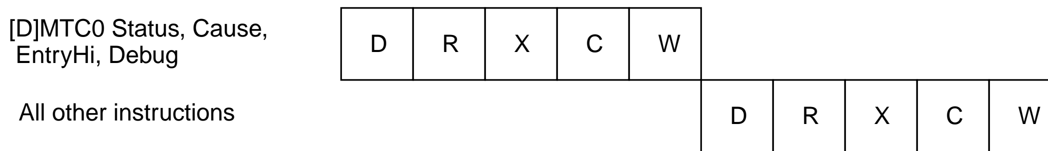


Figure 3-26 Dispatch Restrictions Arising from Certain MTC0 Instructions

Figure 3-27 shows some additional dispatch restrictions for some MTC0 instructions. Moves to the *Status* register and *Debug* register cannot be done until all outstanding floating-point instructions have reached the **K** stage.

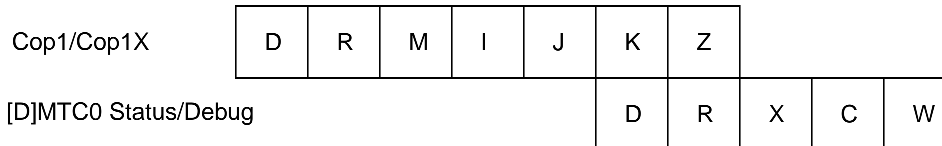


Figure 3-27 Dispatch Restrictions on Certain MTC0 Instructions

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Memory Management

The 20Kc processor supports three processor operating modes: user, kernel, and supervisor, and two addressing modes (32-bit and 64-bit). 32-bit mode is included for compatibility reasons. The 20Kc processor supports a 40-bit virtual address and a 36-bit physical address.

This chapter contains the following sections:

- Section 4.1, "Operating Modes"
- Section 4.2, "Other Modes"
- Section 4.3, "Processor Mode Selection"
- Section 4.4, "Addressing Modes"
- Section 4.5, "Address Space"
- Section 4.6, "Address Segments"
- Section 4.7, "Virtual Address Translation"
- Section 4.8, "Translation Lookaside Buffers"
- Section 4.9, "TLB Instructions"

4.1 Operating Modes

The processor operating modes control the access privileges in the system. This includes access to memory regions, registers in the processor, as well as instructions in the ISA. There are three processor operating modes:

- **Kernel Mode**: This mode enables the highest level of system privilege. Kernel Mode can access all memory and modify any processor register. The innermost core of the operating system runs in kernel mode. The processor is operating in Kernel Mode when the DM bit in the *Debug* register is a zero, and any of the following three conditions is true:
  - The KSU field in the *Status* register contains a value of 00_2
  - The EXL bit in the *Status* register is one
  - The ERL bit in the *Status* register is one

The processor enters Kernel Mode at power-up, or as the result of an interrupt, exception, or error. The processor leaves Kernel Mode and enters User Mode or Supervisor Mode when all of the above three conditions are false, usually as the result of an ERET instruction.

- **Supervisor Mode**: This mode is used by some layered operating systems. It has fewer privileges than kernel mode and is used for less critical sections of the operating system. The processor is operating in Supervisor Mode when all of the following conditions are true:
  - The DM bit in the *Debug* register is a zero
  - The KSU field in the *Status* register contains 01_2
  - The EXL and ERL bits in the *Status* register are both zero

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

- **User Mode**: This mode has the lowest system privilege and it is employed for user programs. The processor is operating in User Mode when all of the following conditions are true:
  - The DM bit in the *Debug* register is a zero
  - The KSU field in the *Status* register contains 10_2
  - The EXL and ERL bits in the *Status* register are both zero
- **Debug Mode**: The processor is operating in Debug Mode if the DM bit in the *Debug* register is a one. If the processor is running in Debug Mode, it has full access to all resources that are available to Kernel Mode operation.

4.2 Other Modes

4.2.1 64-bit Address Enable

Access to 64-bit addresses are enabled under any of the following conditions:

- A legal reference to a kernel address space occurs and the KX bit in the *Status* register is a one
- A legal reference to a supervisor address space occurs and the SX bit in the *Status* register is a one
- A legal reference to a user address space occurs and the UX bit in the *Status* register is a one

Note that the operating mode of the processor is not relevant to 64-bit address enables. That is, a reference to user address space made while the processor is operating in Kernel Mode is controlled by the state of the UX bit, not by the KX bit.

An attempt to reference a 64-bit address space when 64-bit addresses are not enabled results in an Address Error Exception (either AdEL or AdES, depending on the type of reference).

When a TLB miss occurs, the choice of the Exception Vector is also determined by the 64-bit address enable. If 64-bit addresses are not enabled for the reference, the TLB Refill Vector is used. If 64-bit addresses are enabled for the reference, the XTLB Refill Vector is used.

4.2.2 64-bit Operations Enable

Instructions that perform 64-bit operations are legal under any of the following conditions:

- The processor is operating in Kernel Model, Supervisor Mode, or Debug Mode, as described above
- The PX bit in the *Status* register is a one
- The processor is operating in User Mode, as described above, and the UX bit in the *Status* register is a one

An attempt to execute an instruction which performs 64-bit operations when such instructions are not enabled results in a Reserved Instruction Exception.

4.2.3 64-bit FPR Enable

Access to 64-bit FPRs is controlled by the FR bit in the *Status* register. If the FR bit is one, the FPRs are interpreted as 32 64-bit registers that may contain any data type. If the FR bit is zero, the FPRs are interpreted as 32 32-bit registers, any of which may contain a 32-bit data type (W, S). In this case, 64-bit data types are contained in even-odd pairs of registers.

The operation of the processor is UNPREDICTABLE under any of the following conditions:

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

- The FR bit is a zero and an odd register is referenced by an instruction whose data type is 64 bits
- The FR bit is a zero and a floating-point instruction is executed whose data type is L or PS
- 64-bit operations are not enabled, the FR bit is a one, and an instruction references the floating-point registers

4.3 Processor Mode Selection

The different processor modes (operating and addressing) are all specified in the *Status* Coprocessor 0 register. The specific fields involved are:

- **KSU**: The KSU field in the *Status* register (bits [4:3]) specifies the operating mode (00 = Kernel, 01 = Supervisor, 10 = User)
- **KX**: The KX bit in the *Status* register (bit [7]) controls the addressing mode for the kernel operating mode (0 = 32-bit, 1 = 64-bit)
- **SX**: The SX bit in the *Status* register (bit [6]) controls the addressing mode for the supervisor operating mode (0 = 32-bit, 1 = 64-bit)
- **UX**: The UX bit in the *Status* register (bit [5]) controls the addressing mode for the user operating mode (0 = 32-bit, 1 = 64-bit)
- **ERL**: When set to 1, the ERL (error level) bit in the *Status* register (bit [2]) indicates that the processor is in error mode and forces kernel mode operation
- **EXL**: When set to 1, the EXL (exception level) bit in the *Status* register (bit [1]) indicates that the processor is in exception mode and forces kernel mode operation

Table 4-1 shows how the different instruction sets and addressing modes are enabled by the *Status* register bits.

Table 4-1 Processor Modes

KX	SX	UX	KSU	ERL	EXL	Mode Enabled	Addressing Mode
-	-	0	10	0	0	User Mode	32
-	-	1	10	0	0	User Mode	64
-	0	-	01	0	0	Supervisor Mode	32
-	1	-	01	0	0	Supervisor Mode	64
0	-	-	00	0	0	Kernel Mode	32
1	-	-	00	0	0	Kernel Mode	64
0	-	-	-	0	1	Exception Level (Kernel Mode)	32
1	-	-	-	0	1	Exception Level (Kernel Mode)	64
0	-	-	-	1	X	Error Level (Kernel Mode)	32
1	-	-	-	1	X	Error Level (Kernel Mode)	64

4.4 Addressing Modes

In addition to operating modes, the processor supports two addressing modes: 32-bit mode and 64-bit mode. These modes determine whether the processor generates 32-bit virtual addresses or 64-bit virtual addresses. Physical addresses are not affected by the addressing mode.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

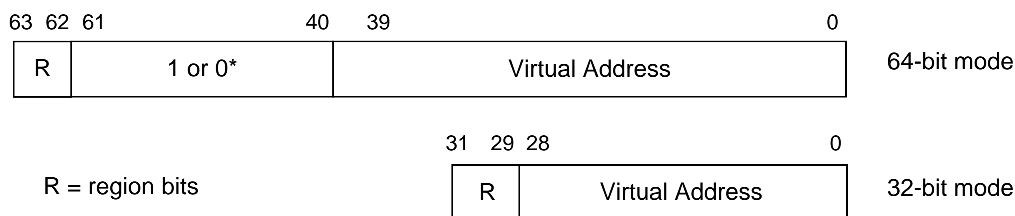
In 32-bit mode:

- Addresses are 32 bits wide
- The address space is divided into five regions
- The maximum user process size is 2 Gigabytes (2^{31})

In 64-bit mode:

- Addresses are 64 bits wide
- The address space is divided into eight regions
- The maximum user process size is 1 Terabyte (2^{40})

The 20Kc processor uses only a portion of the 64-bit address. Each address generated by the processor can be thought of consisting of two parts: region bits and virtual address. Figure 4-1 illustrates the relevant parts of an address for both 32-bit and 64-bit mode operations.



* Note that in 64-bit mode, bits [61:40] are not arbitrary and need to conform to the formats specified in the following sections. In most cases, they are just sign extensions of the MSB of the virtual address.

Figure 4-1 Address Components

4.5 Address Space

An *address space* is the range of all possible addresses that can be generated for a particular addressing mode. There is one 64-bit address space and one 32-bit compatibility address space that is mapped into a subset of the 64-bit address space.

A *segment* is a defined subset of an address space that has self-consistent reference and access behaviors to provide compatibility for 32-bit programs. The 20Kc processor provides a 2^{31}-byte compatibility address space separated into two non-contiguous ranges in which the upper 32 bits of the 64-bit address are the sign extension of bit 31. In the 20Kc processor, a 64-bit segment is part of the 64-bit address space and is 2^{40} bytes in size.

The 20Kc processor implements 36 physical address bits. Therefore, the size of the physical address space is 2^{36} bytes.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Figure 4-2 shows the layout of the address spaces, including the compatibility address space and the segmentation of each address space.

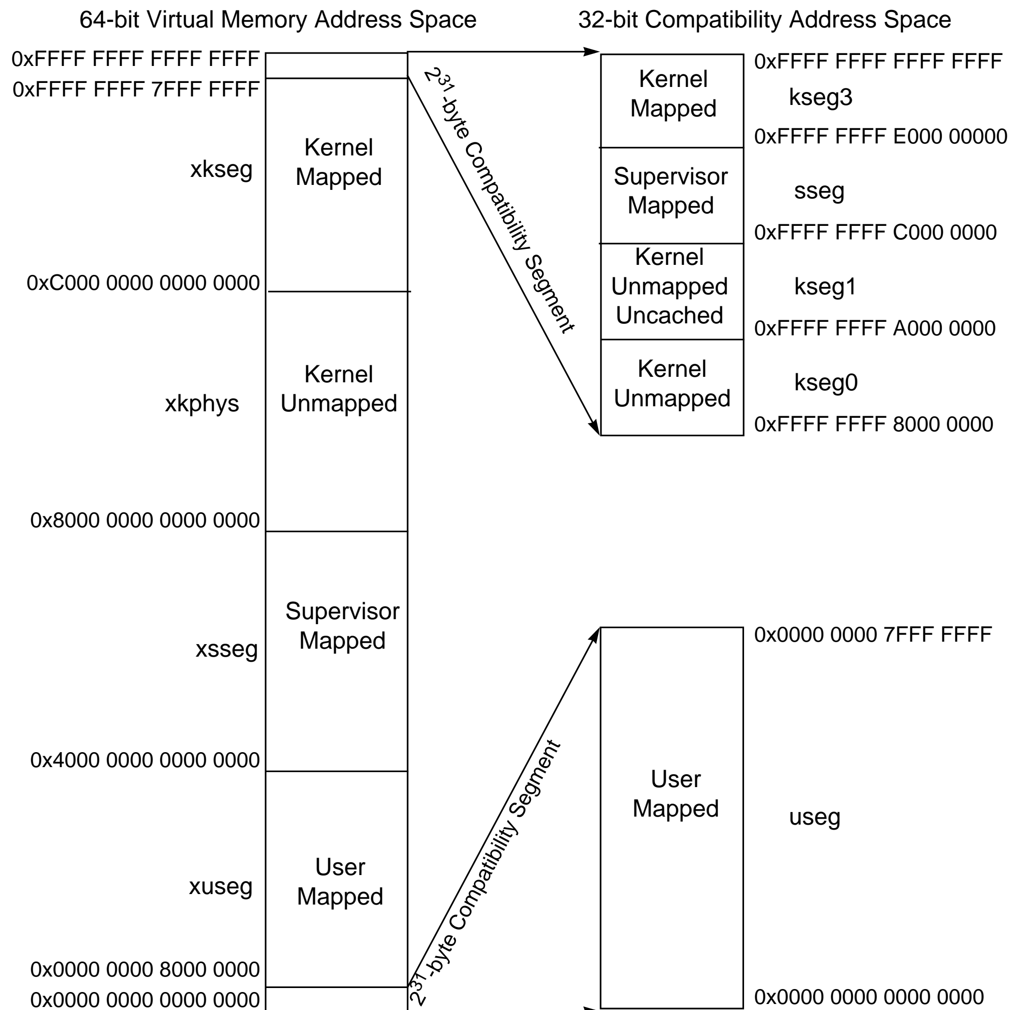


Figure 4-2 Virtual Address Space

Each segment of an address space is classified as *mapped* or *unmapped*. A mapped address is one that is translated through the TLB. An unmapped address is one that is not translated through the TLB and that provides a window into the lowest portion of the physical address space, starting at physical address zero, with a size corresponding to the size of the unmapped segment.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Additionally, the kseg1 segment is classified as *uncached*. References to uncached segments bypass all levels of the cache hierarchy and allow direct access to memory without any interference from the caches.

Table 4-2 Virtual Memory Address Spaces

$VA_{63..62}$	Segment Name(s)	Maximum Address Range	64-bit Address Enable	Associated with Mode	Reference Legal from Mode(s)	Actual Segment Size	Segment Type
11_2	kseg3	0xFFFF FFFF FFFF FFFF through 0xFFFF FFFF E000 0000	Always	Kernel	Kernel	2^{29} bytes	32-bit Compatibility
	sseg ksseg	0xFFFF FFFF DFFF FFFF through 0xFFFF FFFF C000 0000	Always	Supervisor	Supervisor Kernel	2^{29} bytes	32-bit Compatibility
	kseg1	0xFFFF FFFF BFFF FFFF through 0xFFFF FFFF A000 0000	Always	Kernel	Kernel	2^{29} bytes	32-bit Compatibility
	kseg0	0xFFFF FFFF 9FFF FFFF through 0xFFFF FFFF 8000 0000	Always	Kernel	Kernel	2^{29} bytes	32-bit Compatibility
	xkseg	0xFFFF FFFF 7FFF FFFF through 0xC000 0000 0000 0000	KX	Kernel	Kernel	$2^{40} - 2^{31}$ bytes	64-bit
10_2	xkphys	0xBFFF FFFF FFFF FFFF through 0x8000 0000 0000 0000	KX	Kernel	Kernel	Eight 2^{36} byte regions within the 2^{62} byte Segment	64-bit
01_2	xsseg xksseg	0x7FFF FFFF FFFF FFFF through 0x4000 0000 0000 0000	SX	Supervisor	Supervisor Kernel	2^{40} bytes	64-bit
00_2	xuseg xsuseg xkuseg	0x3FFF FFFF FFFF FFFF through 0x0000 0000 8000 0000	UX	User	User Supervisor Kernel	$(2^{40} - 2^{31})$ bytes	64-bit
	useg suseg kuseg	0x0000 0000 7FFF FFFF through 0x0000 0000 0000 0000	Always	User	User Supervisor Kernel	2^{31} bytes	32-bit Compatibility

Each segment of an address space is associated with one of the three processor operating modes: User, Supervisor, or Kernel. A segment that is associated with a particular mode is accessible if the processor is running in that mode or a more privileged mode. For example, a segment associated with User Mode is accessible when the processor is running in User, Supervisor, or Kernel Modes. A segment is not accessible if the processor is running in a less privileged mode than that associated with the segment. For example, a segment associated with Supervisor Mode is not accessible when the processor is running in User Mode and such a reference results in an Address Error exception. The "Reference Legal from Mode(s)" column in Table 4-2 lists the modes from which each segment can be legally referenced.

If a segment has more than one name, each name denotes the mode from which the segment is referenced. For example, the segment name "useg" denotes a reference from user mode, while the segment name "kuseg" denotes a reference to the same segment from kernel mode.

As shown in the "Segment Type" column of Table 4-2, references to 64-bit segments are enabled only if the appropriate 64-bit Address Enable is on as indicated by the "64-bit Address Enable" column of Table 4-2. References to 32-bit Compatibility segments are always enabled.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

4.5.1 Access Control as a Function of Address and Operating Mode

Table 4-3 lists the actions taken by the processor for each section of the 64-bit address space as a function of the processor operating mode. The selection of TLB Refill vector and other special-cased behavior is also listed for each reference.

Table 4-3 Address Space Access and TLB Refill Selection as a Function of Operating Mode

Virtual Address Range			Action when Referenced from Operating Mode		
Symbolic	20Kc **Implementation** ***SEGBITS* = 40, *PABITS* = 36**	**Segment Name(s)**	**User Mode[1]**	**Supervisor Mode**	**Kernel Mode**
0xFFFF FFFF FFFF FFFF through 0xFFFF FFFF E000 0000	0xFFFF FFFF FFFF FFFF through 0xFFFF FFFF E000 0000	kseg3	Address Error	Address Error	Mapped Refill Vector: TLB (KX=0) XTLB(KX=1)
0xFFFF FFFF DFFF FFFF through 0xFFFF FFFF C000 0000	0xFFFF FFFF DFFF FFFF through 0xFFFF FFFF C000 0000	sseg ksseg	Address Error	Mapped Refill Vector[2]: TLB (KX=0) XTLB(KX=1)	Mapped Refill Vector[2]: TLB (KX=0) XTLB(KX=1)
0xFFFF FFFF BFFF FFFF through 0xFFFF FFFF A000 0000	0xFFFF FFFF BFFF FFFF through 0xFFFF FFFF A000 0000	kseg1	Address Error	Address Error	Unmapped, Uncached
0xFFFF FFFF 9FFF FFFF through 0xFFFF FFFF 8000 0000	0xFFFF FFFF 9FFF FFFF through 0xFFFF FFFF 8000 0000	kseg0	Address Error	Address Error	Unmapped See Section 4.5.2, "Address Translation and Cache Coherency Attributes for the kseg0 and kseg1 Segments" on page 40
0xFFFF FFFF 7FFF FFFF through 0xC000 0000 0000 0000 + $2^{SEGBITS}$ - 2^{31}	0xFFFF FFFF 7FFF FFFF through 0xC000 00FF 8000 0000		Address Error	Address Error	Address Error
0xC000 0000 0000 0000 + $2^{SEGBITS}$ - 2^{31} - 1 through 0xC000 0000 0000 0000	0xC000 00FF 7FFF FFFF through 0xC000 0000 0000 0000	xkseg	Address Error	Address Error	Address Error if KX = 0 Mapped if KX = 1 Refill Vector: XTLB
0xBFFF FFFF FFFF FFFF through 0x8000 0000 0000 0000	0xBFFF FFFF FFFF FFFF through 0x8000 0000 0000 0000	xkphys	Address Error	Address Error	Address Error if KX = 0 or in certain address ranges within the Segment Unmapped
0x7FFF FFFF FFFF FFFF through 0x4000 0000 0000 0000 + $2^{SEGBITS}$	0x7FFF FFFF FFFF FFFF through 0x4000 0100 0000 0000		Address Error	Address Error	Address Error

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 4-3 Address Space Access and TLB Refill Selection as a Function of Operating Mode (Continued)

Virtual Address Range		Segment Name(s)	Action when Referenced from Operating Mode		
Symbolic	20Kc **Implementation** ***SEGBITS* = 40, *PABITS* = 36**		**User Mode**[1]	**Supervisor Mode**	**Kernel Mode**
0x4000 0000 0000 0000 + $2^{SEGBITS}$ - 1 through 0x4000 0000 0000 0000	0x4000 00FF FFFF FFFF through 0x4000 0000 0000 0000	xsseg xksseg	Address Error	Address Error if SX = 0 Mapped if SX = 1 Refill Vector: XTLB	Address Error if SX = 0 Mapped if SX = 1 Refill Vector: XTLB
0x3FFF FFFF FFFF FFFF through 0x0000 0000 0000 0000 + $2^{SEGBITS}$	0x3FFF FFFF FFFF FFFF through 0x0000 0100 0000 0000		Address Error	Address Error	Address Error
0x0000 0000 0000 0000 + $2^{SEGBITS}$ - 1 through 0x0000 0000 8000 0000	0x0000 00FF FFFF FFFF through 0x0000 0000 8000 0000	xuseg xsuseg xkuseg	Address Error if UX = 0 Mapped if UX = 1 Refill Vector: XTLB	Address Error if UX = 0 Mapped if UX = 1 Refill Vector: XTLB	Address Error if UX = 0 Mapped if UX = 1 Refill Vector: XTLB
0x0000 0000 7FFF FFFF through 0x0000 0000 0000 0000	0x0000 0000 7FFF FFFF through 0x0000 0000 0000 0000	useg suseg kuseg	Mapped Refill Vector: TLB (UX=0) XTLB(UX=1)	Mapped Refill Vector: TLB (UX=0) XTLB(UX=1)	Unmapped if $Status_{ERL}$=1 See Section 4.5.4, "Address Translation for the kuseg Segment when StatusERL = 1" on page 42 Mapped if $Status_{ERL}$=0 Refill Vector: TLB (UX=0) XTLB(UX=1)

1. See Section 4.5.6, "Special Behavior for Data References in User Mode with StatusUX = 0" on page 43 for the special treatment of the address for data references when the processor is running in User Mode and the UX bit is zero.
2. Note that the Refill Vector for references to sseg/ksseg is determined by the state of the KX bit, not the SX bit. This simplifies the processor implementation by allowing them to treat the entire quadrant of the address space in which $VA_{63..62}$ are 11_2 in the same manner, as well as simplifying operating system software design which does not use Supervisor Mode.

4.5.2 Address Translation and Cache Coherency Attributes for the kseg0 and kseg1 Segments

The kseg0 and kseg1 unmapped segments provide a window into the least significant 2^{29} bytes of physical memory, and, as such, are not translated using the TLB or other address translation unit. The cache coherency attribute of the kseg0 segment is supplied by the K0 field of the *Config* register. The cache coherency attribute for the kseg1 segment is always

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

uncached. Table 4-4 describes how this transformation is done and the source of the cache coherency attributes for each segment.

Table 4-4 Address Translation and Cache Coherency Attributes for the kseg0 and kseg1 Segments

Segment Name	Virtual Address Range	Generates Physical Address	Cache Attribute
kseg1	0xFFFF FFFF BFFF FFFF through 0xFFFF FFFF A000 0000	0x0000 0000 1FFF FFFF through 0x0000 0000 0000 0000	Uncached
kseg0	0xFFFF FFFF 9FFF FFFF through 0xFFFF FFFF 8000 0000	0x0000 0000 1FFF FFFF through 0x0000 0000 0000 0000	From the K0 field of *Config* Register

4.5.3 Address Translation and Cache Coherency Attributes for the xkphys Segment

The xkphys unmapped segment is actually composed of eight address ranges, each of which provides a window into the entire 2^{PABITS} bytes of physical memory and, as such, is not translated using the TLB or other address translation unit. For this segment, the cache coherency attribute is taken from $VA_{61..59}$ and has the same encoding as that shown in Table 4-5. An Address Error Exception occurs if $VA_{58..PABITS}$ are non-zero. If no Address Error Exception occurs, the physical address is taken from $VA_{PABITS-1..0}$. Figure 4-3 shows the interpretation of the various fields of the virtual address when referencing the xkphys segment.

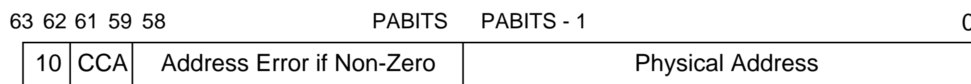


Figure 4-3 Address Interpretation for the xkphys Segment

Table 4-5 Address Translation and Cacheability Attributes for the xkphys Segment

Virtual Address Range		Generates Physical Address	Cache Attribute
Symbolic	20Kc **Implementation** ***PABITS* = 36**		
0xBFFF FFFF FFFF FFFF through 0xB800 0000 0000 0000 + 2^{PABITS}	0xBFFF FFFF FFFF FFFF through 0xB800 0010 0000 0000	Address Error	N/A
0xB800 0000 0000 0000 + 2^{PABITS} - 1 through 0xB800 0000 0000 0000	0xB800 000F FFFF FFFF through 0xB800 0000 0000 0000	0x0000 0000 0000 0000 + 2^{PABITS} - 1 through 0x0000 0000 0000 0000	Uses encoding 7 of Table 4-6 on page 49
0xB7FF FFFF FFFF FFFF through 0xB000 0000 0000 0000 + 2^{PABITS}	0xB7FF FFFF FFFF FFFF through 0xB000 0010 0000 0000	Address Error	N/A
0xB000 0000 0000 0000 + 2^{PABITS} - 1 through 0xB000 0000 0000 0000	0xB000 000F FFFF FFFF through 0xB000 0000 0000 0000	0x0000 0000 0000 0000 + 2^{PABITS} - 1 through 0x0000 0000 0000 0000	Uses encoding 6 of Table 4-6 on page 49
0xAFFF FFFF FFFF FFFF through 0xA800 0000 0000 0000 + 2^{PABITS}	0xAFFF FFFF FFFF FFFF through 0xA800 0010 0000 0000	Address Error	N/A
0xA800 0000 0000 0000 + 2^{PABITS} - 1 through 0xA800 0000 0000 0000	0xA800 000F FFFF FFFF through 0xA800 0000 0000 0000	0x0000 0000 0000 0000 + 2^{PABITS} - 1 through 0x0000 0000 0000 0000	Uses encoding 5 of Table 4-6 on page 49

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 4-5 Address Translation and Cacheability Attributes for the xkphys Segment (Continued)

Virtual Address Range		Generates Physical Address	Cache Attribute
Symbolic	**20Kc Implementation** *PABITS* **= 36**		
0xA7FF FFFF FFFF FFFF through 0xA000 0000 0000 0000 + 2^{PABITS}	0xA7FF FFFF FFFF FFFF through 0xA000 0010 0000 0000	Address Error	N/A
0xA000 0000 0000 0000 + 2^{PABITS} - 1 through 0xA000 0000 0000 0000	0xA000 000F FFFF FFFF through 0xA000 0000 0000 0000	0x0000 0000 0000 0000 + 2^{PABITS} - 1 through 0x0000 0000 0000 0000	Uses encoding 4 of Table 4-6 on page 49
0x9FFF FFFF FFFF FFFF through 0x9800 0000 0000 0000 + 2^{PABITS}	0x9FFF FFFF FFFF FFFF through 0x9800 0010 0000 0000	Address Error	N/A
0x9800 0000 0000 0000 + 2^{PABITS} - 1 through 0x9800 0000 0000 0000	0x9800 000F FFFF FFFF through 0x9800 0000 0000 0000	0x0000 0000 0000 0000 + 2^{PABITS} - 1 through 0x0000 0000 0000 0000	Cacheable (see encoding 3 of Table 4-6 on page 49)
0x97FF FFFF FFFF FFFF through 0x9000 0000 0000 0000 + 2^{PABITS}	0x97FF FFFF FFFF FFFF through 0x9000 0010 0000 0000	Address Error	N/A
0x9000 0000 0000 0000 + 2^{PABITS} - 1 through 0x9000 0000 0000 0000	0x9000 000F FFFF FFFF through 0x9000 0000 0000 0000	0x0000 0000 0000 0000 + 2^{PABITS} - 1 through 0x0000 0000 0000 0000	Uncached (see encoding 2 of Table 4-6 on page 49)
0x8FFF FFFF FFFF FFFF through 0x8800 0000 0000 0000 + 2^{PABITS}	0x8FFF FFFF FFFF FFFF through 0x8800 0010 0000 0000	Address Error	N/A
0x8800 0000 0000 0000 + 2^{PABITS} - 1 through 0x8800 0000 0000 0000	0x8800 000F FFFF FFFF through 0x8800 0000 0000 0000	0x0000 0000 0000 0000 + 2^{PABITS} - 1 through 0x0000 0000 0000 0000	Uses encoding 1 of Table 4-6 on page 49
0x87FF FFFF FFFF FFFF through 0x8000 0000 0000 0000 + 2^{PABITS}	0x87FF FFFF FFFF FFFF through 0x8000 0010 0000 0000	Address Error	N/A
0x8000 0000 0000 0000 + 2^{PABITS} - 1 through 0x8000 0000 0000 0000	0x8000 000F FFFF FFFF through 0x8000 0000 0000 0000	0x0000 0000 0000 0000 + 2^{PABITS} - 1 through 0x0000 0000 0000 0000	Uses encoding 0 of Table 4-6 on page 49

4.5.4 Address Translation for the kuseg Segment when $Status_{ERL}$ = 1

To provide support for the cache error handler, the kuseg segment becomes an unmapped, uncached segment, similar to the kseg1 segment, if the ERL bit is set in the *Status* register. This allows the cache error exception code to operate uncached using GPR *R0* as a base register to save other GPRs before use.

The 20Kc processor transforms the kuseg segment in its entirety. In addition, when the UX bit is a one in the *Status* register, the range of addresses between 2^{32} and 2^{PABITS} – 1 are also treated as an unmapped, uncached segment. However, accesses to xkuseg addresses between 2^{PABITS} and $2^{SEGBITS}$ – 1 would result in address error exceptions.

4.5.5 Special Behavior for the kseg3 Segment when $Debug_{DM}$ = 1

The 20Kc processor implements the EJTAG functionality, and consequently treats the virtual address range 0xFFFF FFFF FF20 0000 through 0xFFFF FFFF FF3F FFFF inclusive as a special memory-mapped region in Debug Mode. Accordingly, the processor:

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

- Explicitly range checks the address range as given and not assume that the entire region between 0xFFFF FFFF FF20 0000 and 0xFFFF FFFF FFFF FFFF is included in the special memory-mapped region.
- Does not enable the special EJTAG mapping for this region in any mode other than in EJTAG Debug Mode.

Even in Debug Mode, normal memory rules might apply in some cases. Refer to Chapter 11, “EJTAG Debug Support,” for details on this mapping.

4.5.6 Special Behavior for Data References in User Mode with $Status_{UX}$ = 0

When the processor is running in User Mode, legal addresses have VA_{31} equal zero, and the 32-bit virtual address is sign-extended (really zero-extended because VA_{31} is zero) into a full 64-bit address. As such, one would expect that the normal address bounds checks on the sign-extended 64-bit address would be sufficient. Unfortunately, there are cases in which a program running on a 32-bit processor can generate a data address that is legal in 32 bits, but which is not appropriately sign-extended into 64 bits. For example, consider the following code example:

```
la          r10, 0x80000000
lw          r10, -4(r10)
```

The results of executing this address calculation on 32-bit and 64-bit processors with UX equal zero is shown below:

32-bit Processor	64-bit Processor
0x8000 0000	0xFFFF FFFF 8000 0000
+ 0xFFFF FFFC	+ 0xFFFF FFFF FFFF FFFC
0x7FFF FFFC	0xFFFF FFFF 7FFF FFFC

On a 32-bit processor, the result of this address calculation results in a valid useg address. On a 64-bit processor such as the 20Kc, however, the sign-extended address in the base register is added to the sign-extended displacement as a 64-bit quantity that results in a carry-out of bit 31, producing a data address that is not properly sign extended.

To provide backward compatibility with 32-bit User Mode code, the 20Kc implements the following special case for data references (and explicitly *not* for instructions references) when the processor is running in User Mode and the UX bit is zero in the *Status* register.

The effective address calculated by a load, store, or prefetch instruction is sign extended from bit 31 into bits [63:32] of the full 64-bit address, ignoring the previous contents of bits [63:32] of the address, before the final address is checked for address error exceptions or used to access the TLB or cache. This special-case behavior is not performed for instruction references.

This results in a properly zero-extended address for all legal data addresses (which cleans up the address shown in the example above), and results in a properly sign-extended address for all illegal data addresses (those in which bit 31 is a one). Code running in Debug Mode, Kernel Mode, or Supervisor Mode with the appropriate 64-bit address enable off is prohibited from generating an effective address in which there is a carry-out of bit 31. If such an address is produced, the operation of the instruction generating such an address is **UNPREDICTABLE**.

4.6 Address Segments

The following sections discuss the User, Kernel, and Supervisor address segments.

4.6.1 User Mode Segments

User Mode operation is in effect when the *Status* register contains the following bit values:

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

- KSU = 10_2
- ERL = 0
- EXL = 0

There is a single uniform address space called User segment (*useg* in 32-bit mode) or Extended User segment *xuseg* in 64-bit mode). Its size is:

- 2 Gigabytes in 32-bit mode
- 1 Terabyte in 64-bit mode

Figure 4-4 shows the address spaces for User Mode.

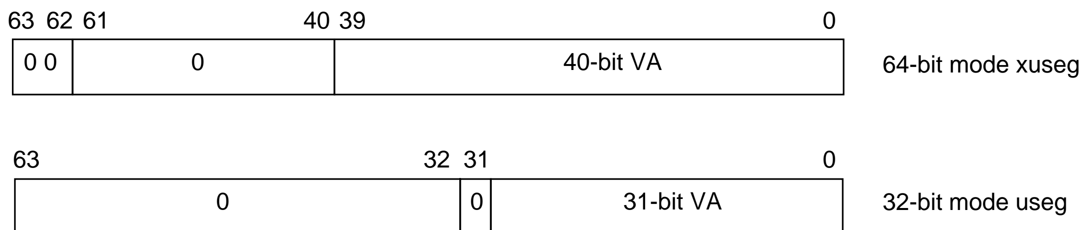


Figure 4-4 User Mode Addressing

4.6.1.1 32-bit User Mode (useg)

In User Mode, when UX = 0 in the *Status* register, User Mode addressing is compatible with the 32-bit addressing model shown in Figure 4-4, and a 2-Gigabyte user address space is available labelled *useg*.

All valid User Mode virtual addresses have their most-significant bit cleared to 0. Any attempt to reference an address with the most significant bit set while in User Mode causes an Address Error exception.

The system maps all references to useg through the TLB, and bit settings within the TLB entry for the page determine the cacheability attribute of a reference.

4.6.1.2 64-bit User Mode (xuseg)

In User Mode, when UX = 1 in the *Status* register, User Mode addressing is extended to the 64-bit model shown in Figure 4-4. In 64-bit User Mode, the processor provides a single, uniform virtual address space of 2^{40} bytes (1 Terabyte), labelled xuseg.

All valid User Mode virtual addresses have bits [63:40] equal to 0. An attempt to reference an address with bits [63:40] not equal to 0 causes an Address Error exception.

4.6.2 Supervisor Mode Segments

Supervisor Mode operation is in effect when the *Status* register contains the following bit values:

- KSU = 01_2
- ERL = 0
- EXL = 0

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Figure 4-5 shows the address spaces for Supervisor Mode.

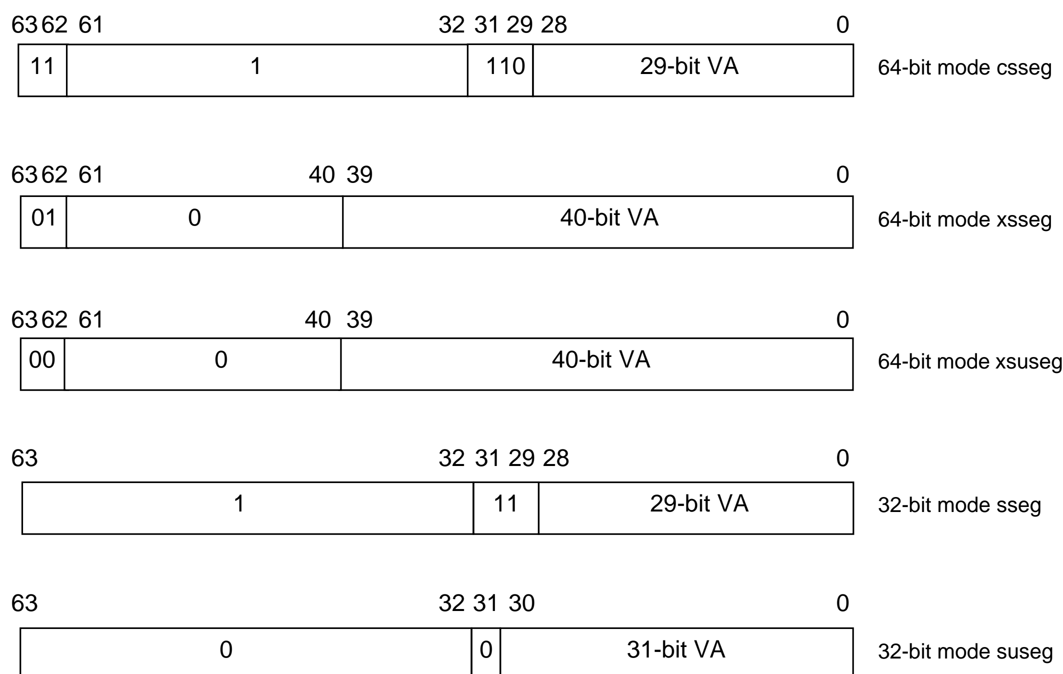


Figure 4-5 Supervisor Mode Addressing

All supervisor segments are mapped by the TLB. In addition, for every user segment, there is a corresponding supervisor segment which covers it. The different supervisor segments are briefly described below.

4.6.2.1 32-bit Supervisor Mode, User Space (suseg)

In Supervisor Mode, when SX = 0 in the Status register and the most significant bit of the 32-bit virtual address is set to 0, the suseg virtual address space is selected. It covers the full 2^{31} bytes (2 GBytes) of the current user address space.

4.6.2.2 32-bit Supervisor Mode, Supervisor Space (sseg)

In Supervisor Mode, when SX = 0 in the *Status* register and the three most-significant bits of the 32-bit virtual address are 110_2, the sseg virtual address space is selected. It is a separate space (does not cover any user space) with a size of 2^{29} bytes (0.5 GByte).

4.6.2.3 64-bit Supervisor Mode, User Space (xsuseg)

In Supervisor Mode, when SX = 1 in the *Status* register and bits [63:62] of the 64-bit virtual address are set to 00_2, the xsuseg virtual address space is selected. It covers the full 2^{40} bytes (1 Terabyte) of the current user address space.

4.6.2.4 64-bit Supervisor Mode, Extended Supervisor Space (xsseg)

In Supervisor Mode, when SX = 1 in the *Status* register and bits [63:62] of the 64-bit virtual address are set to 01_2, the xsseg virtual address space is selected. It is a separate address space (does not cover any user space) with a size of 2^{40} bytes (1 Terabyte).

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

4.6.2.5 64-bit Supervisor Mode, Compatible Supervisor Space (csseg)

In Supervisor Mode, when SX = 1 in the *Status* register and bits [63:62] of the 64-bit virtual address are set to 11_2, the csseg virtual address space is selected. It is a compatible space to the sseg 32-bit mode segment. This means that addressing of the csseg is compatible with the addressing of sseg, as shown in Figure 4-5.

4.6.3 Kernel Mode Segments

Kernel Mode operation is in effect when the *Status* register contains any one of the following bit values:

- KSU = 00_2
- ERL = 1
- EXL = 1

The processor enters kernel mode whenever an exception or error is detected, and remains in that mode until an exception return instruction is executed (ERET). The ERET instruction restores the processor state to the one that existed prior to the exception.

Kernel Mode operation allows access to all the defined regions in the memory map. As mentioned earlier, there are five regions defined for 32-bit mode, and those span the complete 32-bit address space. Therefore, any 32-bit address is always valid in kernel mode. However, in 64-bit addressing mode, there are eight regions defined, and those do not cover the entire 64-bit address space. Therefore, even in kernel mode, an address that falls outside of the defined segments causes an Address Error exception.

Figure 4-6 shows the address spaces for Kernel Mode in 32-bit addressing.

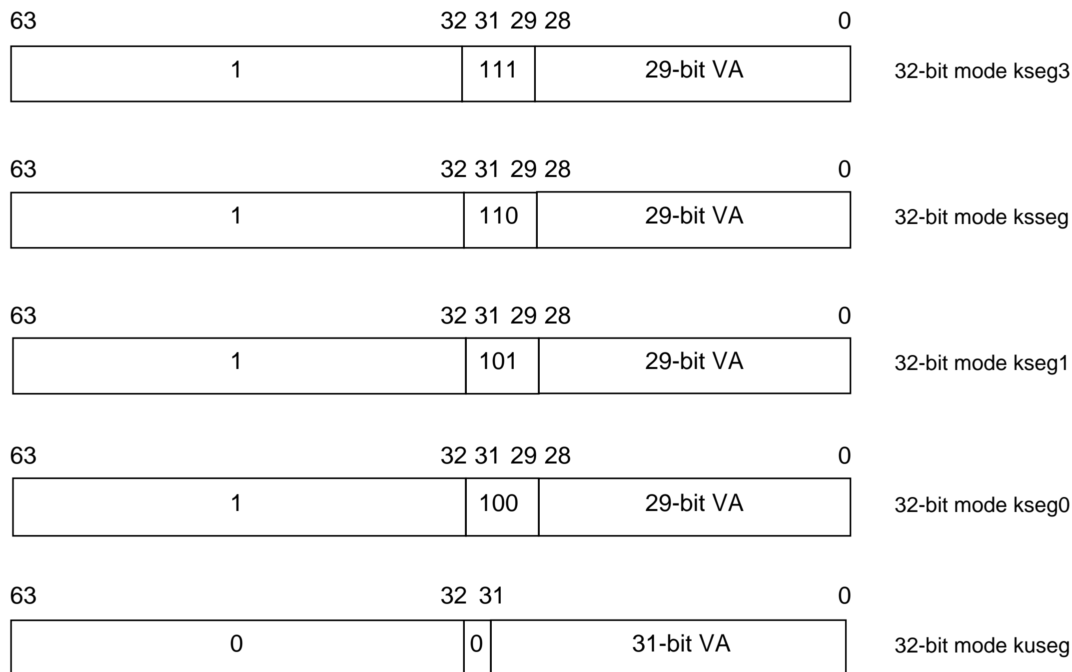


Figure 4-6 Kernel Mode Addressing (32-bit)

Not all kernel segments are mapped by the TLB. In addition, every user or supervisor segment is covered by a corresponding kernel segment. The different kernel segments in 32-bit addressing mode are briefly described below.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

4.6.3.1 32-bit Kernel Mode, User Space (kuseg)

In Kernel Mode, when KX = 0 in the *Status* register and the most significant bit of the 32-bit virtual address is set to 0, the kuseg virtual address space is selected. It covers the full 2^{31} bytes (2 GBytes) of the current user address space.

4.6.3.2 32-bit Kernel Mode, Kernel Space 0 (kseg0)

In Kernel Mode, when KX = 0 in the *Status* register and the three most-significant bits of the 32-bit virtual address are 100_2, the kseg0 virtual address space is selected. It is a separate space (does not cover any user space) with a size of 2^{29} bytes (0.5 GByte). References to *kseg0* are not mapped through the TLB; the physical address is selected by subtracting 0x8000_0000 from the virtual address. The K0 field of the *Config* Coprocessor 0 register determines the cacheability attributes of this segment.

4.6.3.3 32-bit Kernel Mode, Kernel Space 1 (kseg1)

In Kernel Mode, when KX = 0 in the *Status* register and the three most-significant bits of the 32-bit virtual address are 101_2, the *kseg1* virtual address space is selected. It is a separate space (does not cover any user space) with a size of 2^{29} bytes (0.5 GByte). References to *kseg1* are not mapped through the TLB; the physical address is selected by subtracting 0xA000_0000 from the virtual address. All accesses to the *kseg1* space are uncached.

4.6.3.4 32-bit Kernel Mode, Supervisor Space (ksseg)

In Kernel Mode, when KX = 0 in the *Status* register and the three most-significant bits of the 32-bit virtual address are 110_2, the ksseg virtual address space is selected. It has a size of 2^{29} bytes (0.5 GByte) and fully covers the supervisor segment sseg. References to ksseg are mapped through the TLB.

4.6.3.5 32-bit Kernel Mode, Kernel Space 3 (kseg3)

In Kernel Mode, when KX=0 in the *Status* register and the three most-significant bits of the 32-bit virtual address are 111_2, the kseg3 virtual address space is selected. It is a separate space (does not cover any user space) with a size of 2^{29} bytes (1/2 GByte). References to kseg3 are mapped through the TLB.

In 64-bit addressing mode, there are eight total kernel regions. However, most of those just cover existing spaces for user and supervisor modes. Figure 4-7 illustrates the different kernel regions in 64-bit mode, each of which is briefly described below.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

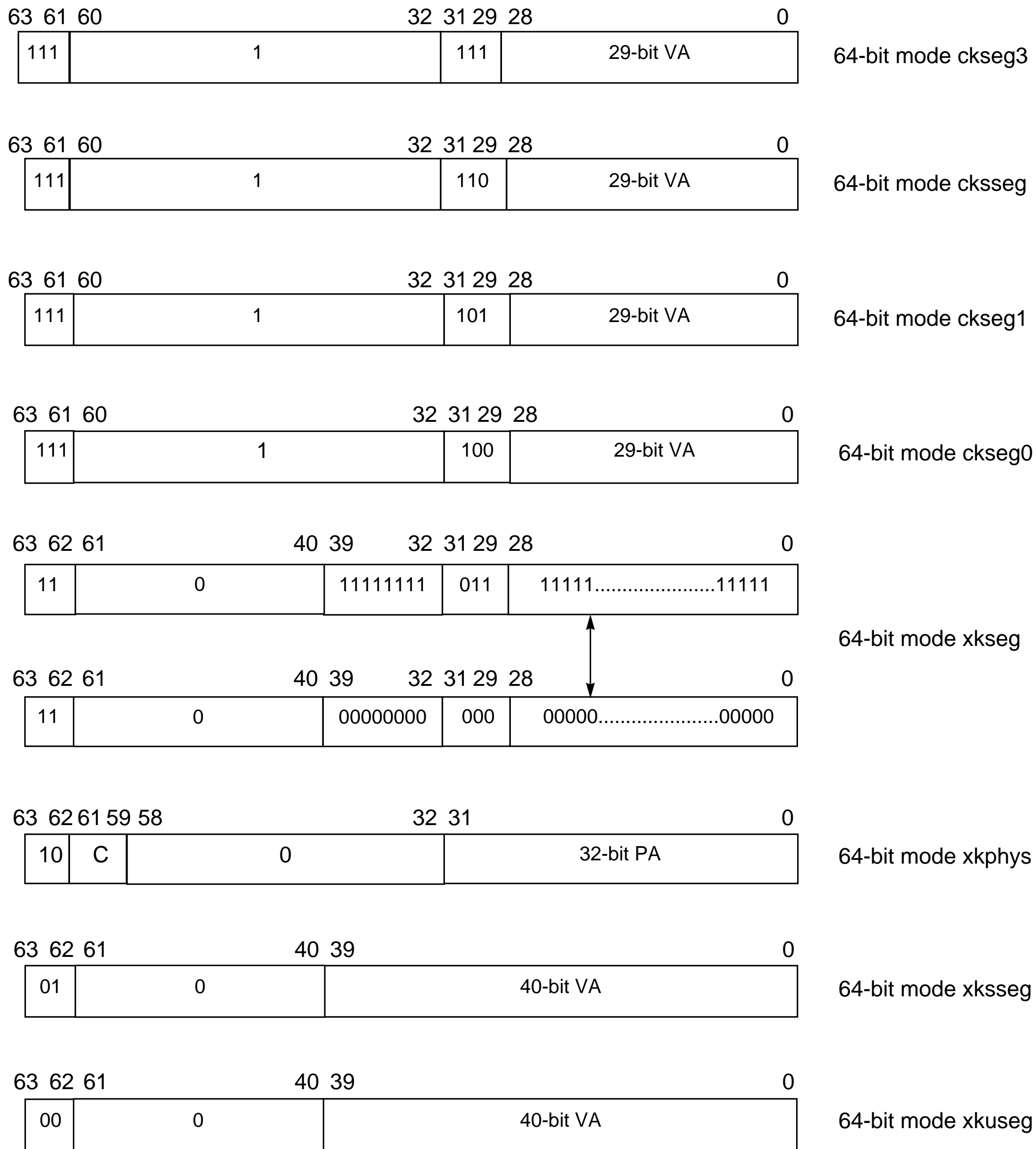


Figure 4-7 Kernel Mode Addressing (64-bit)

4.6.3.6 64-bit Kernel Mode, User Space (xkuseg)

In Kernel Mode, when KX = 1 in the *Status* register and bits [63:62] of the 64-bit virtual address are set to 00_2, the xkuseg virtual address space is selected. It covers the full 2^{40} bytes (1 Terabyte) of the extended user address space.

Note that when ERL = 1 in the *Status* register, the user address region becomes a 2^{31}-byte unmapped uncached address segment.

4.6.3.7 64-bit Kernel Mode, Supervisor Space (xksseg)

In Kernel Mode, when KX = 1 in the *Status* register and bits [63:62] of the 64-bit virtual address are set to 01_2, the xksseg virtual address space is selected. It covers the full 2^{40} bytes (1 Terabyte) of the extended supervisor address space.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

4.6.3.8 64-bit Kernel Mode, Physical Spaces (xkphys)

In Kernel Mode, when KX = 1 in the *Status* register and bits [63:62] of the 64-bit virtual address is set to 10_2, the xkphys address space is selected. The xkphys space is actually a set of eight kernel physical spaces. Each of these physical spaces is 4 GBytes in size.

References to this space are not mapped. Instead, the physical address is taken directly from bits [31:0] of the virtual address. Cacheability attributes are also contained in bits [61:59] of the virtual address. Access with address bits [58:32] not equal to zero cause an Address Error Exception. Table 4-6 lists the xkphys spaces.

Table 4-6 xkphys Spaces

VA[61:59]	Cache Algorithm	Starting Address
0	Cacheable, Non-Coherent, Write Through, No Write Allocate	0x8000 0000 0000 0000
1	Reserved	0x8800 0000 0000 0000
2	Uncached	0x9000 0000 0000 0000
3	Cacheable, Non-Coherent	0x9800 0000 0000 0000
4	Cacheable, Coherent, Exclusive	0xA000 0000 0000 0000
5	Reserved	0xA800 0000 0000 0000
6	Reserved	0xB000 0000 0000 0000
7	Uncached Accelerated	0xB800 0000 0000 0000

4.6.3.9 64-bit Kernel Mode, Extended Kernel Segment (xkseg)

In Kernel Mode, when KX=1 in the *Status* register and bits [63:62] of the 64-bit virtual address are set to 11_2, and bits [31:29] of the virtual address equal one of {000_2, 001_2, 010_2, 011_2}, the xkseg pace is selected. All accesses to this space are mapped through the TLB.

4.6.3.10 64-bit Kernel Mode, Compatibility Spaces (ckseg0, ckseg1, cksseg, ckseg3)

In Kernel Mode, when KX = 1 in the *Status* register and bits [63:61] of the 64-bit virtual address are set to 111_2, and bits [31:29] of the virtual address equal one of {100_2, 101_2, 110_2, 111_2}, one of the 512-MByte compatibility spaces is selected.

- *ckseg0*: This space is selected when bits [31:29] = 100_2. This space is an unmapped region, compatible with the 32-bit address model kseg0. The K0 field of the *Config* register controls cacheability and coherence.
- *ckseg1*: This space is selected when bits [31:29] = 101_2. This space is an unmapped, uncached region. It is compatible with the address model kseg1.
- *cksseg*: This space is selected when bits [31:29] = 110_2. This space is the current supervisor space, and is compatible with the address model ksseg.
- *ckseg3*: This space is selected when bits [31:29] = 111_2. This space is a kernel virtual space, compatible with the address model kseg3. All accesses to this space are mapped through the TLB.

4.6.4 Debug Mode

Debug Mode address space is identical to kernel mode address space with respect to unmapped areas. Mapped areas are only accessible if a valid translation is resident in the TLB. In parallel with this, a debug segment dseg co-exists in the virtual address range 0xFF20_0000 to 0xFF3F_FFFF. The layout is shown in Figure 4-8.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

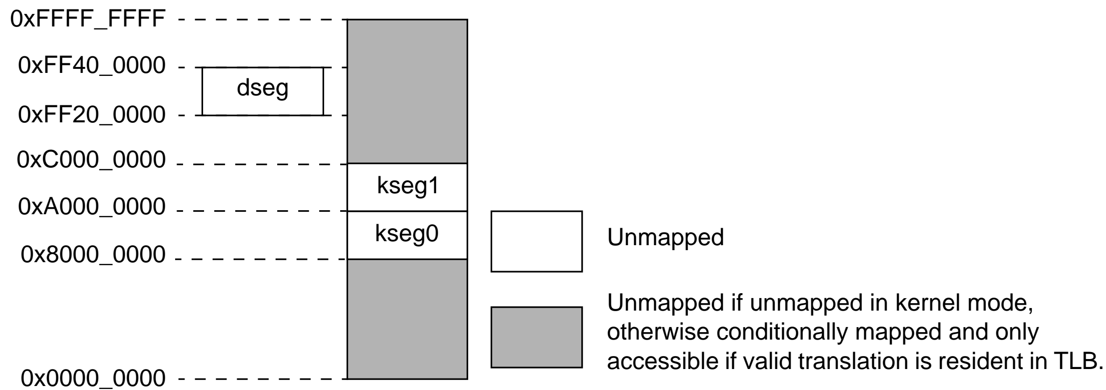


Figure 4-8 Debug Mode Virtual Address Space

Accesses to memory that would normally cause an exception if tried from kernel mode cause the core to re-enter debug mode via a debug mode exception. This includes accesses usually causing a TLB exception, with the result that such accesses are not handled by the usual memory management routines.

The unmapped kseg0 and kseg1 segments from kernel mode address space are available from debug mode, which allows the debug handler to be executed from uncached and unmapped memory.

4.7 Virtual Address Translation

Programs can use either physical or virtual memory addresses:

- Physical addresses are used directly to access specific locations in memory and are by nature fixed.
- Virtual addresses are logical values only, and must be translated to physical addresses before they can be used to access memory.

Translation is essential for multitasking systems, because it allows the operating system to load programs and data in memory independently of the virtual address used. Translation is also used for memory protection, preventing user programs from interfering with each other.

4.7.1 Page Size Support

The 20Kc processor supports the following page sizes: 4 KBytes, 16 KBytes, 64 KBytes, 256 KBytes, 1 MBytes, 4 MBytes, and 16 MBytes. The virtual address bits that select a page are called the *page address* and are translated. The lower bits that select a byte within the page are called the *offset* and are not translated.

For a 4-KByte page, the *offset* consists of bits [11:0]. For the largest page size (16 MBytes), the *offset* increases to bits [23:0]. Accordingly, the page address consists of bits [63:12] for the smallest page, and [63:24] for the largest one. The upper two virtual address bits [63:62] select between user, supervisor, and kernel spaces. The intermediate address bits [61:40] must either be all zeros or all ones, depending on the address region. The only exception is the kernel physical space xkphys, which uses bits [61:59] for the cache attribute of the page.

Figure 4-9 shows the logical translation of a virtual address into a physical address. In this figure, the virtual address is extended with an 8-bit address-space identifier (ASID), which reduces the frequency of TLB flushing during a context switch. This 8-bit ASID contains the number assigned to that process and is stored in the CP0 *EntryHi* register.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

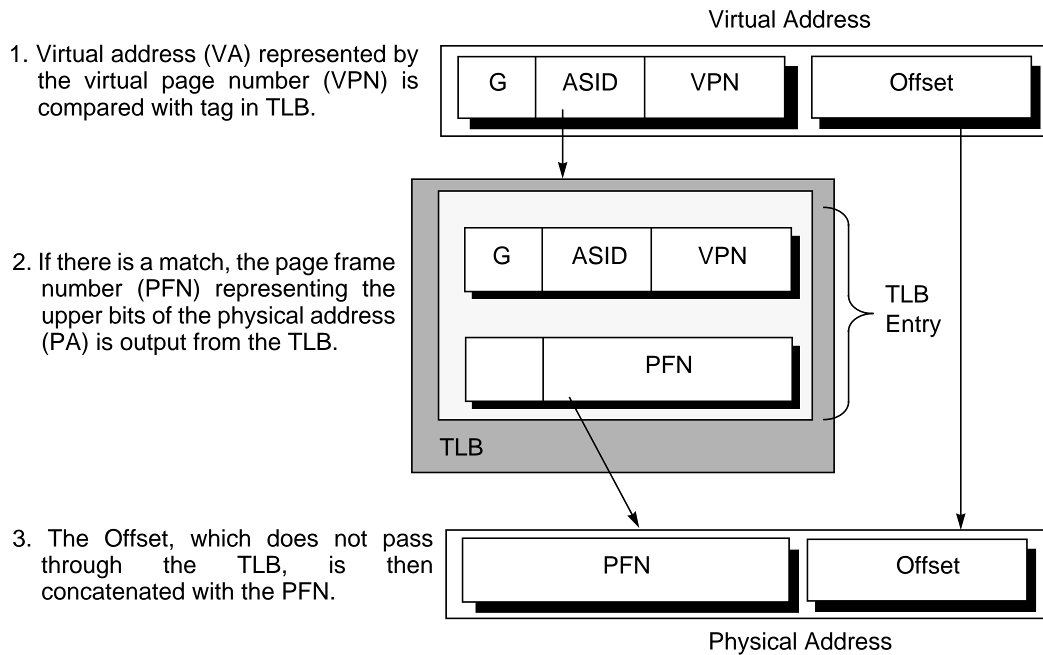


Figure 4-9 Overview of a Virtual-to-Physical Address Translation

If there is a virtual address match in the TLB, the physical address is output from the TLB and concatenated with the *Offset*, which represents an address within the page frame space. The *offset* does not pass through the TLB.

4.7.2 Address Space Identifiers and Global Processes

The 20Kc MMU supports Address Space Identifiers (ASID). These are used to separate address spaces for different processes that are running on the machine. The 20Kc processor supports an 8-bit wide ASID, allowing for 256 separate processes to coexist without interference. In addition, there might be situations where it is desired to have a translation shared among all the processes. In this case, the translation can be marked Global by setting a specific bit in the TLB.

4.7.3 Address Translation Mechanism

Mapped virtual addresses are translated into physical addresses using the on-chip Translation Lookaside Buffers (TLB).

The TLB is a fully associated memory which contains virtual-to-physical translations. Each TLB entry in part can be thought of consisting of two components: a compare part, and a physical translation part. The compare component has a Global bit, an ASID, and finally a Virtual Page Number (VPN) field. The physical translation component has a physical mapping (Page Frame Number, PFN) and other attributes such as cache and coherency attributes (C), a valid bit (V), and a dirty bit (D).

When a translation is requested, the virtual page number and the current ASID are presented to the TLB. All the TLB entries are checked simultaneously for a match. A correct match (TLB hit) occurs when the following conditions are satisfied:

- The current ASID matches the ASID within the TLB entry, or the TLB entry is marked global.
- The relevant bits of the virtual address match the corresponding ones in the virtual page number in the TLB entry. The bits compared depend on the addressing mode as well as the page size of the TLB entry.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

When a TLB entry matches, the physical page number is extracted from the TLB and concatenated with the offset to generate the physical address as shown in Figure 4-9. In addition, the access control bits (C, D, V) are retrieved. Those are used by the processor in the following ways:

- The cache and coherency attributes indicate cacheability attributes of the access, such as cached, uncached, uncached accelerated, and write-through.
- The valid bit (V) indicates if the translation is valid. The valid bit must be set for a valid translation to take place, however, it is not involved in the determination of a matching TLB entry. It is a means for the software to get rid of a translation.
- The dirty bit (D) is used when the translation is for a store request. If the dirty bit is not set, the page is not writable and an exception is triggered.

If no match occurs, an exception is taken and software refills the TLB from the page table in memory.

If there is a virtual address match in the TLB, the physical address is output from the TLB and concatenated with the *Offset*, which represents an address within the page frame space. The *offset* does not pass through the TLB.

The 20Kc core contains a 64-bit virtual address with 40-bit virtual segments. Physical addresses are 36 bits wide. However, its segments are 2^{40} Bytes in size. The top portion of Figure 4-10 shows a virtual address for a 4-Kbyte page size. The width of the *Offset* in Figure 4-10 is defined by the page size. The remaining upper bits of the address represent the virtual page number (VPN).

The bottom portion of Figure 4-10 shows the virtual address for a 16-Mbyte page size. The remaining upper bits of the address represent the VPN.

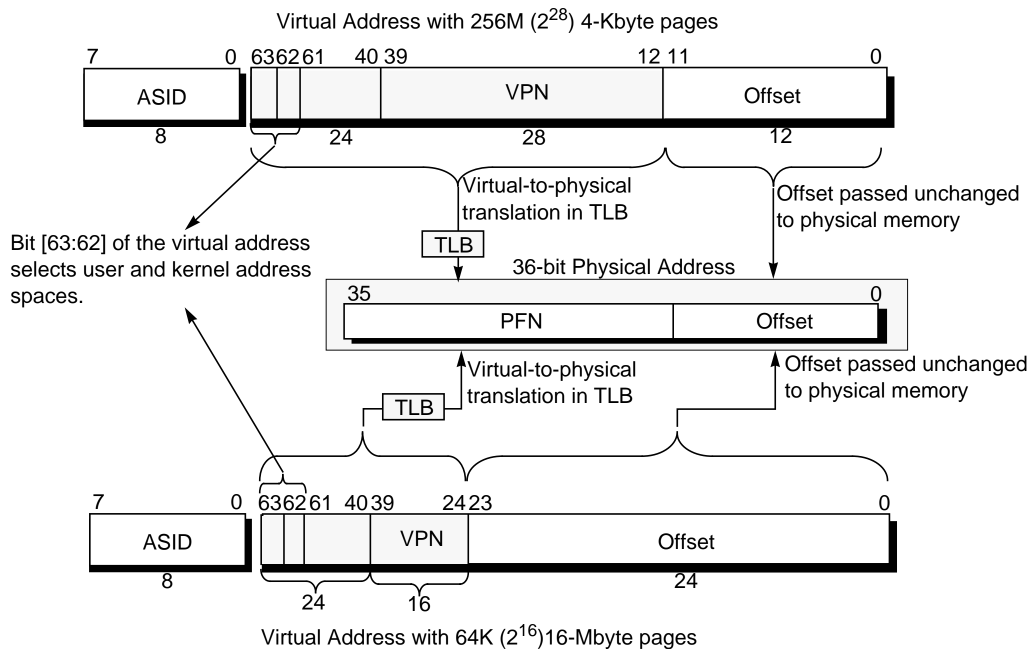


Figure 4-10 20Kc Virtual Address Translation Example

4.8 Translation Lookaside Buffers

The 20Kc processor consists of two address translation buffers: a Micro-TLB and a Joint-TLB.

The Micro-TLB provides translations for load/store instructions and operates as a fully associative cache. Each load/store instruction accesses the Micro-TLB first. If a translation is not found in the Micro-TLB, then the Joint-TLB

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

is accessed. Once the translation is retrieved, it is written back to the Micro-TLB. Therefore, the Micro-TLB contains a subset of translations that are most frequently used.

The Micro-TLB has eight entries to store the different translations. Unlike the Joint-TLB, it is transparent to software. There is no means for the operating system to access the Micro-TLB and modify its contents. Therefore, the hardware ensures that the Micro-TLB is a proper subset of the Joint-TLB, and guarantees that every translation in the Micro-TLB also exists in the Joint-TLB. Each Micro-TLB entry has a page size of 4KBytes. Micro-TLB also provides translation for unmapped virtual address.

The Joint-TLB is the main translation structure in the MMU, even though it is accessed less frequently then the Micro-TLB. It provides translations for instruction cache requests, and for load/store instructions that miss in the Micro-TLB. The Joint-TLB contains 48 dual-entries, each of which provides two physical translations. The physical translation is selected based on the least significant bit of the page address.

4.8.1 20Kc TLB Organization

The 20Kc Micro-TLB and Joint-TLB address translation buffers each contain two logical components: a comparison section and a physical translation section. The comparison section includes the mapping region specifier (R) and the virtual page number (actually, the virtual page number/2 since each entry maps two physical pages, VPN2) of the entry, the ASID, the G(lobal) bit. The physical translation section contains a pair of entries, each of which contains the physical page frame number (PFN), a valid (V) bit, a dirty (D) bit, and a cache coherency field (C). There are two entries in the translation section of each TLB entry because each TLB entry maps an aligned pair of virtual pages and the pair of physical translation entries corresponds to the even and odd pages of the pair. Figure 4-11 shows the logical arrangement of a TLB entry.

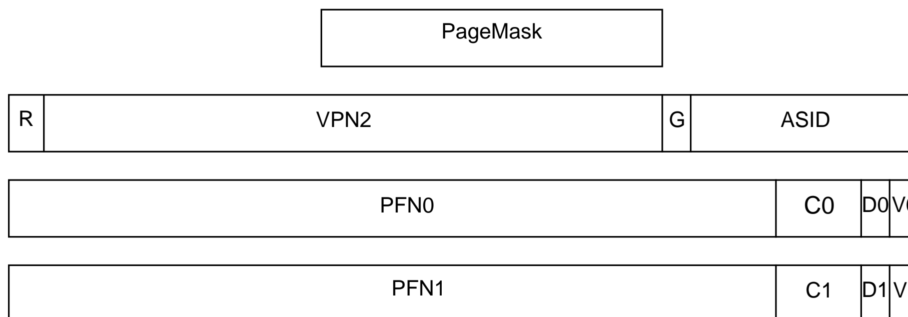


Figure 4-11 Contents of a TLB Entry

The fields of the TLB entry correspond exactly to the fields in the CP0 *PageMask*, *EntryHi*, *EntryLo0*, and *EntryLo1* registers. The even page entries in the TLB (for example, PFN0) come from *EntryLo0*. Similarly, odd page entries come from *EntryLo1*.

When an address translation is requested, the virtual page number and the current process ASID are presented to the TLB. All entries are checked simultaneously for a match, which occurs when all of the following conditions are true:

- The current process ASID (as obtained from the *EntryHi* register) matches the ASID field in the TLB entry, or the G bit is set in the TLB entry.
- Bits [63:62] of the virtual address match the region code in the R field of the TLB entry.
- The appropriate bits of the virtual page number match the corresponding bits of the VPN2 field stored within the TLB entry. The “appropriate” number of bits is determined by the PageMask field in each entry by performing an ANDNOT operation on both the virtual page number and the TLB VPN2 field. This allows each entry of the TLB to support a different page size, as determined by the *PageMask* register at the time that the TLB entry was written.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

If a TLB entry matches the address and ASID presented, the corresponding PFN, C, V, and D bits are read from the translation section of the TLB entry. Which of the two PFN entries is read is a function of the virtual address bit immediately to the right of the section masked with the PageMask entry.

The valid and dirty bits determine the final success of the translation. If the valid bit is off, the entry is not valid and a TLB Invalid exception is raised. If the dirty bit is off and the reference was a store, a TLB Modified exception is raised. If there is an address match with a valid entry and no dirty exception, the PFN and the cache attribute bits are appended to the offset-within-page bits of the address to form the final physical address with attributes.

The TLB lookup process in the 20Kc can be described as follows:

```
found ← 0
for i in 0...TLBEntries-1
          if (TLB[i]R = va63..62) and
             ((TLB[i]VPN2 and not (TLB[i]Mask)) = (vaSEGBITS-1..13 and not
(TLB[i]Mask))) and
             (TLB[i]G or (TLB[i]ASID = EntryHiASID)) then
             # EvenOddBit selects between even and odd halves of the TLB as a
function of
             # the page size in the matching TLB entry
             case TLB[i]Mask
                0000000000002: EvenOddBit ← 12
                0000000000112: EvenOddBit ← 14
                0000000011112: EvenOddBit ← 16
                0000001111112: EvenOddBit ← 18
                0000111111112: EvenOddBit ← 20
                0011111111112: EvenOddBit ← 22
                1111111111112: EvenOddBit ← 24
                otherwise:         UNDEFINED
             endcase
             if vaEvenOddBit = 0 then
                pfn ← TLB[i]PFN0
                v ← TLB[i]V0
                c ← TLB[i]C0
                d ← TLB[i]D0
             else
                pfn ← TLB[i]PFN1
                v ← TLB[i]V1
                c ← TLB[i]C1
                d ← TLB[i]D1
             endif
             if v = 0 then
                InitiateTLBInvalidException(reftype)
             endif
             if (d = 0) and (reftype = store) then
                InitiateTLBModifiedException()
             endif
             # pfnPABITS-1-12..0 corresponds to paPABITS-1..12
             pa ← pfnPABITS-1-12..EvenOddBit-12 || vaEvenOddBit-1..0
             found ← 1
             break
          endif
endfor
if found = 0 then
          InitiateTLBMissException(reftype, VA64Enable)
endif
```

In the 20Kc processor, the VPN2, PFN0, and PFN1 fields of the TLB are pre-masked by the Mask value on a TLB write. This provides the flexibility of eliminating the “and not $TLB[i]_{Mask}$” terms in the pseudocode above. Note that the virtual address must still be masked with the $TLB[i]_{Mask}$ value in either case.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 4-7 demonstrates how the physical address is generated as a function of the page size of the TLB entry that matches the virtual address. The "Even/Odd Select" column of Table 4-7 indicates which virtual address bit is used to select between the even (*EntryLo0*) or odd (*EntryLo1*) entry in the matching TLB entry. The "PA generated from" column specifies how the physical address is generated from the selected PFN and the offset-in-page bits in the virtual address. In this column, PFN is the physical page number as loaded into the TLB from the *EntryLo0* or *EntryLo1* registers, and has the bit range $PFN_{23..0}$, corresponding to $PA_{35..12}$.

Table 4-7 Physical Address Generation

Page Size	Even/Odd Select	PA generated from
4 KBytes	VA_{12}	$PFN_{23..0} \| VA_{11..0}$
16 KBytes	VA_{14}	$PFN_{23..2} \| VA_{13..0}$
64 KBytes	VA_{16}	$PFN_{23..4} \| VA15..0$
256 KBytes	VA_{18}	$PFN_{23..6} \| VA_{17..0}$
1 MBytes	VA_{20}	$PFN_{23..8} \| VA_{19..0}$
4 MBytes	VA_{22}	$PFN_{23..10} \| VA_{21..0}$
16 MBytes	VA_{24}	$PFN_{23..12} \| VA_{23..0}$

4.8.2 TLB Tag and Data Formats

Figure 4-12 shows the format of a TLB *tag* entry. The entry is divided into the following fields:

- Global process indicator (G bit)
- Address space identifier
- Virtual page number
- Compressed page mask

Setting the G bit indicates that the entry is global to all processes and/or threads in the system. In this case, the 8-bit ASID value is ignored since the entry is not relative to a specific thread or process.

The address space identifier (ASID) helps to reduce the frequency of TLB flushing on a context switch. The existence of the ASID allows multiple processes to exist in both the TLB and instruction caches. The ASID value is stored in the *EntryHi* register and is compared to the ASID value of each entry. Figure 4-12 and Table 4-8 show the TLB tag entry format. Figure 4-13 and Table 4-9 show the TLB data array entry format.

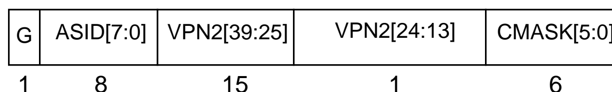


Figure 4-12 TLB Tag Entry Format

Table 4-8 TLB Tag Entry Fields

Field Name	Description
G	Global Bit. When set, indicates that this entry is global to all processes and/or threads and thus disables inclusion of the ASID in the comparison.
ASID[7:0]	Address Space Identifier. Identifies with process or thread this TLB entry is associated with.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 4-8 TLB Tag Entry Fields (Continued)

Field Name	Description
VPN2[39:25], VPN2[24:13]	Virtual Page Number divided by 2. This field contains the upper bits of the virtual page number. Because it represents a pair of TLB pages, it is divided by 2. Bits [31:25] are always included in the TLB lookup comparison. Bits [24:13] are included depending on the page size.
CMASK[5:0]	Compressed Page Mask Value. This field is a compressed version of the page mask. It defines the page size by masking the appropriate VPN2 bits from being involved in a comparison. It is also used to determine which address bit is used to make the even-odd page determination.

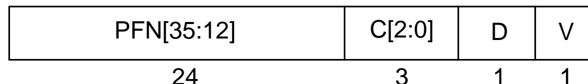


Figure 4-13 TLB Data Array Entry Format

Table 4-9 TLB Data Array Entry Fields

<table>
<tr><th>Field Name</th><th>Description</th></tr>
<tr><td>PFN[35:12]</td><td>Physical Frame Number. Defines the upper bits of the physical address. For page sizes larger than 4 Kbytes, only a subset of these bits is actually used.</td></tr>
<tr><td>C[2:0]</td><td>Cacheability. Contains an encoded value of the cacheability attributes and determines whether the page should be placed in the cache or not. The field is encoded as follows:
<table>
<tr><th>C[2:0]</th><th>Coherency Attribute</th></tr>
<tr><td>000</td><td>Cacheable, noncoherent, write through, no write allocate</td></tr>
<tr><td>001</td><td>Reserved</td></tr>
<tr><td>010</td><td>Uncached</td></tr>
<tr><td>011</td><td>Cacheable, noncoherent</td></tr>
<tr><td>100</td><td>Cacheable, coherent, exclusive</td></tr>
<tr><td>101</td><td>Reserved</td></tr>
<tr><td>110</td><td>Reserved</td></tr>
<tr><td>111</td><td>Uncached accelerated</td></tr>
</table></td></tr>
<tr><td>D</td><td>“Dirty” or Write Enable Bit. Indicates that the page has been written and/or is writable. If this bit is set, stores to the page are permitted. If the bit is cleared, stores to the page cause a TLB Modified exception.</td></tr>
<tr><td>V</td><td>Valid Bit. Indicates that the TLB entry and, thus, the virtual page mapping are valid. If this bit is set, accesses to the page are permitted. If the bit is cleared, accesses to the page cause a TLB Invalid exception.</td></tr>
</table>

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

4.9 TLB Instructions

Table 4-10 lists the 20Kc processor TLB-related instructions. Refer to Chapter 13, “Instruction Set Architecture,” for more information on these instructions.

Table 4-10 TLB Instructions

Opcode	Description of Instruction
TLBP	Translation Lookaside Buffer Probe
TLBR	Translation Lookaside Buffer Read
TLBWI	Translation Lookaside Buffer Write Index
TLBWR	Translation Lookaside Buffer Write Random

4.9.1 Hits, Misses, and Multiple Matches

Each Joint-TLB entry contains a tag portion and a data portion. If a match is found, the upper bits of the virtual address are replaced with the page frame number (PFN) stored in the corresponding entry in the data array of the Joint-TLB. The granularity of Joint-TLB mappings is defined in terms of TLB *pages*. The Joint-TLB supports pages of different sizes ranging from 4 KBytes to 16 MBytes in powers of 4. If a match is found but the entry is invalid, a TLB Invalid exception is taken.

If no match occurs (TLB miss), an exception is taken and software refills the TLB from the page table resident in memory. Software can write over a selected TLB entry or use a hardware mechanism to write into a random entry. In addition, there is a hidden bit in each TLB entry that is cleared on a Reset. This bit is set once the TLB entry is written and is included in the match detection.

The 20Kc processor implements a TLB write-compare mechanism to ensure that multiple TLB matches do not occur. On the TLB write operation, the write value is compared with all other entries in the TLB. If a match occurs, the 20Kc processor takes a machine-check exception, sets the TS bit in the CP0 *Status* register, and completes the write operation. The other matching entries are invalidated by clearing the hidden bit described in the previous paragraph.

Note: To be consistent with other MIPS Technologies devices, it is recommended that all TLB entries be initialized with unique tag values and V bits cleared before the first access to a memory mapped location.

Table 4-11 shows the address bits used for even/odd bank selection depending on page size and the relationship between the legal values in the mask register and the selected page size.

Table 4-11 Mask and Page Size Values

PageMask[11:0]	Page Size	Even/Odd Bank Select Bit
0000_0000_0000	4KB	VAddr[12]
0000_0000_0011	16KB	VAddr[14]
0000_0000_1111	64KB	VAddr[16]
0000_0011_1111	256KB	VAddr[18]
0000_1111_1111	1MB	VAddr[20]
0011_1111_1111	4MB	VAddr[22]
1111_1111_1111	16MB	VAddr[24]

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

4.9.2 Page Sizes and Replacement Algorithm

To assist in controlling both the amount of mapped space and the replacement characteristics of various memory regions, the 20Kc processor provides two mechanisms. First, the page size can be configured on a per entry basis to map a page size of 4 KBytes to 16 MBytes (in multiples of 4). The CP0 *PageMask* register is loaded with the mapping page size, which is then entered into the TLB when a new entry is written. Thus, operating systems can provide special-purpose maps. For example, a typical frame buffer can be memory mapped with only one TLB entry.

The second mechanism controls the replacement algorithm when a TLB miss occurs. To select a TLB entry to be written with a new mapping, the 20Kc processor provides a random replacement algorithm. However, the processor also provides a mechanism whereby a programmable number of mappings can be locked into the TLB via the *Wired* register, thus avoiding random replacement.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Exceptions and Interrupts

The 20Kc processor receives exceptions from a number of sources, including translation lookaside buffer (TLB) misses, arithmetic overflows, I/O interrupts, and system calls. When the processor detects one of these exceptions, the normal sequence of instruction execution is suspended and the processor enters kernel mode.

In kernel mode the core disables interrupts and forces execution of a software exception processor (called a *handler*) located at a fixed address. The handler saves the context of the processor, including the contents of the program counter, the current operating mode, and the status of the interrupts (enabled or disabled). This context is saved so it can be restored when the exception has been serviced.

When an exception occurs, the core loads the *Exception Program Counter* (*EPC*) register with a location where execution can restart after the exception has been serviced. The restart location in the *EPC* register is the address of the instruction that caused the exception or, if the instruction was executing in a branch delay slot, the address of the branch instruction immediately preceding the delay slot.

This chapter contains the following sections:

- Section 5.1, "Exception Conditions"
- Section 5.2, "Exception Types"
- Section 5.3, "Exception Priority"
- Section 5.4, "Exception Vector Locations"
- Section 5.5, "General Exception Processing"
- Section 5.6, "Debug Exception Processing"
- Section 5.7, "Exceptions"
- Section 5.8, "Exception Handling and Servicing Flowcharts"
- Section 5.9, "Interrupts"

5.1 Exception Conditions

When an exception condition occurs, the relevant instruction and all those that follow it in the pipeline are cancelled. Accordingly, any stall conditions and any later exception conditions that may have referenced this instruction are inhibited; there is no benefit in servicing stalls for a cancelled instruction.

When an exception condition is detected on an instruction fetch, the 20Kc processor aborts that instruction and all instructions that follow. When this instruction reaches the W stage, the exception flag causes it to write various CP0 registers with the exception state, change the current program counter (PC) to the appropriate exception vector address, and clear the exception bits of earlier pipeline stages.

This implementation allows all preceding instructions to complete execution and prevents all subsequent instructions from completing. Thus, the value in the *EPC* (*ErrorEPC* for errors, or *DEPC* for debug exceptions) is sufficient to restart execution. It also ensures that exceptions are taken in the order of execution; an instruction taking an exception can itself be killed by an instruction further down the pipeline that takes an exception in a later cycle.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

5.2 Exception Types

The 20Kc processor defines the following exception types listed in Table 5-1. These exceptions types are used in Table 5-2.

Table 5-1 Exception Type Characteristics

Exception Type	Characteristics
Asynchronous Reset	Denotes a reset-type exception that occurs asynchronously to instruction execution. These exceptions always have the highest priority to guarantee that the processor can always be placed in a runnable state.
Asynchronous Debug	Denotes an EJTAG debug exception that occurs asynchronously to instruction execution. These exceptions have very high priority with respect to other exceptions because of the desire to enter Debug Mode, even in the presence of other exceptions, both asynchronous and synchronous.
Asynchronous	Denotes any other type of exception that occurs asynchronously to instruction execution. These exceptions are shown with higher priority than synchronous exceptions mainly for notational convenience. If one thinks of asynchronous exceptions as occurring between instructions, they are either the lowest priority relative to the previous instruction, or the highest priority relative to the next instruction. The ordering of the table above considers them in the second way.
Synchronous Debug	Denotes an EJTAG debug exception that occurs as a result of instruction execution, and is reported precisely with respect to the instruction that caused the exception. These exceptions are prioritized above other synchronous exceptions to allow entry to Debug Mode, even in the presence of other exceptions.
Synchronous	Denotes any other exception that occurs as a result of instruction execution, and is reported precisely with respect to the instruction that caused the exception. These exceptions tend to be prioritized below other types of exceptions, but there is a relative priority of synchronous exceptions with each other.

5.3 Exception Priority

Table 5-2 lists all possible exceptions, and the relative priority of each, highest to lowest.

Table 5-2 Priority of Exceptions

Exception	Description	Type
Reset	The *SI_ColdReset* signal was asserted to the processor.	Asynchronous Reset
Soft Reset	The *SI_Reset* signal was asserted to the processor.	
Debug Single Step (DSS)	An EJTAG Single Step occurred. Prioritized above other exceptions, including asynchronous exceptions, so that one can single-step into interrupt (or other asynchronous) handlers.	Synchronous Debug
Debug Interrupt (DINT)	An EJTAG interrupt (EjtagBrk or DINT) was asserted.	Asynchronous Debug
Imprecise Debug Data Break	An imprecise EJTAG data break condition was asserted.	
Nonmaskable Interrupt (NMI)	The *SI_NMI* signal was asserted to the processor.	Asynchronous
Machine Check	An internal inconsistency was detected by the processor.	Synchronous

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 5-2 Priority of Exceptions

Exception	Description	Type
Bus Error - Instruction fetch	A bus error occurred on an instruction fetch.	Asynchronous
Cache Error - Data access	A cache error occurred on a load or store data reference.	
Bus Error - Data access	A bus error occurred on a load or store data reference.	
Interrupt	An enabled interrupt occurred.	
Deferred Watch	A watch exception, deferred because EXL or ERL was one when the exception was detected, was asserted after both EXL and ERL went to zero.	
Debug Instruction Break	An EJTAG instruction break condition was asserted. This exception is prioritized above instruction fetch exceptions to allow break on illegal instruction addresses.	Synchronous Debug
Watch - Instruction fetch	A watch address match was detected on an instruction fetch. This exception is prioritized above instruction fetch exceptions to allow watch on illegal instruction addresses.	Synchronous
Address Error - Instruction fetch	A non-word-aligned address was loaded into PC.	
TLB/XTLB Refill - Instruction fetch	A TLB miss occurred on an instruction fetch.	
TLB Invalid - Instruction fetch	The valid bit was zero in the TLB entry mapping the address referenced by an instruction fetch.	
Cache Error - Instruction fetch	A cache error occurred on an instruction fetch.	
SDBBP	An EJTAG SDBBP instruction was executed.	Synchronous Debug
Instruction Validity Exceptions	An instruction could not be completed because it was not allowed access to the required resources, or was illegal: Coprocessor unusable, reserved instruction. If both exceptions occur on the same instruction, the Coprocessor Unusable Exception takes priority.	Synchronous
Execution Exception	An instruction-based exception occurred: Integer overflow, trap, system call, breakpoint, floating-point exception.	
Precise Debug Data Break	A precise EJTAG data break on load/store (address match only) condition was asserted. This exception is prioritized above data fetch exceptions to allow break on illegal data addresses.	Synchronous Debug
Watch - Data access	A watch address match was detected on the address referenced by a load or store. This exception is prioritized above data fetch exceptions to allow watch on illegal data addresses.	Synchronous
Address error - Data access	An unaligned address, or an address that was inaccessible in the current processor mode, was referenced by a load or store instruction.	
TLB/XTLB Refill - Data access	A TLB miss occurred on a data access.	
TLB Invalid - Data access	The valid bit was zero in the TLB entry mapping the address referenced by a load or store instruction.	
TLB Modified - Data access	The dirty bit was zero in the TLB entry mapping the address referenced by a store instruction.	

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

5.4 Exception Vector Locations

The Reset, Soft Reset, and NMI exceptions are always vectored to location 0xFFFF_FFFF_BFC0_0000. Debug exceptions are vectored to location 0xFFFF_FFFF_BFC0_0480 or to location 0xFFFF_FFFF_FF20_0200 if the ProbEn bit is 0 or 1, respectively, in the EJTAG Control register (ECR). Addresses for all other exceptions are a combination of a vector offset and a base address. Table 5-3 on page 62 gives the base address as a function of the exception and whether the BEV bit is set in the *Status* register. Table 5-4 on page 62 gives the offsets from the base address as a function of the exception. Table 5-5 on page 62 combines these two tables into one that contains all possible vector addresses as a function of the state that can affect the vector selection.

Table 5-3 Exception Vector Base Addresses

Exception	$Status_{BEV}$	
	0	**1**
Reset, Soft Reset, NMI	0xFFFF_FFFF_BFC0_0000	
Debug (with ProbTrap = 0 in the EJTAG Control register)	0xFFFF_FFFF_BFC0_0480	
Debug (with ProbTrap = 1 in the EJTAG Control register)	0xFFFF_FFFF_FF20_0200	
Cache Error	0xFFFF_FFFF_A000_0000	0xFFFF_FFFF_BFC0_0200
Other	0xFFFF_FFFF_8000_0000	0xFFFF_FFFF_BFC0_0200

Table 5-4 Exception Vector Offsets

Exception	Vector Offset
TLB Refill, EXL = 0	0x000
64-bit XTLB Refill, EXL = 0	0x080
Cache error	0x100
General Exception	0x180
Interrupt, $Cause_{IV}$ = 1	0x200
Reset, Soft Reset, NMI	None (Uses Reset Base Address)

Table 5-5 Exception Vectors

Exception	BEV	EXL	IV	EJTAG ProbTrap	Vector
Reset, Soft Reset, NMI	x	x	x	x	0xFFFF FFFF BFC0 0000
EJTAG Debug	x	x	x	0	0xFFFF FFFF BFC0 0480
EJTAG Debug	x	x	x	1	0xFFFF FFFF FF20 0200
TLB Refill	0	0	x	x	0xFFFF FFFF 8000 0000
XTLB Refill	0	0	x	x	0xFFFF FFFF 8000 0080
TLB Refill	0	1	x	x	0xFFFF FFFF 8000 0180
XTLB Refill	0	1	x	x	0xFFFF FFFF 8000 0180
TLB Refill	1	0	x	x	0xFFFF FFFF BFC0 0200

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 5-5 Exception Vectors (Continued)

Exception	BEV	EXL	IV	EJTAG ProbTrap	Vector
XTLB Refill	1	0	x	x	0xFFFF FFFF BFC0 0280
TLB Refill	1	1	x	x	0xFFFF FFFF BFC0 0380
XTLB Refill	1	1	x	x	0xFFFF FFFF BFC0 0380
Cache Error	0	x	x	x	0xFFFF FFFF A000 0100
Cache Error	1	x	x	x	0xFFFF FFFF BFC0 0300
Interrupt	0	0	0	x	0xFFFF FFFF 8000 0180
Interrupt	0	0	1	x	0xFFFF FFFF 8000 0200
Interrupt	1	0	0	x	0xFFFF FFFF BFC0 0380
Interrupt	1	0	1	x	0xFFFF FFFF BFC0 0400
All others	0	x	x	x	0xFFFF FFFF 8000 0180
All others	1	x	x	x	0xFFFF FFFF BFC0 0380
‘x’ denotes don’t care					

5.5 General Exception Processing

All non-debug related exceptions have the same basic processing flow with the exception of Reset, Soft Reset, and NMI exceptions, which have their own special processing as described below:

- If the EXL bit in the *Status* register is zero, the *EPC* register is loaded with the PC at which execution will be restarted and the BD bit is set appropriately in the *Cause* register. The value loaded into the *EPC* register is the current PC if the instruction is not in the delay slot of a branch, or PC-4 if the instruction is in the delay slot of a branch. If the EXL bit in the *Status* register is set, the *EPC* register is not loaded and the BD bit is not changed in the *Cause* register.
- The CE and ExcCode fields of the *Cause* registers are loaded with the values appropriate to the exception. The CE field is loaded, but not defined, for any exception type other than a coprocessor unusable exception.
- The EXL bit is set in the *Status* register.
- The processor is started at the exception vector.

The value loaded into EPC represents the restart address for the exception and need not be modified by exception handler software in the normal case. Software need not look at the BD bit in the Cause register unless it wishes to identify the address of the instruction that actually caused the exception.

Note that individual exception types can load additional information into other registers. This is noted in the description of each exception type below.

Operation:

```
if Status_EXL = 0
        if InstructionInBranchDelaySlot then
        EPC <- PC - 4
        Cause_BD <- 1
        else
        EPC <- PC
```


Copyright © 2002 MIPS Technologies Inc. All rights reserved.

```
                Cause_BD <- 0
      endif
      if ExceptionType = TLBRefill then
                   vectorOffset <- 0x000
      elseif (ExceptionType = XTLBRefill) then
                   vectorOffset <- 0x080
      elseif (ExceptionType = Interrupt) and
                    (Cause_IV = 1) then
                   vectorOffset <- 0x200
      else
                   vectorOffset <- 0x180
             endif
   else
             vectorOffset <- 0x180
   endif
   Cause_CE <- FaultingCoprocessorNumber
   Cause_ExcCode <- ExceptionType
   Status_EXL <- 1
   if Status_BEV = 1 then
             PC <- 0xFFFF FFFF BFC0 0200 + vectorOffset
   else
             PC <- 0xFFFF FFFF 8000 0000 + vectorOffset
   endif
```

5.6 Debug Exception Processing

All debug exceptions have the same basic processing flow which is described in detail in Chapter 11, “EJTAG Debug Support.”

5.7 Exceptions

The following subsections describe each of the exceptions listed in the same sequence as shown in Table 5-2.

5.7.1 Reset Exception

A reset exception occurs when the *SI_ColdReset* signal is asserted to the processor. This exception is not maskable. When a Reset exception occurs, the processor performs a full reset initialization, including aborting state machines, establishing critical state, and generally placing the processor in a state in which it can execute instructions from uncached, unmapped address space. On a Reset exception, the state of the processor is not defined, with the following exceptions:

- The *Random* register is initialized to the number of TLB entries - 1.
- The *Wired* register is initialized to zero.
- The *Config* register is initialized with its boot state.
- The BEV, TS, SR, NMI, ERL, and RP fields of the *Status* register are initialized to a specified state.
- Watch register enables and Performance Counter register interrupt enables are cleared.
- The *ErrorEPC* register is loaded with PC-4 if the state of the processor indicates that it was executing an instruction in the delay slot of a branch. Otherwise, the *ErrorEPC* register is loaded with PC. Note that this value may or may not be predictable.
- PC is loaded with 0xFFFF_FFFF_BFC0_0000.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

***Cause* Register ExcCode Value:**

None

Additional State Saved:

None

Entry Vector Used:

Reset (0xFFFF_FFFF_BFC0_0000)

Operation:

```
Random <- TLBEntries - 1
Wired <- 0
Config <- ConfigurationState
ConfigK0 <- 2                     # Suggested - see Config register description
Config1 <- ConfigurationState
StatusBEV <- 1
StatusTS <- 0
StatusSR <- 0
StatusNMI <- 0
StatusERL <- 1
StatusRP <- 0
WatchLo[n]I <- 0              # For all implemented Watch registers
WatchLo[n]R <- 0              # For all implemented Watch registers
WatchLo[n]W <- 0              # For all implemented Watch registers
PerfCnt.Control[n]IE <- 0     # For all implemented PerfCnt registers
    if InstructionInBranchDelaySlot then
            ErrorEPC <- PC - 4
    else
            ErrorEPC <- PC
    endif
    PC <- 0xFFFF_FFFF_BFC0_0000
```

5.7.2 Soft Reset Exception

A soft reset exception occurs when the Reset signal is asserted to the processor. This exception is not maskable. When a soft reset exception occurs, the processor performs a subset of the full reset initialization. Although a soft reset exception does not unnecessarily change the state of the processor, it may be forced to do so in order to place the processor in a state in which it can execute instructions from uncached, unmapped address space. Since bus, cache, or other operations may be interrupted, portions of the cache, memory, or other processor state may be inconsistent. In addition to any hardware initialization required, the following state is established on a soft reset exception:

- The BEV, TS, SR, NMI, ERL, and RP fields of the *Status* register are initialized to a specified state.
- Watch register enables and Performance Counter register interrupt enables are cleared.
- The *ErrorEPC* register is loaded with PC-4 if the state of the processor indicates that it was executing an instruction in the delay slot of a branch. Otherwise, the *ErrorEPC* register is loaded with PC. Note that this value may or may not be predictable.
- The PC is loaded with 0xFFFF_FFFF_BFC0_0000.

***Cause* Register ExcCode Value:**

None

Additional State Saved:

None

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Entry Vector Used:

Reset (0xBFC0_0000)

Operation:

```
ConfigK0 <- 2                   # Suggested - see Config register description
StatusBEV <- 1
StatusTS <- 0
StatusSR <- 1
StatusNMI <- 0
StatusERL <- 1
StatusRP <- 0
WatchLo[n]I <- 0                # For all implemented Watch registers
WatchLo[n]R <- 0                # For all implemented Watch registers
WatchLo[n]W <- 0                # For all implemented Watch registers
PerfCnt.Control[n]IE <- 0       # For all implemented PerfCnt registers
if InstructionInBranchDelaySlot then
    ErrorEPC <- PC - 4
else
    ErrorEPC <- PC
endif
PC <- 0xFFFF FFFF BFC0 0000
```

5.7.3 Debug Single Step Exception

A detailed description of this exception can be found in Chapter 11, “EJTAG Debug Support.”

5.7.4 Debug Interrupt Exception

A detailed description of this exception can be found Chapter 11, “EJTAG Debug Support.”

5.7.5 Debug Instruction Break Exception

A detailed description of this exception can be found in Chapter 11, “EJTAG Debug Support.”

5.7.6 Non-Maskable Interrupt (NMI) Exception

A non-maskable interrupt exception occurs when the NMI signal is asserted to the processor. An NMI exception occurs only at instruction boundaries, so it does not cause any reset or other hardware initialization. The state of the cache, memory, and other processor states are consistent and all registers are preserved, with the following exceptions:

- The BEV, TS, SR, NMI, and ERL fields of the *Status* register are initialized to a specified state.
- The *ErrorEPC* register is loaded with PC – 4 if the state of the processor indicates that it was executing an instruction in the delay slot of a branch. Otherwise, the *ErrorEPC* register is loaded with PC.
- The PC is loaded with 0xFFFF_FFFF_BFC0_0000.

NMI exceptions are disabled under the following conditions:

- The processor is operating in Debug Mode
- The NMI Enable (NMIE) bit in the Debug Control Register (DCR) is cleared

For further details on the interaction of NMI with Debug Mode refer to Section 11.5.2.2, "Overview of Data Breakpoint Registers" on page 201.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

***Cause* Register ExcCode Value:**

None

Additional State Saved:

None

Entry Vector Used:

Reset (0xFFFF_FFFF_BFC0_0000)

Operation:

```
SR_BEV <- 1
SR_TS <- 0
SR_SR <- 0
SR_NMI <- 1
SR_ERL <- 1
if InstructionInBranchDelaySlot then
        ErrorEPC <- PC - 4
else
        ErrorEPC <- PC
endif
PC <- 0xFFFF_FFFF_BFC0_0000
```

5.7.7 Machine Check Exception

A machine check exception occurs when the processor detects an internal inconsistency. The following condition causes a machine check exception:

- A machine check exception can only be signalled as a result of a TLB write operation. This occurs when the new entry to be written into the TLB creates the possibility of multiple matching entries on future references. All TLB entries that overlapped with the new one are effectively deleted from the TLB. Their contents remain intact however, and software may read all the entries of the TLB to determine the cause. Note that the TLB operation completes and the new entry is written into the TLB. The TS bit in the Status register is also set to indicate this condition.

***Cause* Register ExcCode Value:**

MCheck

Additional State Saved:

None.

Entry Vector Used:

General exception vector (offset 0x180)

5.7.8 Bus Error Exception — Instruction Fetch or Data Access

A bus error exception occurs when an instruction or data access makes a bus request (due to a cache miss or an uncacheable reference) and that request terminates in an error. Bus error exceptions that occur on an instruction fetch have a higher priority than bus error exceptions that occur on a data access. Note that parity errors detected during bus transactions are reported as cache error exceptions, and not as bus error exceptions.

Bus errors for both instruction fetch and data accesses are imprecise. In case of a bus error, the *EPC* register does not contain the PC of the instruction that actually caused the exception.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

***Cause* Register ExcCode Value:**

IBE: Error on an instruction reference
DBE: Error on a data reference

Additional State Saved:

None

Entry Vector Used:

General exception vector (offset 0x180)

5.7.9 Cache Error Exception

A cache error exception occurs when an instruction or data reference detects a cache tag or data error, or a parity or ECC error is detected on the system bus when a cache miss occurs. The Index Load Tag and Index Store Tag CACHE instructions do not trigger cache error exceptions, however, the software must not rely on this behavior. Future processors may have different behaviors regarding whether these instructions incur cache error exceptions. The reasons for this are:

- The Index Store Tag CACHE instruction is the way to initialize the instruction cache in the first place.
- The Index Load Tag CACHE instruction is a diagnostic instruction that may be used to probe the cache on a cache error exception.

This exception is not maskable. Because the error was in a cache, the exception vector is to an unmapped, uncached address. It is implementation dependent whether a cache error exception resulting from an access to the data cache is reported precisely with respect to the instruction that caused the cache error. When a data cache line being evicted has a bad tag parity, the processor takes a cache error exception and also sends the data to the external bus.

Cache errors for a data access to a non-blocking cache are imprecise and the *ErrorEPC* register does not contain the PC of the instruction that actually caused the exception.

***Cause* Register ExcCode Value**

N/A

Additional State Saved

Table 5-6 CP0 Register States on a Cache Error Exception

Register State	Value
CacheErr	Error state
ErrorEPC	PC

Entry Vector Used

Cache error vector (offset 0x100)

Operation

```
    CacheErr <- ErrorState
StatusERL <- 1
    if InstructionInBranchDelaySlot then
            ErrorEPC <- PC - 4
    else
            ErrorEPC <- PC
    endif
if StatusBEV = 1 then
            PC <- 0xFFFF FFFF BFC0 0200 + 0x100
```


Copyright © 2002 MIPS Technologies Inc. All rights reserved.

```
else
            PC <- 0xFFFF FFFF A000 0000 + 0x100
endif
```

5.7.10 Interrupt Exception

The interrupt exception occurs when one or more of the eight interrupt requests is enabled by the *Status* register and the interrupt input is asserted.

Register ExcCode Value:

Int

Additional State Saved:

Table 5-7 Register States an Interrupt Exception

Register State	Value
$Cause_{IP}$	indicates the interrupts that are pending.

Entry Vector Used:

General exception vector (offset 0x180) if the IV bit in the *Cause* register is 0;
interrupt vector (offset 0x200) if the IV bit in the *Cause* register is 1.

5.7.11 Debug Software Breakpoint Exception

A detailed description of this exception can be found in Section 11.5.4, "Debug Exceptions from Breakpoints" on page 204.

5.7.12 Watch Exception — Instruction Fetch or Data Access

The Watch facility provides a software debugging vehicle by initiating a watch exception when an instruction or data reference matches the address information stored in the *WatchHi* and *WatchLo* registers. A Watch exception is taken immediately if the EXL and ERL bits of the *Status* register are both zero. If either bit is a one at the time that a watch exception would normally be taken, the WP bit in the *Cause* register is set, and the exception is deferred until both the EXL and ERL bits in the Status register are zero. Software can use the WP bit in the *Cause* register to determine if the EPC register points at the instruction that caused the watch exception, or if the exception actually occurred while in kernel mode.

If the EXL or ERL bits are one in the Status register and a single instruction generates both a watch exception (which is deferred by the state of the EXL and ERL bits) and a lower-priority exception, the lower priority exception is taken. It is implementation dependent whether the WP bit is set in this case. The 20Kc processor sets the WP bit only when the instruction completes with no other exception.

The Watch exception can occur on either an instruction fetch or a data access. Watch exceptions that occur on an instruction fetch have a higher priority than watch exceptions that occur on a data access.

The 20Kc processor does not trigger data watch exceptions on a prefetch or cache instruction whose address matches the Watch register.

Register ExcCode Value:

WATCH

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Additional State Saved:

Table 5-8 Register States on a Watch Exception

Register State	Value
$Cause_{WP}$	Indicates that the watch exception was deferred until after both $Status_{EXL}$ and $Status_{ERL}$ were zero. This bit directly causes a watch exception, so software must clear this bit as part of the exception handler to prevent a watch exception loop at the end of the current handler execution.

Entry Vector Used:

General exception vector (offset 0x180)

5.7.13 Address Error Exception — Instruction Fetch/Data Access

An address error exception occurs on an instruction or data access when an attempt is made to execute one of the following:

- A load or store doubleword instruction is executed in which the address is not aligned on a doubleword boundary.
- An instruction is fetched from an address that is not aligned on a word boundary.
- A load or store word instruction is executed in which the address is not aligned on a word boundary.
- A load or store halfword instruction is executed in which the address is not aligned on a halfword boundary.
- A reference is made to a kernel address space from User Mode or Supervisor Mode.
- A reference is made to a supervisor address space from User Mode.
- A reference is made to a a 64-bit address that is outside the range of the 32-bit Compatibility Address Space when 64-bit address references are not enabled.
- A reference is made to an undefined or unimplemented 64-bit address when 64-bit address references are enabled.

Note that in the case of an instruction fetch that is not aligned on a word boundary, the PC is updated before the condition is detected. Therefore, both *EPC* and *BadVAddr* point to the unaligned instruction address. In the case of a data access the exception is taken if either an unaligned address or an address that was inaccessible in the current processor mode was referenced by a load or store instruction.

***Cause* Register ExcCode Value:**

ADEL: Reference was a load or an instruction fetch
ADES: Reference was a store

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Additional State Saved:

Table 5-9 CP0 Register States on an Address Exception Error

Register State	Value
BadVAddr	Failing address
$Context_{VPN2}$	UNPREDICTABLE
$EntryHi_{VPN2}$	UNPREDICTABLE
EntryLo0	UNPREDICTABLE
EntryLo1	UNPREDICTABLE

Entry Vector Used:

General exception vector (offset 0x180)

5.7.14 TLB Refill and XTLB Refill Exceptions

A TLB Refill or XTLB Refill exception occurs in a TLB-based MMU when no TLB entry matches a reference to a mapped address space and the EXL bit is zero in the *Status* register. Note that this is distinct from the case in which an entry matches but has the valid bit off, in which case a TLB Invalid exception occurs. Refill exceptions have distinct exception vector offsets: 0x000 for a 32-bit TLB Refill and 0x080 for a 64-bit extended TLB ("XTLB") refill. The XTLB refill handler is used whenever a reference is made to an enabled 64-bit address space.

***Cause* Register ExcCode Value**

TLBL: Reference was a load or an instruction fetch
TLBS: Reference was a store

Additional State Saved

Table 5-10 CP0 Register States on a TLB/XTLB Refill Exception

Register State	Value
BadVAddr	Failing address
Context	The BadVPN2 field contains $VA_{31:13}$ of the failing address
XContext	The *XContext* BadVPN2 field contains $VA_{SEGBITS-1:13}$, and the *XContext* R field contains $VA_{63:62}$ of the failing address
EntryHi	The *EntryHi* VPN2 field contains $VA_{SEGBITS-1:13}$ of the failing address and the EntryHi R field contains $VA_{63:62}$ of the failing address; the ASID field contains the ASID of the reference that missed
EntryLo0	**UNPREDICTABLE**
EntryLo1	**UNPREDICTABLE**

Entry Vector Used

- TLB Refill vector (offset 0x000) if 64-bit addresses are not enabled and $Status_{EXL}$ = 0 at the time of exception.
- XTLB Refill vector (offset 0x080) if 64-bit addresses are enabled and $Status_{EXL}$ = 0 at the time of exception.
- General exception vector (offset 0x180) in either case if $Status_{EXL}$ = 1 at the time of exception.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

5.7.15 TLB Invalid Exception — Instruction Fetch or Data Access

During an instruction fetch or data access, a TLB invalid exception occurs in one of the following cases:

- No TLB entry in the MMU matches a reference to a mapped address space, and the EXL bit is 1 in the *Status* register.
- A TLB entry in the MMU matches a reference to a mapped address space, but the matched entry has the valid bit off.

***Cause* Register ExcCode Value:**

TLBL: Reference was a load or an instruction fetch
TLBS: Reference was a store

Additional State Saved:

Table 5-11 CP0 Register States on a TLB Invalid Exception

Register State	Value
BadVAddr	Failing address
Context	The BadVPN2 field contains $VA_{31:13}$ of the failing address
XContext	The *XContext* BadVPN2 field contains $VA_{SEGBITS-1:13}$ of the failing address, and the *XContext* R field contains $VA_{63:62}$ of the failing address
EntryHi	The VPN2 field contains $VA_{SEGBITS-1:13}$ of the failing address, and the *XContext* R field contains $VA_{63:62}$ of the failing address. The ASID field contains the ASID of the reference that missed.
EntryLo0	UNPREDICTABLE
EntryLo1	UNPREDICTABLE

Entry Vector Used:

General exception vector (offset 0x180)

5.7.16 Execution Exception — System Call

The system call exception is one of the six execution exceptions. All of these exceptions have the same priority. A system call exception occurs when a SYSCALL instruction is executed.

***Cause* Register ExcCode Value:**

Sys

Additional State Saved:

None

Entry Vector Used:

General exception vector (offset 0x180)

5.7.17 Execution Exception — Breakpoint

The breakpoint exception is one of the six execution exceptions. All of these exceptions have the same priority. A breakpoint exception occurs when a BREAK instruction is executed.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

***Cause* Register ExcCode Value:**

Bp

Additional State Saved:

None

Entry Vector Used:

General exception vector (offset 0x180)

5.7.18 Execution Exception — Reserved Instruction

A Reserved Instruction Exception occurs if any of the following conditions is true:

- An instruction was executed that specifies an encoding of the opcode field (Table 13-19) that is flagged with “∗” (reserved), “β” (higher-order ISA), “⊥” (64-bit) if 64-bit operations are not enabled, or an unimplemented “e” (ASE).
- An instruction was executed that specifies a *SPECIAL* opcode encoding of the function field (Table 13-20) that is flagged with “∗” (reserved), “β” (higher-order ISA), or “⊥” (64-bit) if 64-bit operations are not enabled.
- An instruction was executed that specifies a *REGIMM* opcode encoding of the rt field (Table 13-21) that is flagged with “∗” (reserved).
- An instruction was executed that specifies an unimplemented *SPECIAL2* opcode encoding of the function field (Table 13-22) that is flagged with an unimplemented “q” (partner available), “^” (64-bit) if 64-bit operations are not enabled, or an unimplemented “s” (EJTAG).
- An instruction was executed that specifies a *COPz* opcode encoding of the rs field (Table 13-23) that is flagged with “*” (reserved), “b” (higher-order ISA), “^” (64-bit) if 64-bit operations are not enabled, or an unimplemented “e” (ASE), assuming that access to the coprocessor is allowed. If access to the coprocessor is not allowed, a Coprocessor Unusable Exception occurs instead. For the *COP1* opcode, some implementations of previous ISAs reported this case as a Floating-Point Exception, setting the Unimplemented Operation bit in the Cause field of the *FCSR* register.
- An instruction was executed that specifies an unimplemented *COP0* opcode encoding of the function field when rs is *CO* (Table 13-26) that is flagged with “*” (reserved), or an unimplemented “s” (EJTAG), assuming that access to Coprocessor 0 is allowed. If access to the coprocessor is not allowed, a Coprocessor Unusable Exception occurs instead.
- An instruction was executed that specifies a *COP1* opcode encoding of the function field when rs is S, D, or W (Table 13-27, Table 13-28, Table 13-29) that is flagged with “*” (reserved), “b” (higher-order ISA), “^” (64-bit) if 64-bit operations are not enabled, or an unimplemented “e” (ASE), assuming that access to Coprocessor 1 is allowed. If access to the coprocessor is not allowed, a Coprocessor Unusable Exception occurs instead. Some implementations of previous ISAs reported this case as a Floating-Point Exception, setting the Unimplemented Operation bit in the Cause field of the *FCSR* register.
- An instruction was executed that specifies a *COP1* opcode encoding when rs is L or PS (Table 13-30, Table 13-31) and 64-bit operations are not enabled, or with a function field encoding that is flagged with “*” (reserved), “b” (higher-order ISA), or an unimplemented “e” (ASE), assuming that access to coprocessor 1 is allowed. If access to the coprocessor is not allowed, a Coprocessor Unusable Exception occurs instead. Some implementations of previous ISAs reported this case as a Floating-Point Exception, setting the Unimplemented Operation bit in the Cause field of the *FCSR* register.
- An instruction was executed that specifies a COP1X opcode encoding of the function field (Table 13-32) that is flagged with “*” (reserved), or any execution of the COP1X opcode when 64-bit operations are not enabled, assuming that access to coprocessor 1 is allowed. If access to the coprocessor is not allowed, a Coprocessor Unusable Exception occurs instead. Some implementations of previous ISAs reported this case as a Floating-Point Exception, setting the Unimplemented Operation bit in the Cause field of the *FCSR* register.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

- A branch occurs in a branch delay slot in certain circumstances. Note that a branch in a branch delay slot is not allowed in the MIPS architecture, and the behavior is defined as “unpredictable” for that case.

***Cause* Register ExcCode Value**

RI

Additional State Saved

None

Entry Vector Used

General exception vector (offset 0x180)

5.7.19 Execution Exception — Coprocessor Unusable

A coprocessor unusable exception occurs if any of the following conditions are true:

- A COP0 or Cache instruction was executed while the processor was running in a mode other than Debug Mode or Kernel Mode, and the CU0 bit in the *Status* register was a zero.
- A COP1, COP1X, LWC1, SWC1, LDC1, SDC1, or MOVCI (Special opcode function field encoding) instruction was executed and the CU1 bit in the *Status* register was a zero.
- A COP2, LWC2, SWC2, LDC2, or SDC2 instruction was executed, and the CU2 bit in the *Status* register was a zero.

***Cause* Register ExcCode Value:**

CpU

Additional State Saved:

Table 5-12 Register States on a Coprocessor Unusable Exception

Register State	Value
$Cause_{CE}$	Unit number of the coprocessor being referenced

Entry Vector Used:

General exception vector (offset 0x180)

5.7.20 Execution Exception — Integer Overflow

The integer overflow exception is one of the six execution exceptions. All of these exceptions have the same priority. An integer overflow exception occurs when selected integer instructions result in a 2’s complement overflow.

***Cause* Register ExcCode Value:**

Ov

Additional State Saved:

None

Entry Vector Used:

General exception vector (offset 0x180)

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

5.7.21 Execution Exception — Trap

The trap exception is one of the six execution exceptions. All of these exceptions have the same priority. A trap exception occurs when a trap instruction results in a TRUE value.

***Cause* Register ExcCode Value:**

Tr

Additional State Saved:

None

Entry Vector Used:

General exception vector (offset 0x180)

5.7.22 Precise Debug Data Break Exception

A detailed description of this exception can be found in Section 11.3.5.6, "Debug Data Break Load/Store Precise Exception on Address".

5.7.23 Imprecise Debug Data Break Exception

A detailed description of this exception can be found in Section 11.3.5.7, "Debug Data Break Load/Store Imprecise Exception on Data".

5.7.24 TLB Modified Exception — Data Access

During a data access, a TLB modified exception occurs on a *store* reference to a mapped address if the following condition is true:

- The matching TLB entry in a TLB-based MMU is valid, but not dirty (indicating that the page is not writeable).

***Cause* Register ExcCode Value:**

Mod

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Additional State Saved:

Table 5-13 Register States on a TLB Modified Exception

Register State	Value
BadVAddr	Failing address
Context	The BadVPN2 field contains $VA_{31:13}$ of the failing address.
XContext	The *XContext* BadVPN2 field contains $VA_{SEGBITS-1:13}$ of the failing address, and the *XContext* R field contains $VA_{63:62}$ of the failing address.
EntryHi	The VPN2 field contains $VA_{SEGBITS-1:13}$ of the failing address, and the *XContext* R field contains $VA_{63:62}$ of the failing address. The ASID field contains the ASID of the reference that missed.
EntryLo0	UNPREDICTABLE
EntryLo1	UNPREDICTABLE

Entry Vector Used:

General exception vector (offset 0x180)

5.8 Exception Handling and Servicing Flowcharts

The remainder of this chapter contains flowcharts for the following exceptions and guidelines for their handlers:

- General exceptions and their exception handler
- TLB miss exception and their exception handler
- Reset, soft reset and NMI exceptions, and a guideline to their handler.
- Debug exceptions

Generally speaking, the exceptions are handled by hardware (HW); the exceptions are then serviced by software (SW). Note that unexpected debug exceptions to the debug exception vector at 0xFFFF_FFFF_BFC0_0200 may be viewed as a reserved instruction since uncontrolled execution of a SDBBP instruction caused the exception. The DERET instruction must be used at return from the debug exception handler, in order to leave debug mode and return to non-debug mode. The DERET instruction returns to the address in the DEPC register.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Exceptions other than Reset, Soft Reset, NMI, or first-level miss.
Note: Interrupts can be masked by IE or IMs and Watch is masked if EXL =1

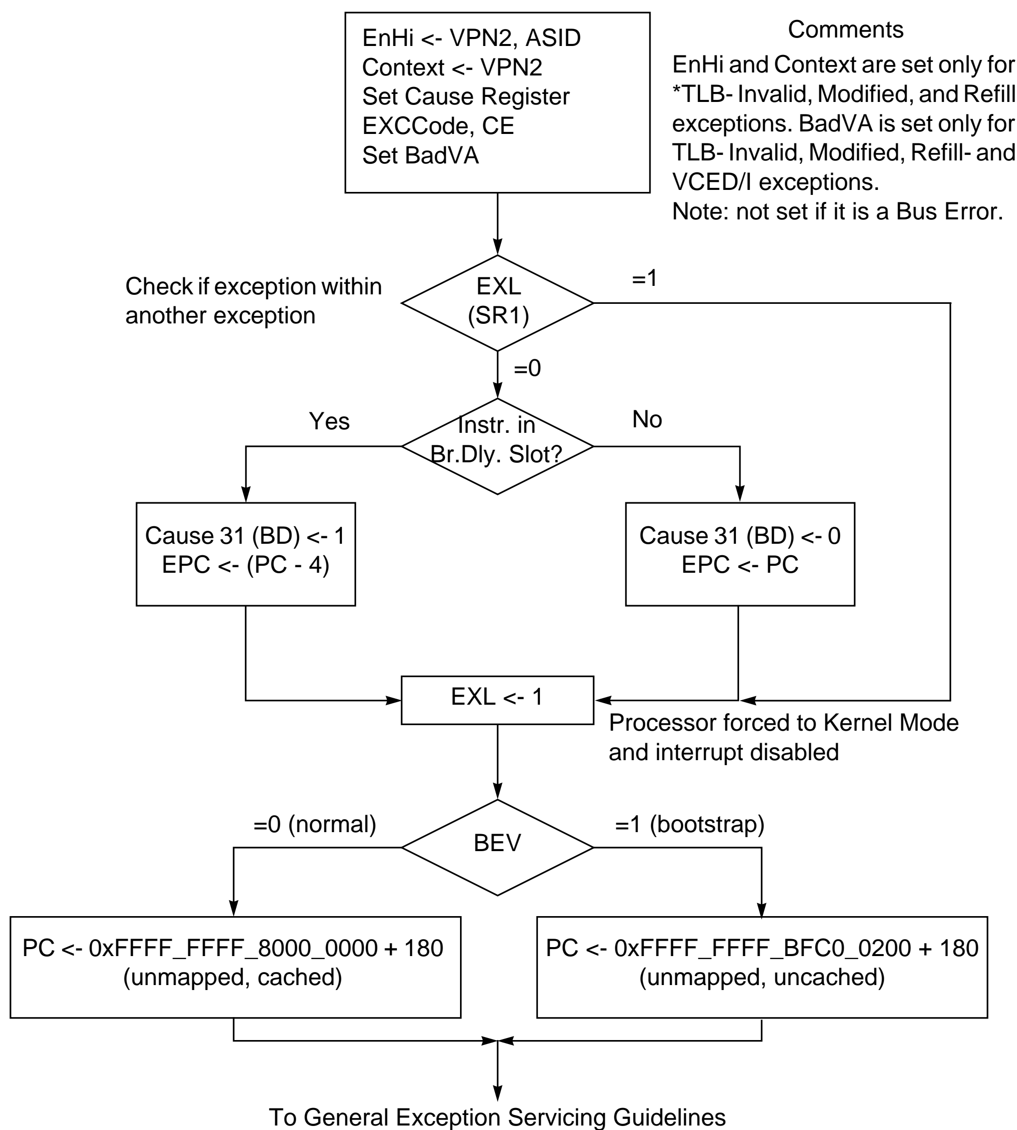


Figure 5-1 General Exception Handler (HW)

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

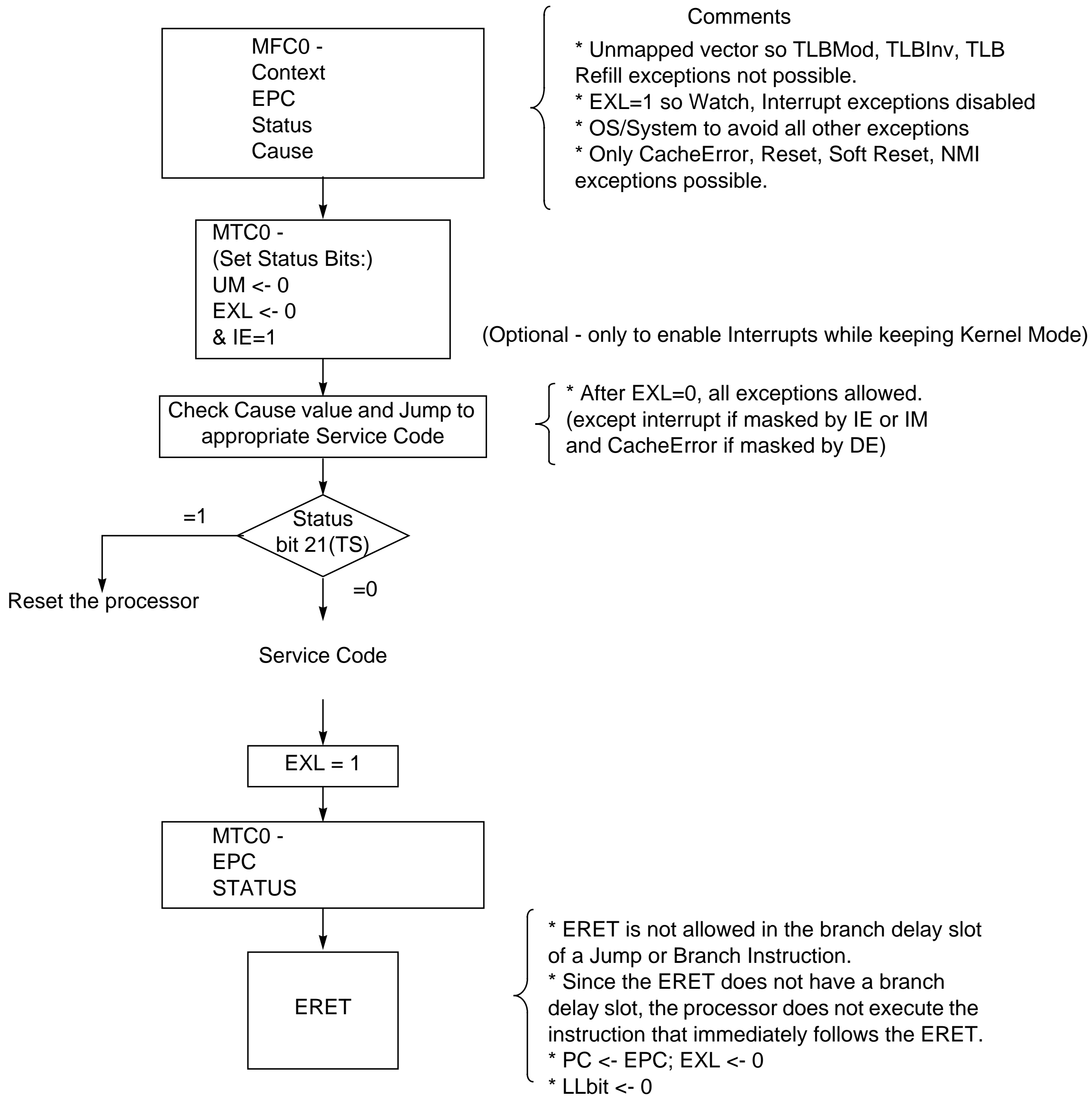


Figure 5-2 General Exception Servicing Guidelines (SW)

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

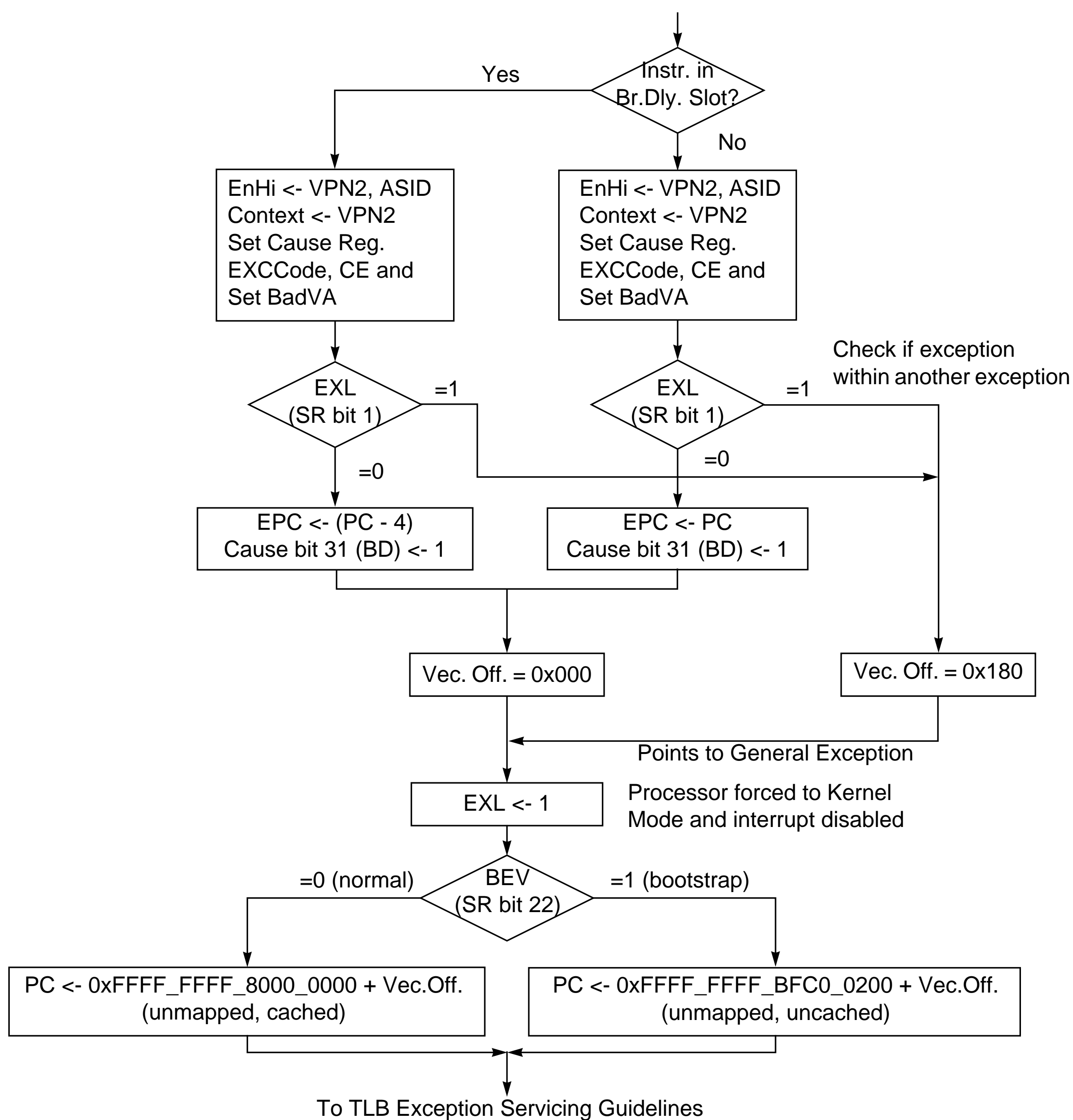


Figure 5-3 TLB Miss Exception Handler (HW)

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

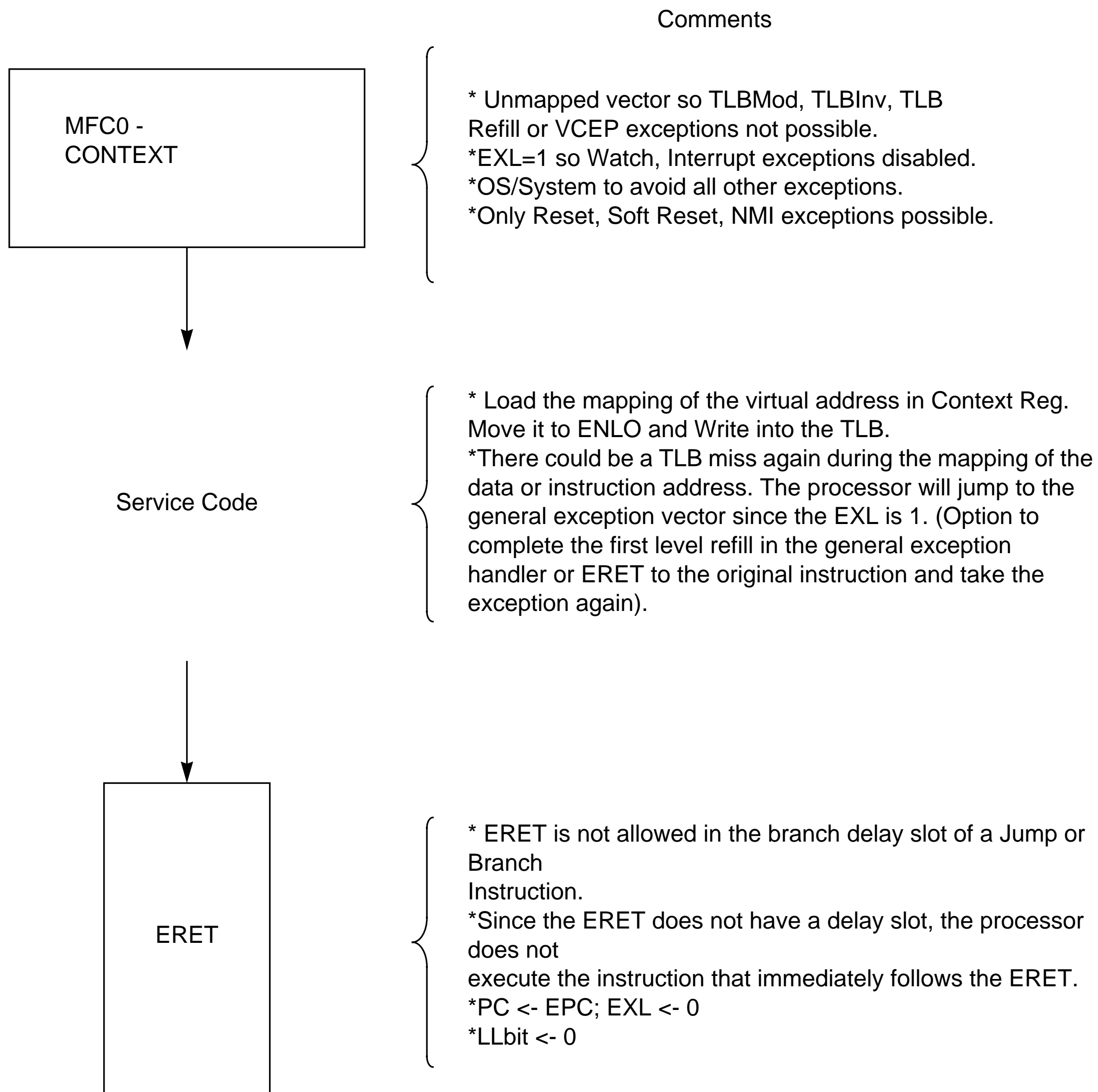


Figure 5-4 TLB Exception Servicing Guidelines (SW)

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

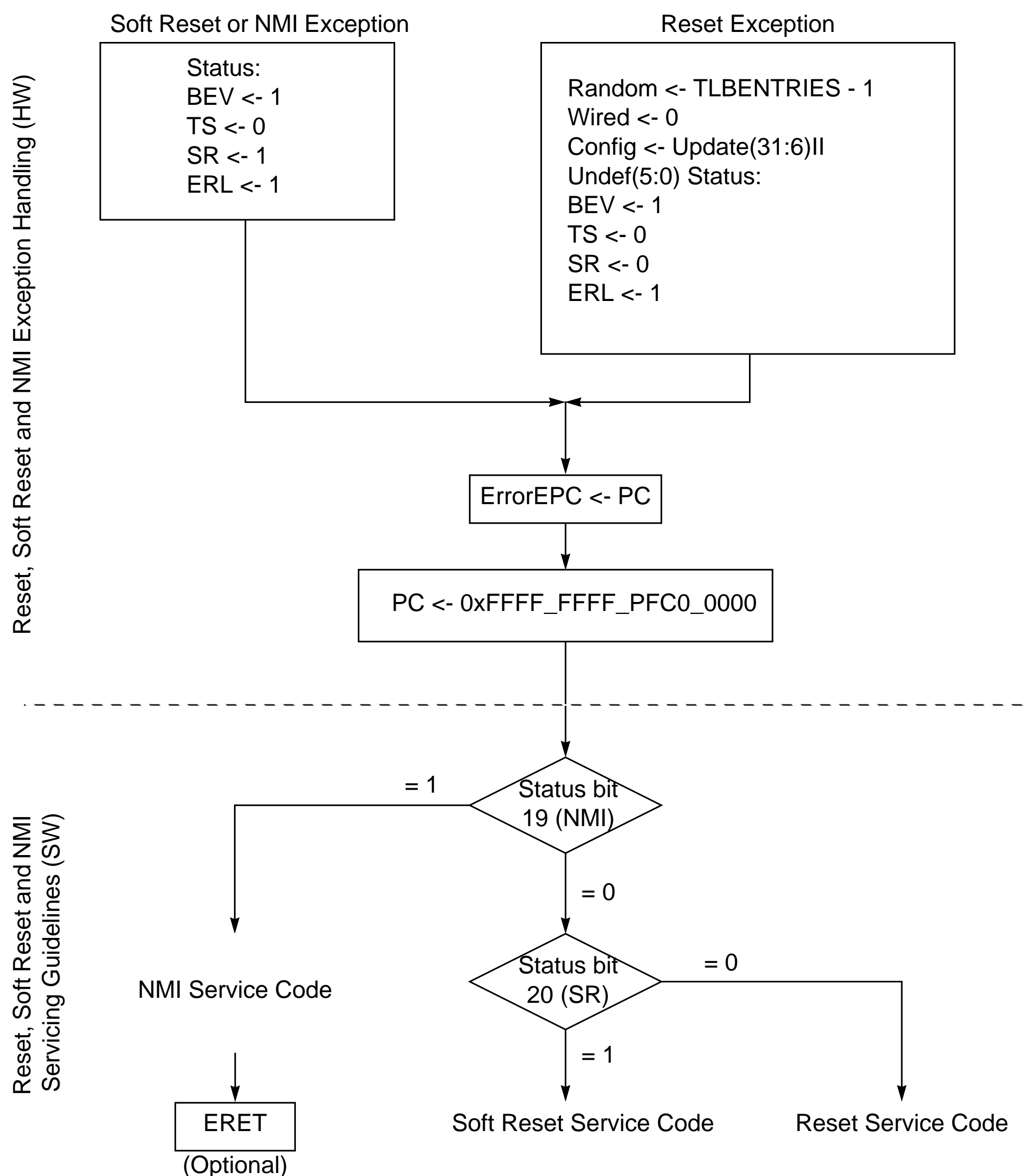


Figure 5-5 Reset, Soft Reset and NMI Exception Handling and Servicing Guidelines

5.9 Interrupts

The 20Kc processor supports eight interrupt requests, broken down into four categories:

- Software interrupts - Two software interrupt requests are made via software writes to bits IP0 and IP1 of the *Cause* register.
- Hardware interrupts - Six hardware interrupt requests numbered 0 through 5 are made via implementation-dependent external requests to the processor.
- Timer interrupt - A timer interrupt is raised when the *Count* and *Compare* registers reach the same value.
- Performance counter interrupt - A performance counter interrupt is raised when the most significant bit of the counter is a one, and the interrupt is enabled by the IE bit in the performance counter control register.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Timer interrupts, performance counter interrupts, and hardware interrupt 5 are combined to create the ultimate hardware interrupt 5. Timer interrupt is multiplexed with the hardware interrupt 5 based on the value of bit [24] of the *Config* register. The Performance counter interrupt is then OR-ed with the multiplexed value to create the final hardware interrupt 5.

The current interrupt requests are visible via the IP field in the *Cause* register on any read of that register (not just after an interrupt exception has occurred). The mapping of *Cause* register bits to the various interrupt requests is shown in Table 5-14.

Table 5-14 Mapping of Interrupts to the *Cause* and *Status* Registers

		Cause Register Bit		*Status* Register Bit	
Interrupt Type	**Interrupt Number**	**Number**	**Name**	**Number**	**Name**
Software Interrupt	0	8	IP0	8	IM0
	1	9	IP1	9	IM1
Hardware Interrupt	0	10	IP2	10	IM2
	1	11	IP3	11	IM3
	2	12	IP4	12	IM4
	3	13	IP5	13	IM5
	4	14	IP6	14	IM6
Hardware Interrupt, Timer Interrupt, or Performance Counter Interrupt	5	15	IP7	15	IM7

For each bit of the IP field in the *Cause* register there is a corresponding bit in the IM field in the *Status* register. An interrupt is only taken when all of the following are true:

- An interrupt request bit is a one in the IP field of the *Cause* register.
- The corresponding mask bit is a one in the IM field of the *Status* register. The mapping of bits is shown in Table 5-14.
- The IE bit in the *Status* register is a one.
- The DM bit in the *Debug* register is a zero
- The EXL and ERL bits in the *Status* register are both zero.

Logically, the IP field of the *Cause* register is bit-wise ANDed with the IM field of the *Status* register, the eight resultant bits are ORed together and that value is ANDed with the IE bit of the *Status* register. The final interrupt request is then asserted only if both the EXL and ERL bits in the *Status* register are zero, and the DM bit in the *Debug* register is zero, corresponding to a non-exception, non-error, non-debug processing mode.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Coprocessor Registers

The 20Kc processor implements both Coprocessor 0 (CP0) and Coprocessor 1 (CP1) of the MIPS architecture.

The System Control Coprocessor 0 (CP0) provides the register interface to the 20Kc processor and supports memory management, address translation, exception handling, and other privileged operations. Each CP0 register has a unique number that identifies it; this number is referred to as the *register number*. For instance, the *PageMask* register is register number 5. For more information on the EJTAG registers, refer to Chapter 11, “EJTAG Debug Support.”

The System Control Coprocessor 1 (CP1) provides the register interface to the 20Kc processor to support floating-point operations. As with CP0, each CP1 register also has a unique register number that identifies it.

This chapter contains the following sections:

- Section 6.1, "CP0 Register Summary"
- Section 6.2, "CP0 Registers"
- Section 6.3, "CP0 Hazards"
- Section 6.4, "CP1 Register Summary"
- Section 6.5, "CP1 Registers"

6.1 CP0 Register Summary

Table 6-1 lists the CP0 registers in numerical order. The individual registers are described throughout this chapter.

Table 6-1 CP0 Registers

Register Number	Register Name	Function
0	Index[1]	Index into the TLB array
1	Random[1]	Randomly generated index into the TLB array
2	EntryLo0[1]	Low-order portion of the TLB entry for even-numbered virtual pages
3	EntryLo1[1]	Low-order portion of the TLB entry for odd-numbered virtual pages
4	Context[1, 2]	Pointer to page table entry in memory
5	PageMask[1]	Controls the variable page sizes in TLB entries
6	Wired[1]	Controls the number of fixed (“wired”) TLB entries
7	Reserved	Reserved
8	BadVAddr[2]	Reports the address for the most recent address-related exception
9	Count[2]	Processor cycle count
10	EntryHi[1]	High-order portion of the TLB entry
11	Compare[2]	Timer interrupt control

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 6-1 CP0 Registers (Continued)

Register Number	Register Name	Function
12	Status[2]	Processor status and control
13	Cause[2]	Cause of last general exception
14	EPC[2]	Program counter at last exception
15	PRId	Processor identification and revision
16	Config/Config1	Configuration register
17	LLAddr	Load linked address
18	WatchLo[2]	Low-order watchpoint address
19	WatchHi[2]	High-order watchpoint address
20	XContext[1, 2]	Pointer to page table entry in extended memory
21 - 22	Reserved	Reserved
23	Debug[3]	Debug control and exception status
24	DEPC[3]	Program counter at last debug exception
25	PerfCount	Performance counter register
26	DErrCtl/IErrCtl	Error Checking and control register
27	CacheErr	Provides an interface to the cache error detection logic
28	ITagLo/IDataLo, DTagLo/DDataLo	Low-order portion of cache tag interface
29	ITagHi/IDataHi DTagHi/DDataHi	High-order portion of cache tag interface
30	ErrorEPC[2]	Program counter at last error
31	DESAVE[3]	Debug handler scratchpad register
1. Registers used in memory management.		
2. Registers used in exception processing.		
3. Registers used in debug.		

6.2 CP0 Registers

The CP0 registers provide the interface between the ISA and the architecture. Each register is discussed below. The registers are presented in numerical order, first by register number, then by select field number.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

For each register described below, field descriptions include the read/write properties of the field and the reset state of the field. For the read/write properties of the field, the following notation is used:

Table 6-2 CP0 Register Field Types

Read/Write Notation	Hardware Interpretation	Software Interpretation
R/W	A field in which all bits are readable and writable by software and, potentially, by hardware. Hardware updates of this field are visible by software read. Software updates of this field are visible by hardware read. If the reset state of this field is “Undefined,” either software or hardware must initialize the value before the first read will return a predictable value. This should not be confused with the formal definition of UNDEFINED behavior.	
R	A field that is either static or is updated only by hardware. If the Reset State of this field is either “0” or “Preset”, hardware initializes this field to zero or to the appropriate state, respectively, on powerup. If the Reset State of this field is “Undefined”, hardware updates this field only under those conditions specified in the description of the field.	A field to which the value written by software is ignored by hardware. Software may write any value to this field without affecting hardware behavior. Software reads of this field return the last value updated by hardware. If the Reset State of this field is “Undefined,” software reads of this field result in an UNPREDICTABLE value except after a hardware update done under the conditions specified in the description of the field.
0	A field that hardware does not update, and for which hardware can assume a zero value.	A field to which the value written by software must be zero. Software writes of non-zero values to this field may result in UNDEFINED behavior of the hardware. Software reads of this field return zero as long as all previous software writes are zero. If the Reset State of this field is “Undefined,” software must write this field with zero before it is guaranteed to read as zero.

6.2.1 Index Register (CP0 Register 0, Select 0)

The *Index* register is a 32-bit read/write register that contains the index used to access the TLB for TLBP, TLBR, and TLBWI instructions.

The operation of the processor is UNDEFINED if a value greater than or equal to the number of TLB entries is written to the *Index* register.

Index Register Format

31	30 … 6	5 … 0
P	0	Index

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 6-3 Index Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	Bit(s)			
P	31	Probe Failure. Hardware writes this bit during execution of the TLBP instruction to indicate whether a TLB match occurred: 0: A match occurred, and the Index field contains the index of the matching entry 1: No match occurred and the Index field is **UNPREDICTABLE**	R	Undefined
0	30:6	Must be written as zero; returns zero on read.	0	0
Index	5:0	TLB index. Software writes this field to provide the index to the TLB entry referenced by the TLBR and TLBWI instructions. Hardware writes this field with the index of the matching TLB entry during execution of the TLBP instruction. If the TLBP fails to find a match, the contents of this field are **UNPREDICTABLE**.	R/W	Undefined

6.2.2 Random Register (CP0 Register 1, Select 0)

The *Random* register is a read-only register whose value is used to index the TLB during a TLBWR instruction. The width of the Random field is calculated in the same manner as that described for the *Index* register above.

The value of the register varies between an upper and lower bound as follow:

- A lower bound is set by the number of TLB entries reserved for exclusive use by the operating system (the contents of the *Wired* register). The entry indexed by the *Wired* register is the first entry available to be written by a TLB Write indexed operation.
- An upper bound is set by the total number of TLB entries minus 1.

Within the required constraints of the upper and lower bounds, the processor selects values for the Random register according to the Not Last Used algorithm. The value is selected in a round-robin manner within the upper and lower bounds. However, when the selected value is the last entry hit, the next value is selected instead. This algorithm avoids a potential live lock condition that exists for implementations that simply increment the Random field every 'n' cycles.

The processor initializes the *Random* register to the upper bound on a Reset exception and when the *Wired* register is written.

Random Register Format

31 6	5 0
0	Random

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 6-4 Random Register Field Descriptions

Fields				
Name	**Bit(s)**	**Description**	**Read/ Write**	**Reset State**
0	31:6	Must be written as zero; returns zero on read.	0	0
Random	5:0	TLB Random Index	R	TLB Entries - 1

6.2.3 EntryLo0, EntryLo1 (CP0 Registers 2 and 3, Select 0)

The pair of *EntryLo* registers act as the interface between the TLB and the TLBR, TLBWI, and TLBWR instructions. The *EntryLo0* register stores the entries for even pages and *EntryLo1* register stores the entries for odd pages.

The contents of the *EntryLo0* and *EntryLo1* registers are undefined after an address error, TLB invalid, TLB modified, or TLB refill exceptions.

EntryLo0, EntryLo1 Register Format

63 30	29 6	5 3	2	1	0
0	PFN	C	D	V	G

Table 6-5 EntryLo0, EntryLo1 Register Field Descriptions

Fields				
Name	**Bit(s)**	**Description**	**Read/ Write**	**Reset State**
0	63:30	Reserved. Should be ignored on writes; returns zero on read.	R	0
PFN	29:6	Page Frame Number. Corresponds to bits 35:12 of the physical address.	R/W	Undefined
C	5:3	Coherency attribute of the page. See Table 6-6.	R/W	Undefined
D	2	“Dirty” or write-enable bit, indicating that the page has been written, and/or is writable. If this bit is a one, stores to the page are permitted. If this bit is a zero, stores to the page cause a TLB Modified exception.	R/W	Undefined
V	1	Valid bit, indicating that the TLB entry, and thus the virtual page mapping are valid. If this bit is a one, accesses to the page are permitted. If this bit is a zero, accesses to the page cause a TLB Invalid exception.	R/W	Undefined
G	0	Global bit. On a TLB write, the logical AND of the G bits in both the EntryLo0 and EntryLo1 registers become the G bit in the TLB entry. If the TLB entry G bit is a one, ASID comparisons are ignored during TLB matches. On a read from a TLB entry, the G bits of both EntryLo0 and EntryLo1 reflect the state of the TLB G bit.	R/W	Undefined

Table 6-6 lists the encoding of the C field of the *EntryLo0* and *EntryLo1* registers and the K0 field of the *Config* register. In prior MIPS implementations, there was little consistent interpretation or usage of the Cache Coherency Attribute

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

encodings. Table 6-6 describes the 20Kc-specific definitions that are permitted by the MIPS64 architecture and should not be interpreted as being required by the MIPS64 architecture.

Table 6-6 Cache Coherency Attributes

C[5:3] Value	Cache Coherency Attributes
0	Cacheable, Noncoherent, Write Through, No Write Allocate
1	Reserved
2	Uncached
3	Cacheable, Noncoherent (Writeback)
4	Cacheable, Coherent (Writeback)
5	Reserved
6	Reserved
7	Uncached Accelerated

6.2.4 Context Register (CP0 Register 4, Select 0)

The *Context* register is a read/write register containing a pointer to an entry in the page table entry (PTE) array. This array is an operating system data structure that stores virtual-to-physical translations. During a TLB miss, the operating system loads the TLB with the missing translation from the PTE array. The *Context* register is primarily intended for use with the TLB Refill handler, but is also loaded by hardware on an XTLB Refill and may be used by software in that handler. The *Context* register duplicates some of the information provided in the *BadVAddr* register but is organized in such a way that the operating system can directly reference an 8-byte page table entry (PTE) in memory.

A TLB exception (TLB Refill, TLB Invalid, or TLB Modified) causes bits $VA_{31:13}$ of the virtual address to be written into the BadVPN2 field of the *Context* register. The PTEBase field is written and used by the operating system.

The BadVPN2 field of the *Context* register is not defined after an address error exception and this field may be modified by hardware during the address error exception sequence.

Context Register Format

63 ... 23	22 ... 4	3 2 1 0
PTEBase	BadVPN2	0

Table 6-7 Context Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	Bit(s)			
PTEBase	63:23	This field is for use by the operating system and is normally written with a value that allows the operating system to use the *Context* Register as a pointer into the current PTE array in memory.	R/W	Undefined
BadVPN2	22:4	This field is written by hardware on a TLB miss. It contains bits $VA_{31:13}$ of the virtual address that missed.	R	Undefined
0	3:0	Must be written as zero; returns zero on read.	0	0

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

6.2.5 PageMask Register (CP0 Register 5, Select 0)

The *PageMask* register is a read/write register used for reading from and writing to the TLB. It holds a comparison mask that sets the variable page size for each TLB entry as shown in Table 6-9.

PageMask Register Format

31 25	24 13	12 0
0	Mask	0

Table 6-8 PageMask Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	**Bit(s)**			
Mask	24:13	The Mask field is a bit mask in which a "1" indicates that the corresponding bit of the virtual address should not participate in the TLB match.	R/W	Undefined
0	31:25, 12:0	Must be written as zero; returns zero on read.	0	0

Table 6-9 Values for the Mask Field of the PageMask Register

Page Size	Bit											
	24	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**
4 KBytes	0	0	0	0	0	0	0	0	0	0	0	0
16 KBytes	0	0	0	0	0	0	0	0	0	0	1	1
64 KBytes	0	0	0	0	0	0	0	0	1	1	1	1
256 KBytes	0	0	0	0	0	0	1	1	1	1	1	1
1 MByte	0	0	0	0	1	1	1	1	1	1	1	1
4 MByte	0	0	1	1	1	1	1	1	1	1	1	1
16 Mbyte	1	1	1	1	1	1	1	1	1	1	1	1

6.2.6 Wired Register (CP0 Register 6, Select 0)

The *Wired* register is a read/write register that specifies the boundary between the wired and random entries in the TLB as shown in Figure 6-1. The width of the Wired field is calculated in the same manner as that described for the *Index* register above. Wired entries are fixed, non-replaceable entries that are not overwritten by a TLBWR instruction. Wired entries can be overwritten by a TLBWI instruction.

The *Wired* register is set to zero by a Reset exception. Writing the *Wired* register causes the *Random* register to reset to its upper bound.

The operation of the processor is undefined if a value greater than or equal to the number of TLB entries is written to the *Wired* register.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

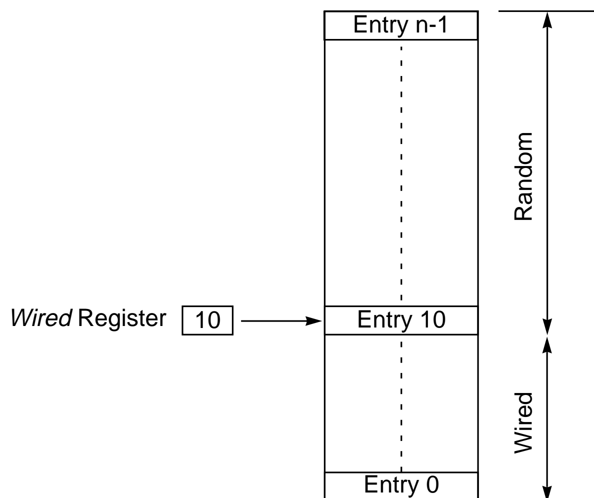


Figure 6-1 Wired and Random Entries in the TLB

Wired Register Format

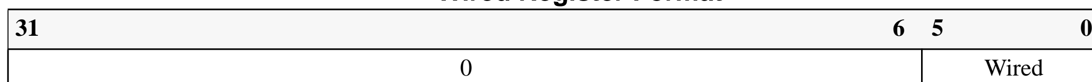


Table 6-10 Wired Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	Bit(s)			
0	31:6	Must be written as zero; returns zero on read.	0	0
Wired	5:0	TLB wired boundary.	R/W	0

6.2.7 BadVAddr Register (CP0 Register 8, Select 0)

The *BadVAddr* register is a read-only register that captures the most recent virtual address that caused one of the following exceptions:

- Address error (AdEL or AdES)
- TLB/XTLB Refill
- TLB Invalid (TLBL, TLBS)
- TLB Modified

The *BadVAddr* register does not capture address information for cache or bus errors, since neither is an addressing error.

BadVAddr Register Format

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 6-11 BadVAddr Register Field Description

Fields		Description	Read/ Write	Reset State
Name	Bits			
BadVAddr	63:0	Bad virtual address	R	Undefined

6.2.8 Count Register (CP0 Register 9, Select 0)

The Count register acts as a timer, incrementing at a constant rate whether or not an instruction is executed, retired, or any forward progress is made through the pipeline. The 20Kc processor increments the counter by one every clock.

The *Count* register continues to count when the processor enters a low power mode, as might occur after executing the Wait instruction. The Count register can be written for functional or diagnostic purposes, including at reset or to synchronize processors.

The Count register continues incrementing while the processor is in debug mode.

Count Register Format

31 ... 0
Count

Table 6-12 Count Register Field Description

Fields		Description	Read/ Write	Reset State
Name	Bits			
Count	31:0	Interval counter.	R/W	Undefined

6.2.9 EntryHi Register (CP0 Register 10, Select 0)

The *EntryHi* register contains the virtual address match information used for TLB read, write, and access operations.

A TLB exception (TLB Refill, XTLB Refill, TLB Invalid, or TLB Modified) causes the bits corresponding to the R and VPN2 fields to be written into the *EntryHi* register. The ASID field is written by software with the current address space identifier value and is used during the TLB comparison process to determine TLB match.

The VPN2 and R fields of the *EntryHi* register are not defined after an address error exception and these fields can be modified by hardware during the address error exception sequence. Software writes of the *EntryHi* register (via MTC0 or DMTC0) do not cause the implicit write of address-related fields in the *BadVAddr*, *Context*, or *XContext* registers.

EntryHi Register Format

63 62	61 ... 40	39 ... 13	12 ... 8	7 ... 0
R	0	VPN2	0	ASID

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 6-13 EntryHi Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	Bits			
R	63:62	Virtual memory region, corresponding to $VA_{63:62}$. 00: xuseg: user address region 01: xsseg: supervisor address region. 10: Reserved 11: kseg: kernel address region This field is written by hardware on a TLB exception or on a TLB read, and is written by software before a TLB write.	R/W	Unspecified
VPN2	39:13	$VA_{39:13}$ of the virtual address (virtual page number / 2). This field is written by hardware on a TLB exception or on a TLB read, and is written by software before a TLB write.	R/W	Unspecified
ASID	7:0	Address space identifier. This field is written by hardware on a TLB read and by software to establish the current ASID value for TLB write and against which TLB references match each entry's TLB ASID field.	R/W	Unspecified
0	61:40, 12:8	Must be written as zero; returns zero on read.	0	0

6.2.10 Compare Register (CP0 Register 11, Select 0)

The *Compare* register acts in conjunction with the *Count* register to implement a timer and timer interrupt function. The *Compare* register maintains a stable value and does not change on its own.

When the value of the *Count* register equals the value of the *Compare* register, an interrupt request is multiplexed with hardware interrupt 5 to set interrupt bit IP(7) in the *Cause* register. This causes an interrupt as soon as the interrupt is enabled.

For diagnostic purposes, the *Compare* register is a read/write register. In normal use, however, the *Compare* register is write-only. Writing a value to the *Compare* register, as a side effect, clears the timer interrupt.

Compare Register Format

31 ... 0
Compare

Table 6-14 Compare Register Field Description

Fields		Description	Read/ Write	Reset State
Name	Bit(s)			
Compare	31:0	Interval count compare value	R/W	Undefined

6.2.11 Status Register (CP0 Register 12, Select 0)

The *Status* register (SR) is a read/write register that contains the operating mode, interrupt enabling, and the diagnostic states of the processor. Fields of this register combine to create operating modes for the processor, as follows:

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Interrupt Enable: Interrupts are enabled when all of the following conditions are true:

- IE = 1
- EXL = 0
- ERL = 0
- DM = 0

If these conditions are met, the settings of the IM and IE bits enable the interrupt.

Operating Modes: The following CPU Status register bit settings are required for user, supervisor, and kernel modes.

- User Mode: KSU = 10, EXL = 0, ERL = 0, and DM = 0
- Supervisor Mode: KSU = 01, EXL = 0, ERL = 0, and DM = 0
- Kernel Mode: KSU = 00, or EXL = 1, or ERL = 1, and DM = 0

Coprocessor Accessibility: The Status register CU bits control coprocessor accessibility. If any coprocessor is unusable, an instruction that accesses it generates an exception.

Coprocessor 0 is always enabled in kernel mode, regardless of the setting of the CU0 bit.

Status Register Format

31 28	27	26	25	24	23	22	21	20	19	18	17	16	15 8	7	6	5	4 3	2	1	0
CU3-CU0	RP	FR	RE	0	PX	BEV	TS	SR	NMI	0	CE	DE	IM7-IM0	KX	SX	UX	KSU	ERL	EXL	IE

Table 6-15 Status Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	Bits			
CU3:CU0	31:28	Controls access to Coprocessors 1 and 0: CU0 controls accesses to CP0, while CU1 controls accesses to CP1. Each bit is defined as follows: 0: access not allowed 1: access allowed Coprocessor 0 can always be accessed when the processor is running in Kernel Mode or Debug Mode, independent of the state of the CU_0 bit. Execution of all floating-point instructions, including those encoded with the COP1X opcode, is controlled by the CU1 enable. CU2 and CU3 are not used. They are ignored on write and are read as zeros.	R/W	Unspecified
RP	27	Setting this bit initiates reduced power mode. Resetting the bit returns the 20Kc processor to normal operation. Refer to Chapter 10, “Power Management,” for more information.	R/W	0
FR	26	Controls the floating-point register mode: 0: Floating-point registers can contain any 32-bit data type. 64-bit datatypes are stored in even-odd pairs of registers. 1: Floating-point registers can contain any data type.	R/W	Unspecified

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 6-15 Status Register Field Descriptions (Continued)

Fields		Description	Read/ Write	Reset State
Name	Bits			
RE	25	Used to enable reverse-endian memory references while the processor is running in user mode: 0: User Mode uses configured endianness 1: User Mode uses reversed endianness Debug Mode, Kernel Mode, and Supervisor Mode references are not affected by the state of this bit.	R/W	Unspecified
0	24	Ignored on a write and read as zero.	R	0
PX	23	Enables access to 64-bit operations in User Mode without enabling 64-bit addressing: 0: 64-bit operations are not enabled 1: 64-bit operations are enabled	R/W	Unspecified
BEV	22	Controls the location of exception vectors: 0: Normal 1: Bootstrap	R/W	1
TS	21	Indicates that the TLB has detected a match on multiple entries. This detection occurs only on a write to the TLB. When such a detection occurs, the processor initiates a machine check exception and sets this bit. The existing TLB entry or entries is retained but ceases to participate in address translation. The entry being written is stored into the TLB, and the processor continues operation. This condition can be corrected by software. Software should clear this bit before resuming normal operation. Software writes to this bit may not cause a 0-to-1 transition.	R/W	0
SR	20	Indicates that the entry through the reset exception vector was due to a Soft Reset: 0: Not Soft Reset (NMI or Reset) 1: Soft Reset	R/W	1 for Soft Reset; 0 otherwise
NMI	19	Indicates that the entry through the reset exception vector was due to an NMI 0: Not NMI (Soft Reset or Reset) 1: NMI	R/W	1 for NMI; 0 otherwise
0	18	Must be written as zero; returns zero on read.	0	0
CE	17	Enables forcing cache parity error for store instructions: 0: data parity is computed 1: value in the PA field of the *DErrCtl* register is used instead	R/W	Unspecified
DE	16	Disables exceptions caused by cache parity errors. 0: enabled 1: disabled	R/W	Unspecified

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 6-15 Status Register Field Descriptions (Continued)

Fields		Description	Read/ Write	Reset State
Name	Bits			
IM	15:8	Interrupt Mask: Controls the enabling of each of the external, internal and software interrupts. 0: interrupt request disabled 1: interrupt request enabled	R/W	Unspecified
KX	7	Enables the following behavior: Access to 64-bit Kernel Segments Use of the XTLB Refill Vector for references to Kernel Segments 0: Access to 64-bit Kernel Segments disabled, TLB Refill Vector used for references to Kernel Segments 1: Access to 64-bit Kernel Segments enabled, XTLB Refill Vector used for references to Kernel Segments	R/W	Unspecified
SX	6	Enables the following behavior: Access to 64-bit Supervisor Segments Use of the XTLB Refill Vector for references to Supervisor Segments 0: Access to 64-bit Supervisor Segments disabled, TLB Refill Vector used for references to Supervisor Segments 1: Access to 64-bit Supervisor Segments enabled, XTLB Refill Vector used for references to Supervisor Segments	R/W	Unspecified
UX	5	Enables the following behavior: Access to 64-bit User Segments Use of the XTLB Refill Vector for references to User Segments Execution of instructions which perform 64-bit operations while the processor is operating in User Mode 0: Access to 64-bit User Segments disabled, TLB Refill Vector used for references to User Segments, execution of instructions which perform 64-bit operations is disallowed while the processor is running in User Mode 1: Access to 64-bit User Segments enabled, XTLB Refill Vector used for references to User Segments, execution of instructions which perform 64-bit operations is allowed while the processor is running in User Mode	R/W	Unspecified
KSU	4:3	The encoding of this field denotes the base operating mode of the processor. The encoding of this field is: 00_2: Kernel Mode 01_2: Supervisor Mode 10_2: User Mode 11_2: Reserved	R/W	Unspecified

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 6-15 Status Register Field Descriptions (Continued)

Fields		Description	Read/ Write	Reset State
Name	Bits			
ERL	2	Error Level; Set by the processor when a Reset, Soft Reset, NMI or Cache Error exception are taken. 0: normal level 1: error level When ERL is set: • The processor is running in kernel mode • Interrupts are disabled • The ERET instruction uses the return address held in ErrorEPC instead of EPC • The entire 2^{32} bytes of kuseg are treated as an unmapped and uncached region. In addition when the UX bit is a one in the Status register, the range of addresses between 2^{32} and 2^{36}- 1 are also treated as an unmapped, uncached Segment. Accesses to xkuseg to addresses between 2^{36} and 2^{40} - 1 however would result in address error exceptions. This allows main memory to be accessed in the presence of cache errors. The operation of the processor is **UNDEFINED** if the ERL bit is set while the processor is executing instructions from kuseg.	R/W	1
EXL	1	Exception Level; Set by the processor when any exception other than Reset, Soft Reset, NMI or Cache Error exception are taken. 0: normal level 1: exception level When EXL is set: • The processor is running in Kernel Mode • Interrupts are disabled. • TLB/XTLB Refill exceptions will use the general exception vector instead of the TLB/XTLB Refill vectors. • EPC and $Cause_{BD}$ will not be updated if another exception is taken	R/W	Unspecified
IE	0	Interrupt Enable: Acts as the master enable for software and hardware interrupts: 0: disable interrupts 1: enables interrupts	R/W	Unspecified

6.2.12 Cause Register (CP0 Register 13, Select 0)

The *Cause* register primarily describes the cause of the most recent exception. In addition, fields also control software interrupt requests and the vector through which interrupts are dispatched. With the exception of the IP[1:0], IV, and WP fields, all fields in the Cause register are read-only.

Cause Register Format

31	30	29 28	27 24	23	22	21 16	15 10	9 8	7	6 2	1 0
BD	0	CE	0	IV	WP	0	IP[7:2]	IP[1:0]	0	Exc Code	0

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 6-16 Cause Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	Bit(s)			
BD	31	Indicates whether the last exception taken occurred in a branch delay slot: 0: Not in delay slot 1: In delay slot Note that the BD bit is not updated on a new exception if the EXL bit is set.	R	Undefined
CE	29:28	Coprocessor unit number referenced when a Coprocessor Unusable exception is taken. This field is loaded by hardware on every exception but is unpredictable for all exceptions except for Coprocessor Unusable.	R	Undefined
IV	23	Indicates whether an interrupt exception uses the general exception vector or a special interrupt vector: 0: Use the general exception vector (0x180) 1: Use the special interrupt vector (0x200)	R/W	Undefined
WP	22	Indicates that a watch exception was deferred because $Status_{EXL}$ or $Status_{ERL}$ was a one at the time the watch exception was detected. This bit indicates that the watch exception was deferred and causes the exception to be initiated once $Status_{EXL}$ and $Status_{ERL}$ are both zero. As such, software must clear this bit as part of the watch exception handler to prevent a watch exception loop.	R/W	Undefined
IP[7:2]	15:10	Indicates an external interrupt is pending: 15: Hardware interrupt 5 (or timer interrupt or perf. counter) 14: Hardware interrupt 4 13: Hardware interrupt 3 12: Hardware interrupt 2 11: Hardware interrupt 1 10: Hardware interrupt 0	R	Undefined
IP[1:0]	9:8	Controls the request for software interrupts: 9: Request software interrupt 1 8: Request software interrupt 0	R/W	Undefined
Exc Code	6:2	Exception code — see Table 6-17.	R	Undefined
0	30, 27:24, 21:16, 7, 1:0	Must be written as zero; returns zero on read.	0	0

Table 6-17 Cause Register Exc Code Field Descriptions

Exception Code Value	Mnemonic	Description
0	Int	Interrupt

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 6-17 Cause Register Exc Code Field Descriptions (Continued)

Exception Code Value	Mnemonic	Description
1	Mod	TLB modification exception
2	TLBL	TLB exception (load or instruction fetch)
3	TLBS	TLB exception (store)
4	AdEL	Address error exception (load or instruction fetch)
5	AdES	Address error exception (store)
6	IBE	Bus error exception (instruction fetch)
7	DBE	Bus error exception (data reference: load or store)
8	Sys	Syscall exception
9	Bp	Breakpoint exception
10	RI	Reserved instruction exception
11	CpU	Coprocessor Unusable exception
12	Ov	Integer Overflow exception
13	Tr	Trap exception
14	-	Reserved
15	FPE	Floating-Point Exception
16-22	-	Reserved
23	WATCH	Reference to WatchHi/WatchLo address
24	MCheck	Machine check
25-29	-	Reserved
30	CacheErr	Cache error. In normal mode a cache error exception has a dedicated vector and the Cause register is not updated. If a cache error occurs while in debug mode, this code is used to indicate that reentry to debug mode was caused by a cache error.
31	-	Reserved

6.2.13 Exception Program Counter (CP0 Register 14, Select 0)

The *Exception Program Counter (EPC)* is a read/write register that contains the address at which processing resumes after an exception has been serviced. All bits of the *EPC* register are significant and must be writable.

For synchronous (precise) exceptions, the *EPC* contains one of the following:

- The virtual address of the instruction that was the direct cause of the exception.
- The virtual address of the immediately preceding branch or jump instruction, when the exception causing instruction is in a branch delay slot and the Branch Delay bit in the *Cause* register is set.

On new exceptions, the processor does not write to the *EPC* register when the EXL bit in the *Status* register is set. However, the register can still be written via the MTC0 instruction.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

EPC Register Format

63 ... 0
EPC

Table 6-18 EPC Register Field Description

Fields		Description	Read/ Write	Reset State
Name	Bit(s)			
EPC	63:0	Exception Program Counter.	R/W	Undefined

6.2.14 Processor Identification (CP0 Register 15, Select 0)

The *Processor Identification (PRId)* register is a 32-bit read-only register that contains information identifying the manufacturer, manufacturer options, processor identification, and revision level of the processor.

Processor Identification Register Format

31 ... 24	23 ... 16	15 ... 8	7 ... 0
Company Options	Company ID	Processor ID	Revision

Table 6-19 PRId Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	Bit(s)			
Company Options	31:24	Available to the designer or manufacturer of the processor for company-dependent options. SOC designers can hardwire the setting using input port *SI_PRIdOpt[7:0]*.	R	Preset to the state of *SI_PRIdOpt[7:0]*
Company ID	23:16	Identifies the company that designed or manufactured the processor. The setting denotes "MIPS Technologies".	R	0x01
Processor ID	15:8	Identifies the type of processor. This field allows software to distinguish between the various types of MIPS Technologies processors. The setting denotes "MIPS64 20Kc Processor Core".	R	0x82
Revision	7:0	The revision number of the processor. This field allows software to distinguish between one revision and another of the same processor type. The revision is broken into the following subfields: [7:5] Major revision [4:2] Minor revision [1:0] Patch revision The revision field is initially set to 0x20 and will be modified on each revision of the processor.	R	Preset

6.2.15 Config Register (CP0 Register 16, Select 0)

The *Config* register specifies various configuration and capabilities information. Most of the fields in the *Config* register are initialized by hardware during the Reset exception process, or are constant. One field, K0, must be initialized by software in the Reset exception handler.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Config Register Format — Select 0

31	30 28	27	26	25	24	23	22	21 16	15	14 13	12 10	9 7	6 4	3	2 0
M	EC	DD	LP	SP	TI	TD	TF	0	BE	AT	AR	MT	0	VI	K0

Table 6-20 Config Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	Bits			
M	31	Denotes that the Config1 register is implemented at a select field value of 1.	R	1
EC	30:28	Processor to System (External) clock ratio 0: Reserved 1: 2:1 2: 3:1 3: 4:1 4: 5:1 5: 6:1 6: 7:1 7: 8:1	R	Externally Set
DD	27	Indicates double data rate bus mode: 0: synchronous bus mode 1: double data rate bus mode	R	0
LP	26	Indicates a bus configuration with low packaging cost: 0: 64-bit bus 1: 32-bit bus	R	Externally Set
SP	25	System Command Parity. This bit determines when parity is checked on the SysCmd and SysAD buses. 0 - SysCmd parity is never checked. SysAD parity is not checked for command transactions. i.e. when SysAD contains address. 1 - SysCmd parity is always checked and SysAD parity is checked for command transactions.	R/W	0
TI	24	Timer Disable. The Timer interrupt is multiplexed with the hardware interrupt 5 based on the value of this bit. If this bit is set, hardware interrupt 5 is selected. If this bit is cleared the timer interrupt is selected. This multiplexed value is then ORed with the performance counter interrupt to create the final hardware interrupt 5.	R/W	0

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 6-20 Config Register Field Descriptions (Continued)

Fields		Description	Read/ Write	Reset State
Name	Bits			
TD	23	Test Done. The value of *Config* register bit TD is multiplexed on external pin *TS_BistDone* and while the core is not in the processor monitor mode and BIST is not active. TD TF Description 0 0 No New Action 0 1 Reserved 1 0 Output a pass signature on the external pin 1 1 Output a fail signature on the external pin	R/W	0
TF	22	Test Fail. Refer to TD field. The values of *Config* register bit TF are multiplexed on external pin *TS_BistFail* while the core is not in the processor monitor mode and BIST is not active.	R/W	0
0	21:16	Must be written as zero.	0	Unspecified
BE	15	Endian mode in which the processor is running: 0: Little endian 1: Big endian	R	Externally Set
AT	14:13	Architecture type implemented by the processor: 0: MIPS32 1: MIPS64 with 32-bit addresses only 2: MIPS64 with 32/64-bit addresses 3: Reserved	R	2
AR	12:10	Architecture revision level: 0: Revision 1 1-7: Reserved	R	0
MT	9:7	MMU Type: 0: None 1: Standard TLB 2: Standard BAT 3: Standard fixed mapping 4: Reserved 5: Reserved 6: Reserved 7: Reserved	R	1
0	6:4	Must be written as zero; returns zero on read.	0	0
VI	3	I-Cache is virtual.	R	1

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 6-20 Config Register Field Descriptions (Continued)

Fields		Description	Read/ Write	Reset State
Name	Bits			
K0	2:0	Kseg0 coherency algorithm. See Table 6-6 for the encoding of this field.	R/W	2

6.2.16 Config1 Register (CP0 Register 16, Select 1)

The *Config1* register is an adjunct to the Config register and encodes additional capabilities information. All fields in the Config1 register are read-only.

The instruction and data cache configuration parameters include encodings for the number of sets per way, the line size, and the associativity. The total cache size for a cache is therefore:

Associativity * Line Size * Sets Per Way = 4 ways * 32 bytes * 256 sets = 32 Kbytes

Config1 Register Format — Select 1

31	30 25	24 22	21 19	18 16	15 13	12 10	9 7	6 5	4	3	2	1	0
0	MMU Size	IS	IL	IA	DS	DL	DA	0	PC	WR	CA	EP	FP

Table 6-21 Config1 Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	Bits			
0	31	This bit is reserved to indicate that a Config2 register is present. With this revision of the architecture, writes to this bit must be ignored, and it must read as zero. Since the 20Kc processor does not implement a Config2 register, this bit is always zero.	R	0
MMU Size - 1	30:25	Number of entries in the TLB minus one.	R	47
IS	24:22	ICache sets per way: 0: 64 1: 128 2: 256 3: 512 4: 1024 5: 2048 6: 4096 7: Reserved In the 20Kc processor, this value is 0x02 to indicate 256 ICache sets per way.	R	2

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 6-21 Config1 Register Field Descriptions (Continued)

Fields		Description	Read/ Write	Reset State
Name	Bits			
IL	21:19	ICache line size: 0: No ICache present 1: 4 bytes 2: 8 bytes 3: 16 bytes 4: 32 bytes 5: 64 bytes 6: 128 bytes 7: Reserved In the 20Kc processor, this value is 0x04 to indicate a 32-byte ICache line size.	R	4
IA	18:16	ICache associativity: 0: Direct mapped 1: 2-way 2: 3-way 3: 4-way 4: 5-way 5: 6-way 6: 7-way 7: 8-way In the 20Kc processor, this value is 0x03 to indicate a 4-way set associative instruction cache.	R	3
DS	15:13	DCache sets per way: 0: 64 1: 128 2: 256 3: 512 4: 1024 5: 2048 6: 4096 7: Reserved In the 20Kc processor, this value is 0x02 to indicate 256 data cache sets per way.	R	2

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 6-21 Config1 Register Field Descriptions (Continued)

Fields		Description	Read/ Write	Reset State
Name	Bits			
DL	12:10	DCache line size: 0: No DCache present 1: 4 bytes 2: 8 bytes 3: 16 bytes 4: 32 bytes 5: 64 bytes 6: 128 bytes 7: Reserved In the 20Kc processor, this value is 0x04 to indicate a 32-byte data cache line size.	R	4
DA	9:7	DCache associativity: 0: Direct mapped 1: 2-way 2: 3-way 3: 4-way 4: 5-way 5: 6-way 6: 7-way 7: 8-way In the 20Kc processor, this value is 0x03 to indicate a 4-way set associative data cache.	R	3
PC	4	Performance Counter registers implemented: 0: No performance counter registers implemented 1: At least one performance counter register implemented This bit is always 1 to indicate that the 20Kc processor implements one performance register.	R	1
WR	3	Watch registers implemented: 0: No watch registers implemented 1: At least one watch register implemented This bit is always 1 to indicate that the 20Kc processor implements one watch register.	R	1
CA	2	Code compression (MIPS16) implemented: 0: No code compression 1: Code compression This bit is always 0 to indicate that the 20Kc processor does not support code compression.	R	0

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 6-21 Config1 Register Field Descriptions (Continued)

Fields		Description	Read/ Write	Reset State
Name	Bits			
EP	1	EJTAG implemented: 0: No EJTAG implemented 1: EJTAG implemented This bit is always 1 to indicate that the 20Kc processor implements EJTAG.	R	1
FP	0	FPU implemented: 0: No FPU 1: FPU This bit is always 1 to indicate that the 20Kc processor implements a FPU.	R	1
0	6:5	Must be written as zero; returns zero on read.	0	0

6.2.17 Load Linked Address (CP0 Register 17, Select 0)

The *LLAddr* register contains the physical address read by the most recent Load Linked (LL) instruction. This register is for diagnostic purposes only and serves no function during normal operation.

Load Linked Address Register Format

63 — 36	35 — 0
0	PAddr[35:0]

Table 6-22 LLAddr Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	Bit(s)			
0	63:36	Must be written as zero; returns zero on read.	0	0
PAddr[35:0]	35:0	Physical address read by the most recent Load Linked instruction.	R	Undefined

6.2.18 WatchLo Register (CP0 Register 18)

The *WatchLo* and *WatchHi* registers together provide the interface to a watchpoint debug facility that initiates a watch exception if an instruction or data access matches the address specified in the registers. As such, they duplicate some functions of the EJTAG debug solution. Watch exceptions are taken only if the EXL and ERL bits are zero in the *Status* register. If either bit is a one, the WP bit is set in the *Cause* register, and the watch exception is deferred until both the EXL and ERL bits are zero.

The 20Kc processor implements one pair of WatchLo and WatchHi registers, referencing them via the select field of the MTC0/MFC0 and DMTC0/DMFC0 instructions. Software may determine that there is at least one pair of *WatchLo* and *WatchHi* registers implemented via the WR bit of the *Config1* register. See the discussion of the M bit in the *WatchHi* register description below.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

The *WatchLo* register specifies the base virtual address and the type of reference (instruction fetch, load, store) to match. Software may determine which enables are supported by a particular Watch register pair by setting all three enable bits and reading them back to see which ones were actually set. The 20Kc processor supports all three in the register pair.

WatchLo Register Format

63 ... 3	2	1	0
VAddr	I	R	W

Table 6-23 WatchLo Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	Bits			
VAddr	63:3	The virtual address to match. Note that this is a doubleword address, since bits [2:0] are used to control the type of match.	R/W	Undefined
I	2	If this bit is set, watch exceptions are enabled for instruction fetches that match the address.	R/W	Undefined
R	1	If this bit is set, watch exceptions are enabled for loads that match the address.	R/W	Undefined
W	0	If this bit is set, watch exceptions are enabled for stores that match the address.	R/W	Undefined

6.2.19 WatchHi Register (CP0 Register 19)

The *WatchLo* and *WatchHi* registers together provide the interface to a watchpoint debug facility that initiates a watch exception if an instruction or data access matches the address specified in the registers. As such, they duplicate some functions of the EJTAG debug solution. Watch exceptions are taken only if the EXL and ERL bits are zero in the *Status* register. If either bit is a one, the WP bit is set in the *Cause* register, and the watch exception is deferred until both the EXL and ERL bits are zero.

The 20Kc processor implements one pair of WatchLo and WatchHi registers, referencing them via the select field of the MTC0/MFC0 and DMTC0/DMFC0 instructions. Software may determine that there is at least one pair of *WatchLo* and *WatchHi* registers implemented via the WR bit of the *Config1* register. If the M bit is one in the *WatchHi* register reference with a select field of ‘*n*’, another *WatchHi/WatchLo* pair are implemented with a select field of ‘*n+1*’. The 20Kc processor implements only one *WatchHi/WatchLo* pair.

The *WatchHi* register contains information that qualifies the virtual address specified in the *WatchLo* register: an ASID, a G(lobal) bit, and an optional address mask. If the G bit is 1, any virtual address reference that matches the specified address will cause a watch exception. If the G bit is a 0, only those virtual address references for which the ASID value in the *WatchHi* register matches the ASID value in the *EntryHi* register cause a watch exception. The optional mask field provides address masking to qualify the address specified in *WatchLo*.

WatchHi Register Format

31	30	29 ... 24	23 ... 16	15 ... 12	11 ... 3	2 ... 0
M	G	0	ASID	0	MASK	0

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 6-24 WatchHi Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	Bit(s)			
M	31	If this bit is one, another pair of *WatchHi*/*WatchLo* registers is implemented at an MTC0 or MFC0 select field value of 1. In the 20Kc processor, this bit is always zero to indicate that only one pair or *WatchHi*/*WatchLo* registers is implemented.	R	0
G	30	If this bit is one, any address that matches that specified in the *WatchLo* register causes a watch exception. If this bit is zero, the ASID field of the *WatchHi* register must match the ASID field of the *EntryHi* register to cause a watch exception.	R/W	Undefined
0	29:24	Must be written as zero; returns zero on read.	0	0
ASID	23:16	ASID value which is required to match that in the *EntryHi* register if the G bit is zero in the *WatchHi* register.	R/W	Undefined
0	15:12	Must be written as zero; returns zero on read.	0	0
Mask	11:3	Bit mask that qualifies the address in the *WatchLo* register. Any bit in this field that is a set inhibits the corresponding address bit from participating in the address match. Software can determine how many mask bits are implemented by writing ones in this field and then reading back the result.	R/W	Undefined
0	2:0	Must be written as zero; returns zero on read.	0	0

6.2.20 XContext Register (CP0 Register 20)

The *XContext* register is a read/write register containing a pointer to an entry in the page table entry (PTE) array. This array is an operating system data structure that stores virtual-to-physical translations. During a TLB miss, the operating system loads the TLB with the missing translation from the PTE array. The *XContext* register is primarily intended for use with the XTLB Refill handler, but is also loaded by hardware on a TLB Refill. However, it is unlikely to be useful to software in the TLB Refill Handler. The *XContext* register duplicates some of the information provided in the *BadVAddr* register, but is organized in such a way that the operating system can directly reference a 16-byte structure in memory that describes the mapping.

A TLB exception (TLB Refill, XTLB Refill, TLB Invalid, or TLB Modified) causes bits [63:62] of the virtual address to be written into the R field and bits [39:13] of the virtual address to be written into the BadVPN2 field of the *XContext* register. The PTEBase field is written and used by the operating system.

The BadVPN2 field of the *Context* register is not defined after an address error exception and this field can be modified by hardware during the address error exception sequence.

The PTEBase fields of the *Context* and *XContext* registers do not share storage so software can write different values to these fields.

XContext Register Format

63 ... 33	32 31	30 ... 4	3 ... 0
PTEBase	R	BadVPN2 ($VA_{39..13}$)	0

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 6-25 XContext Register Field Descriptions

Field		Description	Read/ Write	Reset State
Name	Bits			
PTEBase	63:33	This field is for use by the operating system and is normally written with a value that allows the operating system to use the *Context* Register as a pointer into the current PTE array in memory	R/W	Unspecified
R	32:31	The *Region* field contains bits 63:62 of the virtual address. 00 = xuseg 01 = xsseg: supervisor address region. 10 = Reserved 11 = xkseg	R	Unspecified
BadVPN2	30:4	The *Bad Virtual Page Number*/2 field is written by hardware on a TLB exception. It contains bits VA[39:13] of the virtual address that caused the exception.	R	Unspecified
0	3:0	Must be written as zero; returns zero on read.	0	0

6.2.21 Debug Register (CP0 Register 23)

The Debug register is used to control the debug exception and provide information about the cause of the debug exception and when re-entering at the debug exception vector due to a normal exception in debug mode. The read only information bits are updated every time the debug exception is taken or when a normal exception is taken when already in debug mode.

Only the DM bit and the EJTAGver field are valid when read from non-debug mode; the value of all other bits and fields is UNPREDICTABLE. Operation of the processor is UNDEFINED if the Debug register is written from non-debug mode.

Some of the bits and fields are only updated on debug exceptions and/or exceptions in debug mode, as shown below:

- DSS, DBp, DDBL, DDBS, DIB, DINT are updated on both debug exceptions and on exceptions in debug modes
- DExcCode is updated on exceptions in debug mode, and is undefined after a debug exception
- Halt and Doze are updated on a debug exception, and is undefined after an exception in debug mode
- DBD is updated on both debug and on exceptions in debug modes

All bits and fields are undefined when read from normal mode, except those explicitly described to be defined, e.g. EJTAGver and DM.

Debug Register Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18
DBD	DM	DCR	LSNM	Doze	Halt	CountDM	IBusEP	MCheckP	CacheEP	DBusEP	IEXI	DDBSImpr	DDBLImpr

17 15	14 10	9	8	7 6	5	4	3	2	1	0
EJTAGVer	DExcCode	NoSSt	SSt	0	DINT	DIB	DDBS	DDBL	DBp	DSS

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 6-26 Debug Register Field Descriptions

Fields		Description	Read/ Write	Reset
Mnemonic	Bit(s)			
DBD	31	Indicates whether the last debug exception or exception in debug mode, occurred in a branch delay slot: 0: Not in delay slot 1: In delay slot	R	Undefined
DM	30	Indicates that the processor is operating in debug mode: 0: Processor is operating in non-debug mode 1: Processor is operating in debug mode	R	0
NoDCR	29	Indicates whether the dseg memory segment is present. 0: dseg is present 1: No dseg present	R	0
LSNM	28	Controls access of load/store between dseg and remain memory: 0: Load/stores in dseg address range goes to dseg 1: Load/stores in dseg address range goes to remain memory	R/W	0
Doze	27	Indicates that the processor was low power mode when a debug exception occurred: 0: Processor not in low power mode when debug exception occurred 1: Processor in low power mode when debug exception occurred	R	Undefined
Halt	26	Indicates that the internal system bus clock was stopped when the debug exception occurred: 0: Internal system bus clock stopped 1: Internal system bus clock running	R	Undefined
CountDM	25	Indicates the Count register behavior in debug mode. Encoding of the bit is: 0: Count register stopped in debug mode 1: Count register is running in debug mode	R	1
IBusEP	24	Instruction fetch Bus Error exception Pending. Set when an instruction fetch bus error event occurs or if a 1 is written to the bit by software. Cleared when a Bus Error exception on instruction fetch is taken by the processor. If IBusEP is set when IEXI is cleared, a Bus Error exception on an instruction fetch is taken by the processor and the IBusEP bit is cleared.	R/W1	0
MCheckP	23	Unused in 20Kc processor	R	0
CacheEP	22	Indicates if a Cache Error is pending. Set when a cache error event occurs, or if a 1 is written to the bit by software. Cleared when a Cache Error exception is taken by the processor. If CacheEP is set when IEXI is cleared, a Cache Error exception is taken by the processor and the CacheEP bit is cleared. In debug mode, a Cache Error exception applies to a Debug Mode Cache Error exception.	R/W1	0

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 6-26 Debug Register Field Descriptions (Continued)

Fields		Description	Read/ Write	Reset
Mnemonic	Bit(s)			
DBusEP	21	Data access Bus Error exception Pending. Set when a data access bus error event occurs, or if a 1 is written to the bit by software. Cleared when a Bus Error exception on a data access is taken by the processor. If DBusEP is set when IEXI is cleared, a Bus Error exception on a data access is taken by the processor and the DBusEP bit is cleared.	R/W1	0
IEXI	20	Imprecise Error eXception Inhibit controls exceptions taken due to imprecise error indications. Set when the processor takes a debug exception or another exception in debug mode. Cleared by execution of the DERET instruction. This bit can be modified by debug mode software. When this bit is set, the imprecise error exceptions generated by a bus error on instruction fetch or data access, cache error, or machine check, are inhibited and deferred until the bit is cleared.	R/W	0
DDBSImpr	19	Indicates that a Debug Data Break Store Imprecise exception on a store was the cause of the debug exception, or that an imprecise data hardware break on a store was indicated after another debug exception occurred. Cleared on exception in Debug Mode. 0: No imprecise match of data hardware breakpoint on a store 1: Imprecise match of data hardware breakpoint on a store	R	Undefined
DDBLImpr	18	Indicates that a Debug Data Break Load Imprecise exception on a load was the cause of the debug exception, or that an imprecise data hardware break on a load was indicated after another debug exception occurred. Cleared on exception in Debug Mode. 0: No imprecise match of data hardware breakpoint on a load 1: Imprecise match of data hardware breakpoint on a load	R	Undefined
EJTAGVer	17:15	EJTAG version. 0: Versions 1 and 2.0 1: Version 2.5 2: Version 2.6 3-7: Reserved	R	2
DExcCode	14:10	Indicates the cause of the latest exception in debug mode. The field is encoded as the Exc Code field in the *Cause* register for those normal exceptions that may occur in debug mode. Value is undefined after a debug exception.	R	Undefined
NoSSt	9	Indicates whether the single-step feature controlled by the SSt bit is available. 0: Single-step feature available 1: Single-step feature not available	R	0
SSt	8	Controls if debug single step exception is enabled: 0: No debug single step exception enabled 1: Debug single step exception enabled	R/W	0
0	7:6	Reserved. Must be written as zero; returns zero on read.	0	0

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 6-26 Debug Register Field Descriptions (Continued)

Fields		Description	Read/ Write	Reset
Mnemonic	Bit(s)			
DINT	5	Indicates that a debug interrupt exception occurred. Cleared on exception in debug mode. 0: No debug interrupt exception 1: Debug interrupt exception	R	Undefined
DIB	4	Indicates that a debug instruction break exception occurred. Cleared on exception in debug mode. 0: No debug instruction exception 1: Debug instruction exception	R	Undefined
DDBS	3	Indicates that a debug data break exception occurred on a store. Cleared on exception in debug mode. 0: No debug data exception on a store 1: Debug instruction exception on a store	R	Undefined
DDBL	2	Indicates that a debug data break exception occurred on a load. Cleared on exception in debug mode. 0: No debug data exception on a load 1: Debug instruction exception on a load	R	Undefined
DBp	1	Indicates that a debug software breakpoint exception occurred. Cleared on exception in debug mode. 0: No debug software breakpoint exception 1: Debug software breakpoint exception	R	Undefined
DSS	0	Indicates that a debug single step exception occurred. Cleared on exception in debug mode. 0: No debug single step exception 1: Debug single step exception	R	Undefined

6.2.22 Debug Exception Program Counter Register (CP0 Register 24)

The Debug Exception Program Counter (*DEPC*) register is a read/write register that contains the address at which processing resumes after a debug exception or debug mode exception has been serviced.

For precise debug and debug mode precise exceptions, the *DEPC* contains either:

- The virtual address of the instruction that was the direct cause of the debug exception, or
- The virtual address of the immediately preceding branch or jump instruction, when the debug exception causing instruction is in a branch delay slot, and the Debug Branch Delay (BDB) bit in the *Debug* register is set.

For imprecise debug exceptions (debug interrupt) and imprecise exceptions in Debug Mode, the *DEPC* contains the virtual address of the instruction where execution should resume after the debug handler code is executed.

Debug Exception Program Counter Register Format

63 ... 0
DEPC

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 6-27 Debug Register Formats

Fields		Description	Read/ Write	Reset
Mnemonic	Bit(s)			
DEPC	63:0	The *DEPC* register is updated with the virtual address of the instruction that caused the debug exception. If the instruction is in the branch delay slot, the virtual address of the immediately preceding branch or jump instruction is placed in this register. Execution of the DERET instruction causes a jump to the address in the *DEPC*.	R/W	Undefined

6.2.23 Performance Counter Registers (CP0 Register 25, Selects 0, 1)

The 20Kc processor supports a single performance counter that provides the capability to count events or cycles for use in performance analysis. Each performance counter consists of a pair of registers: a 32-bit control register and a 32-bit counter register. The performance counter uses select 0 for control and select 1 for the counter.

The performance counter can be configured to count events or cycles under a specified set of conditions that are determined by the control register. The counter register increments once for each enabled event. When bit 31 of the counter register is one (the counter overflows), the performance counter optionally requests an interrupt that is combined with hardware interrupt 5 to set interrupt bit IP(7) in the *Cause* register. Counting continues after a counter register overflows, whether or not an interrupt is requested or taken.

Performance Counter Control Register Format

31	30 ... 11	10 ... 5	4	3	2	1	0
M	0	Event	IE	U	S	K	EXL

Table 6-28 Performance Counter Control Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	Bits			
M	31	This bit is cleared to indicate that the 20Kc processor supports one performance counter.	R	0
0	30:11	Must be written as zero; returns zero on read	0	0

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 6-28 Performance Counter Control Register Field Descriptions (Continued)

<table>
<tr><th colspan="2">Fields</th><th rowspan="2">Description</th><th rowspan="2">Read/
Write</th><th rowspan="2">Reset State</th></tr>
<tr><th>Name</th><th>Bits</th></tr>
<tr><td>Event</td><td>10:5</td><td>Selects the event to be counted by the corresponding Counter Register. The encodings for this field are as follows:
0x00 - counts CPU cycles
0x01 - counts dispatched/issued instructions
0x02 - counts fetch groups entering CPU execution pipes
0x03 - counts instructions completed in FPU datapath (computational instructions only)
0x04 - counts taken TLB refill exceptions
0x05 - counts branches that mispredicted before completing execution
0x06 - counts branches that completed execution
0x07 - counts taken Joint-TLB exceptions
0x08 - counts replays due to load-dependent speculative dispatch
0x09 - counts instruction requests from the IFU to the BIU
0x0A - counts taken FPU exceptions
0x0B - counts the total number of replays due to:
• LSU requested replays
• Load-dependent speculative dispatch
• FPU exception prediction
0x0C - counts JR instructions that mispredicted using the Return Prediction Stack (RPS)
0x0D - counts JR instruction that completed execution
0x0E - counts LSU requested replays
0x0F - counts instruction that completed execution (with or without exception)
0x10 - 0x3F - Reserved</td><td>R/W</td><td>Unspecified</td></tr>
<tr><td>IE</td><td>4</td><td>Interrupt Enable. Enables the interrupt request when the corresponding counter overflows (bit 31 of the counter is one).
Note that this bit simply enables the interrupt request. The actual interrupt is still gated by the normal interrupt masks and enable in the Status register.
0: Performance counter interrupt disabled
1: Performance counter interrupt enabled</td><td>R/W</td><td>0</td></tr>
<tr><td>U</td><td>3</td><td>Enables event counting in User Mode.
0: Disable event counting in User Mode
1: Enable event counting in User Mode</td><td>R/W</td><td>Unspecified</td></tr>
<tr><td>S</td><td>2</td><td>Enables event counting in Supervisor Mode.
0: Disable event counting in Supervisor Mode
1: Enable event counting in Supervisor Mode</td><td>R/W</td><td>Unspecified</td></tr>
</table>

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 6-28 Performance Counter Control Register Field Descriptions (Continued)

Fields		Description	Read/ Write	Reset State
Name	**Bits**			
K	1	Enables event counting in Kernel Mode. Unlike the usual definition of Kernel Mode, this bit enables event counting only when the EXL and ERL bits in the *Status* register are zero. 0: Disable event counting in Kernel Mode 1: Enable event counting in Kernel Mode	R/W	Unspecified
EXL	0	Enables event counting when the EXL bit in the *Status* register is one and the ERL bit in the *Status* register is zero. 0: Disable event counting while EXL = 1, ERL = 0 1: Enable event counting while EXL = 1, ERL = 0 Counting is never enabled when the ERL bit in the *Status* register is one.	R/W	Unspecified

The Counter Register associated with each performance counter increments once for each enabled event. Table 6-29 describes the Performance Counter Register fields.

Performance Counter Register Format

31 … 0
Event Count

Table 6-29 Performance Counter Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	**Bits**			
Event Count	31:0	Increments once for each event that is enabled by the corresponding Control Register. When bit 31 is one, an interrupt request is made if the IE bit in the Control Register is one.	R/W	Unspecified

6.2.24 DErrCtl Register (CP0 Register 26, Select 0)

The *DErrCtl* register provides a diagnostic interface with the error detection mechanism in the data cache. This register is used to read and write parity to and from the primary data cache data arrays. An Index Load Tag cache instruction causes the even byte parity of the accessed doubleword of data, retrieved into the *DDataHi* and *DDataLo* registers, to be written into the PA field of the *DErrCtl* register. If the CE bit in the *Status* register is set, a store hit will use the value in the PA field instead of the computed data parity to write into the cache.

DErrCtl Register Format

31 … 8	7 … 0
0	PA

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 6-30 DErrCtl Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	Bits			
0	31:8	Must be written as zero.	0	Unspecified
PA	7:0	Data parity. An Index Load Tag cache instruction computes the even byte parity of the accessed doubleword of data and writes it in this field. Bits 0-7 correspond to bytes 0-7 of the data. If CE in the *Status* register is 1, a store forces the value in this field into the cache as data parity. If CE is 0, the data parity will be computed.	R/W	Unspecified

6.2.25 IErrCtl Register (CP0 Register 26, Select 1)

The *IErrCtl* register provides a diagnostic interface with the error detection mechanism in the instruction cache. This register is used only to read parity from the primary instruction cache data arrays. An Index Load Tag cache instruction causes the even parity of the accessed doubleword of data, retrieved into the *IDataHi* and *IDataLo* registers, to be written into the PA field of the *IErrCtl* register.

IErrCtl Register Format

31 ... 8	7	6 ... 0
0	PA	0

Table 6-31 IErrCtl Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	Bits			
0	31:8	Must be written as zero.	0	Unspecified
PA	7	Data parity. An Index Load Tag cache instruction computes the even parity of the accessed doubleword of data and writes it in this field. Unlike the parity in *DErrCtl*, this field is a single bit for the entire doubleword.	R	Unspecified
0	6:0	Must be written as zero.	0	Unspecified

6.2.26 CacheErr Register (CP0 Register 27, Select 0)

The CacheErr register provides an interface with the cache error detection logic.

CacheErr Register Format

31 30	29	28	27	26	25	24	23 22	... 15	14 13	12 ... 5	4 ... 0
ER	ED	ET	ES	EE	EB	0	EW	0	WA	IN	0

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 6-32 CacheErr Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	Bits			
ER	31:30	Indicate the type of reference that encountered an error: 00: Instruction Cache 01: Prefetch Buffer 10: Data Cache 11: Fill Store Buffer	R	Unspecified
ED	29	Indicates a data error in ICache, PFB, DCache, or FSB: 0: No data error detected 1: Data error detected	R	Unspecified
ET	28	Indicates a tag error in ICache or DCache: 0: No tag error detected 1: Tag error detected	R	Unspecified
ES	27	Indicates data error on external request: 0: No error due to external request 1: Error due to external request	R	Unspecified
EE	26	Indicates a bus parity error: 0: No bus parity error 1: Bus parity error	R	Unspecified
EB	25	Indicates that an instruction error occurred in addition to a data error: 0: No additional instruction error 1: Additional instruction error	R	Unspecified
Reserved	24	Must be written as zero	0	Unspecified
EW	23	Indicates tag error on external request.	R	Unspecified
Reserved	22:15	Must be written as zero	0	Unspecified
WA	14:13	The cache way at which the error was detected.	R	Unspecified
IN	12:5	The cache index at which the error was detected.	R	Unspecified
Reserved	4:0	Must be written as zero	0	Unspecified

6.2.27 ITagLo Register (CP0 Register 28, Select 0)

The *ITagLo* and *ITagHi* registers act as the interface to the cache virtual tag array and are intended for diagnostic operation only. The Index Store Tag and Index Load Tag operations of the CACHE instruction use the *ITagLo* and *ITagHi* registers as the source or sink of tag information, respectively.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

ITagLo Register Format

31 … 8	7	6	5	4	3 … 1	0
VTagLo	PState	0	L	F	0	P

Table 6-33 ITagLo Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	Bits			
VTagLo	31:8	Virtual address bits 36:13.	R/W	Unspecified
PState	7	State for the cache tag. 0: Invalid 1: Valid	R/W	Unspecified
Reserved	6	Must be written as zero	0	Undefined
L	5	Lock bit for the Instruction cache tag. 0: Unlocked 1: Locked	R/W	Unspecified
F	4	Fill bit for the Least Recently Filled (LRF) algorithm.	R/W	Unspecified
Reserved	3:1	Must be written as zero	R	0
P	0	Even parity bit for the cache tag.	R/W	Unspecified

6.2.28 IDataLo Register (CP0 Register 28, Select 1)

The *IDataLo* and *IDataHi* registers are read-only registers that act as the interface to the cache data array and are intended for diagnostic operations only. The Index Load Tag operation of the CACHE instruction reads the corresponding data values into the *IDataLo* and *IDataHi* registers.

IDataLo Register Format

31 … 0
DATA

Table 6-34 IDataLo Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	Bit(s)			
DATA	31:0	Low-order data read from the data array (as opposed to the tag array) of the Instruction cache.	R	Undefined

Certain fields of the opcodes fetched from memory are somtimes modified before they are stored in the Instruction cache. The data in the *IDataLo* register reflects this modification. The algorithm used for modification is referred to as "instruction recoding" and is defined below:

- On all shift instructions (i.e. SLL, SRL, SRA, DSLL, DSRL, DSRA, DSLL32, DSRL32, DSRA32, SLLV, SRLV, SRAV, DSLLV, DSRLV, DSRAV), opcode field 25:21 is swapped with the opcode field 20:16 to form the recoded opcode.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

- For all branch instructions, recoded opcode field[14:0] = Program Counter [16:2] + raw opcode field [14:0] + 1.

6.2.29 DTagLo Register (CP0 Register 28, Select 2)

The *DTagLo* and *DTagHi* registers act as the interface to the cache physical tag array and are intended for diagnostic operation only. The Index Store Tag and Index Load Tag operations of the CACHE instruction use the *DTagLo* and *DTagHi* registers as the source or sink of tag information, respectively.

Software is allowed to write zeros into the *DTagLo* and *DTagHi* registers and then use the Index Store Tag cache operation to initialize the cache tags to a valid state at power-up.

DTagLo Register Format

31	30 ... 8	7 6	5	4	3 1	0
0	PTagLo	PState	L	F	0	P

Table 6-35 DTagLo Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	Bits			
Reserved	31	Must be written as zero.	R	0
PTagLo	30:8	Physical address bits 35:13.	R/W	Unspecified
PState	7:6	State for the cache tag. 0: Invalid 1: Reserved 2: Clean Exclusive 3: Dirty	R/W	Unspecified
L	5	Lock bit for the cache tag. 0: Unlocked 1: Locked	R/W	Unspecified
F	4	Fill bit for the Least Recently Filled (LRF) algorithm.	R/W	Unspecified
Reserved	3:1	Must be written as zero	0	Unspecified
P	0	Even parity bit for the cache tag.	R/W	Unspecified

6.2.30 DDataLo Register (CP0 Register 28, Select 3)

The *DDataLo* and *DDataHi* registers are read-only registers that act as the interface to the cache data array and are intended for diagnostic operations only. The Index Load Tag operation of the CACHE instruction reads the corresponding data values into the *DDataLo* and *DDataHi* registers.

DDataLo Register Format

31 ... 0
DATA

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 6-36 DDataLo Register Field Description

Fields		Description	Read/ Write	Reset State
Name	Bit(s)			
DATA	31:0	Low-order data read from the data cache.	R	Undefined

6.2.31 ITagHi Register (CP0 Register 29, Select 0)

The *ITagLo* and *ITagHi* registers act as the interface to the cache virtual tag array and are intended for diagnostic operation only. The Index Store Tag and Index Load Tag operations of the CACHE instruction use the *ITagLo* and *ITagHi* registers as the source or sink of tag information, respectively.

Software is allowed to write zeros into the *ITagLo* and *ITagHi* registers and then use the Index Store Tag cache operation to initialize the cache tags to a valid state at power-up.

ITagHi Register Format

31 ... 18	17	16	15 ... 8	7 ... 3	2 ... 0
0	BE	G	ASID	SEG	TG

Table 6-37 ITagHi Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	Bits			
Reserved	31:18	Must be written as zero	0	Unspecified
BE	17	Indicates endianness of cache line: 0: little endian 1: big endian	R/W	Unspecified
G	16	Global bit in cache tag.	R/W	Unspecified
ASID	15:8	ASID to store to or read from the cache tag.	R/W	Unspecified
SEG	7:3	Segment bits. Virtual address bits 63:58.	R/W	Unspecified
TG	2:0	Virtual tag. Virtual address bits 39:37.	R/W	Unspecified

6.2.32 IDataHi Register (CP0 Register 29, Select 1)

The *IDataLo* and *IDataHi* registers are read-only registers that act as the interface to the cache data array and are intended for diagnostic operations only. The Index Load Tag operation of the CACHE instruction reads the corresponding data values into the *IDataLo* and *IDataHi* registers.

IDataHi Register Format

31 ... 0
DATA

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 6-38 IDataHi Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	Bit(s)			
DATA	31:0	High-order data read from the data array (as opposed to the tag array) of the Instruction cache.	R	Undefined

The data stored in this register is also recoded using the same instruction recoding algorithm defined for the *IDataLo* register.

6.2.33 DTagHi Register (CP0 Register 29, Select 2)

The *DTagLo* and *DTagHi* registers act as the interface to the cache physical tag array and are intended for diagnostic operation only. The Index Store Tag and Index Load Tag operations of the CACHE instruction use the *DTagLo* and *DTagHi* registers as the source or sink of tag information, respectively. Software writes to this register are ignored, and reads always return zero.

DTagHi Register Format

31 ... 0
0

Table 6-39 DTagHI Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	Bits			
0	31:0	Ignored on write. Returns zero when read.	R	Unspecified

6.2.34 DDataHi Register (CP0 Register 29, Select 3)

The *DDataLo* and *DDataHi* registers are read-only registers that act as the interface to the cache data array and are intended for diagnostic operations only. The Index Load Tag operation of the CACHE instruction reads the corresponding data values into the *DDataLo* and *DDataHi* registers.

DDataHi Register Format

31 ... 0
DATA

Table 6-40 DDataHi Register Field Description

Fields		Description	Read/ Write	Reset State
Name	Bit(s)			
DATA	31:0	High-order data read from the data cache.	R	Undefined

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

6.2.35 ErrorEPC (CP0 Register 30, Select 0)

The *ErrorEPC* register is a read-write register, similar to the *EPC* register, except that *ErrorEPC* is used on error exceptions. All bits of the *ErrorEPC* register are significant and must be writable. It is also used to store the program counter on Reset, Soft Reset, and non-maskable interrupt (NMI) exceptions.

The *ErrorEPC* register contains the virtual address at which instruction processing can resume after servicing an error. This address can be:

- The virtual address of the instruction that caused the exception
- The virtual address of the immediately preceding branch or jump instruction when the error causing instruction is in a branch delay slot

Unlike the *EPC* register, there is no corresponding branch delay slot indication for the *ErrorEPC* register.

ErrorEPC Register Format

63 ... 0
ErrorEPC

Table 6-41 ErrorEPC Register Field Description

Fields		Description	Read/Write	Reset State
Name	Bit(s)			
ErrorEPC	63:0	Error Exception Program Counter	R/W	Undefined

6.2.36 DESAVE Register (CP0 Register 31)

The Debug Exception Save (DESAVE) register is a read/write register that functions as a simple memory location. This register is used by the debug exception handler to save one of the GPRs that is then used to save the rest of the context to a pre-determined memory area (such as in the EJTAG Probe). This register allows the safe debugging of exception handlers and other types of code where the existence of a valid stack for context saving cannot be assumed.

DESAVE Register Format

63 ... 0
DESAVE

Table 6-42 DESAVE Register Description

Bit(s)	Mnemonic	Description	R/W	Reset
63:0	DESAVE	Debug exception save contents.	R/W	Undefined

6.3 CP0 Hazards

Because resources controlled via Coprocessor 0 affect the operation of various pipeline stages of a MIPS64 processor, manipulation of these resources may produce results that are not detectable by subsequent instructions for some number of execution cycles. When no hardware interlock exists between one instruction that causes an effect that is visible to a second instruction, a *CP0 hazard* exists.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

The 20Kc processor implements hardware interlocks to resolve all functional CP0 instruction hazards. Table 6-43 shows hazards which are not handled by the interlocks. The duration of the hazard is indicated by either the number of dispatch cycles (number of SSNOPs) or until the specified instruction. A hazard exists regarding when the *LLAddr* register is updated after a LL instruction. However, this register is for diagnostic purposes only and serves no function during normal operation. A deferred watch exception also causes a hazard because the WP bit is not set in the *Cause* register immediately. Also, changes to $EntryHi_{ASID}$ and *Status* are not guaranteed to affect the instruction fetches until an ERET instruction is executed.

Table 6-43 CP0 Hazard Spacing

Producer	→	Consumer	Hazard On	Duration
LL	→	MFC0	LLAddr	4
deferred watch exception	→	MFC0	$Cause_{WP}$	2
$EntryHi_{ASID}$	→	instruction fetches	$EntryHi_{ASID}$	ERET
Status	→	instruction fetches	Status	ERET
Watch register write	→	instruction taking exception	Watch	ERET
TLBW	→	instruction fetches	TLB	ERET
Compare	→	instruction not seeing timer interrupt	Timer interrupt	4
CACHE instruction	→	instruction fetches	Instruction cache	ERET

6.4 CP1 Register Summary

Table 6-44 lists the CP1 floating-point registers in numerical order. The individual registers are described throughout this section.

Table 6-44 CP1 Registers

Register Number	Register Name	Function
0	FIR	Floating-Point Implementation register. Contains information that identified the FPU.
25	FCCR	Floating-Point Condition Codes register.
26	FEXR	Floating-Point Exceptions register.
28	FENR	Floating-Point Enables register.
31	FCSR	Floating-Point Control and Status register.

6.5 CP1 Registers

The CP1 registers provide the interface between the ISA and the floating-point unit. Each register is discussed below, with the registers presented in numerical order, first by register number, then by select field number.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

For each register described below, field descriptions include the read/write properties of the field, and the reset state of the field. For the read/write properties of the field, the notation listed in Table 6-45 is used:

Table 6-45 CP1 Register Field Types

Read/Write Notation	Hardware Interpretation	Software Interpretation
R/W	A field in which all bits are readable and writable by software and, potentially, by hardware. Hardware updates of this field are visible by software read. Software updates of this field are visible by hardware read. If the reset state of this field is “Undefined,” either software or hardware must initialize the value before the first read will return a predictable value. This should not be confused with the formal definition of UNDEFINED behavior.	
R	A field that is either static or is updated only by hardware. If the Reset State of this field is either “0” or “Preset”, hardware initializes this field to zero or to the appropriate state, respectively, on powerup. If the Reset State of this field is “Undefined”, hardware updates this field only under those conditions specified in the description of the field.	A field to which the value written by software is ignored by hardware. Software may write any value to this field without affecting hardware behavior. Software reads of this field return the last value updated by hardware. If the Reset State of this field is “Undefined,” software reads of this field result in an UNPREDICTABLE value except after a hardware update done under the conditions specified in the description of the field.
0	A field that hardware does not update, and for which hardware can assume a zero value.	A field to which the value written by software must be zero. Software writes of non-zero values to this field may result in UNDEFINED behavior of the hardware. Software reads of this field return zero as long as all previous software writes are zero. If the Reset State of this field is “Undefined,” software must write this field with zero before it is guaranteed to read as zero.

6.5.1 Floating-Point Implementation Register (CP1 Register 0)

The Floating-Point Implementation Register (*FIR*) is a 32-bit read-only register that contains information identifying the capabilities of the floating-point unit, the floating-point processor identification, and the revision level of the floating-point unit. Table 6-46 describes the *FIR* register fields.

Floating-Point Implementation Register Format

31 ... 20	19	18	17	16	15 ... 8	7 ... 0
0	3D	PS	D	S	ProcessorID	Revision

Table 6-46 FIR Register Field Descriptions

Fields		Description	Read/Write	Reset State
Name	Bits			
0	31:20	Reserved for future use; reads as zero	0	0

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 6-46 FIR Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	Bits			
3D	19	Indicates that the MIPS-3D ASE is implemented: 0: MIPS-3D not implemented 1: MIPS-3D implemented This bit is always 1 to indicate that MIPS-3D is implemented.	R	1
PS	18	Indicates that the paired-single (PS) floating-point data type and instructions are implemented: 0: PS floating not implemented 1: PS floating implemented This bit is always 1 to indicate that paired-single floating-point data types are implemented.	R	1
D	17	Indicates that the double-precision (D) floating-point data type and instructions are implemented: 0: D floating not implemented 1: D floating implemented This bit is always 1 to indicate that double-precision floating-point data types are implemented.	R	1
S	16	Indicates that the single-precision (S) floating-point data type and instructions are implemented: 0: S floating not implemented 1: S floating implemented This bit is always 1 to indicate that single-precision floating-point data types are implemented.	R	1
Processor ID	15:8	Identifies the floating-point processor. This value should normally match the corresponding field of the *PRId* CP0.	R	0x82
Revision	7:0	Revision number of the floating-point unit. Allows software to distinguish between one revision and another of the same floating-point processor type.	R	Preset

6.5.2 Floating-Point Condition Codes Register (CP1 Register 25)

The Floating-Point Condition Codes Register (*FCCR*) is an alternative way to read and write the floating-point condition code values that also appear in *FCSR*. Unlike *FCSR*, all eight FCC bits are contiguous in *FCCR*. Table 6-47 describes the *FCCR* register fields.

Floating-Point Condition Codes Register Format

31 ... 8	7 ... 0
0	FCC

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 6-47 FCCR Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	Bits			
0	31:8	Must be written as zero; returns zero on read	0	0
FCC	7:0	Floating-point condition code. Refer to the description of this field in the *FCSR* register.	R/W	Undefined

6.5.3 Floating-Point Exceptions Register (CP1 Register 26)

The Floating-Point Exceptions Register (*FEXR*) is an alternative way to read and write the Cause and Flags fields that also appear in *FCSR*. Table 6-48 describes the *FEXR* register fields.

Floating-Point Exceptions Register Format

31 18	17 12	11 7	6 2	1 0
0	Cause	0	Flags	0
	E V Z O U I		V Z O U I	

Table 6-48 FEXR Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	Bits			
0	31:18, 11:7, 1:0	Must be written as zero; returns zero on read	0	0
Cause	17:12	Cause bits. Refer to the description of this field in the *FCSR* register.	R/W	Undefined
Flags	6:2	Flags bits. Refer to the description of this field in the *FCSR* register.	R/W	Undefined

6.5.4 Floating-Point Enables Register (CP1 Register 28)

The Floating-Point Enables Register (*FENR*) is an alternative way to read and write the Enables, FS, and RM fields that also appear in *FCSR*. Table 6-49 describes the *FENR* register fields.

Floating-Point Exceptions Register Format

31 12	11 7	6 3	2	1 0
0	Enables	0	FS	RM
	V Z O U I			

Table 6-49 FENR Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	Bits			
0	31:12, 6:3	Must be written as zero; returns zero on read	0	0

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 6-49 FENR Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	Bits			
Enables	11:7	Enable bits. Refer to the description of this field in the *FCSR* register.	R/W	Undefined
FS	2	Flush to Zero bit. Refer to the description of this field in the *FCSR* register.	R/W	Undefined
RM	1:0	Rounding mode. Refer to the description of this field in the *FCSR* register.	R/W	Undefined

6.5.5 Floating-Point Control and Status Register (CP1 Register 31)

The Floating-Point Control and Status Register (*FCSR*) is a 32-bit register that controls the operation of the floating-point unit. Access to *FCSR* is not privileged; it can be read or written by any program that has access to the floating-point unit (via the coprocessor enables in the *Status* register). Table 6-50 describes the *FCSR* register fields.

Floating-Point Control and Status Register Format

31 25	24	23	22	21	20 18	17 12	11 7	6 2	1 0
FCC	FS	FCC	FO	FN	0	Cause	Enables	Flags	RM
7 6 5 4 3 2 1		0				E V Z O U I	V Z O U I	V Z O U I	

Table 6-50 FCSR Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	Bit			
FCC	31:25, 23	Floating-point condition codes. These bits record the result of floating-point compares and are tested for floating-point conditional branches and conditional moves. The FCC bit to use is specified in the compare, branch, or conditional move instruction. For backward compatibility with previous MIPS ISAs, the FCC bits are separated into two non-contiguous fields.	R/W	Undefined
FS	24	Flush to Zero. When FS is one, denormalized results are flushed to zero instead of causing an Unimplemented Operation exception. It is implementation dependent whether denormalized operand values are flushed to zero before the operation is carried out.	R/W	Undefined
FO	22	MADD Flush Override. This mode bit is only defined for the 20Kc FPU. When FO is set, a denorm intermediate result of any MADD instruction is not flushed nor denormalized according to the FS bit. The intermediate result maintains an internal normalized format to improve accuracy.	R/W	Undefined
FN	21	Flush to nearest. When FN is set, denormalized results are flushed to either zero or min-norm depending on whether the denorm result is closer to zero of closer to min-norm, instead of causing an Unimplemented Operation exception.	R/W	Undefined
0	20:18	Reserved for future use. Must be written as zero; returns zero on read.	0	0

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 6-50 FCSR Register Field Descriptions (Continued)

Fields		Description	Read/ Write	Reset State
Name	Bit			
Cause	17:12	Cause bits. These bits indicate the exception conditions that arise during execution of an FPU arithmetic instruction. A bit is set to 1 if the corresponding exception condition arises during the execution of an instruction and is set to 0 otherwise. By reading the registers, the exception condition caused by the preceding FPU arithmetic instruction can be determined. Refer to Section 6.2.12, "Cause Register (CP0 Register 13, Select 0)" for descriptions of the Cause bits.	R/W	Undefined
Enables	11:7	Enable bits. These bits control whether or not a trap is taken when an IEEE exception condition occurs for any of the five conditions. The trap occurs when both an Enable bit and the corresponding Cause bit are set either during an FPU arithmetic operation or by moving a value to FCSR or one of its alternative representations. Note that Cause bit E has no corresponding Enable bit; the non-IEEE Unimplemented Operation exception is defined by MIPS as always enabled.	R/W	Undefined
Flags	6:2	Flag bits. This field shows any exception conditions that have occurred for completed instructions since the flag was last reset by software. When a FPU arithmetic operation raises an IEEE exception condition that does not result in a Floating-Point Exception (i.e., the Enable bit was off), the corresponding bit(s) in the Flag field are set, while the others remain unchanged. Arithmetic operations that result in a Floating-Point Exception (i.e., the Enable bit was on) do not update the Flag bits. This field is never reset by hardware and must be explicitly reset by software.	R/W	Undefined
RM	1:0	Rounding mode. This field indicates the rounding mode used for most floating-point operations (some operations use a specific rounding mode).	R/W	Undefined

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Chapter 7

Caches

The 20Kc processor contains a 32-Kbyte Instruction Cache and a 32-KByte Data Cache. Each cache can each be accessed in a single processor cycle. The two caches are distinct, and each cache can be accessed in the same cycle.

This chapter contains the following sections.

- Section 7.1, "Instruction Cache"
- Section 7.2, "Data Cache"
- Section 7.3, "Cache Protocol"
- Section 7.4, "Cache Attributes"

Table 7-1 lists the instruction and data cache attributes.

Table 7-1 Instruction and Data Cache Attributes

Parameter	Instruction	Data
Size	32 KBytes	32 KBytes
Number of Cache Sets	256	256
Lines Per Set (Associativity)	4-way set associative	4-way set associative
Line Size	32 bytes	32 bytes
Read Unit	16 bytes	8 bytes
Write Unit	32 bytes	32 bytes
Write Policy	N/A	write-back
Access Mode	Virtually indexed/Virtually tagged	Physically indexed/Physically tagged
Cache Locking	per line	per line

7.1 Instruction Cache

The instruction cache is an on-chip memory block of 32 KBytes. The Instruction cache is virtually indexed and virtually tagged. This allow the cache access to proceed without waiting for the virtual-to-physical address translation. The tag contains 32 bits of virtual address, 1 valid bit, 1 lock bit, 1 LRF replacement bit, 1 Global bit, 8 ASID bits, 1 Big Endian bit, and 1 parity bit.

The 20Kc processor supports instruction cache-locking. Cache locking allows critical code or data segments to be locked into the cache on a “per-line” basis, enabling the system programmer to maximize the efficiency of the system cache.

The cache locking function is always enabled on all instruction cache entries. Entries can then be marked as locked or unlocked on a per entry basis using the CACHE instruction.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

7.1.1 Memory Management implications of the Virtual I-Cache

There are certain implications of a virtual I-Cache that anyone developing software for the 20Kc should be aware of.

7.1.1.1 Virtual I-Cache synchronization with Joint Translation Buffer (JTB)

Since the I-Cache contains translation information similar to that contained in the Joint Translation Buffer (JTB), any programmer manipulating the contents of the JTB must take the necessary steps to keep the I-Cache synchronized with the JTB. This is not done automatically by the hardware. In order to keep the virtual I-Cache synchronized with the JTB, software is required to meet the following two constraints:

- If a virtual-to-physical address mapping is changed such that the same virtual address now points to a different physical address, and the ASID is unchanged after this operation, then the 20Kc processor requires that the I-Cache be flushed to eliminate any old mappings.
- If the ASIDs are incremented such that they rollover and get reused, then the 20Kc processor requires that the I-Cache be flushed so that stale Virtual Address mappings corresponding to a former life of an ASID in the I-Cache are eliminated.

7.1.1.2 Virtual I-Cache hits

The JTB lookup is bypassed on I-Cache hits. As a result it is possible that a cache line exists in the I-Cache but the corresponding translation in the JTB is replaced. Software should therefore never make the assumption that an instruction that hits in the I-Cache has a corresponding translation currently present in the JTB.

One such example of an incorrect assumption would be the following sequence of events:

- The 20Kc dispatches an instruction. The instruction was present in the ICache but its corresponding translation was replaced in the JTB prior to the fetching of this instruction from the ICache.
- This instruction when dispatched by the 20Kc processor takes an exception.
- The exception handler attempts to examine the opcode of the instruction that took the exception via a Load instruction (e.g.LW). Since the Data Cache (DCache) is physically indexed and physically tagged, any lookup of the DCache is preceded by a lookup of the JTB. Since the corresponding entry has been removed from the JTB, this will result in a TLB Refill exception. The exception handler was unprepared for this exception, because it assumed that the instruction would not take a TLB Refill instruction.

7.2 Data Cache

The data cache is an on-chip memory block of up to 32 KBytes. The data cache is physically indexed and physically tagged. The tag contains 23 bits of physical address, 1 valid bit, 1 lock bit, 1 LRF replacement bit, 1 dirty or modified bit, and 1 parity bit.

In addition to instruction cache locking, the 20Kc processor also supports a data cache locking mechanism identical to the instruction cache. Critical data segments to be locked into the cache on a “per-line” basis. The locked contents can be updated on a store hit, but cannot be selected for replacement on a store miss.

The cache locking function is always enabled on all data cache entries. Entries can then be marked as locked or unlocked on a per entry basis using the CACHE instruction.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

7.3 Cache Protocol

The 20Kc processor implements the three state MEI (Modified-Exclusive-Invalid). A cache line in the data cache can be in one of the three states: Modified, Exclusive or Invalid. A cache line in the Instruction cache can be in one of the two states i.e: Exclusive or Invalid.

Figure 7-1 shows the instruction cache state diagram. Figure 7-2 shows the data cache state diagram.

Figure 7-1 Instruction Cache State Diagram

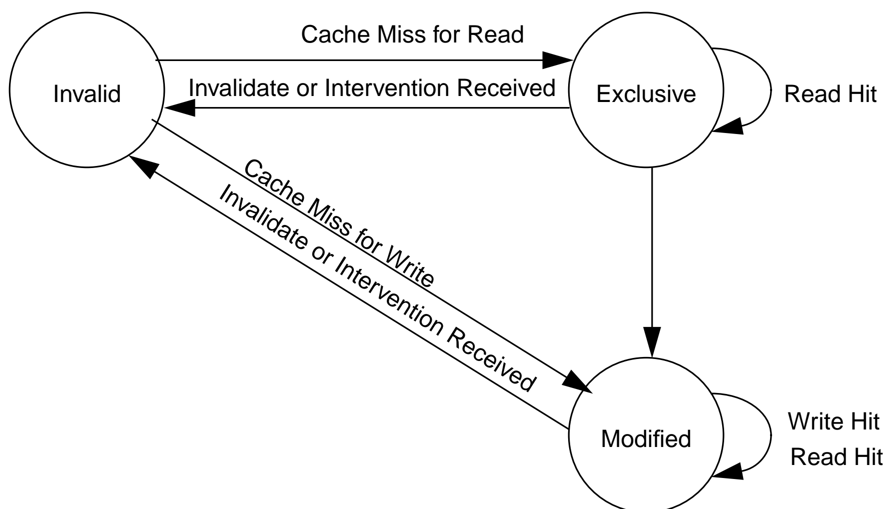


Figure 7-2 Data Cache State Diagram

7.4 Cache Attributes

The 20Kc processor supports the following different cache attributes.

- Uncached
- Uncached Accelerated
- Non-Coherent Write Back
- Coherent Write Back
- Non-Coherent, write through, no write-allocate

7.4.1 Uncached

Addresses in a memory area indicated as uncached are never read from the cache. When reads to this area result in fetches from memory, the read data is not refilled into the cache.

Stores to such addresses are written directly to main memory, without changing the contents of the data cache.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

This protocol is applicable to both instruction and data references.

7.4.2 Uncached Accelerated

Addresses in a memory area indicated as uncached accelerated are not read from the data cache. When reads to this area result in fetches from memory, the read data is not refilled into the cache.

Stores to such addresses are gathered in the uncached accelerated write buffer, providing they meet certain requirements described later. They are then sent out on the bus as a gathered write. For a description of the write buffer, refer to Chapter 8, “Bus Interface Unit.” This is intended as a mechanism to improve the write bandwidth on uncached data.

This protocol is applicable only for data references. Uncached Accelerated Gathering is not supported in Reverse Endian mode. Reverse Endian mode is enabled for user mode accesses when Status.RE is set to 1.

7.4.2.1 Gathering Mechanism for Uncached Accelerated Stores

Uncached accelerated gathering is supported for word and doubleword stores only. Gathering of uncached accelerated stores starts on cache-line aligned addresses (32- byte aligned addresses).

Uncached accelerated word or doubleword stores that do not meet the conditions required to start gathering are treated like regular uncached stores.

Once an Uncached Accelerated Store meets the requirements needed to start gathering, a 32-byte gather buffer is reserved for this store. All subsequent uncached accelerated word or doubleword stores write sequentially into this buffer, independent of the address associated with these stores. The uncached accelerated buffer is tagged with the address of the first store. Even though the address of subsequent stores is not checked, a software programmer is required to use appropriate addresses for all subsequent stores. An appropriate address would be either "identical" or "sequential". Indentical addresses imply all the stores use the same address. This may be the appropriate mode for certain devices. Sequential addresses imply that the address is incremented after every store, based on the granularity of the write. This would be the mode of addressing required for memory.

An uncached accelerated buffer can be explicitly flushed under program control via the following mechanisms:

- An uncached accelerated byte store: This store is treated as a NOP and causes the gather buffer to be flushed.
- A Sync instruction: The SYNC instruction causes the gather buffer to be flushed.

An uncached accelerated buffer is implicitly flushed under the following conditions.

- The last word in the entry being gathered is written.
- TLB hit on virtual address of a load instruction matches original virtual address associated with the gather buffer.
- An exception occurs.

When an uncached accelerated buffer is flushed, the address sent out on the system interface is the address associated with the first uncached accelerated store. It is important to use "identical" or "sequential" addressing since gathering can be interrupted by asynchronous exceptions such as interrupts. As described in the previous paragraph, this will cause flushing of the buffer. When the store instruction that was interrupted resumes execution after the asynchronous exception is handled, it will be treated as an uncached store and needs to have the correct address associated with it in order to ensure correct operation. The 20Kc does not check to see if the program has violated the "sequential" or "identical" addressing mode. That responsibility lies with the programmer.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Caveats:

- Any uncached stores and any uncached loads to unrelated addresses that occur between uncached accelerated stores that are part of a gather sequence go out of order. The uncached accelerated stores are not ordered with respect to these load or stores.
- The only constraint imposed on the gathering is that doubleword stores are only allowed to write to doubleword aligned locations in the buffer. For example, if uncached accelerated gathering starts with a Store Word (SW), it may not be followed by a Store Double (SD).
- Uncached accelerated stores that are halfword stores (SH), or unaligned stores (SWL, SWR, SDL, SDR) and store conditional (SC and SCD) are not intended to be used. In the present implementation of the 20Kc processor, they are treated as NOPS. However, this behavior is not guaranteed for subsequent implementations. Programmers must not use these stores and expect a specific behavior.

7.4.3 Non-Coherent Write-Back

Loads and instruction fetches first search the cache, reading main memory only if the desired data does not reside in the cache. Data returned from main memory is written into the caches. If the line that is being replaced in the cache contains data that is more recent than main memory (i.e. in the modified state), that line is written out to memory, before the refill occurs.

In the case of data store operations, the cache is first searched to see if the target address is cache resident. If it is resident, the cache contents are updated. If the line is not present in the cache, main memory is accessed. The data returned from main memory is then written into the cache. Again if the line being replaced is in the modified state, it is written back to memory before the line is refilled. The store is completed once the refill from memory is completed.

When a cache miss triggers a memory access to this space, the corresponding request on the System Interface contains a qualifier indicating that the address is intended to be non-coherent. This qualifier can be used in a multimaster environment to determine the appropriate course of action.

This protocol is applicable only for both instruction and data references. However, a line in the instruction cache can never be in the modified state, hence there is never any writeback.

7.4.4 Coherent Exclusive Write-Back

The sequence of events for this cache attribute are identical to those in the preceding section on Non-Coherent Writeback, with the following exception.

When a cache miss triggers a memory access to this space, the corresponding request on the System Interface contains a qualifier indicating that the address is intended to be Coherent Exclusive. This qualifier can be used in a multi-master environment to determine the appropriate course of action.

This protocol is applicable only for both instruction and data references. However, a line in the instruction cache can never be in the modified state, hence there is never any writeback.

7.4.5 Non-Coherent, Write-Through with No Write-Allocate

Loads and instruction fetches first search the cache, reading main memory only if the desired data does not reside in the cache. Data returned from main memory is written into the caches. If the line that is being replaced in the cache contains data that is more recent than main memory (that is, in the modified state), that line is written out to memory, before the refill occurs.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

In the case of data store operations, the cache is first searched to see if the target address is cache resident. If it is resident, the cache contents are updated. If the line is not present in the cache, the cache is bypassed and the data is written directly to main memory.

This protocol is applicable only for data references.

7.4.6 Encoding

The above cache protocols can be selected by one of the two following mechanisms:

- For mapped addresses they are controlled on a per-page basis by appropriately setting the “C” field of the *EntryLo0* and *EntryLo1* registers before a TLBWI or TLBWR operation.
- For unmapped addresses in the xkphys segment, they are determined by the CCA field of the virtual address. The CCA field corresponds to bits [61:59] of the virtual address.
- For unmapped address in the kseg0 segment, they are determined by the “K0” field of the *Config* Register. The K0 field corresponds to bits [2:0] of the *Config* Register.

The encoding used for the three bit “Cache Coherency Attribute” is identical in all of the above three cases and is given in Table 6-6 on page 88.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Bus Interface Unit

The MIPS 20Kc System interface is a high-performance interface that provides a high-speed connection between the MIPS 20Kc processor and a single SOC controller. This interface is also referred to as the MIPS GigaBytes per second (MGB™) bus protocol. While the new bus protocol is fundamentally different from the SysAD style interfaces used in MIPS processors, such as the R4000 and R5000, some features of SysAD have been retained. Throughout this document, the device communicating with the 20Kc processor through the System interface is referred to as the SOC controller.

The 20Kc System interface contains a 32-bit unidirectional multiplexed address and data bus driven by the processor (*EB_PrcAD[31:0]*), and a separate 64-bit unidirectional multiplexed address and data bus driven by the SOC controller (*EB_SysAD[63:0]*). The *EB_SysAD* bus can also be configured in 32-bit mode to minimize pin count.

The 20Kc System interface allows the processor to access external resources needed to satisfy cache misses and uncached operations, while permitting an SOC controller to access some of the processor internal resources.

This chapter contains the following sections:

- Section 8.1, "20Kc System Interface Features"
- Section 8.2, "Bus Encoding (64-bit EB_SysAD Mode)"
- Section 8.3, "Processor and External Request Protocols (64-bit EB_SysAD Mode)"
- Section 8.4, "Bus Encoding (32-bit EB_SysAD Mode)"
- Section 8.5, "Processor and External Request Protocols (32-bit EB_SysAD Mode)"
- Section 8.6, "20Kc Signal Descriptions"

8.1 20Kc System Interface Features

The following is a brief summary of features:

- Support for both processor and external requests and responses.
- Multiplexed unidirectional 32-bit address and data bus driven by the processor with associated 7-bit command bus.
- Multiplexed unidirectional 64-bit address/data bus driven by the SOC controller with associated 14-bit command bus. This bus also contains a 32-bit option for pin-constrained applications.
- Support for multiple outstanding split transactions and out-of-order data return.
- Credit-based flow control.

8.1.1 Processor and External Requests

There are two broad categories of requests: *processor requests* and *external requests*. When a system event occurs, the processor issues either a single request or a series of requests—called processor requests—through the System interface, to access an external resource and service the event. For this to work, the 20Kc processor must be connected to an SOC controller that is compatible with the 20Kc System interface protocol and can coordinate accesses to system resources.

The SOC controller only performs intervention or invalidate external requests. The SOC controller can access and manipulate data residing in the processor cache through these types of requests.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

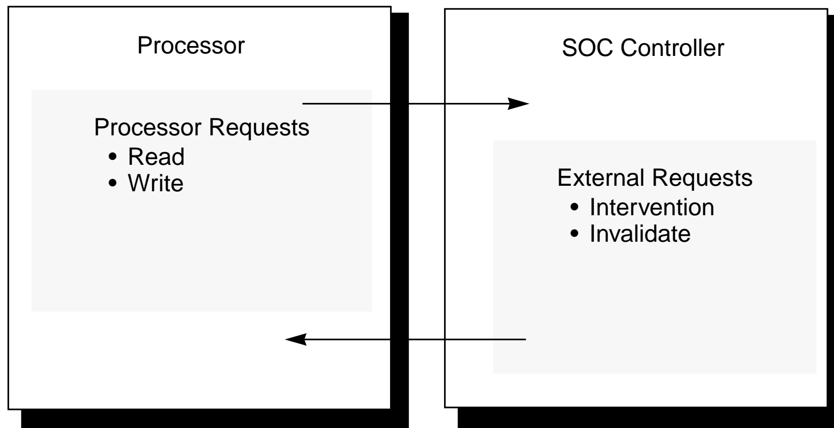


Figure 8-1 Processor and SOC Controller Requests

There are two types of responses on the System interface: data responses and state responses. A data response can be returned by either the SOC controller (for processor read requests) or the processor (for interventions that hit dirty lines in the data cache). A state response is only returned by the processor for Intervention and Invalidate requests from the SOC controller. State responses return the state of the targeted cache line. Figure 8-2 illustrates all the possible request/response combinations.

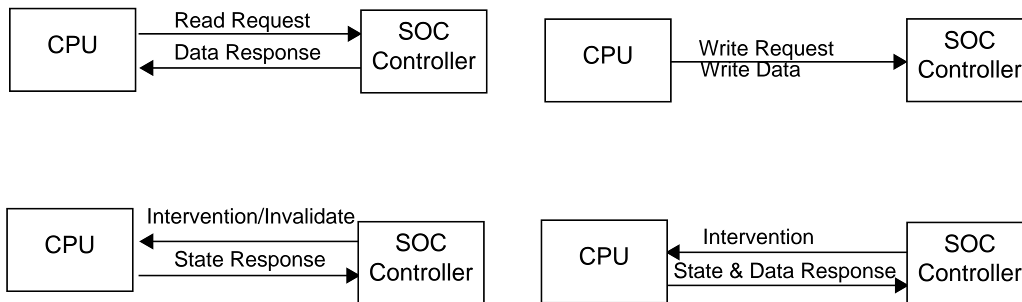


Figure 8-2 Request/Response Combinations

8.1.2 Multiplexed Unidirectional 32-bit Processor Address/Data Bus

The 20Kc System interface contains a 32-bit unidirectional multiplexed address/data bus, *EB_PrcAD[31:0]*, that is always driven by the processor. The minimum transfer size is 8 bits. The bus supports transfer sizes of 64 bits, requiring two cycles per data transfer on *EB_PrcAD[31:0]*. The *EB_PrcAD[31:0]* address/data bus is parity protected by the *EB_PrcADP[3:0]* bus which provides even parity. Each parity bit on *EB_PrcADP[3:0]* corresponds to one byte on *EB_PrcAD[31:0]*. In addition, a seven-bit command bus, *EB_PrcCmd[6:0]*, is driven at the same time as *EB_PrcAD[31:0]* and provides information on the type of operation.

8.1.3 Multiplexed Unidirectional 64-bit SOC Controller Address/Data Bus

The 20Kc System interface contains a 64-bit unidirectional multiplexed address/data bus, *EB_SysAD[63:0]*, that is always driven by the SOC controller. The minimum transfer size is 8 bits. The bus supports transfer sizes of 64 bits. The *EB_SysAD[63:0]* address/data bus is parity protected by the *EB_SysADP[7:0]* bus which provides even parity. Each parity bit on *EB_SysADP[7:0]* corresponds to one byte on *EB_SysAD[63:0]*. In addition, a 14-bit command bus, *EB_SysCmd[13:0]*, is driven at the same time as *EB_SysAD[63:0]* and provides information on the type of operation. The *EB_SysAD* bus can also be configured in 32-bit mode for pin-constrained applications.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

8.1.4 Support for Multiple Outstanding Split Transactions

The 20Kc System interface supports split transactions, which do not require the bus requestor to wait for one request to be completed before issuing a new one. In addition, the bus requestor also can receive and process new requests while waiting for the completion of a prior request. The 20Kc processor can issue up to six outstanding read operations.

Because multiple requests can be pending on the System interface, a means to identify the data replies back to their corresponding requests is needed to support out-of-order return. To satisfy this requirement, each request is tagged by the processor when it is issued. When the data response is ready, the SOC controller sends the same tag back to the processor with the data. Requests from the SOC controller to the 20Kc processor and the corresponding replies are tagged in a similar way.

The transaction tag format is given in Figure 8-3. It is made up of two components: a 4-bit transaction ID field and a 1-bit device ID field. The SOC controller is assigned Device ID 1, while the 20Kc core is assigned Device ID 0. With this configuration, transaction tags 0-15 are reserved for processor requests, while transaction tags 16-31 are reserved for SOC controller requests. The transaction tag is part of the command bus as described later.

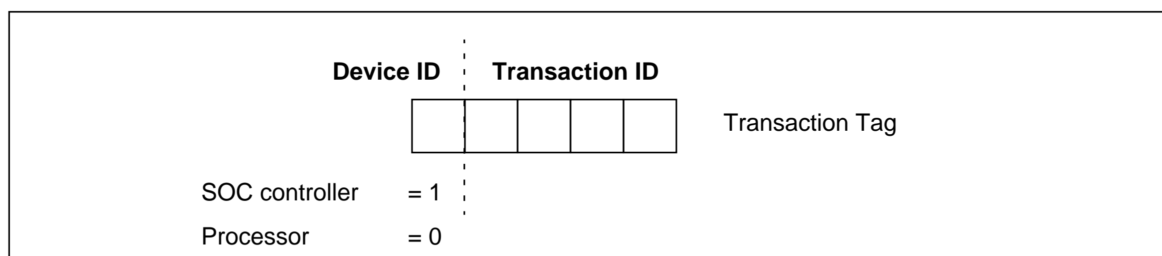


Figure 8-3 Transaction Tags

8.1.5 Credit-Based Flow Control

Flow control on the 20Kc System interface is based on credits and occurs on both processor initiated and SOC controller initiated transactions. On the processor side there are two counters: a read resource counter (RdRscCount) and a write resource counter (WrRscCount). Each of these counters reflect the available read or write resources in the SOC controller. When the processor issues a read operation to the SOC controller, the RdRscCount counter decrements by one internally. The SOC controller responds to the transaction by asserting the *EB_SysRdCredit* signal for one clock at some point before, during, or after the transaction. The assertion of this signal causes the RdRscCount counter to increment.

In a credit-based flow control scheme, the processor knows at any given time the amount and type of free resources in the SOC controller. Therefore the processor does not issue more requests than the SOC controller can handle. In return, the SOC controller must accept processor requests for which there exists corresponding free resources, because there is no means of retrying a request once it has been issued by the processor.

All read and write credit counters default to zero on power-up because the processor does not know the level of flow control resources that the SOC controller implements. The 20Kc processor supports up to 16 read credits and 16 write credits. During the initialization phase (after reset) the SOC controller asserts the *EB_SysRdCredit* and *EB_SysWrCredit* signals between 1 and 16 clocks each, depending on the number of resources implemented. Figure 8-4 shows an example of how the RdRscCount and WrRscCount counters are programmed during initialization by asserting the *EB_SysRdCredit* and *EB_SysWrCredit* signals, respectively.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

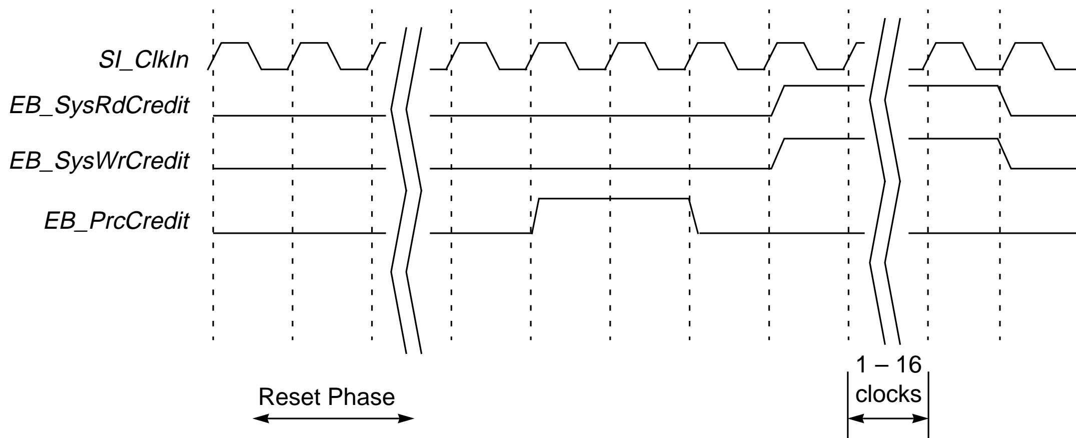


Figure 8-4 Credit Counter Programming

8.1.5.1 SOC Controller Resource Control

As shown in Figure 8-6, the processor maintains two separate counters: a read resource counter (RdRscCount) and a write resource counter (WrRscCount). Each of these counters reflect the available read or write resources in the SOC controller. The RdRscCount tells the processor the number of Read requests which the SOC controller can handle, while the WrRscCount indicates the number of write requests (including their associated data) the processor can issue to the SOC controller. In this example, the SOC controller supports four outstanding read operations and two outstanding write operations.

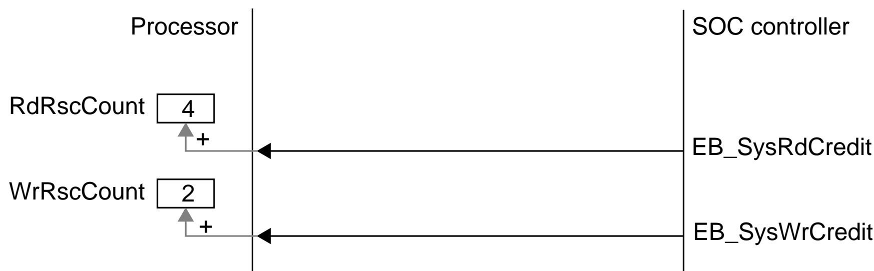


Figure 8-5 SOC Controller Resource Control

Whenever the processor issues a Read or a Write request, it also decrements the corresponding counter. Once a counter reaches zero, the processor stops issuing any more requests. During a read operation, once the SOC controller completes a request and frees up a resource, it asserts the *EB_SysRdCredit* signal for one clock. This causes the RdRscCount to increment by one. Each additional cycle the *EB_SysRdCredit* is asserted represents a new credit, so in effect the *EB_SysRdCredit* signal acts as an increment-enable for the RdRscCount counter. Note that once a credit is issued by the SOC controller, it cannot be revoked later on.

In Figure 8-5, the SOC controller has asserted the *EB_SysRdCredit* signal four times during initialization, indicating that it supports up to four outstanding read operations. When the processor initiates the initial read operation on the bus, the counter decrements to 3, indicating that three more read operations can occur prior to the time data for the current read cycle is returned. At some point before, during, or after completion of the first read cycle the SOC controller asserts *EB_SysRdCredit*. The assertion of this signal causes the RdRscCount counter to increment from 3 back to 4.

Similarly, when the processor issues a write operation to the SOC controller, the WrRscCount counter decrements by one internally. In Figure 8-5, the SOC controller has asserted the *EB_SysWrCredit* signal two times during initialization, indicating that it supports up to two outstanding write operations. The SOC controller responds to the transaction by asserting the *EB_SysWrCredit* signal at some point before, during, or after completion of the transaction. The assertion of this signal causes the WrRscCount counter to increment by one. For example, when the processor initiates a write

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

operation on the bus, the counter decrements to one, indicating that one more write operation can be initiated by the processor prior to the SOC controller finishing the current write cycle. At some point during or after the write data transfer is complete, the SOC controller asserts *EB_SysWrCredit*. The assertion of this signal causes the WrRscCount counter to increment from 1 back to 2.

8.1.5.2 Processor Resource Control

Flow control from the processor to the SOC controller works in the same way. This is required since both the processor and the SOC controller can issue requests to each other. However, as seen in Figure 8-6, there is only one type of processor resource, one counter in the SOC controller to keep track of it, and one credit signal from the processor to the SOC controller (*EB_PrcCredit*) acting as an increment-enable to the PrcRscCount counter. The processor asserts the *EB_PrcCredit* signal two times during initialization to indicate to the SOC controller that it supports two outstanding operations.

Figure 8-6 Processor Resource Control

8.1.5.3 Flow Control Implications on SOC controller design.

A designer of an SOC controller for the 20Kc processor needs to be cognizant of the following 20Kc specific aspects of the flow control scheme. These are aspects particular to the 20Kc processor and not to the more general protocol.

- The Bus Interface Unit (BIU) within the 20Kc processor maintains a single shared queue for processor requests which is maintained in FIFO order. If a read request is stalled at the head of the queue for lack of read credits, a write request that is behind the read request will be unable to get out on the bus even if write credits are available. The same comment applies to a write request stalled at the head of the queue for lack of write credits. It will prevent any subsequent read requests in the queue from being dispatched.
- Intervention Responses from the processor require a resource within the 20Kc processor that is shared with write requests. This resource is the internal writeback buffer. It is possible that the writeback buffer is full because the processor does not have the write credits required to send write requests out on the bus. An Intervention request received at such a point can only make progress once a write credit is received, to free up the writeback buffer resource. Since write requests can be stalled in the BIU behind read requests waiting for read credits, it is important that the SOC controller provide the 20Kc processor at least 1 read credit and 1 write credit while expecting the 20Kc processor to respond to an Intervention Request.

8.2 Bus Encoding (64-bit EB_SysAD Mode)

This section contains the *EB_PrcCmd[6:0]* and *EB_SysCmd[13:0]* bus encoding in 64-bit *EB_SysAD* mode.

8.2.1 PrcCmd/SysCmd Bus Encoding (Command Cycles)

In 64-bit EB_SysAD mode, the 20Kc System interface contains a 7-bit *EB_PrcCmd* bus and a 14-bit *EB_SysCmd* bus that contain request and identification information for the transaction. *EB_PrcCmd[6:0]* is driven by the processor. *EB_SysCmd[13:0]* is driven by the SOC controller. The encoding for each bus is identical.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Figure 8-7 shows the command bus format for the processor and SOC controller. In 64-bit EB_SysAD bus mode, where the SOC controller is 64 bits, the command information is transferred as a single 14-bit value. On the processor side, where the bus width is 32 bits, the command is transferred in two cycles. The lower seven bits are transferred first on *EB_PrcCmd[6:0]*, followed by the upper seven bit in the following clock.

There are two types of command transfers. These transfers determine the format of the command bus. During the address phase of a transaction, the 'command' format is transferred as shown in Figure 8-7.

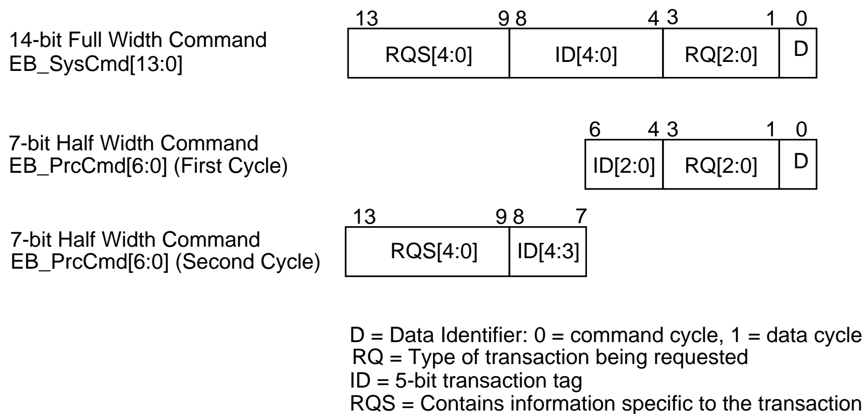


Figure 8-7 Command Bus Formats During a Command Cycle

8.2.1.1 Processor and SOC Controller Request Encoding

Bits [3:1] of the *EB_PrcCmd* bus and bits [3:1] of the *EB_SysCmd* bus comprise the RQ[2:0] field, which encodes the type of processor and SOC controller requests as shown in Table 8-1.

Table 8-1 Request Encoding

RQ[2:0]	Request Type
0	Read request (Processor Only)
1	Reserved
2	Write Request (Processor Only)
3	Reserved
4	Invalidate Request (SOC Only)
5	Reserved
6	Intervention Request (SOC Only)
7	Reserved

8.2.1.2 Processor Read Request Encoding

A processor read request occurs when the *EB_PrcCmd[3:1]* field contains a value of 0b000. Bits [13:9] of the command bus are loaded into the RQS[4:0] field.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Processor Read Request Type Encoding

The RQS[4:3] bits encode the type of processor read request as shown in Table 8-2. This information is useful in handling the associated external response.

Table 8-2 Encoding of RQS[4:3] for Processor Read Requests

RQS[4:3]	Request Type
0	Reserved
1	Coherent Block Read - Exclusive
2	Noncoherent Block Read
3	Double/Single/Partial Word Read

Processor Block Read Transfer Size Encoding

Block read requests are used to transfer 32 bytes of data in a single transaction. The lower two bits of the RQS field, RQS[1:0], indicate the size of the block read transfer as shown in Table 8-3. Note that during a block read request, the RQS[2] bit is reserved and is not used in the encoding process.

Table 8-3 Encoding of RQS[1:0] for Processor Block Read Requests

RQS[1:0]	Block Transfer Size
0	Reserved
1	8 words
2	Reserved
3	Reserved

Processor Double/Single/Partial Word Read Encoding

The 20Kc System interface allows for the transfer of doubleword, word, and partial word data types. A partial word read requests the transfer of less than 32 bits of data. A single-word read requests 32 bits of data. A doubleword read requests 64 bits of data. During these types of requests, the lower three bits of the RQS field, RQS[2:0], encode the number of bytes valid for the transaction as shown in Table 8-4. Refer to Table 8-14 for all valid combinations of address and access size.

Table 8-4 Encoding of RQS[2:0] for Processor D/S/P Word Read Requests

RQS[2:0]	Number of Bytes Valid
0	1 byte valid
1	2 bytes valid
2	3 bytes valid
3	4 bytes valid
4	5 bytes valid
5	6 bytes valid
6	7 bytes valid
7	8 bytes valid

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

8.2.1.3 Processor Write Request Encoding

The upper five bits of the command bus, RQS[4:0], contain information specific to the transaction indicated by the RQ[2:0] field.

Processor Write Request Type Encoding

During the address cycle of processor write requests the upper two bits of the command bus, RQS[4:3], contains the type of write operation as shown in Table 8-5. This information is useful in handling the associated write data.

Table 8-5 Encoding of RQS[4:3] for Processor Write Requests

RQS[4:3]	Write Cause Indication
0	Uncached Accelerated Block Write
1	Reserved
2	Block Write
3	Double/Single/Partial Word Write

Processor Block Write Transfer Size Encoding

Block write requests are used to transfer 32 bytes of data in a single transaction. The lower two bits of the RQS field, RQS[1:0], indicate the size of the block write transfer as shown in Table 8-6. Note that during a block write request the RQS[2] bit is reserved and is not used in the encoding process.

Table 8-6 Encoding of RQS[1:0] for Processor Block Write Requests

RQS[1:0]	Block Transfer Size
0	Reserved
1	8 words
2	Reserved
3	Reserved

Processor Double/Single/Partial Word Write Request Encoding

The 20Kc System interface allows for the transfer of doubleword, word, and partial word data types. A partial word write requests the transfer of less than 32 bits of data. A single-word write requests 32 bits of data. A doubleword write requests 64 bits of data. During these types of requests, the lower three bits of the RQS field, RQS[2:0], encode the number of bytes valid for the transaction as shown in Table 8-7. Refer to Table 8-14 for all valid combinations of address and access size.

Table 8-7 Encoding of RQS[2:0] for Processor D/S/P Word Write Requests

RQS[2:0]	Number of Bytes Valid
0	1 byte valid
1	2 bytes valid
2	3 bytes valid
3	4 bytes valid
4	5 bytes valid

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 8-7 Encoding of RQS[2:0] for Processor D/S/P Word Write Requests

RQS[2:0]	Number of Bytes Valid
5	6 bytes valid
6	7 bytes valid
7	8 bytes valid

Processor Uncached Accelerated Block Write Request Encoding

During the address cycle of processor uncached accelerated block write requests, RQS[2:0] contain the number of valid words in the block transfer as shown in Table 8-8. This value is needed because the transfer size on the system bus is always eight words regardless of the actual number of valid words contained within the block. Note that uncached accelerated writes are always block-aligned, and the first valid word is the one located at the block-aligned address.

Table 8-8 Encoding of RQS[2:0] for a Processor Uncached Accelerated Block Write Request

RQS[2:0]	Number of Valid Words
0	1 word valid
1	2 words valid
2	3 words valid
3	4 words valid
4	5 words valid
5	6 words valid
6	7 words valid
7	8 words valid

8.2.1.4 External Invalidation Request Encoding

An external invalidation request is selected when the RQ[2:0] field contains a value of 0b100. During this type of request, the RQS[1:0] field is reserved.

8.2.1.5 External Intervention Request Encoding

An external intervention request is selected when the RQ[2:0] field contains a value of 0b110. The RQS[1:0] bits determine the type of intervention request as shown in Table 8-9. Note that during an intervention request, RQS[4:2] are reserved and are not used in the encoding process.

Table 8-9 Encoding of RQS[1:0] for External Intervention Requests

RQS[1:0]	Type of Intervention
0	Reserved
1	Change Cache Line State to Invalid
2	Reserved
3	Reserved

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

8.2.2 PrcCmd/SysCmd Bus Encoding (Data Cycles)

The 20Kc System interface contains a 7-bit *EB_PrcCmd* bus and a 14-bit *EB_SysCmd* bus that contain request and identification information for the transaction. *EB_PrcCmd[6:0]* is driven by the processor. *EB_SysCmd[13:0]* is driven by the SOC controller. The encoding for each bus is identical.

Figure 8-8 below shows the command bus format for the processor and SOC controller. In 64-bit *EB_SysAD* bus mode, where the SOC controller is 64 bits, the command information is transferred as a single 14-bit value at the same time as address. On the processor side where the bus width is 32 bits, the command is transferred in two cycles. The lower seven bits are transferred first on *EB_PrcCmd[6:0]*, followed by the upper seven bits in the following clock.

There are two types of command transfers (data and address). These transfers determine the format of the command bus. During the data phase of a transaction the 'data' format is transferred on the command bus as shown in Figure 8-8.

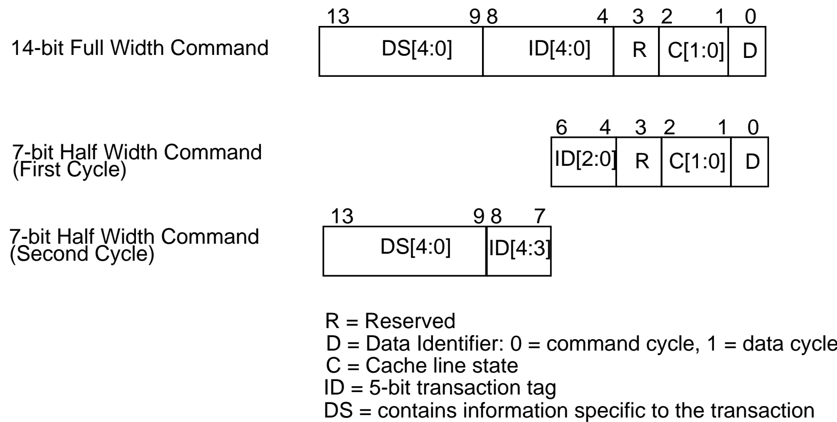


Figure 8-8 Command Bus Formats During a Data Cycle

8.2.2.1 Cache Line State Encoding

The C[1:0] field is loaded with bits [2:1] of the command bus. The C[1:0] field indicates the state of the cache line during a state response, which occurs in response to an external request. Table 8-10 shows the encoding of the C[1:0] field.

Table 8-10 Encoding of the C[1:0] Field

C[1:0]	Cache State
0	Invalid
1	Reserved
2	Clean Exclusive
3	Dirty Exclusive

8.2.2.2 Data Status Bit Encoding

The DS[4:0] field, represented by bits [13:9] of the command bus as shown in Figure 8-8 above, indicates the status of data on the bus at any given point in the transaction. The encoding for this field is shown in Table 8-11.

Table 8-11 Encoding of the DS[4:0] Field

DS[4:0]	Name	Value	Data Status
DS[4]	Last Data	1	Last data element
		0	Not the last data element

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 8-11 Encoding of the DS[4:0] Field

DS[4:0]	Name	Value	Data Status
DS[3]	Resp Data	1	Data is response data
		0	Data is not response data
DS[2]	Err Data	1	Data is erroneous
		0	Data is error free
DS[1]	Check Enable	1	Check data and parity
		0	Do not check data and parity
DS[0]	Reserved		

If DS[2] is asserted on the SysCmd bus during a read, the processor takes a Bus Error exception. (Bus Error exceptions typically are raised by the SOC controller for events such as bus time-outs, local bus parity errors, and invalid physical memory addresses or access types.) Block read requests must complete and are not aborted by Bus Error being signalled in this way. Note that during a write, the SOC controller does not drive the SysCmd bus, so equivalent errors must be signalled by the SOC controller via an interrupt exception. Also note that DS[2] serves other purposes as well, such as reporting parity errors on data that is being flushed out of the cache.

8.3 Processor and External Request Protocols (64-bit EB_SysAD Mode)

This section discusses the data transfer protocols associated with processor and external requests. The 20Kc processor supports two bus combinations for the processor and the SOC controller: 32-bit processor with 32-bit SOC controller, and 32-bit processor with 64-bit SOC controller. This section focuses on 64-bit SOC mode. Refer to Section 8.5, "Processor and External Request Protocols (32-bit EB_SysAD Mode)" for more information on 32-bit SOC mode.

The processor drives the *EB_PrcAD* bus and the SOC controller drives the *EB_SysAD* bus. The highest performance is achieved using a 64-bit *EB_SysAD* bus for the SOC controller. The 32-bit SOC mode is provided for applications which may be pin-constrained.

8.3.1 Processor Requests

A processor request is a single request or a series of requests through the 20Kc System interface used to access some external resource. The 20Kc System interface supports both read and write processor requests in block and non-block modes.

The 20Kc System interface supports the following types of requests:

- Processor read requests
- Processor write requests

Block reads are used for cache line fills and are triggered by instruction or data cache misses. Non-block reads are used for uncached load or fetch accesses. Block writes can result because of writebacks from the data cache or uncached accelerated operations. Non-block write requests are issued when the processor executes uncached stores, or write-through stores.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

8.3.1.1 Processor Read Requests

When a processor issues a read request, the SOC controller must access the specified resource and return the requested data. A processor read request can be split from the return of the requested data by the SOC controller, allowing the processor to place another request on the bus prior to the time that data from the first request is returned by the SOC controller. A processor read request is completed after the last word of response data has been received from the SOC controller.

The 20Kc System interface defines three types of read operations that can be initiated by the processor:

- Coherent Block Read Exclusive
- Noncoherent Block Read
- Double/Single/Partial Word Read

The processor issues a Read request when there are available read resources in the SOC controller as indicated by the value in the RdRscCount counter.

Processor Block Read Requests

Figure 8-9 shows a back-to-back block read operation in 64-bit EB_SysAD mode. In this example the SOC controller supports three outstanding read operations. The processor drives address and command information on *EB_PrcAD* and *EB_PrcCmd*, indicated by Adr 1 on *EB_PrcAD*. The address driven out on the *EB_PrcAD* is not block aligned, but corresponds to the actual address of the instruction that triggered the read request. This address can therefore be byte, word, halfword or doubleword aligned. In the next clock, the processor drives the second address onto the bus, denoted by Adr 2 on *EB_PrcAD*. Prior to the first address being driven the internal RdRscCount counter contains a value of 3. The driving of the first address onto the bus causes the internal RdRscCount counter to decrement by one (from 3 to 2) as shown. The driving of the second address onto the bus in the next clock causes the internal RdRscCount counter to decrement by one (from 2 to 1) as shown. Two read cycles are now outstanding on the bus.

At some later point, indicated by the break in the diagram, the SOC controller drives data for the first transaction, along with the *EB_SysDataVld* signal. The SOC controller continues to drive *EB_SysDataVld* as long as valid data is on the bus. However, the 20Kc System interface only requires that the *EB_SysRdCredit* signal be asserted for one clock. Note that the assertion of *EB_SysRdCredit* has no timing relationship to the data being driven and can be driven before, during, or after the transaction. The assertion of *EB_SysRdCredit* causes the internal RdRscCount counter to increment by 1 (from 1 to 2). When the SOC controller drives data for the second transaction, the RdRscCount counter increments by 1 (from 2 to 3).

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

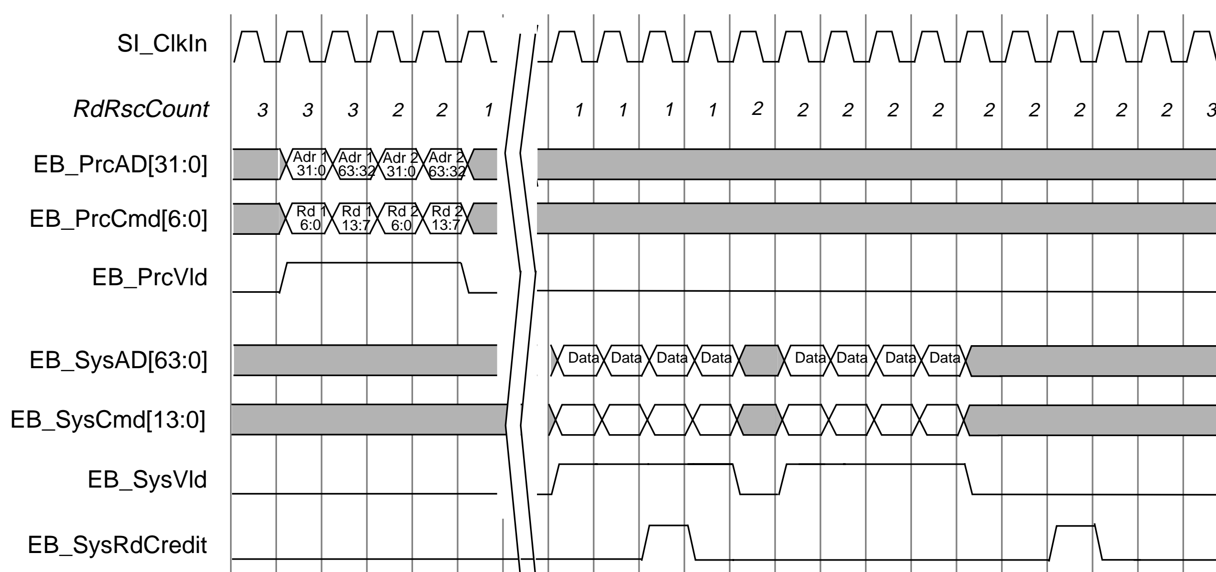


Figure 8-9 32-bit EB_PrcAD/64-bit EB_SysAD Block Read Protocol

Processor Double/Single/Partial Word Read Requests

Figure 8-10 shows a back-to-back non-block read operation in 64-bit EB_SysAD mode. The processor drives address and command information on *EB_PrcAD* and *EB_PrcCmd*, indicated by Adr 1 on *EB_PrcAD*. In the next clock, the processor drives the second address onto the bus, denoted by Adr 2 on *EB_PrcAD*. In this example the SOC controller allows up to three outstanding read cycles on the bus at any given time. Therefore, prior to the first address being driven, the internal RdRscCount counter contains a value of 3. The driving of the first address onto the bus causes the internal RdRscCount counter to decrement by one (from 3 to 2) as shown. The driving of the second address onto the bus in the next clock causes the internal RdRscCount counter to decrement by one (from 2 to 1) as shown. Two read cycles are now outstanding on the bus.

At some later point, indicated by the first break in the diagram, the SOC controller drives data for the first transaction, along with the *EB_SysDataVld* signal. In this example the SOC controller asserts *EB_SysRdCredit* one clock after the first data transfer, causing the internal RdRscCount counter to increment by 1 (from 1 to 2). The assertion of the second *EB_SysRdCredit* occurs five clocks drives after data for the second transaction, causing the RdRscCount counter to increment by 1 (from 2 to 3).

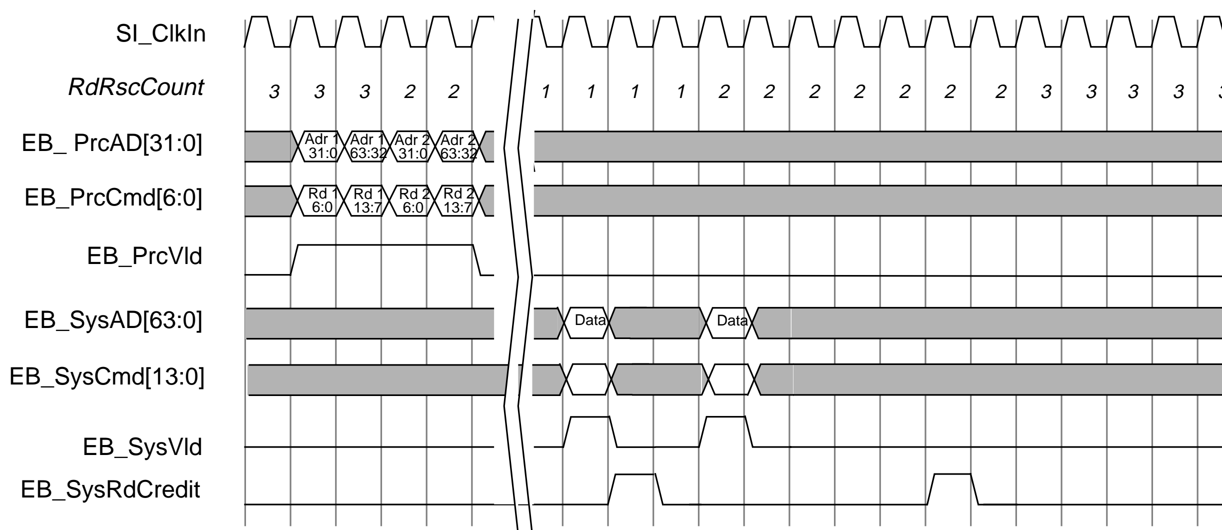


Figure 8-10 32-bit EB_PrcAD/64-bit EB_SysAD Double/Single/Partial Word Read Protocol

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

8.3.1.2 Processor Write Requests

When a processor issues a write request, the specified resource is accessed and the data is written to it. A processor write request is complete after the last word of data has been transmitted to the SOC controller.

The processor issues a Write request when there are write resources available in the SOC controller as determined by the value in the WrRscCount counter.

Processor Block Writes

In Figure 8-11, the processor drives address information on *EB_PrcAD[31:0]*, and command information on *EB_PrcCmd[6:0]*. In this example the SOC controller allows two outstanding write cycles on the bus at any given time. Therefore, prior to address being driven the internal WrRscCount counter contains a value of 2. The driving of address onto the bus causes the internal WrRscCount counter to decrement by one (from 2 to 1) as shown. Note that the assertion of *EB_SysWrCredit* by the SOC controller has no timing relationship to the data. *EB_SysWrCredit* can be driven before, during, or after the transaction.

In this example, two write operations are initiated on the bus prior to *EB_SysWrCredit* being asserted by the SOC controller. After the first transaction is driven onto the bus the counter decrements from 2 to 1. The driving of address and data for the second transaction causes the counter to decrement from 1 to 0. The assertion of *EB_SysWrCredit* during the second transaction causes the WrRscCount counter to increment by 1 (from 0 to 1). Note that the second assertion of *EB_SysWrCredit* by the SOC controller is not shown.

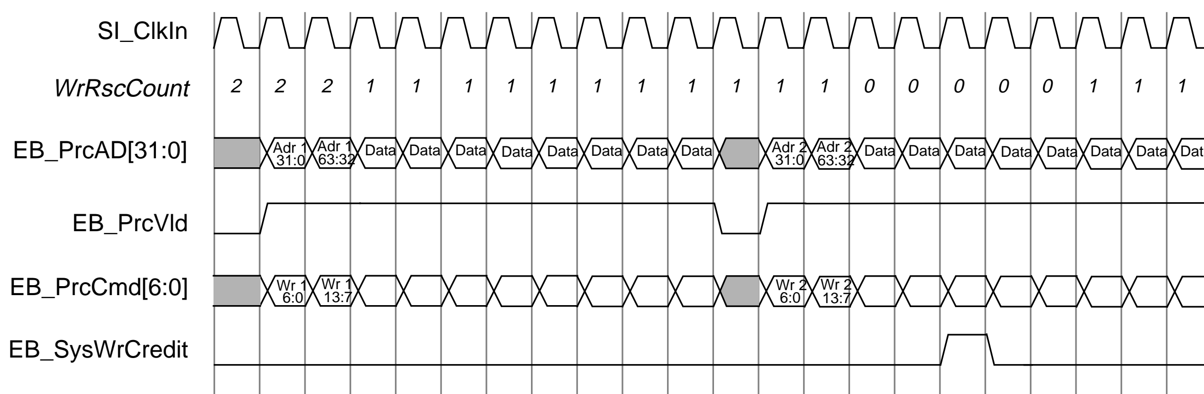


Figure 8-11 32-bit EB_PrcAD Block Write Protocol

Processor Double/Single/Partial Word Write Requests

In Figure 8-12, the processor drives address information on *EB_PrcAD* and command information on *EB_PrcCmd*. Data is driven in the following clock. In this example, the SOC controller supports up to two outstanding write cycles on the bus at any given time. Therefore, prior to address being driven the internal WrRscCount counter contains a value of 2. The processor drives address onto the bus, causing the internal WrRscCount counter to decrement by one (from 2 to 1) as shown. At some later time, indicated by the break in the timing diagram, the SOC controller asserts *EB_SysWrCredit*, indicating that it has accepted the data on the bus. When the processor samples this signal asserted, the WrRscCount counter is incremented.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

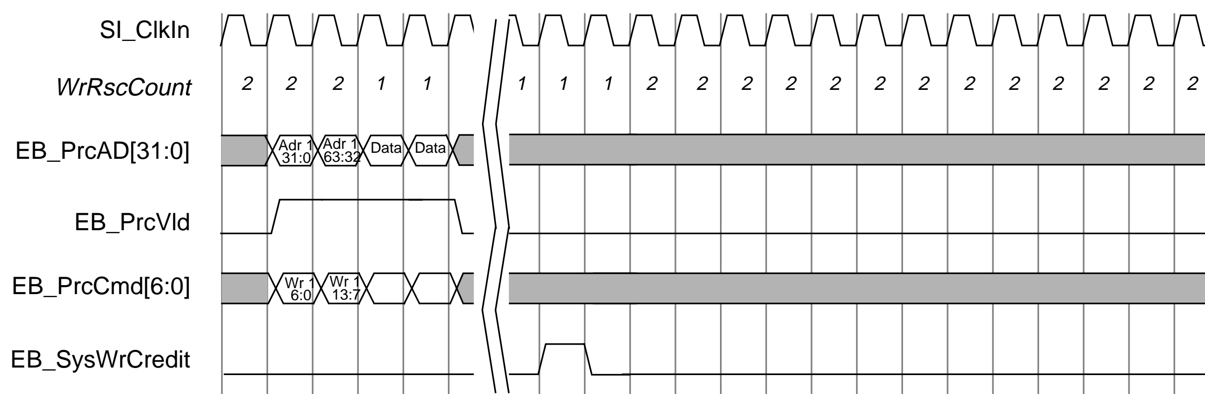


Figure 8-12 32-bit EB_PrcAD Double/Single/Partial Word Write Protocol

Processor Uncached Accelerated Block Write Request

During the address cycle of processor uncached accelerated requests, the RQS[2:0] field contains the number of valid words in the block transfer. The 20Kc System interface allows between 1 to 8 words of data to be transferred in an uncached accelerated transaction. For the encoding of the RQS[2:0] field, refer to Processor Double/Single/Partial Word Write Request Encoding.

8.3.2 External Requests

External requests include intervention and invalidate.

Intervention requests require the processor to return the state of the cache line at the specified physical address. Under certain conditions related to the state of the cache line and the nature of the intervention request, the contents of the primary cache line can be returned. The state of the line is modified by this request.

Invalidate requests specify a cache line, in the primary and secondary caches of the processor, that must be marked invalid.

8.3.2.1 External Intervention Request

The external intervention request is selected when the RQ[2:0] field contains a value of 0b110. The RQS[1:0] field determines the type of Intervention request.

The SOC controller issues an intervention when the following conditions are satisfied:

- There is a processor resource to receive the external request.
- The SOC controller has a resource available to receive the potential data response.

The 20Kc System interface supports two external requests of any type, even in the middle of a burst read or write. Figure 8-13 illustrates an intervention example with an associated data response that proceeds in the following manner:

- The SOC controller issues an intervention request to the processor.
- The processor returns a state response embedded in the command bus value that indicates the line was dirty. At the same time the processor also returns a data response on the data bus, completing the *Intervention* transaction.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

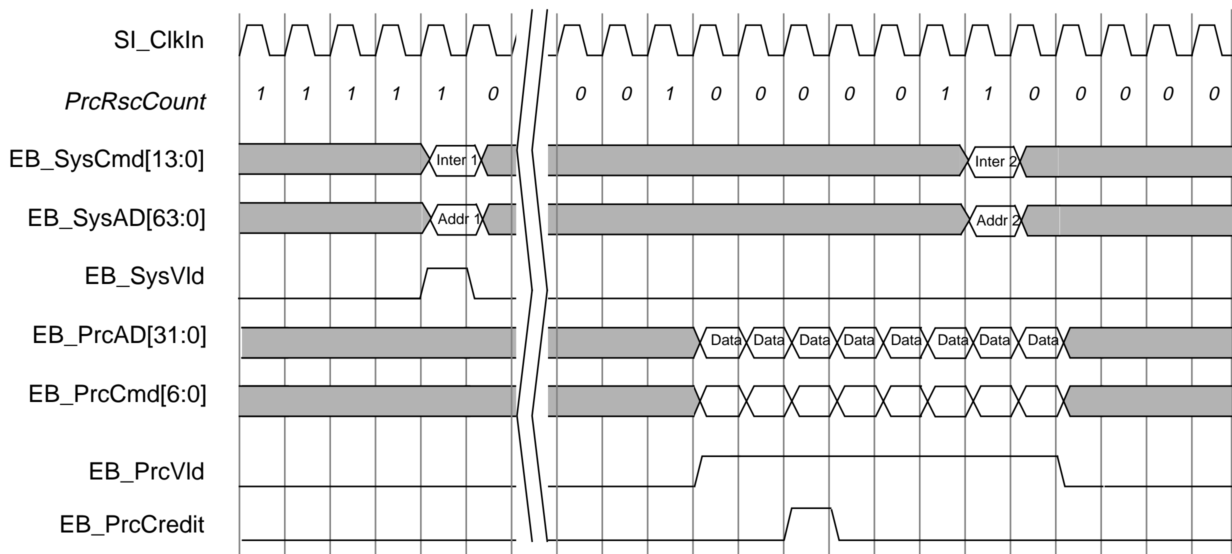


Figure 8-13 External Intervention with Associated Data Response

In Figure 8-13, the internal PrcRscCount counter has been programmed with a value of 1, indicating that only one intervention is supported on the bus at any given time. The count decrements from 1 to 0 when the intervention request (Inter 1) is placed in the *EB_SysCmd[13:0]* bus. At some later point, indicated by the break in the diagram, the processor asserts the *EB_PrcCredit* signal for one clock and drives data and command information onto the bus. The assertion of *EB_PrcCredit* causes the PrcRscCount counter to increment from 0 back to 1. In this example, another intervention request was pending. As soon as the SOC controller samples *EB_PrcCredit* asserted, it issues another intervention request (Inter 2). This in turn causes the PrcRscCount counter to again decrement from 1 to 0.

Figure 8-14 shows an external intervention request with no data response that proceeds in the following manner:

- The SOC controller issues an intervention request to the processor.
- At some later point, indicated by the break in the diagram, the processor responds by asserting *EB_PrcVld*. The processor data and command buses are driven in the same clock as *EB_PrcVld* is asserted. The *EB_PrcCmd* bus contains the state response, and the *EB_PrcAD* bus is ignored.

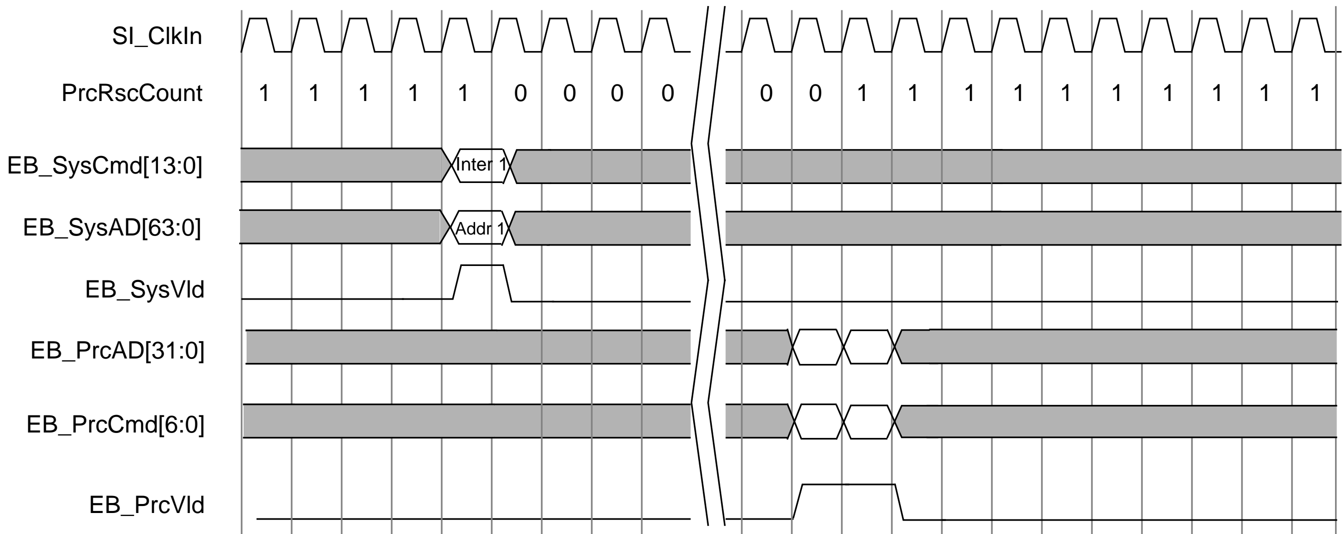


Figure 8-14 External Intervention with No Data Response

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

8.3.2.2 External Invalidation Request

The SOC controller issues an invalidation when the following conditions are satisfied:

- There is a processor resource to receive the external request.
- There is no conflicting pending processor request.

Figure 8-15 shows an external invalidation request.

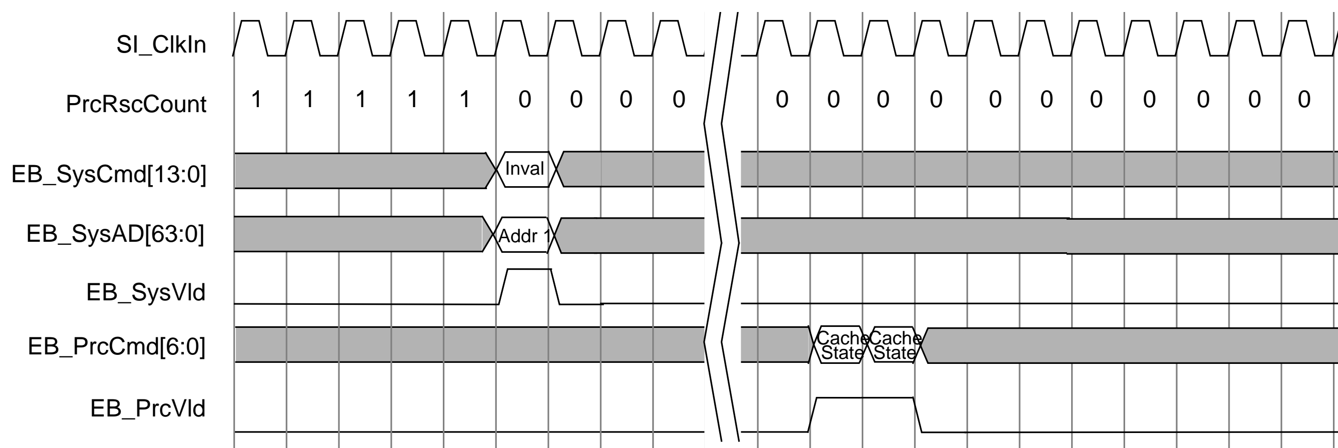


Figure 8-15 External Invalidation

In Figure 8-15, the processor receives the invalidation request on *EB_SysCmd[13:0]* and at some later point, indicated by the break in the diagram, responds by asserting *EB_PrcVld*. The processor drives the cache state onto the *EB_PrcCmd[6:0]* bus in the same clock as *EB_PrcVld* is asserted.

8.3.3 Coherency Conflicts

Coherency conflicts occur when there are concurrent requests by the processor and the SOC controller to the same cache line. Due to the point-to-point (unidirectional) buses in the 20Kc System interface, it is not always possible to detect those conflicts in time. Both the SOC controller and the processor must have some logic to handle these types of situations. Section 8.3.3.1, "Conflict Resolution" describes how the processor handles coherency conflicts, and Section 8.3.3.2, "External Observable Behavior" describes the corresponding behavior observed by the SOC controller.

8.3.3.1 Conflict Resolution

The processor implements the most straightforward solution when conflicts are present. The processor attempts to replicate the behavior of two concurrent requests in transit at the same time, even though it may detect the conflict ahead of time:

Table 8-12 Conflict Resolution

External Request	Conflicting External Request	Resolution
Block Read	Intervention Exclusive	Conflicting external requests have no effect on block read requests. The processor sends a state response of ***Invalid*** and also sends the read request with no ordering constraints.
	Invalidate	
Block Write	Intervention Exclusive	Conflicting external requests have no effect on block write requests. The block write request issues to the System interface, followed by a state response of ***Invalid.***
	Invalidate	

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

8.3.3.2 External Observable Behavior

Table 8-13 lists what the SOC controller should expect in response to its request, including both conflicting and non-conflicting cases.

Table 8-13 Acceptable Responses to External Requests

Processor Request	Possible Responses
Invalidate	The processor returns a state response of ***Invalid, Clean Exclusive***, or ***Dirty Exclusive***.
	The processor issues a read request to the same line then the processor returns a state response of ***Invalid.***
	The processor issues a write request to the same line then the processor returns a state response of ***Invalid.***
Intervention Exclusive	The processor returns a state response of ***Invalid*** or ***Clean Exclusive.***
	The processor returns a state response of ***Dirty Exclusive*** and a data response.
	The processor issues a read request to the same line then the processor returns a state response of ***Invalid.***
	The processor issues a write request to the same line then the processor returns a state response of ***Invalid.***

8.3.3.3 Implications of Coherency Conflicts on SOC Controller design

If there is a pending read request from the 20Kc for the same line as an invalidate/intervention request from the SOC or the 20Kc issues a read request for the same line before issuing the invalidate/intervention response, the 20Kc will send an invalid response no matter when the SOC controller replies to the read request. Even though the 20Kc responds with an invalid response, it will proceed to refill its internal data cache with the read response from the SOC when it receives it.The SOC controller must therefore retry the invalidate/intervention request in order to remove that line from the internal cache. This retry needs to happen only if the state response is ***Invalid.***

Illustrated below are some examples of situations where such a retry is required. In the following examples, each of the numbered steps implies a time ordered sequence.

Example 1:

1) The 20Kc sends a read request for address A to the SOC.

2) The SOC sends an intervention for address A to the 20Kc. This step can occur after or concurrently with step 1 above.

3) The 20Kc sends an invalid response to the SOC.

4) The SOC sends the read response for address A to the 20Kc.

Example 2:

1) The SOC sends an intervention for address A to the 20Kc

2) The 20Kc sends a read request for address A to the SOC

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

3) The 20Kc sends an invalid response to the SOC

4) The SOC sends the read response for address A to the 20Kc

Example 3:

1) The SOC sends an intervention for address A to the 20Kc

2) The 20Kc sends a read request for address A to the SOC. This step can occur after or concurrently with step 1 above.

3) The SOC sends the read response for address A to the 20Kc

4) The 20Kc sends an invalid response to the SOC

Example 4:

1) The 20Kc sends a read request for address A to the SOC

2) The SOC sends an intervention for address A to the 20Kc

3) The SOC sends the read response for address A to the 20Kc

4) The 20Kc sends an invalid response to the SOC

8.3.4 Data Ordering

Block data transfers require a convention to specify the order in which the different doublewords in the block are transferred on the bus. The 20Kc System interface performs and expects all data transfers to be in linear order, critical word first; the requested doubleword is always the first to be transferred on the bus. Figure 8-16 shows the data transfer ordering for the four possible requested doubleword addresses (00_2, 01_2, 10_2, 11_2).

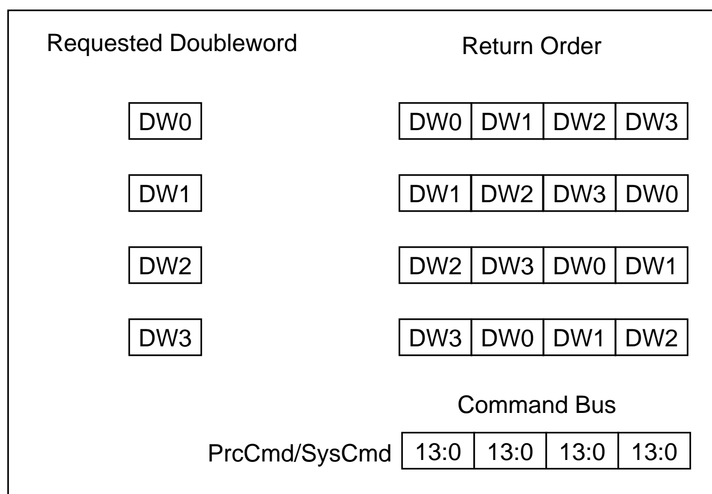


Figure 8-16 64-bit Block Data Ordering

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

All transfers on the 20Kc System interface must be 64 bits wide. Each transfer equates to 64 bits of data per data transfer along with 14 bits of command information. However, in 32-bit systems, twice as many data transfers are required to move the requested data as in a 64-bit system. In addition, the 7-bit width of the command bus requires two clocks to transfer the entire 14-bit command bus value. In this case the lower seven bits of the command bus is transferred along with the lower 32 bits of the 64-bit doubleword, and the upper seven bits of the command bus is transferred along with the upper 32 bits of the 64-bit doubleword. This is shown in the following figure.

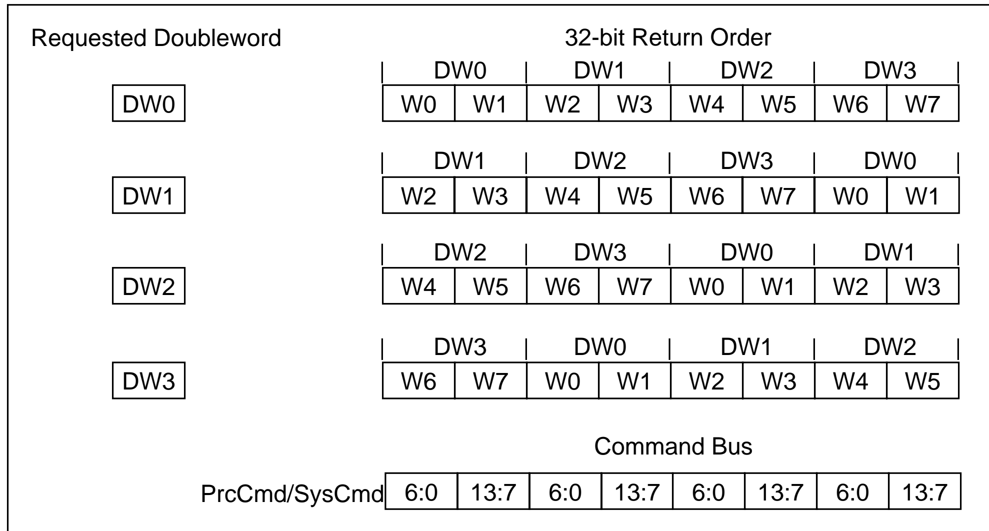


Figure 8-17 32-bit Block Data Ordering

8.3.5 Data Alignment

Data on the 20Kc System interface must always be aligned. The valid byte lines depend upon the position of the data with respect to the aligned doubleword. For example, in little-endian mode, on a byte request where the address modulo 8 is 4, *EB_SysAD[39:32]* are valid during the data cycle. Figure 8-18 illustrates such an example and contrasts it to the big-endian case.

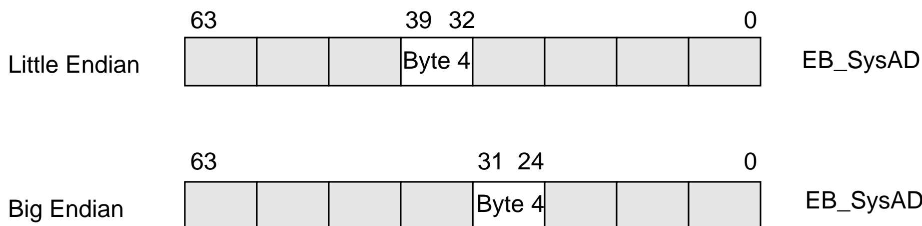


Figure 8-18 Data Alignment Example

The valid bytes in a given doubleword on the system bus depend on the low-order address bits, the size of the access, and the endianness. Refer to Table 2-1 for more information on the valid bytes.

8.3.6 Dual Multiplexed Address and Data Buses

The 20Kc System interface contains a 32-bit processor unidirectional multiplexed address/data bus, and a 32- or 64-bit SOC controller unidirectional multiplexed address/data bus. The *EB_PrcAD[31:0]* bus is always driven by the processor, and the *EB_SysAD[63:0]* bus is always driven by the SOC controller. The minimum transfer size on the bus

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

is 64 bits. The 20Kc System interface also supports a 32-bit *EB_SysAD* bus. The number of address and data transfers required depends on the configuration of the processor and SOC controller as shown in Table 8-14.

Table 8-14 20Kc System Interface Address and Data Transfer Requirements

Cycle Type	Address/Data Transfers	Configuration	
		32-bit EB_PrcAD 32-bit EB_SysAD	32-bit EB_PrcAD 64-bit EB_SysAD
Block Read (32 bytes)	Address:	2 clocks on EB_PrcAD	2 clocks on EB_PrcAD
	Data:	8 clocks on EB_SysAD	4 clocks on EB_SysAD
Double/Single/ Partial Word Read	Address:	2 clocks on EB_PrcAD	2 clocks on EB_PrcAD
	Data:	2 clocks on EB_SysAD[1]	1 clocks on EB_SysAD
Block Write (32 bytes)	Address:	2 clocks on EB_PrcAD	2 clocks on EB_PrcAD
	Data:	8 clocks on EB_PrcAD	8 clocks on EB_PrcAD
Double/Single/ Partial Word Write	Address:	2 clocks on EB_PrcAD	2 clocks on EB_PrcAD
	Data:	2 clocks on EB_PrcAD[2]	2 clocks on EB_PrcAD
Intervention (no data response)	Address:	2 clocks on EB_SysAD	1 clocks on EB_SysAD
	Data:	2 clocks on EB_PrcAD	2 clocks on EB_PrcAD
Intervention (with data response)	Address:	2 clocks on EB_SysAD	1 clocks on EB_SysAD
	Data:	8 clocks on EB_PrcAD	8 clocks on EB_PrcAD
Invalidate	Address:	2 clocks on EB_SysAD	1 clocks on EB_SysAD
	Data:	2 clocks on EB_PrcAD[3]	2 clocks on EB_PrcAD

1. 64 bits = minimum read size
2. 64 bits = minimum write size
3. No data cycles on invalidates

The 20Kc System interface allows for a full 32-byte cache line to be retrieved in one eight-transfer burst. Both the processor and the SOC controller address/data and control buses are parity protected.

- The *EB_PrcADP[1:0]* signals provide parity for the processor address/data bus.
- The *EB_SysADP[7:0]* signals provide parity for the SOC controller address/data bus.
- The *EB_PrcCmdP* signal provides parity for the processor control bus.
- The *EB_SysCmdP[1:0]* signals provide parity for the SOC controller control bus.

The processor parity bits map to the address/data bytes as shown in Table 8-15.

Table 8-15 Processor Address/Data Bus and Corresponding Parity Bit

Data	Parity Bit
EB_PrcAD[31:24]	*EB_PrcADP[3]*
EB_PrcAD[23:16]	*EB_PrcADP[2]*
EB_PrcAD[15:8]	*EB_PrcADP[1]*
EB_PrcAD[7:0]	*EB_PrcADP[0]*

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

The SOC controller parity bits map to the system address/data bytes as shown in Table 8-16.

Table 8-16 System Address Bus and Check Bits

Data	Parity Bit
EB_SysAD[63:56]	*EB_SysADP[7]*
EB_SysAD[55:48]	*EB_SysADP[6]*
EB_SysAD[47:40]	*EB_SysADP[5]*
EB_SysAD[39:32]	*EB_SysADP[4]*
EB_SysAD[31:24]	*EB_SysADP[3]*
EB_SysAD[23:16]	*EB_SysADP[2]*
EB_SysAD[15:8]	*EB_SysADP[1]*
EB_SysAD[7:0]	*EB_SysADP[0]*

8.4 Bus Encoding (32-bit EB_SysAD Mode)

This section contains the *EB_PrcCmd[6:0]* and *EB_SysCmd[6:0]* bus encoding for command and data cycles in 32-bit EB_SysAD mode. Note that even through the processor and SOC controller bus widths are 32 bits wide, transfers are still made in 64-bit quantities over two clocks. In the first clock, the lower 32 bits of data are transferred along with the lower seven bits of command information on *EB_PrcCmd[6:0]* or *EB_SysCmd[6:0]*. In the second clock, the upper 32 bits of data are transferred along with the upper seven bits of command information on *EB_PrcCmd[6:0]* or *EB_SysCmd[6:0]*. This section discusses 32-bit SOC mode. Refer to Section 8.2, "Bus Encoding (64-bit EB_SysAD Mode)" for more information on the 64-bit SOC mode.

8.4.1 PrcCmd/SysCmd Bus Encoding (Command Cycles)

In 32-bit EB_SysAD mode, the 20Kc System interface contains a 7-bit processor command bus and a 7-bit EB_SysAD command bus that contain request and identification information for the transaction. *EB_PrcCmd[6:0]* is driven by the processor and contains the command information. *EB_SysCmd[6:0]* is driven by the SOC controller. The encoding for each bus is identical.

Figure 8-19 shows the command bus format for the processor and SOC controller. In 32-bit bus mode, where the SOC controller is 32 bits, the command is transferred in two consecutive clocks. The lower seven bits is transferred first followed by the upper seven bits in the following clock. The same is true for transfers on the *EB_PrcCmd[6:0]* command bus.

There are two types of command transfers (data and address). These transfers determine the format of the command bus. During the address phase of a transaction the ‘command’ format is transferred as shown in Figure 8-19.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

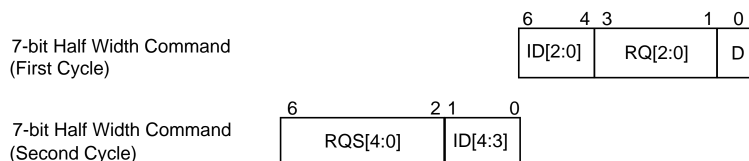


Figure 8-19 Command Bus Formats During a Command Cycle (32-bit EB_SysAD Mode)

8.4.1.1 Processor Request Encoding

The *EB_PrcCmd[6:0]* bus contains command information about each of the above requests. The encoding for the *EB_SysCmd* bus is identical. The RQ[2:0] field in Figure 8-19 encodes the different type of processor requests shown in Table 8-17.

Table 8-17 Request Encoding

RQ[2:0]	Request Type
0	Read Request (Processor only)
1	Reserved
2	Write Request (Processor only)
3	Reserved
4	Invalidate Request (SOC Controller only)
5	Reserved
6	Intervention Request (SOC Controller only)
7	Reserved

8.4.1.2 Processor Read Request Encoding

A processor read request occurs when the RQ[2:0] field contains a value of 0b000.

Processor Read Request Type Encoding

The upper two bits of the RQS[4:3] field encode the type of processor read request as shown in Table 8-18. This information is useful in handling the associated external response.

Table 8-18 Encoding of RQS[4:3] for Processor Read Requests

RQS[4:3]	Request Type
0	Reserved
1	Coherent Block Read, Exclusive
2	Noncoherent Block Read
3	Double/Single/Partial Word Read

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Processor Block Read Transfer Size Encoding

Block read requests transfer 32 bytes of data in a single transaction. The lower two bits of the RQS field, RQS[1:0], indicate the size of the block read transfer as shown in Table 8-19. Note that during a block read request, the RQS[2] bit is reserved and is not used in the encoding process.

Table 8-19 Encoding of RQS[1:0] for Processor Block Read Requests

RQS[1:0]	Block Transfer Size
0	Reserved
1	8 words
2	Reserved
3	Reserved

Processor Double/Single/Partial Word Read Encoding

The 20Kc System interface allows for the transfer of doubleword, word, and partial word data types. A partial word read requests the transfer of less than 32 bits of data. A single word read requests 32 bits of data. A doubleword read requests 64 bits of data. During these types of requests, the lower three bits of the RQS field, RQS[2:0], encode the number of bytes valid for the transaction as shown in Table 8-20. Note that *EB_PrcAD[2:0]* contains the address of the first valid byte in the doubleword. Refer to Table 8-14 for all valid combinations of address and access size.

Table 8-20 Encoding of RQS[2:0] for Processor D/S/P Word Read Requests

RQS[2:0]	Number of Bytes Valid
0	1 byte valid
1	2 bytes valid
2	3 bytes valid
3	4 bytes valid
4	5 bytes valid
5	6 bytes valid
6	7 bytes valid
7	8 bytes valid

8.4.1.3 Processor Write Request Encoding

The upper five bits of the command bus, RQS[4:0], contains information specific to the transaction indicated by the RQ[2:0] field.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Processor Write Request Type Encoding

The upper two bits of this field, RQS[4:3], indicate the type of write operation. During the address cycle of processor write requests, RQS[4:3] contain the type of write operation as shown in Table 8-21. This information is useful in handling the associated write data.

Table 8-21 Encoding of RQS[4:3] for Processor Write Requests

RQS[4:3]	Write Cause Indication
0	Uncached Accelerated Block Write
1	Reserved
2	Block Write
3	Double/Single/Partial Word Write

Processor Block Write Transfer Size Encoding

Block write requests transfer 32 bytes of data in a single transaction. The lower two bits of the RQS field, RQS[1:0], indicate the size of the block write transfer as shown in Table 8-22. Note that during a block write request, the RQS[2] bit is reserved and is not used in the encoding process.

Table 8-22 Encoding of RQS[1:0] for Processor Block Write Requests

RQS[1:0]	Block Transfer Size
0	Reserved
1	8 words
2	Reserved
3	Reserved

Processor Double/Single/Partial Word Write Request Encoding

The 20Kc System interface allows for the transfer of doubleword, word, and partial word data types. A partial word write requests the transfer of less than 32 bits of data. A single-word write requests 32 bits of data. A doubleword write requests 64 bits of data. During these types of requests, the lower three bits of the RQS field, RQS[2:0], encode the number of bytes valid for the transaction as shown in Table 8-23. Note that *EB_PrcAD[2:0]* contains the address of the first valid byte in the doubleword. Refer to Table 8-14 for all valid combinations of address and access size.

Table 8-23 Encoding of RQS[2:0] for Processor D/S/P Word Write Requests

RQS[2:0]	Number of Bytes Valid
0	1 byte valid
1	2 bytes valid
2	3 bytes valid
3	4 bytes valid
4	5 bytes valid
5	6 bytes valid
6	7 bytes valid
7	8 bytes valid

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Processor Uncached Accelerated Block Write Request Encoding

During the address cycle of processor uncached accelerated block write requests, RQS[2:0] contain the number of valid words in the block transfer as shown in Table 8-24. This value is needed because the transfer size on the system bus is always eight words regardless of the actual number of valid words contained within the block. Note that uncached accelerated writes are always block-aligned, and the first valid word is the one located at the block-aligned address.

Table 8-24 Encoding of RQS[2:0] for Processor Uncached Accelerated Write Requests

RQS[2:0]	Number of Words Valid
0	1 word valid
1	2 words valid
2	3 words valid
3	4 words valid
4	5 words valid
5	6 words valid
6	7 words valid
7	8 words valid

8.4.1.4 External Invalidation Request Encoding

The external invalidation request is selected when the RQ[2:0] field contains a value of 0b100. During this type of request, the RQS[1:0] field is reserved.

8.4.1.5 External Intervention Request Encoding

The external intervention request is selected when the RQ[2:0] field contains a value of 0b110. The RQS[1:0] bits determine the type of intervention request as shown in Table 8-25. Note that during an intervention request, RQS[4:2] are reserved and are not used in the encoding process.

Table 8-25 Encoding of RQS[1:0] for External Intervention Requests

RQS[1:0]	Type of Intervention
0	Reserved
1	Change Cache Line State to Invalid
2	Reserved
3	Reserved

8.4.1.6 PrcCmd/SysCmd Bus Encoding (Data Cycles)

In 32-bit EB_SysAD mode, the 20Kc System interface contains two identical 7-bit command buses that contain request and identification information for the transaction. EB_PrcCmd[6:0] is driven by the processor and contains the command information. *EB_SysCmd[6:0]* is driven by the SOC controller. The encoding for each bus is identical.

Figure 8-20 below shows the command bus format for the processor and SOC controller. In 32-bit EB_SysAD mode, where the bus width is 32 bits, the command is transferred in two consecutive clocks. The lower seven bits are transferred first, followed by the upper seven bits in the following clock. This is true for both the *EB_PrcCmd[6:0]* and *EB_SysCmd[6:0]* buses.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

There are two types of command transfers (data and address). These transfers determine the format of the command bus. During the data phase of a transaction the ‘data’ format is transferred on the command bus as shown in Figure 8-20.

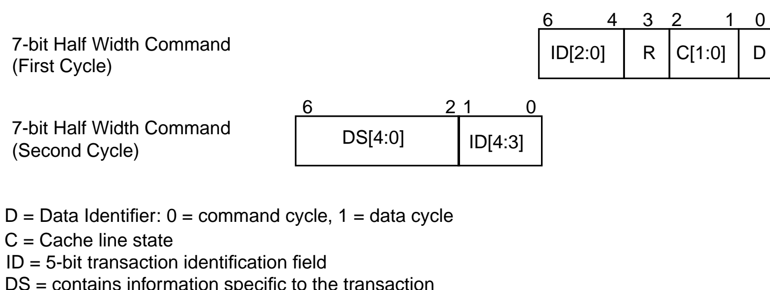


Figure 8-20 Command Bus Formats During a Data Cycle

8.4.1.7 Cache Line State Encoding

The C[1:0] field is loaded with bits [2:1] of the command bus. The C[1:0] field indicates the state of the cache line during a state response, which occurs in response to an external request. Table 8-26 shows the encoding of the C[1:0] field.

Table 8-26 Encoding of the C[1:0] Field

C[1:0]	Cache State
0	Invalid
1	Reserved
2	Clean Exclusive
3	Dirty Exclusive

8.4.1.8 Data Status Bit Encoding

The DS[4:0] field in Figure 8-20 above indicates the status of data on the bus at any given point in the transaction. The encoding for this field is shown in Table 8-27.

Table 8-27 Encoding of the DS[4:0] Field

DS[4:0]	Name	Value	Data Status
DS[4]	Last Data	1	Last data element
		0	Not the last data element
DS[3]	Resp Data	1	Data is response data
		0	Data is not response data
DS[2]	Err Data	1	Data is erroneous
		0	Data is error free
DS[1]	Check Enable	1	Check data and parity
		0	Do not check data and parity
DS[0]	Reserved		

If DS[2] is asserted on the SysCmd bus during a read, the processor takes a Bus Error exception. (Bus Error exceptions typically are raised by the SOC controller for events such as bus time-outs, local bus parity errors, and invalid physical

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

memory addresses or access types.) Block read requests must complete and are not aborted by Bus Error being signalled in this way. Note that during a write, the SOC controller does not drive the SysCmd bus, so equivalent errors must be signalled by the SOC controller via an interrupt exception. Also note that DS[2] serves other purposes as well, such as reporting parity errors on data that is being flushed out of the cache.

8.5 Processor and External Request Protocols (32-bit EB_SysAD Mode)

This section discusses the data transfer protocols associated with processor and external requests in 32-bit EB_SysAD mode. The 20Kc System interface supports two bus combinations for the processor and the SOC controller. This section focuses on 32-bit EB_SysAD mode. Refer to Section 8.3, "Processor and External Request Protocols (64-bit EB_SysAD Mode)" for more information on 64-bit EB_SysAD mode.

The processor drives the *EB_PrcAD* bus, and the SOC controller drives the *EB_SysAD* bus. The highest performance is achieved using a 64-bit *EB_SysAD* bus for the SOC controller.

8.5.1 Processor Requests

A processor request can be a single request, or a series of requests through the 20Kc System interface used to access some external resource. The 20Kc System interface supports both read and write processor requests in block and non-block modes.

Block reads are used for cache line fills and are triggered by instruction or data cache misses. Non-block reads are used for uncached load or fetch accesses. Block writes can result because of writebacks from the data cache, or uncached accelerated operations. Non-block write requests are issued when the processor executes uncached stores, or write-through stores.

8.5.1.1 Processor Read Requests

When a processor issues a read request, the SOC controller must access the specified resource and return the requested data. A processor read request can be split from the return of the requested data by the SOC controller, allowing the processor to place another request on the bus prior to the time that data from the first request is returned by the SOC controller. A processor read request is completed after the last word of response data has been received from the SOC controller.

The 20Kc System interface defines three types of read operations that can be initiated by the processor:

- Coherent block read, exclusive
- Noncoherent block read
- Double/Single/Partial word read

The processor issues a Read request when there are adequate read resources available in the SOC controller as determined by the value of the RdRscCount counter.

Processor Block Read Requests

Figure 8-21 shows a back-to-back block read operation in 32-bit EB_SysAD mode. The processor drives address and command information on *EB_PrcAD* and *EB_PrcCmd*, indicated by Adr 1 on *EB_PrcAD*. The address driven out on the *EB_PrcAD* is not block aligned, but corresponds to the actual address of the instruction that triggered the read request. This address can therefore be byte, word, halfword or doubleword aligned. In the next clock, the processor drives the second address onto the bus, denoted by Adr 2 on *EB_PrcAD*. In this example the SOC controller supports up to three outstanding read cycles on the bus at any given time. Therefore, prior to the first address being driven the internal RdRscCount counter contains a value of 3. The driving of the first address onto the bus causes the internal RdRscCount

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

counter to decrement by one (from 3 to 2) as shown. The driving of the second address onto the bus in the next clock causes the internal RdRscCount counter to decrement by one (from 2 to 1) as shown. Two read cycles are now outstanding on the bus.

At some later point, indicated by the break in the diagram, the SOC controller drives data for the first transaction along with the *EB_SysDataVld* signal. The SOC controller continues to drive *EB_SysDataVld* as long as valid data is on the bus. However, the 20Kc System interface only requires that the *EB_SysRdCredit* signal be asserted for one clock either before, during, or after completion of the transaction. In this example, assertion of the first *EB_SysRdCredit* by the SOC controller occurs about halfway through the first data transaction, causing the internal RdRscCount counter to increment by 1 (from 1 to 2). The second *EB_SysRdCredit* issued by the SOC controller also occurs about half way through the second data transaction, causing the internal RdRscCount counter to increment by 1 (from 2 to 3).

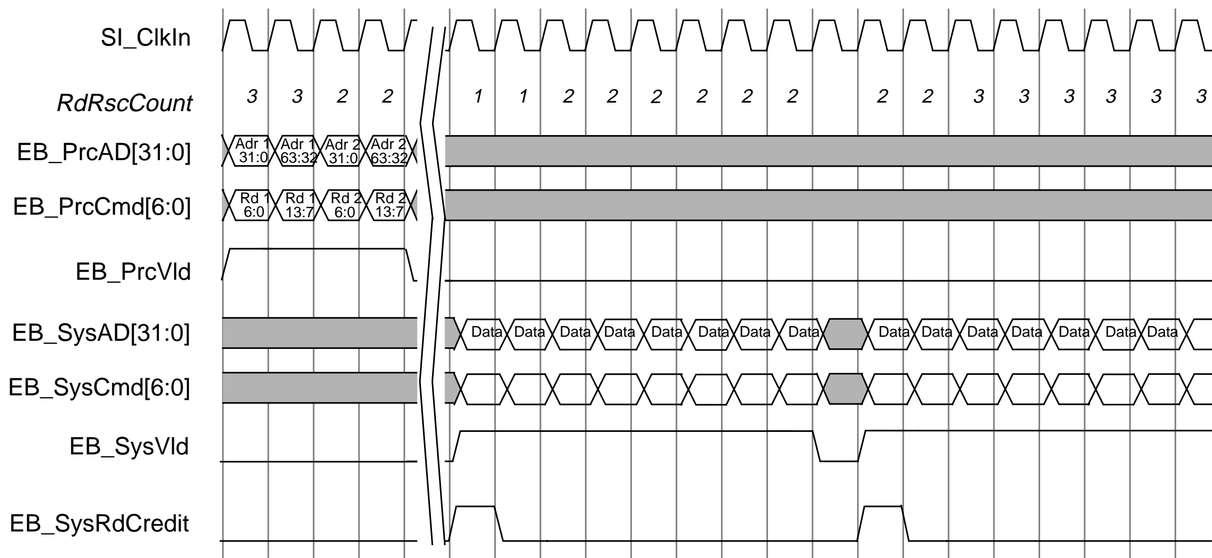


Figure 8-21 32-bit EB_PrcAD/32-bit EB_SysAD Block Read Protocol

Processor Double/Single/Partial Word Read Requests

Figure 8-22 shows a back-to-back non-block read operation in 32-bit EB_SysAD mode. In this diagram the processor drives address and command information onto *EB_PrcAD* and *EB_PrcCmd*, indicated by Adr 1 on EB_PrcAD. In the next clock the processor drives the second address onto the bus, denoted by Adr 2 on *EB_PrcAD*. In this example, the SOC controller supports up to three outstanding read operations on the bus at any given time. Therefore, prior to the first address being driven the internal RdRscCount counter contains a value of 3. The processor drives the first address onto the bus, causing the internal RdRscCount counter to decrement by one (from 3 to 2) as shown. The driving of the second address onto the bus in the next clock causes the internal RdRscCount counter to decrement by one (from 2 to 1) as shown. Two read cycles are now outstanding on the bus.

At some later point, indicated by the first break in the diagram, the SOC controller drives data for the first transaction along with the *EB_SysDataVld* signal. At some point before, during, or after the first transaction the SOC controller asserts *EB_SysRdCredit*, causing the internal RdRscCount counter to increment by 1 (from 1 to 2). At some point after the assertion of the first data transfer, the SOC controller then drives data for the second transaction, indicated by the second break in the diagram, then again asserts EB_SysRdCredit, causing the RdRscCount counter to increment from 2 to 3. Note that there is no timing relationship between the transfer of data and the assertion of *EB_SysRdCredit*.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

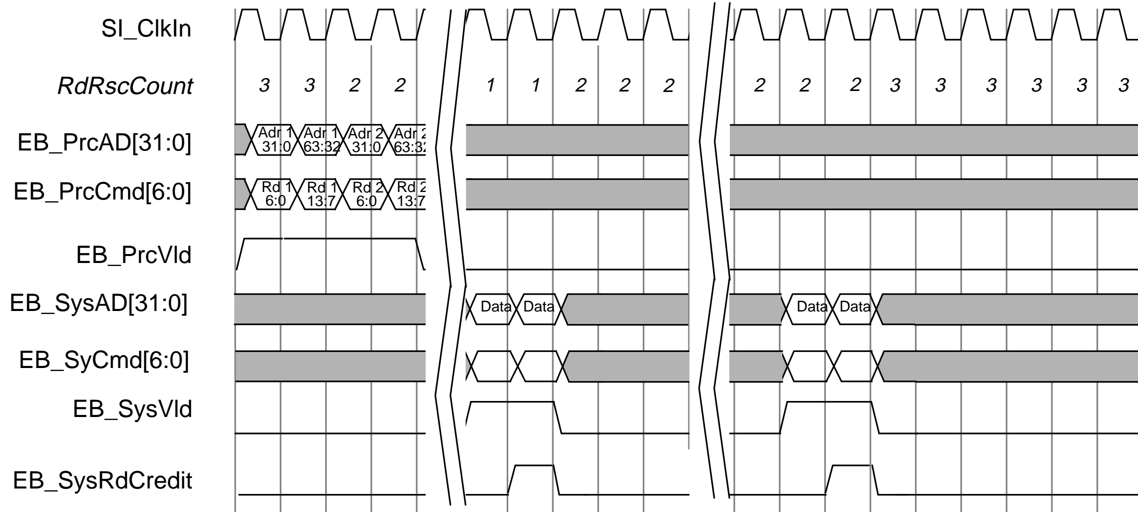


Figure 8-22 32-bit EB_PrcAD/32-bit EB_SysAD Double/Single/Partial Word Read Protocol

8.5.1.2 Processor Write Requests

When a processor issues a write request, the specified resource is accessed and the data is written to it. A processor write request is complete after the last word of data has been transmitted to the SOC controller.

The processor issues a Write request when there are write resources available in the SOC controller as indicated by the value in the WrRscCount counter.

Processor Block Writes

Figure 8-23 shows a processor block write in 32-bit EB_SysAD mode. The processor drives address information on EB_PrcAD, and command information on *EB_PrcCmd*. Data is driven in the following clock. In this example the SOC controller allows two outstanding write cycles on the bus at any given time. Therefore, prior to address being driven the internal WrRscCount counter contains a value of 2. The driving of address onto the bus causes the internal WrRscCount counter to decrement by one (from 2 to 1) as shown. The SOC controller can assert *EB_SysWrCredit* at any point before, during, or after the transaction is completed. A few clocks later the processor drives address for the second transaction onto the bus, causing the counter to again decrement from 1 to 0. Once the processor samples *EB_SysWrCredit* asserted, the WrRscCount counter is incremented (from 0 to 1). Note that the assertion of *EB_SysWrCredit* for the second block write is not shown in the figure.

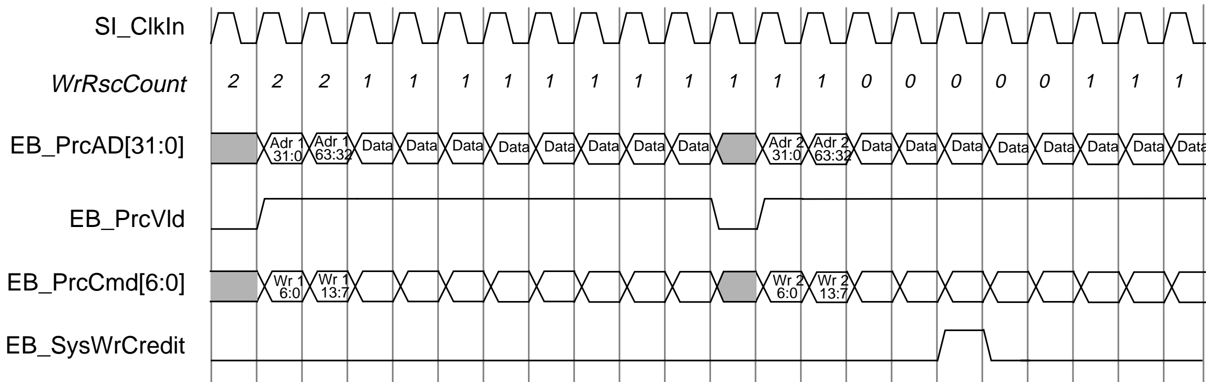


Figure 8-23 32-bit EB_PrcAD Block Write Protocol

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Processor Double/Single/Partial Word Write Requests

In Figure 8-24, the processor drives address information on *EB_PrcAD*, and command information on *EB_PrcCmd*. Data is driven in the following clock. In this example, the SOC controller supports two outstanding write cycles on the bus at any given time. Therefore, prior to address being driven the internal WrRscCount counter contains a value of 2. The driving of address onto the bus causes the internal WrRscCount counter to decrement by one (from 2 to 1) as shown. At some later time, indicated by the break in the timing diagram, the SOC controller asserts *EB_SysWrCredit*, indicating that it has accepted the data on the bus. Once the processor samples this signal asserted, the WrRscCount counter is incremented.

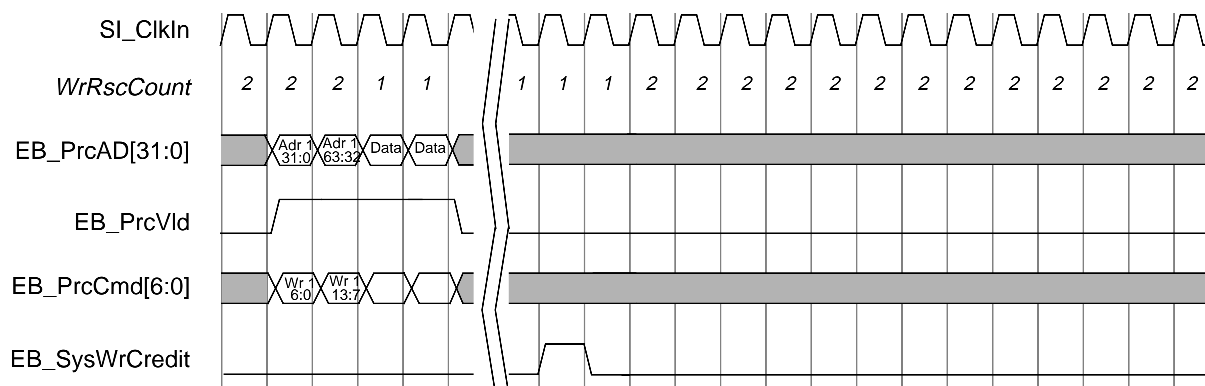


Figure 8-24 32-bit EB_PrcAD Double/Single/Partial Word Write Protocol

Processor Uncached Accelerated Write Request

During the address cycle of processor uncached accelerated requests, RQS[2:0] contain the number of valid words in the block transfer. The 20Kc System interface allows between 1 to 8 words of data to be transferred in an uncached accelerated transaction. For the encoding of the RQS[2:0] field, refer to the section entitled “Processor Uncached Accelerated Block Write Request Encoding” on page 160.

8.5.1.3 External Requests

External requests include intervention and invalidate. These requests apply to the data cache only, not the instruction cache.

Intervention requests require the processor to return the state of the cache line at the specified physical address. Under certain conditions related to the state of the cache line and the nature of the intervention request, the contents of the cache line can be returned. The state of the line is modified by this request.

Invalidate requests specify a cache line that must be marked invalid in the processor data cache.

External Intervention Request

The external intervention request is selected when the RQ[2:0] field contains a value of 0b110. The RQS[1:0] bits determine the type of Intervention request. The SOC controller issues an intervention when the following conditions are satisfied:

- There is a processor resource to receive the external request.
- The SOC controller has a resource available to receive the potential data response.

The 20Kc System interface supports two external requests of any type. Figure 8-25 illustrates an intervention example with an associated data response that proceeds in the following manner:

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

- The SOC controller issues an intervention request to the processor.
- The processor returns a state response embedded in the command bus value that indicates if the line was dirty. At the same time the processor also returns a data response on the data bus, completing the Intervention transaction.

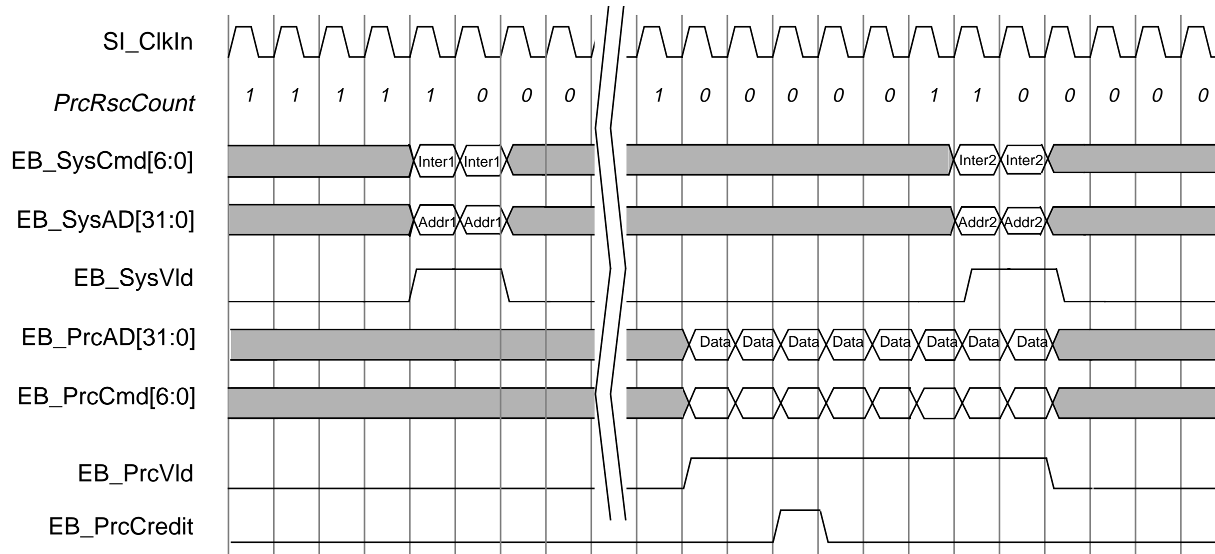


Figure 8-25 External Intervention with Associated Data Response

In Figure 8-25, the internal PrcRscCount counter has been programmed with a value of 1, indicating that only one intervention is supported on the bus at any given time. The count decrements from 1 to 0 when the intervention request (Inter 1) is placed in the *EB_SysCmd[6:0]* bus. At some later point, indicated by the break in the diagram, the processor asserts the *EB_PrcCredit* signal for one clock and drives data and command information onto the bus. The assertion of *EB_PrcCredit* causes the PrcRscCount counter in the SOC controller to increment from 0 back to 1. In this example a second intervention request is pending, but could not be generated until after the assertion of *EB_PrcCredit* by the processor. As soon as the SOC controller samples *EB_PrcCredit* asserted, it issues another intervention request (Inter 2). This in turn causes the PrcRscCount counter to again decrement from 1 to 0 as shown in Figure 8-25.

Figure 8-26 shows an example of an external intervention with no data response that proceeds in the following manner.

- The SOC controller issues an intervention request to the processor.
- At some later point, indicated by the break in the diagram, responds by asserting *EB_PrcVld*. The processor data and command buses are driven in the same clock as *EB_PrcVld* is asserted. The command bus contains the state response, while the data bus is ignored.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

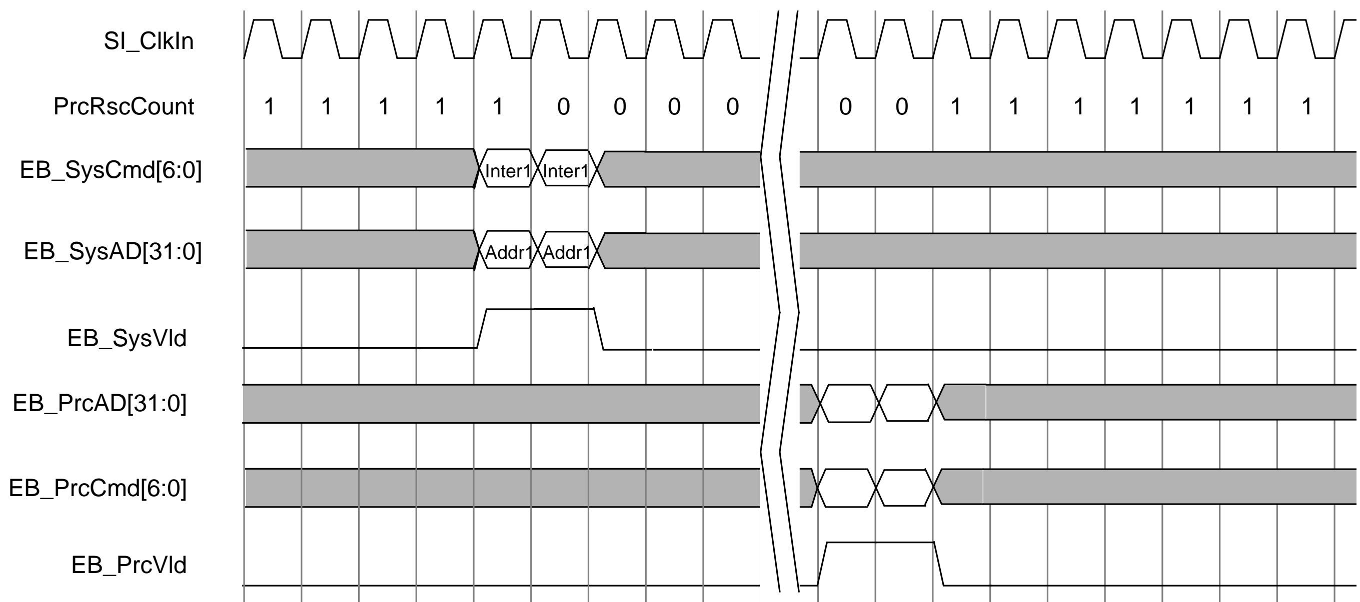


Figure 8-26 External Intervention with No Data Response

External Invalidation Request

The SOC controller issues an invalidation when the following conditions are satisfied:

- There is a processor resource to receive the external request.
- There is no conflicting pending processor request.

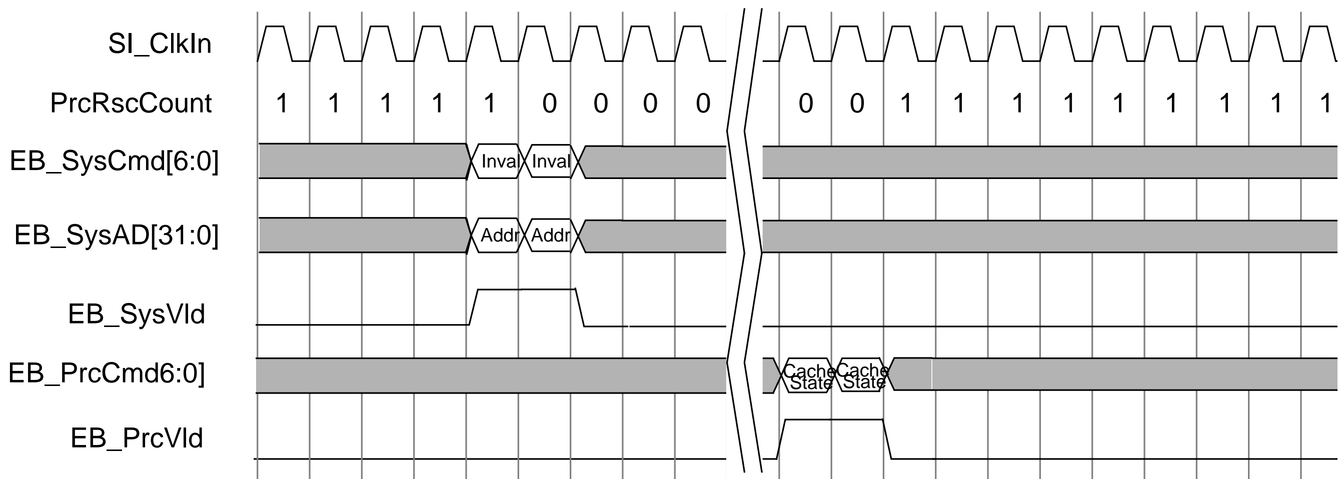


Figure 8-27 External Invalidation

In Figure 8-27, the processor receives the invalidation request on *EB_SysCmd[6:0]* and at some later point, indicated by the break in the diagram, responds by asserting *EB_PrcVld*. The processor drives the cache state onto the *EB_PrcCmd[6:0]* bus in the same clock as *EB_PrcVld* is asserted.

8.6 20Kc Signal Descriptions

Figure 8-28 shows the signals that make up the System interface of the 20Kc processor core.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

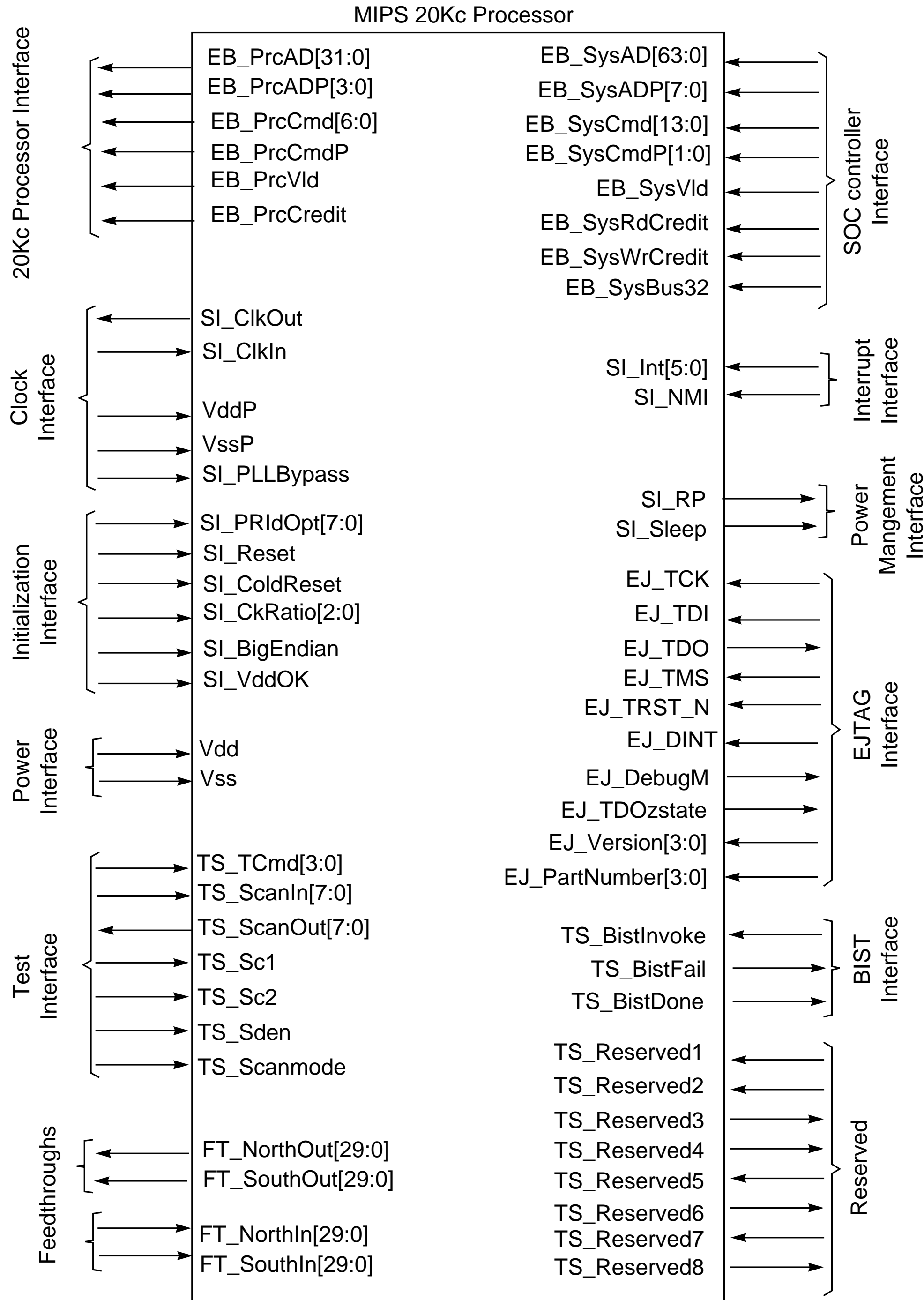


Figure 8-28 20Kc System Interface Signal Groupings

The pin direction key for the signal descriptions is shown in Table 8-28 below.

The names of interface signals present on a 20Kc core are prefixed with a unique string, that identifies their primary function. Table 8-29 defines the prefixes used for 20Kc core interface signals.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 8-28 20Kc Core Signal Direction Key

Dir	Description
I	Input to the 20Kc core. Unless otherwise noted, these signals are sampled on the rising edge of the clock signal.
O	Output of the 20Kc core. Unless otherwise noted, these signals are driven on the rising edge of the clock signal.
S	Static Input to the 20Kc core. These signals must not change state while *SI_VddOk* is asserted.

Table 8-29 Signal Prefix Key

Prefix	Description
EB_	External Bus signals
SI_	System Interface signals
EJ_	EJTAG interface signals
TS_	Test signals
FT_	Feedthrough signals

The 20Kc signal descriptions are listed in Table 8-30. The 20Kc core signals are described alphabetically by function.

Table 8-30 20Kc Core Signal Descriptions

<table>
<tr><th>Pin Name</th><th>Direction</th><th>Description</th></tr>
<tr><td colspan="3">External Bus Interface: SOC Interface Signals</td></tr>
<tr><td>EB_SysAD[63:0]</td><td>I</td><td>Core Interface address/data. This bus is driven by the SOC interface controller. The SOC controller drives address information on intervention or invalidate requests, and data information on a processor read.</td></tr>
<tr><td>EB_SysADP[7:0]</td><td>I</td><td>Core Interface address/data parity. The EB_SysADP[7:0] parity bits correspond to the EB_SysAD[63:0] data bus as follows:
<table>
<tr><th>Parity Bit</th><th>Corresponding Data Byte</th></tr>
<tr><td>EB_SysADP[0]</td><td>EB_SysAD[7:0]</td></tr>
<tr><td>EB_SysADP[1]</td><td>EB_SysAD[15:8]</td></tr>
<tr><td>EB_SysADP[2]</td><td>EB_SysAD[23:16]</td></tr>
<tr><td>EB_SysADP[3]</td><td>EB_SysAD[31:24]</td></tr>
<tr><td>EB_SysADP[4]</td><td>EB_SysAD[39:32]</td></tr>
<tr><td>EB_SysADP[5]</td><td>EB_SysAD[47:40]</td></tr>
<tr><td>EB_SysADP[6]</td><td>EB_SysAD[55:48]</td></tr>
<tr><td>EB_SysADP[7]</td><td>EB_SysAD[63:56]</td></tr>
</table></td></tr>
<tr><td>EB_SysCmd[13:0]</td><td>I</td><td>Core Interface command. This bus is driven by the SOC interface controller during address and data transactions and provides information about the type of transaction.</td></tr>
</table>

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 8-30 20Kc Core Signal Descriptions

<table>
<tr><th>Pin Name</th><th>Direction</th><th>Description</th></tr>
<tr><td>EB_SysCmdP[1:0]</td><td>I</td><td>Core Interface command parity. This two-bit bus provides parity for the 14-bit EB_SysCmd bus. The EB_SysCmdP[1:0] parity bits correspond to the EB_SysCmd[13:0] command bus as follows:
<table>
<tr><th>Parity Bit</th><th>Corresponding Command Bus Bits</th></tr>
<tr><td>EB_SysCmdP[0]</td><td>EB_SysCmd[6:0]</td></tr>
<tr><td>EB_SysCmdP[1]</td><td>EB_SysCmd[13:7]</td></tr>
</table></td></tr>
<tr><td>EB_SysVld</td><td>I</td><td>Core Interface command valid. This signal is asserted whenever information on the SOC command bus is valid.</td></tr>
<tr><td>EB_SysRdCredit</td><td>I</td><td>Flow control from SOC interface for reads. This signal is asserted by the SOC interface controller to indicate that the current read operation has successfully completed. The 20Kc core responds by incrementing its read resource counter.</td></tr>
<tr><td>EB_SysWrCredit</td><td>I</td><td>Flow control from SOC interface for writes. This signal is asserted by the SOC interface controller to indicate that the current write operation has successfully completed. The 20Kc core responds by incrementing its write resource counter.</td></tr>
<tr><td>EB_SysBus32</td><td>S</td><td>Assertion of this signal at power-up causes the 20Kc core to operate in 32-bit SysAD bus mode. If this signal is deasserted at power-up, the 20Kc core operates in 64-bit SysAD mode.</td></tr>
<tr><td colspan="3">External Bus Interface: Core Interface Signals</td></tr>
<tr><td>EB_PrcAD[31:0]</td><td>O</td><td>Processor Core Bus - address/data. The processor drives address on read and write operations, and data on write operations.</td></tr>
<tr><td>EB_PrcADP[3:0]</td><td>O</td><td>Processor Core Bus - address/data parity. This bus provides byte parity for the EB_PrcAD[31:0] bus. The relationship of parity bits to data bytes is as follows:
<table>
<tr><th>Parity Bit</th><th>Corresponding Data Byte</th></tr>
<tr><td>EB_PrcADP[0]</td><td>EB_PrcAD[7:0]</td></tr>
<tr><td>EB_PrcADP[1]</td><td>EB_PrcAD[15:8]</td></tr>
<tr><td>EB_PrcADP[2]</td><td>EB_PrcAD[23:16]</td></tr>
<tr><td>EB_PrcADP[3]</td><td>EB_PrcAD[31:24]</td></tr>
</table></td></tr>
<tr><td>EB_PrcCmd[6:0]</td><td>O</td><td>Processor Core Bus - command. This bus is driven by the core along with address and data transactions and provides information about the type of transaction.</td></tr>
<tr><td>EB_PrcCmdP</td><td>O</td><td>Processor Core Bus - command parity. This one-bit bus provides a single parity for the seven-bit EB_PrcCmd bus.</td></tr>
<tr><td>EB_PrcVld</td><td>O</td><td>Processor Core Bus - command valid. This signal is asserted whenever information on the core command bus is valid.</td></tr>
<tr><td>EB_PrcCredit</td><td>O</td><td>Flow control to SOC interface. This signal is asserted by the 20Kc core to indicate that the current read or write operation has successfully completed. The SOC interface controller responds by incrementing its resource counter.</td></tr>
<tr><td colspan="3">System Interface: Clock Signals</td></tr>
<tr><td>SI_ClkIn</td><td>I</td><td>SOC differential clock input - high asserted. All inputs and outputs, except the EJTAG and Interrupt interface signals, are driven or captured on the rising edge of SI_ClkIn.</td></tr>
</table>

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 8-30 20Kc Core Signal Descriptions

<table>
<tr><th>Pin Name</th><th>Direction</th><th>Description</th></tr>
<tr><td>SI_ClkOut</td><td>O</td><td>SOC clock output - high asserted. This pin is used for silicon debugging of the PLL and is not used during normal operation.</td></tr>
<tr><td>SI_PLLBypass</td><td>S</td><td>The 20Kc core contains an internal PLL to synchronize its high frequency clock to the SOC clock inputs. This PLL is bypassed whenever this signal is asserted at power-up. This signal must be asserted when the device is in cache test mode.</td></tr>
<tr><td colspan="3">System Interface: Interrupt Interface Signals</td></tr>
<tr><td>SI_Int[5:0]</td><td>I</td><td>These signals are driven by external logic and when asserted indicate the corresponding interrupt exception to the 20Kc core.</td></tr>
<tr><td>SI_NMI</td><td>I</td><td>When sampled asserted, this signal causes the 20Kc core to take an NMI exception.</td></tr>
<tr><td colspan="3">System Interface: Initialization Interface Signals</td></tr>
<tr><td>SI_Reset</td><td>I</td><td>Reset signal. This signal must be asserted for any reset sequence: power-on, cold, or warm. It can be asserted synchronously or asynchronously for a cold reset or power-on reset, and must be synchronously initiated for a warm reset. It must always be deasserted synchronously.

This signal and the SI_ColdReset signal determine the nature of the reset sequence.</td></tr>
<tr><td>SI_ColdReset</td><td>I</td><td>Cold reset signal. This signal must be asserted during either a power-on reset or a cold reset. It can be asserted synchronously or asynchronously for a cold reset or power-on reset. It must always be deasserted synchronously.

This signal and the SI_Reset signal determine the nature of the reset sequence.</td></tr>
<tr><td>SI_CkRatio[2:0]</td><td>S</td><td>SOC clock to core clock multiplier ratio.
<table>
<tr><th>SI_CkRatio[2:0]</th><th>Core Clock to SOC Clock Ratio</th></tr>
<tr><td>000</td><td>Reserved</td></tr>
<tr><td>001</td><td>2:1</td></tr>
<tr><td>010</td><td>3:1</td></tr>
<tr><td>011</td><td>4:1</td></tr>
<tr><td>100</td><td>5:1</td></tr>
<tr><td>101</td><td>6:1</td></tr>
<tr><td>110</td><td>7:1</td></tr>
<tr><td>111</td><td>8:1</td></tr>
</table></td></tr>
<tr><td>SI_BigEndian</td><td>S</td><td>Indicates the base endianness of the 20Kc core.
<table>
<tr><th>SI_BigEndian</th><th>Base Endian Mode</th></tr>
<tr><td>0</td><td>Little Endian</td></tr>
<tr><td>1</td><td>Big Endian</td></tr>
</table></td></tr>
<tr><td>SI_VddOk</td><td>I</td><td>Indicates all power supplies are stable.</td></tr>
<tr><td>SI_PRIdOpt[7:0]</td><td>S</td><td>These signals are used as the upper eight bits of the CP0 PRId register.</td></tr>
</table>

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 8-30 20Kc Core Signal Descriptions

Pin Name	Direction	Description
System Interface: Power Management Signals		
SI_RP	O	This signal represents the state of the RP bit in the CP0 *Status* register.
SI_Sleep	O	This signal is asserted by the core whenever the WAIT instruction is executed. The assertion of this signal indicates that the core is in power-down mode.
EJTAG Interface Signals		
EJ_PartNumber[3:0]	S	The state of these pins reflects the PartNumber[3:0] field in the *Device ID* register. This value is used by the SOC vendor to program part of the EJTAG unit *Device ID* register.
EJ_TCK	I	Test Clock Input (TCK) for the EJTAG Test Access Port (TAP).
EJ_TDI	I	Test Data Input (TDI) for the EJTAG TAP.
EJ_TDO	O	Test Data Output (TDO) for the EJTAG TAP.
EJ_TMS	I	Test Mode Select Input (TMS) for the EJTAG TAP.
EJ_TRST_N	I	Test Reset Input (TRST_N) for the EJTAG TAP. At power-up the assertion of *EJ_TRST_N* causes the TAP controller to be reset.
EJ_DINT	I	Debug exception request when this signal is asserted in a 20Kc clock period after being deasserted in the previous 20Kc clock period. The request is cleared when debug mode is entered. Requests when in debug mode is ignored.
EJ_DebugM	O	This signal is asserted by the core whenever it is in debug mode.
EJ_TDOzstate	O	Output drive indication for the core pin outputting the *EJ_TDO* signal. When asserted, the core pin outputting the *EJ_TDO* must be 3-stated.
EJ_Version[3:0]	S	The state of these pins reflects the Version[3:0] field in the *Device ID* register. This value is used by the SOC vendor to program part of the EJTAG unit *Device ID* register.
Power Interface		
VddP	I	Quiet Vdd for processor core PLL.
VssP	I	Quiet Vss for processor core PLL.
Vdd	I	Core power supply.
Vss	I	Core ground.
Core Test Interface		
TS_TCmd[3:0]	S	Core test operation control signals.
TS_Sc1	I	Scan clock 1.
TS_Sc2	I	Scan clock 2.
TS_Sden	I	Scan data enable.
TS_Scanmode	I	Used to stop processor clocks during scan. Asserted when shifting in scan vectors.
TS_ScanIn[7:0]	I	Core scan input signals.
TS_ScanOut[7:0]	O	Core scan output signals.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 8-30 20Kc Core Signal Descriptions

Pin Name	Direction	Description
BIST Interface		
TS_BistInvoke	I	Starts BIST when asserted. Should be deasserted when BIST testing is over.
TS_BistFail	O	Asserted to indicate that BIST has failed. Asserted to indicate the TestFail when BIST is not active.
TS_BistDone	O	Asserted when BIST is completed. Asserted to indicate the TestDone when BIST is not active.
Feedthroughs		
FT_NorthIn[29:0]	I	Feedthrough inputs on north side of core.
FT_SouthOut[29:0]	O	Feedthrough outputs on south side of core.
FT_SouthIn[29:0]	I	Feedthrough inputs on south side of core.
FT_NorthOut[29:0]	O	Feedthrough outputs on north side of core.
Reserved		
TS_Reserved1	I	Reserved. Must connect to Vss (logic 0).
TS_Reserved2	I	Reserved. Must connect to Vss (logic 0).
TS_Reserved3	O	Reserved.
TS_Reserved4	O	Reserved.
TS_Reserved5	I	Reserved. Must connect to Vss (logic 0).
TS_Reserved6	O	Reserved.
TS_Reserved7	I	BIST algorithm select and retention control. Tied to 0 if BIST retention is not used.
TS_Reserved8	O	Reserved.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Reset and Initialization

The 20Kc processor supports the following three types of resets.

- **Power-On Reset:** The Power-On Reset sequence starts when the power supplies are turned on and completely initializes the internal state of the processor without saving any state information.
- **Cold Reset:** The Cold Reset sequence completely initializes the internal state of the processor while the power supply remains stable. It is similar to the Power-On Reset, the only difference being that the power supplies are stable throughout this sequence. The clocks are restarted during this sequence.
- **Warm Reset:** The Warm Reset sequence restarts the processor while preserving some architectural state. The clocks are not restarted during this sequence.

9.1 Processor Reset Signals

This 20Kc processor has three reset related input signals: *SI_VddOk*, *SI_ColdReset*, and *SI_Reset*.

SI_Reset: The *SI_Reset* signal must be asserted for any reset sequence. It can be asserted synchronously or asynchronously for a Cold or Power-On Reset but must be asserted synchronously for a Warm Reset. *SI_Reset* must always be deasserted synchronously with *SI_ClkIn*.

ColdReset: The *SI_ColdReset* signal must be asserted for Power-On Reset or a Cold Reset. It can be asserted synchronously or asynchronously but must be deasserted synchronously with *SI_ClkIn*.

SI_VddOk: The *SI_VddOk* signal when asserted, indicates to the processor that all the power supply inputs to the processor i.e. *Vdd*, *VddP* and *Vss* are stable and that the input clock *SI_ClkIn* is also stable. The system configuration input pins described in the following section are sampled only when *SI_VddOk* is asserted.

9.2 Processor Initialization Signals

This section describes the system configuration input pins: *SI_CkRatio[2:0]*, *EB_SysBus32*, and *SI_BigEndian*. These pins are sampled only while *SI_VddOk* is asserted. Prior to the assertion of *SI_VddOk*, default values are assumed for these pins. These pins are sampled only while *SI_VddOk* is asserted; therefore if they are changed on-the-fly, correct operational behavior does not occur.

SI_CkRatio[2:0]: This set of pins provides the multiplier to be used on the *SI_ClkIn* input in order to generate the internal clock for the processor. Table 9-1 shows the encoding used for this set of pins.

Table 9-1 Clock Multiplier Ratios

SI_CkRatio[2:0]	Multiplier Ratio
000	Reserved
001	2:1
010	3:1
011	4:1

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 9-1 Clock Multiplier Ratios

SI_CkRatio[2:0]	Multiplier Ratio
100	5:1
101	6:1
110	7:1
111	8:1

SI_BigEndian: This pin is used to configure the system to operate in Big-Endian mode. When this pin is asserted, the system operates in Big-Endian mode.

9.3 Reset Sequences

This section describes the following reset sequences:

- Section 9.3.1, "Power-On Reset Sequence"
- Section 9.3.2, "ColdReset Sequence"
- Section 9.3.3, "Warm Reset Sequence"

9.3.1 Power-On Reset Sequence

A Power-On Reset starts with the application of power supplies. The 20Kc processor has separate power supplies for core and PLL.The PLL gets the same voltage level as the core, however it requires a quiet and separate source of the same voltage.

- The *Vdd* and *Vss* pins supply the voltage required by the core.
- The PLL is fed by the *VddP* and *VssP* pins.

The sequence for a Power-On Reset is as follows:

1. The input *SI_VddOk* is held low and the input signals *SI_Reset* and *SI_ColdReset* are held high.
2. *Vdd* and *VddP* are applied to the part.
3. *SI_ClkIn* is applied concurrent with or after *Vdd*.
4. The configuration pins *SI_CkRatio[2:0]*, *SI_BigEndian*, *EB_SysBus32*, and *SI_PLLBypass* are applied concurrently with or after *Vdd*.
5. Once *Vdd*, *VddP*, and *SI_ClkIn* are stable, and the configuration pins have been stable for at least five *SI_ClkIn* cycles, *SI_VddOk* is asserted.
6. Once *SI_VddOk* is asserted the configuration input pins are sampled.
7. If the JTAG interface is used during power up, it can be activated after *SI_VddOk* is asserted.
8. *SI_ColdReset* and *SI_Reset* must be asserted for a minimum of 120 microseconds after the assertion of *SI_VddOk*. This time is allowed in order to get the PLL to lock.
9. *SI_ColdReset* and *SI_Reset* are then deasserted simultaneously. It is allowable to not deassert them simultaneously but in that case *SI_ColdReset* must be deasserted before *SI_Reset*. *SI_Reset* must not be deasserted before *SI_ColdReset*.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

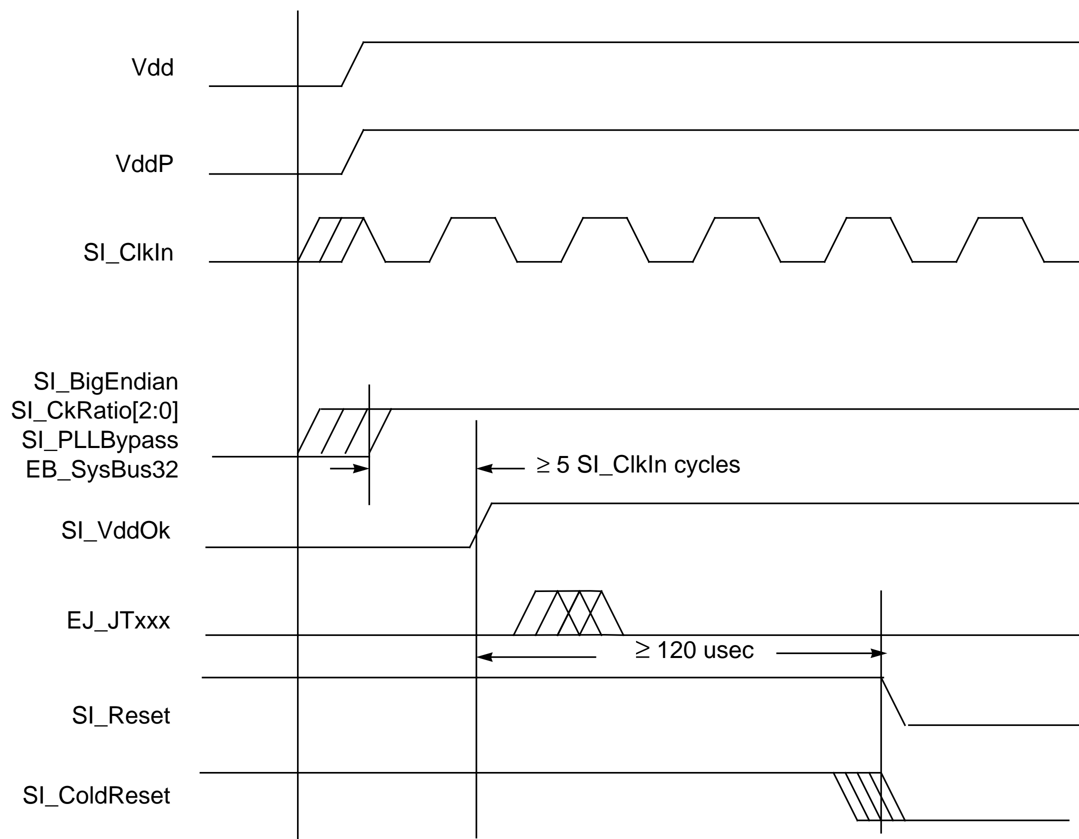


Figure 9-1 Power-On Reset Sequence

9.3.2 ColdReset Sequence

A Cold Reset (also called Hard Reset) follows almost the same sequence as a Power-On Reset with the only difference being that all the power supplies are expected to be stable coming into this sequence. A Cold Reset sequence is required whenever the state of any of the configuration input pins *SI_CkRatio[2:0]*, *SI_BigEndian*, *EB_SysBus32*, and *SI_PLLBypass* is to be changed. The clocks are restarted on a Cold Reset.

The sequence for a Cold Reset is as follows:

1. *SI_ColdReset* and *SI_Reset* are asserted.
2. *SI_VddOk* is deasserted.
3. After a minimum of five *SI_ClkIn* cycles, *SI_VddOk* is asserted.
4. Once *SI_VddOk* is asserted, system configuration input pins are sampled.
5. If the JTAG interface is used during power up, it can be activated after *SI_VddOk* is asserted.
6. *SI_ColdReset* and *SI_Reset* must be asserted for a minimum of 120 microseconds after the assertion of *SI_VddOk*. This time is allowed in order to get the PLL to lock.
7. *SI_ColdReset* and *SI_Reset* are then deasserted simultaneously. It is also allowable to deassert *SI_ColdReset* before *SI_Reset,* and a Cold Reset will be performed, but in that case, the SR (Soft Reset) bit in the Status Register (CP0 Register 12, Select 0) will be set.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

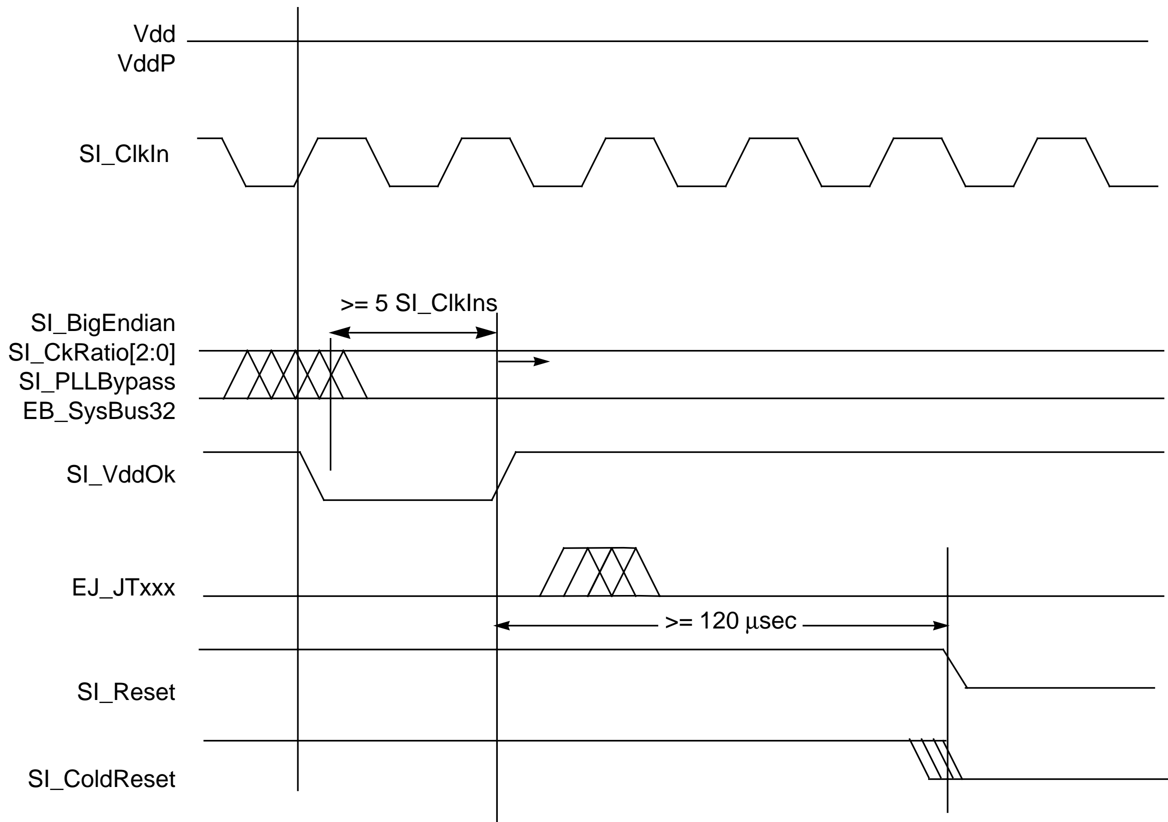


Figure 9-2 Cold Reset Sequence

9.3.3 Warm Reset Sequence

The sequence for a Warm Reset (also called Soft Reset) is as follows:

1. *SI_Reset* is asserted.
2. *SI_Reset* may be deasserted after a minimum of 10 *SI_ClkIn* cycles.

Note: During a Warm Reset sequence, none of the configuration parameters can be changed. If that is required, a Cold Reset sequence must be performed. The SR bit in the Status Register will be set following a Warm Reset.

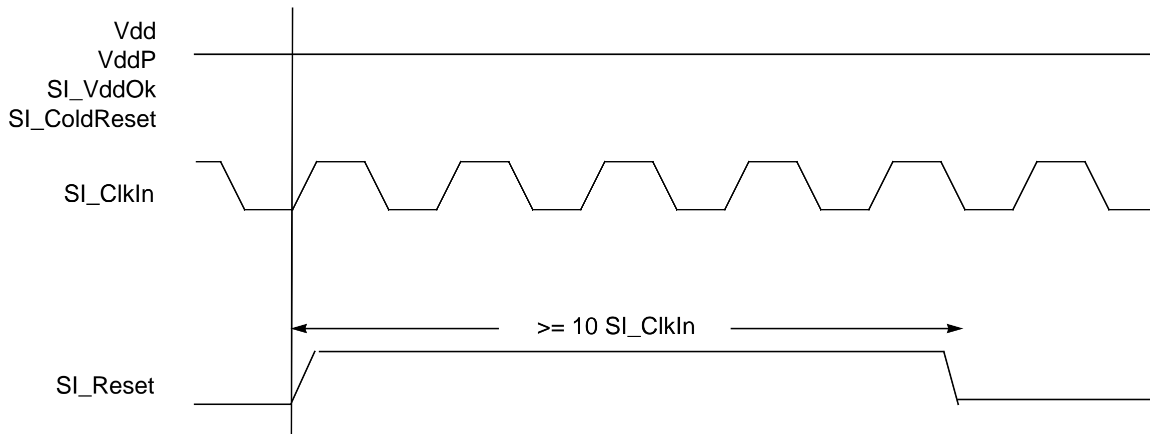


Figure 9-3 Warm Reset Sequence

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Power Management

The 20Kc core offers a number of power management features, including low-power design, active power management, and two power-down modes of operation.

This chapter contains the following sections:

- Section 10.1, "Register-Controlled Power Management"
- Section 10.2, "Instruction-Controlled Power Management"

10.1 Register-Controlled Power Management

This section describes register-controlled power management or Doze mode. The RP bit in the CP0 *Status* register provides a standard software mechanism for placing the system into a low power state. Two additional bits, EXL and ERL, support the power management function by allowing the power state to be changed based on the setting of these bits in the CP0 register if an exception or error occurs while the processor is in a low power state.

If an interrupt is taken while the device is in power down mode, that interrupt might need to be serviced depending on the needs of the application. The interrupt causes an exception which in turn causes the EXL bit to be set.

The core is placed in low-power mode by writing a logic one to the RP status bit. The internal clock generation logic reduces the core clock frequency to the system clock frequency at the next system clock rising edge. While in the low-power state, the processor maintains PLL lock. With the exception of the core clock frequency, the processor is functioning normally. The internal timer continues to run but at the lower clock frequency. Instruction fetch, dispatch, and execution continue. However, if the processor detects an exception, the clock generation logic temporarily returns the core clock to normal frequency until the exception has been cleared.

Low-power mode is exited by asserting any interrupt pin, asserting *SI_Reset*, or writing a logic zero to the RP bit. Upon detecting any one of the above conditions, the processor increases the core clock frequency to the system clock frequency multiplied by the clock ratio. This event occurs at the next system clock rising edge.

10.2 Instruction-Controlled Power Management

The second mechanism for invoking power-down mode is through execution of the WAIT instruction (Sleep mode). As in Doze mode, the internal core clock frequency is reduced to that of the system clock frequency to reduce power, but the pipeline is frozen so no further instructions are executed.

However, the internal timer still runs (at this reduced frequency), and the Bus Interface Unit (BIU) is still active to handle interrupts as well as other bus transactions, including cache interventions and invalidations. Once the core is in Sleep mode, any unmasked interrupt or reset caused by the following pins (*SI_Int[5:0], SI_NMI, SI_Reset, SI_ColdReset*, and *EJ_DINT*) causes the core to exit this state and resume normal operation.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

EJTAG Debug Support

This chapter describes the Enhanced Joint Test Action Group (EJTAG) debug features supported by the 20Kc processor and contains the following sections:

- Section 11.1, "EJTAG Components and Options"
- Section 11.2, "Register and Memory Map Overview"
- Section 11.3, "EJTAG Processor Core Extensions"
- Section 11.4, "Debug Control Register"
- Section 11.5, "Hardware Breakpoints"
- Section 11.6, "EJTAG Test Access Port"

EJTAG is a hardware/software subsystem that provides comprehensive debugging and performance tuning capabilities to MIPS microprocessors and to system-on-a-chip components having MIPS processor cores. It exploits the infrastructure provided by the IEEE 1149.1 JTAG Test Access Port (TAP) standard to provide an external interface, and extends the MIPS instruction set and privileged resource architectures to provide a standard software architecture for integrated system debugging.

For more information on EJTAG and the IEEE 1149.1 standard, refer to Ref [1] and Ref [4].

11.1 EJTAG Components and Options

EJTAG hardware support consists of several distinct components: extensions to the 20Kc processor, the EJTAG Test Access Port, the Debug Control Register, and the Hardware Breakpoint Unit. Figure 11-1 shows the relationship between these components in the 20Kc EJTAG implementation.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

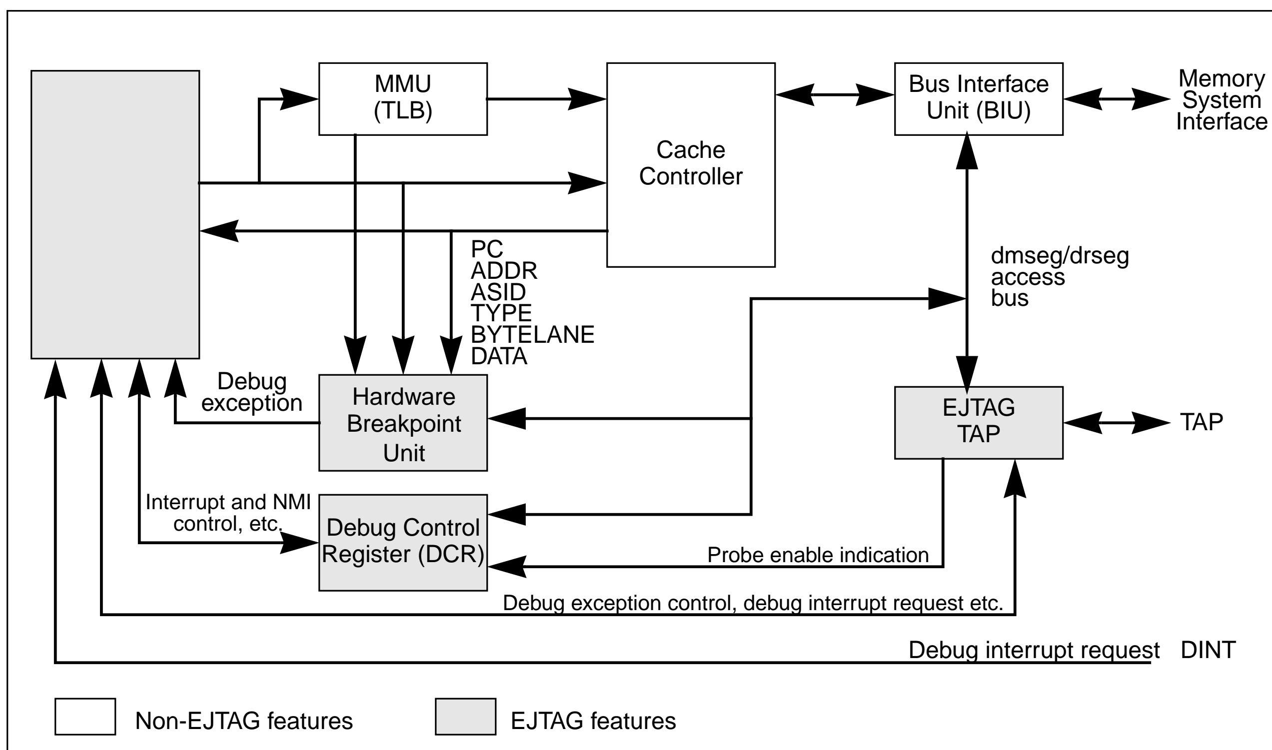


Figure 11-1 Simplified Block Diagram of EJTAG Components

11.1.1 EJTAG Extensions to the MIPS Processor Core

The 20Kc processor supports EJTAG-specific instructions, additional system coprocessor (CP0) registers, a single-step mode of execution, and vectoring to Debug Exceptions, which puts the processor in a special Debug Mode of execution, as described in Section 11.3, "EJTAG Processor Core Extensions".

11.1.2 Debug Control Register

The Debug Control Register (DCR) is a memory-mapped register that is provided as part of the processor core. It indicates the availability and status of EJTAG features. The memory-mapped region containing the DCR is available to software only in Debug Mode.

Refer to Section 11.4, "Debug Control Register" for more information on the DCR.

11.1.3 Hardware Breakpoint Unit

The hardware breakpoint unit implements memory-mapped registers that control the instruction and data hardware breakpoints. The memory-mapped region containing the hardware breakpoint registers is accessible to software only in Debug Mode.

The 20Kc core provides the following hardware breakpoint support:

- Four independent instruction hardware breakpoints
- Two independent data hardware breakpoints

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

The presence or absence of hardware breakpoint capability is indicated to debug software in the DCR. Refer to Section 11.5, "Hardware Breakpoints" for more information on hardware breakpoints.

11.1.4 EJTAG Test Access Port

The EJTAG Test Access Port (TAP) provides a standard JTAG TAP interface to the EJTAG system. It is necessary for host-based debugging and processor access to external debug memory. The presence or absence of off-board EJTAG memory is indicated to debug software via the Debug Control Register (*DCR*). Refer to Section 11.6, "EJTAG Test Access Port" for more information on the TAP.

11.2 Register and Memory Map Overview

This section summarizes the registers and special memory that are used for the EJTAG debug solution. More detailed information regarding these registers and memory locations is provided in the relevant sections.

11.2.1 Coprocessor 0 Register Overview

Table 11-1 summarizes the Coprocessor 0 (CP0) registers. These registers are accessible by the debug software executed on the processor; they provide debug control and status information. General information about the debug CP0 registers is found in Section 11.3.10, "EJTAG Coprocessor 0 Registers".

Table 11-1 Overview of Coprocessor 0 Registers for EJTAG

Register Name	Register Mnemonic	Functional Description
Debug	DEBUG	Debug indications and controls for the processor, including information about recent debug exception.
Debug Exception Program Counter	DEPC	Program counter at last debug exception or exception in Debug Mode.
Debug Exception Save	DESAVE	Scratchpad register available for the debug handler.

11.2.2 Memory-Mapped EJTAG Register Overview

The memory-mapped EJTAG registers are located in the drseg part of the debug segment (dseg). They are accessible by the debug software when the processor is executing in Debug Mode. These registers provide both miscellaneous debug control and control of hardware breakpoints. General information about the debug segment and registers is found in Section 11.3.3, "Debug Mode Address Space" and Section 11.3.3.2, "Access to drseg (EJTAG Registers) Address Range".

11.2.2.1 Debug Control Register Overview

Table 11-2 summarizes the Debug Control Register (*DCR*), which provides miscellaneous debug control.

Table 11-2 Overview of Debug Control Register as Memory-Mapped Register for EJTAG

Register Name	Register Mnemonic	Functional Description	Reference
Debug Control Register	DCR	Indicates availability of instruction and data value breakpoints and controls the enabling of interrupts and NMIs in Non-Debug Mode.	See Section 11.4, "Debug Control Register"

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

11.2.2.2 Instruction Hardware Breakpoint Register Overview

Table 11-3 summarizes the instruction hardware breakpoint registers, which are controlled through a number of memory-mapped registers. Certain registers are provided for each implemented instruction hardware breakpoint, as indicated with an “n”. General information about the instruction hardware breakpoint registers is found in Section 11.5.2, "Overview of Instruction and Data Breakpoint Registers".

Table 11-3 Overview of Instruction Hardware Breakpoint Registers

Register Name	Register Mnemonic	Functional Description	Reference
Instruction Breakpoint Status	*IBS*	Indicates number of instruction hardware breakpoints and status on a previous match.	See Section 11.5.6.1, "Instruction Breakpoint Status (IBS) Register"
Instruction Breakpoint Address n	*IBAn*	Address to compare for breakpoint n.	See Section 11.5.6.2, "Instruction Breakpoint Address n (IBAn) Register"
Instruction Breakpoint Address Mask n	*IBMn*	Mask for address comparison for breakpoint n.	See Section 11.5.6.3, "Instruction Breakpoint Address Mask n (IBMn) Register"
Instruction Breakpoint ASID n	*IBASIDn*	ASID value to compare for breakpoint n.	See Section 11.5.6.4, "Instruction Breakpoint ASID n (IBASIDn) Register"
Instruction Breakpoint Control n	*IBCn*	Control of breakpoint n comparison of ASID and generated event on match.	See Section 11.5.6.5, "Instruction Breakpoint Control n (IBCn) Register"

11.2.3 Data Hardware Breakpoint Register Overview

Table 11-4 summarizes the data hardware breakpoints, which are controlled through a number of memory-mapped registers. Certain registers are provided for each implemented data hardware breakpoint, as indicated with an “n”. General information about the data hardware breakpoint registers is found in Section 11.5.7, "Data Breakpoint Registers".

Table 11-4 Overview of Data Hardware Breakpoint Registers

Register Name	Register Mnemonic	Functional Description	Reference
Data Breakpoint Status	*DBS*	Indicates number of data hardware breakpoints and status on a previous match.	See Section 11.5.7.1, "Data Breakpoint Status (DBS) Register"
Data Breakpoint Address n	*DBAn*	Address to compare for breakpoint n.	See Section 11.5.7.2, "Data Breakpoint Address n (DBAn) Register"
Data Breakpoint Address Mask n	*DBMn*	Mask for address comparison for breakpoint n.	See Section 11.5.7.3, "Data Breakpoint Address Mask n (DBMn) Register"
Data Breakpoint ASID n	*DBASIDn*	ASID value to compare for breakpoint n.	See Section 11.5.7.4, "Data Breakpoint ASID n (DBASIDn) Register"
Data Breakpoint Control n	*DBCn*	Control of breakpoint n match on load/store, data bytes, access to data bytes, comparison of ASID, and generated event on match.	See Section 11.5.7.5, "Data Breakpoint Control n (DBCn) Register"

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 11-4 Overview of Data Hardware Breakpoint Registers (Continued)

Register Name	Register Mnemonic	Functional Description	Reference
Data Breakpoint Value n	*DBVn*	Data value to match for breakpoint n.	See Section 11.5.7.6, "Data Breakpoint Value n (DBVn) Register"

11.2.3.1 Memory-Mapped EJTAG Memory Overview

The memory-mapped EJTAG memory is located in the debug segment (dseg). It is accessible by the debug software when the processor is executing in Debug Mode. All accesses to this segment are handled by the EJTAG probe through the Test Access Port (TAP), whereby the processor has access to dedicated debug memory even if no debug memory was originally located in the system. General information about the debug segment and memory is found in Section 11.3.3, "Debug Mode Address Space" and Section 11.3.3.1, "Access to dmseg (EJTAG Memory) Address Range".

11.2.3.2 EJTAG Test Access Port Registers

The probe accesses EJTAG Test Access Port (TAP) registers (shown in Table 11-5) through the TAP; they are not accessible from the processor. These registers allow control of the target processor through the TAP. General information about the TAP registers is found in Section 11.6.5, "Data Registers".

Table 11-5 Overview of Test Access Port Registers

Register Name	Register Mnemonic	Functional Description	Reference
Instruction	(none)	Controls selection of the accessed data registers and controls setting and clearing of the EJTAGBOOT indication.	See Section 11.6.4, "Instruction Register and Special Instructions"
Device ID	(none)	Identifies device and accessed processor in the device.	See Section 11.6.5.1, "Device Identification (ID) Register (TAP Instruction IDCODE)"
Implementation	(none)	Identifies main debug features implemented and accessible through the TAP.	See Section 11.6.5.2, "Implementation Register (TAP Instruction IMPCODE)"
Data	(none)	Data register for processor accesses used to support the EJTAG memory.	See Section 11.6.5.3, "Data Register (TAP Instruction DATA or ALL)"
Address	(none)	Address register for processor access used to support the EJTAG memory.	See Section 11.6.5.4, "Address Register (TAP Instruction ADDRESS or ALL)"
EJTAG Control	ECR	Control register for most EJTAG features used through the TAP.	See Section 11.6.5.5, "EJTAG Control Register (ECR) (TAP Instruction CONTROL or ALL)"
Fastdata	(none)	Provides efficient block transfers between dmseg and target memory.	See Section 11.6.5.8, "Fastdata Register (TAP Instruction FASTDATA)"
Bypass	(none)	Provides a one-bit shift path through the TAP.	See Section 11.6.5.9, "Bypass Register (TAP Instruction BYPASS, EJTAGBOOT, NORMALBOOT or Unused)"

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

11.2.4 Register Field Notations

Usual field notation is used in the registers, but Table 11-6 defines the addition of R/W0 and R/W1.

Table 11-6 Register Field Read/Write Notations

Read/Write Notation	Hardware Interpretation	Software Interpretation
R/W0	Similar to the R/W interpretation, except a software write of value 1 to this bit is ignored.	
R/W1	Similar to the R/W interpretation, except a software write of value 0 to this bit is ignored.	

11.3 EJTAG Processor Core Extensions

This section gives an overview of the processor's EJTAG debug behavior. Some of these features are described elsewhere in this manual, in which case cross references are used. Some information is also duplicated in order to give a more complete view of the debug features. The extensions for EJTAG provide the following major features:

- Debug Mode, associated exceptions and dedicated debug vector
- Instruction set extensions: SDBBP (Software Debug Breakpoint) and DERET (Debug Exception Return)
- CP0 registers: *Debug*, *DEPC*, and *DESAVE*
- Memory-mapped debug segment (dseg)
- Interrupt and NMI control from Debug Control Register (*DCR*)
- Single step
- Debug interrupt request signal

11.3.1 Debug Mode Execution

Debug Mode is entered only through a debug exception. It is exited as a result of either execution of a DERET instruction or application of a reset or soft reset.

When the processor is operating in Debug Mode it has access to the same resources, instructions, and CP0 registers as in Kernel Mode. Restrictions on Kernel Mode access (non-zero coprocessor references, access to extended addressing controlled by UX, SX, KX, etc.) apply equally to Debug Mode, but Debug Mode provides some additional capabilities as described in this chapter.

Other processor modes (Kernel Mode, Supervisor Mode, User Mode) are collectively considered as Non-Debug Mode. Debug software can determine if the processor is in Non-Debug Mode or Debug Mode through the DM bit in the Debug register, or the *EJ_DebugM* port as described in Chapter 8, "Bus Interface Unit."

11.3.2 Debug Mode Instruction Set

The full native ISA of the processor is accessible in Debug Mode. Coprocessor loads and stores to the dseg memory segment are not supported. The operation of the processor is UNDEFINED if a coprocessor load or store to dseg is executed in Debug Mode. Refer to Section 11.3.3, "Debug Mode Address Space" for more information on the dseg address space.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

11.3.3 Debug Mode Address Space

Debug Mode access to unmapped address space is identical to that of Kernel Mode. Mapped areas are accessible as in Kernel Mode, but only if a valid translation is possible immediately by the MMU. If a valid translation is not possible, a TLB exception is generated, but the TLB fill sequence is not started.

In addition, an optional unmapped debug segment dseg (EJTAG area) appears in the address range 0xFFFF FFFF FF20 0000 to 0xFFFF FFFF FF3F FFFF. The dseg thereby appears in the kseg3 part of the compatibility segment, and access to kseg3 is possible with dseg provided as described in Section 11.3.3.1, "Access to dmseg (EJTAG Memory) Address Range" on page 188 and Section 11.3.3.2, "Access to drseg (EJTAG Registers) Address Range" on page 188. Access to dseg is uncached and unmapped.

The presence of the dseg is indicated by the $Debug_{NoDCR}$, and debug software must check the $Debug_{NoDCR}$ bit before trying to access the dseg segment.

Table 11-7 Presence of dseg Segment

NoDCR bit in Debug Register	dseg Presence
0	dseg is present
1	No dseg is present

Conditions for access to dseg are described in Section 11.3.3.1, "Access to dmseg (EJTAG Memory) Address Range" and Section 11.3.3.2, "Access to drseg (EJTAG Registers) Address Range". Refer to Ref [1] for more information on the layout of the virtual address space.

A memory access that causes an exception if tried from Kernel Mode causes re-entry into Debug Mode by an exception while in Debug Mode (see Section 11.3.6.3, "Debug Mode Exception Processing" on page 195). These accesses include those that cause TLB exceptions. Such accesses therefore are not handled by the usual memory management routines. Coprocessor loads and stores to dseg are not allowed, as described in Section 11.3.2, "Debug Mode Instruction Set" on page 186. Updating and handling of cached areas is the same as that in Kernel Mode.

The dseg segment is subdivided into the dmseg (EJTAG memory) segment from 0xFFFF FFFF FF20 0000 to 0xFFFF FFFF FF2F FFFF and the drseg (EJTAG registers) segment from 0xFFFF FFFF FF30 0000 to 0xFFFF FFFF FF3F FFFF. The dmseg segment is used when the probe services the memory segment. The drseg segment is used when the memory-mapped debug registers are accessed. Table 11-8 shows the subdivision and attributes for the segments.

Table 11-8 Physical Address and Cache Attribute for dseg, dmseg, and drseg

Segment Name	Subsegment Name	Virtual Address	Reference Address	Cache Attribute
dseg	dmseg	0xFFFF FFFF FF20 0000 to 0xFFFF FFFF FF2F FFFF	Because the dseg address range is serviced exclusively by the EJTAG features, there are no physical addresses per se. Rather the lower 21 bits of the virtual address are used to select the appropriate reference in either EJTAG memory or registers. References are not mapped through the TLB, nor do the accesses appear on the external system memory interface.	Uncached
	drseg	0xFFFF FFFF FF30 0000 to 0xFFFF FFFF FF3F FFFF		

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

There are no timing requirements with respect to transactions to dmseg, which the probe services. Therefore a system watchdog must be disabled during dseg transactions, so accesses can take any amount of time without being terminated.

11.3.3.1 Access to dmseg (EJTAG Memory) Address Range

Table 11-9 shows the behavior of processor accesses in Debug Mode to the dmseg address range from 0xFFFF FFFF FF20 0000 to 0xFFFF FFFF FF2F FFFF.

Table 11-9 Access to dmseg Address Range

LSNM bit in Debug Register	ProbEn bit in DCR register	Transaction	Access
1	x	Load/Store	Kernel Mode address space
x	1	Fetch	dmseg
0	1	Load/Store	
x	0	Fetch	See comments below regarding behavior when ProbEn is 0
0	0	Load/Store	
'x' denotes don't care			

From Table 11-9, the cases with ProbEn equal to 0 for dmseg accesses are not expected to happen. Debug software must read the state of the ProbEn bit in the *DCR* register before attempting to reference dmseg. If debug software references dmseg when ProbEn is 0, the reference hangs until it is satisfied by the probe. The probe must not assume that a reference to dmseg never occurs when the ProbEn bit is dynamically cleared to 0, because there is an inherent race between the debug software sampling the ProbEn bit as 1 and the probe clearing it to 0.

The protocol for accesses to dmseg does not allow a transaction to be aborted once started, except by a reset or soft reset. Transactions of all sizes are allowed to dmseg.

11.3.3.2 Access to drseg (EJTAG Registers) Address Range

Table 11-10 shows the behavior of processor accesses in Debug Mode to the drseg address range from 0xFFFF FFFF FF30 0000 to 0xFFFF FFFF FF3F FFFF.

Table 11-10 Access to drseg Address Range

LSNM bit in Debug Register	Transaction	Access
1	Load/Store	Kernel Mode address space
0		drseg (see comments below the table)
x	Fetch	Operation of the processor is UNDEFINED at fetch
'x' denotes don't care		

Instruction fetches from drseg are not allowed. The operation of the processor is UNDEFINED if the processor tries to fetch from drseg.

When the NoDCR bit is 0 in the *Debug* register it indicates that the processor is allowed to access the entire drseg segment, therefore all transactions to drseg are responded to.

The *DCR* register, at offset 0x0000 in drseg, is always available. Debug software is expected to read the *DCR* register to determine if the hardware breakpoint registers exist in drseg. The value returned in response to a read of any

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

unimplemented memory-mapped register is UNPREDICTABLE, and writes are ignored to any unimplemented register in drseg.

Only doubleword size transactions are allowed for drseg. Operation of the processor is UNDEFINED for other transaction sizes.

11.3.4 Debug Mode Handling of Processor Resources

Unless otherwise specified, the processor resources in Debug Mode are handled identically to those in Kernel Mode. Some identical cases are described in the following subsections for emphasis. In addition, see the following related sections for more information:

- Section 11.3, "EJTAG Processor Core Extensions" covering exception handling in Debug Mode.
- Section 11.3.7, "Interrupts and NMIs" for handling in both Debug and Non-Debug Modes.
- Section 11.3.8, "Reset and Soft Reset of the Processor" for handling in both Debug and Non-Debug Modes.

11.3.4.1 Coprocessors

A Debug Mode Coprocessor Unusable exception is raised under the same conditions as for a Coprocessor Unusable exception in Kernel Mode (see Section 11.3.5, "Debug Exceptions"). Therefore Debug Mode software cannot reference Coprocessors 1 through 3 without first setting the respective enable in the *Status* register.

11.3.4.2 Random Register

The *Random* register is not frozen in Debug Mode.

11.3.4.3 Count Register

The *Count* register is not frozen in Debug Mode.

11.3.4.4 WatchLo/WatchHi Registers

The *WatchLo*/*WatchHi* registers (CP0 Registers 18 and 19) are prohibited from matching any instruction executed in Debug Mode.

11.3.4.5 LL / SC Instruction Pair

A DERET instruction does not clear the LLbit. Neither does the occurrence of a debug exception. The value of the LLbit is not directly visible by software. For more information on the DERET instruction, refer to Ref [1].

11.3.5 Debug Exceptions

This section describes issues related to debug exceptions. Debug exceptions force the processor from Non-Debug Mode into Debug Mode.

Exceptions can occur in Debug Mode, and these are denoted as debug mode exceptions. These exceptions are handled differently from exceptions that occur in Non-Debug Mode, and are described in Section 11.3.5.3, "General Debug Exception Processing".

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

11.3.5.1 Debug Exception Priorities

Refer to Table 5-1 on page 60 for a detailed description of exception priorities.

11.3.5.2 Debug Exception Vector Location

The same debug exception vector location is used for all debug exceptions. The ProbTrap bit in the EJTAG Control Register (*ECR*) in the Test Access Port (TAP) determines the vector location.

Table 11-11 Debug Exception Vector Locations

ProbTrap Bit in ECR Register	Debug Exception Vector Address
0	0xFFFF FFFF BFC0 0480
1	0xFFFF FFFF FF20 0200 in dmseg

11.3.5.3 General Debug Exception Processing

All debug exceptions have the same basic processing flow:

- The *DEPC* register is loaded with the PC at which execution can be restarted, and the DBD bit is set to indicate whether the last debug exception occurred in a branch delay slot. The value loaded into the *DEPC* register is either the current PC (if the instruction is not in the delay slot of a branch) or the PC of the branch or jump (if the instruction is in the delay slot of a branch or jump).
- The DSS, DBp, DDBL, DDBS, DIB, DINT, DDBLImpr, and DDBSImpr bits in the *Debug* register are updated appropriately depending on the debug exception.
- DExcCode field in the *Debug* register is undefined.
- Halt and Doze bits in the *Debug* register are updated appropriately.
- IEXI bit is set to inhibit imprecise exceptions at the start of the debug handler.
- DM bit in the *Debug* register is set to 1.
- The processor begins fetching instructions from the debug exception vector.

The value loaded into the *DEPC* register represents the restart address from the debug exception and does not need to be modified by the debug exception handler software. Debug software need only look at the DBD bit in the *Debug* register if it wishes to identify the address of the instruction that actually caused a precise debug exception.

The DSS, DBp, DDBL, DDBS, DIB, DINT, DDBLImpr, and DDBSImpr bits in the *Debug* register indicate the occurrence of distinct debug exceptions, except when a Debug Data Break Load/Store Imprecise exception occurs (see section Section 11.5.4.3, "Imprecise Debug Exception Caused by Data Breakpoint"). Note that occurrence of an exception while in Debug Mode clears these bits. The handler can thereby determine whether a debug exception or an exception in Debug Mode occurred.

No other CP0 registers or fields are changed due to the debug exception, thus no additional state is saved.

The overall exception processing flow is shown below:

Operation:

```
if (InstructionInBranchDelaySlot) then
        DEPC ← BranchInstructionPC
        Debug_DBD ← 1
    else
        DEPC ← PC
```


Copyright © 2002 MIPS Technologies Inc. All rights reserved.

```
    Debug_DBD ← 0
endif
Debug_DSS, DBp, DDBL, DDBS, DIB, DINT, DDBLImpr and DDBSImpr ← DebugExceptionType
Debug_DExcCode ← UNPREDICTABLE
Debug_Halt ← Halt StatusAt Debug Exception
Debug_Doze ← Doze StatusAt Debug Exception
Debug_IEXI ← 1
Debug_DM ← 1
if ECR_ProbTrap = 1 then
    PC ← 0xFFFF FFFF FF20 0200
else
    PC ← 0xFFFF FFFF BFC0 0480
endif
```

11.3.5.4 Debug Breakpoint Exception

A Debug Breakpoint exception occurs when an SDBBP instruction is executed. The contents of the *DEPC* register and the DBD bit in the *Debug* register indicate that the SDBBP instruction caused the debug exception. For more information on the SDBBP instruction, refer to Ref [1].

Debug Register Debug Status Bit Set

DBp

Additional State Saved

None

Entry Vector Used

Debug exception vector

11.3.5.5 Debug Instruction Break Exception

A Debug Instruction Break exception occurs when an instruction hardware breakpoint matches an executed instruction. The DEPC register and DBD bit in the Debug register indicate the instruction that caused the instruction hardware breakpoint match.

Debug Register Debug Status Bit Set

DIB

Additional State Saved

None

Entry Vector Used

Debug exception vector

11.3.5.6 Debug Data Break Load/Store Precise Exception on Address

A Debug Data Break Load/Store exception occurs when a data hardware breakpoint matches the load/store address of an executed load/store instruction. The *DEPC* register and DBD bit in the *Debug* register indicate the load/store instruction that caused the data hardware breakpoint to match, as this is a precise debug exception. The load/store instruction that caused the debug exception has not completed (it has not updated the destination register or memory location), and the instruction therefore is executed on return from the debug handler.

Debug Register Debug Status Bit Set

DDBL for a load instruction or DDBS for a store instruction

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Additional State Saved

None

Entry Vector Used

Debug exception vector

11.3.5.7 Debug Data Break Load/Store Imprecise Exception on Data

A Debug Data Break Load/Store Imprecise exception occurs when a data hardware breakpoint matches a load/store access of an executed load/store instruction. The *DEPC* register and the DBD bit in the *Debug* register indicate an instruction later in the execution flow instead of the load/store instruction that caused the data hardware breakpoint to match. The DDBLImpr/DDBSImpr bits in the *Debug* register indicate that a Debug Data Break Load/Store Imprecise exception occurred. The instruction that caused the Debug Data Break Load/Store Imprecise exception will have completed. It updates its destination register, and is not executed on return from the debug handler.

Imprecise debug exceptions from data hardware breakpoints are indicated together with another debug exception if the load/store transaction that made the data hardware breakpoint match did not complete until after another debug exception occurred. In this case, the other debug exception was the cause of entering Debug Mode, so the *DEPC* register and the DBD bit in *Debug* register point to this instruction. DDBLImpr/DDBSImpr are set concurrently with the status bit for that debug exception.

Debug Register Debug Status Bit Set

DDBLImpr for a load instruction or DDBSImpr for a store instruction

Additional State Saved

None

Entry Vector Used

Debug exception vector

11.3.5.8 Debug Single Step Exception

When single-step mode is enabled, a Debug Single Step exception occurs each time the processor has taken a single execution step in Non-Debug Mode. An execution step is a single instruction, or an instruction pair consisting of a jump/branch instruction and the instruction in the associated delay slot. The SSt bit in the *Debug* register enables Debug Single Step exceptions. They are disabled on the first execution step after a DERET.

The *DEPC* register points to the instruction on which the Debug Single Step exception occurred, which is also the next instruction to execute when returning from Debug Mode. The debug software can examine the system state before this instruction is executed. Thus the *DEPC* does not point to the instruction(s) that have just executed in the execution step, but points to the instruction following the execution step. The Debug Single Step exception never occurs on an instruction in a jump/branch delay slot, because the jump/branch and the instruction in the delay slot are always executed in one execution step; thus the DBD bit in the *Debug* register is never set for a Debug Single Step exception.

Exceptions occurring on the instruction(s) in the execution step are taken regardless, so if a non-debug exception occurs (other than reset or soft reset), a Debug Single Step exception is taken on the first instruction in the non-debug exception handler. The non-debug exception occurs during the execution step, and the instruction(s) that received a non-debug exception counts as the execution step.

Debug exceptions are unaffected by single-step mode; returning to an SDBBP instruction with single step enabled causes a Debug Breakpoint exception with the *DEPC* register pointing to the SDBBP instruction.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

To ensure proper functionality of single-step execution, the Debug Single Step exception has priority over all exceptions, except resets and soft resets.

A Debug Single Step exception is only possible when the NoSSt bit in the *Debug* register is 0.

Debug Register Debug Status Bit Set

DSS

Additional State Saved

None

Entry Vector Used

Debug exception vector

11.3.5.9 Debug Interrupt Exception

The Debug Interrupt exception is an asynchronous debug exception that is taken as soon as possible, but with no specific relation to the executed instructions. The *DEPC* register and the DBD bit in the *Debug* register reference the instruction at which execution can be resumed after Debug Interrupt exception service.

Debug interrupt requests are ignored when the processor is in Debug Mode, and pending requests are cleared when the processor takes any debug exception, including debug exceptions other than Debug Interrupt exceptions.

A debug interrupt restarts the pipeline if the pipeline was stopped by a WAIT instruction, or it restarts the processor clock if the clock was stopped due to a low-power mode.

Debug Register Debug Status Bit Set

DINT

Additional State Saved

None

Entry Vector Used

Debug exception vector

The following sources can cause Debug Interrupt exceptions:

- The *EJ_DINT* signal from the probe can request a debug interrupt on a low (0) to high (1) transition. The DINTsup bit in the *Implementation* register in the Test Access Port (TAP) indicates that the *EJ_DINT* signal from the probe to the target processor is implemented.
- The EjtagBrk bit in the *EJTAG Control* register requests a Debug Interrupt exception when set (see Section 11.6.5.5, "EJTAG Control Register (ECR) (TAP Instruction CONTROL or ALL)"). This provides similar DINT functionality from the probe, but with higher latency.
- The EJTAGBOOT feature allows a debug interrupt to be requested immediately after a reset or soft reset has occurred (see Section 11.6.4.2, "EJTAGBOOT and NORMALBOOT Instructions").

11.3.6 Debug Mode Exceptions

The handling of exceptions generated in Debug Mode, other than reset and soft reset, differs from those exceptions generated in Non-Debug Mode in that only the *Debug* and *DEPC* registers are updated. All other CP0 registers are unchanged by an exception taken in Debug Mode. The exception vector is equal to the debug exception vector (see section Section 11.3.5.2, "Debug Exception Vector Location"), and the processor stays in Debug Mode.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Reset and soft reset are handled as when occurring in Non-Debug Mode (see section Section 11.3.8, "Reset and Soft Reset of the Processor").

11.3.6.1 Exceptions Taken in Debug Mode

Only some Non-Debug Mode exception events cause exceptions while in Debug Mode. Remaining events are blocked. Exceptions occurring in Debug Mode have the same relative priorities as the Non-Debug Mode exceptions for the same exception event. These exceptions are called Debug Mode <*Non-Debug Mode exception name*>. For example, a Debug Mode Breakpoint exception is caused by execution of a BREAK instruction in Debug Mode, and a Debug Mode Address Error exception is caused by an address error due to an instruction executed in Debug Mode.

Table 11-12 lists all the Debug Mode exceptions with their corresponding non-debug exception event names, priorities, and handling.

Table 11-12 Exception Handling in Debug Mode

Priority	Event in Debug Mode	Debug Mode Handling
Highest	Reset	Reset and soft reset handled as for Non-Debug Mode, see Section 11.3.8, "Reset and Soft Reset of the Processor"
	Soft Reset	
	Debug Single Step	Blocked
	Debug Interrupt	
	Debug Data Break Load/Store Imprecise	
	NMI	
	Machine Check	Re-enter Debug Mode
	Interrupt	Blocked
	Deferred Watch	
	Debug Instruction Break, DIB	
	Watch on instruction fetch	
	Address error on instruction fetch	Re-enter Debug Mode
	TLB refill on instruction fetch	
	TLB Invalid on instruction fetch	
	Cache error on instruction fetch	
	Bus error on instruction fetch	
	Debug Breakpoint; execution of SDBBP instruction	Re-enter Debug Mode as for execution of the BREAK instruction

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 11-12 Exception Handling in Debug Mode

<table>
<tr><th>Priority</th><th>Event in Debug Mode</th><th>Debug Mode Handling</th></tr>
<tr><td rowspan="9">Lowest</td><td>Other execution-based exceptions</td><td>Re-enter Debug Mode</td></tr>
<tr><td>Debug Data Break Load/Store address match only (precise debug data break).</td><td rowspan="2">Blocked</td></tr>
<tr><td>Watch on data access</td></tr>
<tr><td>Address error on data access</td><td rowspan="6">Re-enter Debug Mode</td></tr>
<tr><td>TLB Refill on data access</td></tr>
<tr><td>TLB Invalid on data access</td></tr>
<tr><td>TLB Modified on data access</td></tr>
<tr><td>Cache error on data access</td></tr>
<tr><td>Bus error on data access</td></tr>
</table>

Exceptions that are blocked in Debug Mode are simply ignored, not causing updates in any state.

Handling of the exceptions causing Debug Mode re-entry are described below.

11.3.6.2 Exceptions on Imprecise Errors

Exceptions on imprecise errors are possible in Debug Mode. A bus error on an instruction fetch or data access or a cache error might occur.

The IEXI bit in the *Debug* register blocks imprecise error exceptions on entry or re-entry into Debug Mode. This bit can be cleared by the debug exception handler once sufficient context has been saved to allow a safe re-entry into Debug Mode and the debug handler. The IEXI bit is cleared by execution of the DERET instruction due to an imprecise exception.

Pending exceptions due to instruction fetch bus errors, data access bus errors, or cache errors are indicated and controlled by the IBusEP, DBusEP, and CacheEP bits in the *Debug* register, respectively.

Those bits required to indicate the possible imprecise errors in an implementation are implemented as R/W, otherwise they are read only.

11.3.6.3 Debug Mode Exception Processing

All exceptions that are allowed in Debug Mode (except for reset and soft reset) have the same basic processing flow:

- The *DEPC* register is loaded with the PC at which execution is restarted and the DBD bit is set appropriately in the Debug register. The value loaded into the *DEPC* register is either the current PC (if the instruction is not in the delay slot of a branch or jump) or the PC of the branch or jump if the instruction is in the delay slot of a branch or jump).
- The DSS, DBp, DDBL, DDBS, DIB, DINT, DDBLImpr, and DDBSImpr bits in the *Debug* register are all cleared to differentiate from debug exceptions where at least one of the bits are set.
- The DExcCode field in the *Debug* register is updated to indicate the type of exception that occurred.
- The Halt and Doze bits in the *Debug* register are UNPREDICTABLE.
- The IEXI bit is set to inhibit imprecise exceptions at the start of the debug handler.
- The DM bit in the *Debug* register is unchanged, leaving the processor in Debug Mode.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

- The processor is started at the debug exception vector, specified in Section 11.3.5.2, "Debug Exception Vector Location".

The value loaded into the *DEPC* register represents the restart address for the exception; typically debug software does not need to modify this value at the location of the debug exception. Debug software need not look at the DBD bit in the *Debug* register unless it wishes to identify the address of the instruction that actually caused the exception in Debug Mode.

It is the responsibility of the debug handler to save the contents of the *Debug*, *DEPC*, and *DESAVE* registers before nested entries into the handler at the debug exception vector can occur. The handler returns to the debug exception handler by a jump instruction, not a DERET, in order to keep the processor in Debug Mode.

The cause of the exception in Debug Mode is indicated through the DExcCode field in the *Debug* register, and the same codes are used for the exceptions as those for the ExcCode field in the *Cause* register when the exceptions with the same names occur in Non-Debug Mode, with addition of the code 30 (decimal) with the mnemonic CacheErr for cache errors.

No other CP0 registers or fields are changed due to the exception in Debug Mode.

The overall processing flow for exceptions in Debug Mode is shown below:

Operation:

```
if (InstructionInBranchDelaySlot) then
    DEPC ← Branch Instruction PC
    Debug_DBD ← 1
else
    DEPC ← PC
    Debug_DBD ← 0
endif
Debug_DSS, DBp, DDBL, DDBS, DIB, DINT, DDBLImpr and DDBSImpr ← 0
Debug_DExcCode ← Debug Exception Type
Debug_Halt ← UNPREDICTABLE
Debug_Doze ← UNPREDICTABLE
Debug_IEXI ← 1
if ECR_ProbTrap = 1 then
    PC ← 0xFFFF FFFF FF20 0200
else
    PC ← 0xFFFF FFFF BFC0 0480
endif
```

11.3.7 Interrupts and NMIs

Interrupts and NMIs are handled as described in the following subsections.

11.3.7.1 Interrupts

Interrupts are requested through either asserted external hardware signals or internal software-controllable bits. Interrupt exceptions are disabled when any of the following conditions are true:

- The processor is operating in Debug Mode
- The Interrupt Enable (IntE) bit in the Debug Control Register (*DCR*) is cleared (see Section 11.4, "Debug Control Register")
- Another mechanism disables the interrupt exception

A pending interrupt is indicated through the Cause register, even if Interrupt exceptions are disabled.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

11.3.7.2 NMIs

An NMI is requested on the asserting edge of the NMI signal to the processor, and an internal indicator holds the NMI request until the NMI exception is actually taken.

NMI exceptions are disabled when either of the following is true:

- The Processor is operating in Debug Mode
- The NMI Enable (NMIE) bit in the Debug Control Register (*DCR*) is cleared, see Section 11.4, "Debug Control Register"

If an asserting edge on the NMI signal to the core is detected while NMI exceptions are disabled, then the NMI request is held pending and is deferred until NMI exceptions are no longer disabled. Then the NMI is taken.

A pending NMI is indicated in the NMIpend bit in the *DCR* even if NMI exceptions are disabled.

11.3.8 Reset and Soft Reset of the Processor

This section covers the handling of issues with respect to resets and soft resets. For EJTAG features, there is no difference between a reset and a soft reset occurring to the processor; they behave identically in both Debug Mode and Non-Debug Mode.

11.3.8.1 EJTAGBOOT Feature

The EJTAGBOOT feature allows a debug interrupt to be requested immediately after a reset or soft reset has occurred. The debug handler is executed at reset or soft reset through a Debug Interrupt exception, instead of through execution of the usual reset handler.

When EJTAGBOOT is indicated at reset or soft reset, it is possible to take a Debug Interrupt exception and execute the debug handler from the probe even if no instructions can be fetched from the reset handler. Control of EJTAGBOOT is described in Section 11.6.4.2, "EJTAGBOOT and NORMALBOOT Instructions".

11.3.8.2 Reset Occurred Indication through Test Access Port

The Rocc bit in the EJTAG Control register is set at both reset and soft reset in order to indicate the event to the probe. Refer to Section 11.6.5.5, "EJTAG Control Register (ECR) (TAP Instruction CONTROL or ALL)" for more information on the EJTAG Control Register.

11.3.8.3 Reset of Other Debug Features

The operation of processor resets and soft resets also apply to resets of the following:

- Debug Control Register (DCR), see Section 11.4, "Debug Control Register".
- Hardware Breakpoint, see Section 11.5, "Hardware Breakpoints".
- Test Access Port (TAP) EJTAG Control Register, see Section 11.6, "EJTAG Test Access Port".

11.3.9 EJTAG Instructions

The SDBBP and DERET instructions are added to the processor's instruction set as part of the required EJTAG features. Refer to Ref [1] for more information on the instructions.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

11.3.10 EJTAG Coprocessor 0 Registers

The Coprocessor 0 registers for EJTAG are shown in Table 11-13 with reference to further descriptions.

Table 11-13 Coprocessor 0 Registers for EJTAG

Register Number	Sel	Register Name	Function	Reference
23	0	*Debug*	Debug indications and controls for the processor	Refer to Section 6.1, "CP0 Register Summary" for more information on this register.
24	0	*DEPC*	Program counter at last debug exception or exception in Debug Mode	Refer to Section 6.1, "CP0 Register Summary" for more information on this register.
31	0	*DESAVE*	Debug exception save register	Refer to Section 6.1, "CP0 Register Summary" for more information on this register.

11.4 Debug Control Register

The Debug Control Register (*DCR*) controls and provides information about debug issues. The width of the register is 64 bits. The *DCR* is located in the drseg at offset 0x0000.

The Debug Control Register (*DCR*) provides the following key features:

- Interrupt and NMI control when in Non-Debug Mode
- NMI pending indication
- Availability indicator of instruction and data hardware breakpoints

The DataBrk and InstBrk bits within the *DCR* indicate the types of hardware breakpoints implemented. Debug software is expected to read hardware breakpoint registers for additional information on the number of implemented breakpoints. Refer to Section 11.5, "Hardware Breakpoints" for descriptions of the hardware breakpoint registers.

Hardware and software interrupts can be disabled in Non-Debug Mode using the *DCR* IntE bit. This bit is a global interrupt enable used along with several other interrupt enables that enable specific mechanisms. The NMI interrupt can be disabled in Non-Debug Mode using the NMIE bit; a pending NMI is indicated through the NMIpend bit. Pending interrupts are indicated in the *Cause* register, and pending NMIs are indicated in the *DCR* register NMIpend bit, even when disabled. Hardware and software interrupts and NMIs are always disabled in Debug Mode.

The ProbEn bit reflects the state of the ProbEn bit from the EJTAG Control Register (*ECR*). Through this bit, the probe can indicate to the debug software running on the CPU if it expects to service dmseg accesses.

Figure 11-2 shows the format of the *DCR*; Table 11-14 describes the *DCR* fields. The reset values in Table 11-14 take effect on both hard resets and soft resets.

63 30	29	28 18	17	16	15 5	4	3	2	1	0
0	ENM	0	DataBrk	InstBrk	0	IntE	NMIE	NMIpend	SRstE	ProbEn

Figure 11-2 DCR Register Format

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 11-14 DCR Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	Bits			
ENM	29	Endianness in which the processor is running in kernel and Debug Mode: 0: Little endian 1: Big endian	R	Preset to endianness at reset
DataBrk	17	Indicates if data hardware breakpoint is implemented: 0: No data hardware breakpoint implemented 1: Data hardware breakpoint implemented	R	1
InstBrk	16	Indicates if instruction hardware breakpoint is implemented: 0: No instruction hardware breakpoint implemented 1: Instruction hardware breakpoint implemented	R	1
IntE	4	Hardware and software interrupt enable for Non-Debug Mode, in conjunction with other disable mechanisms: 0: Interrupt disabled 1: Interrupt enabled depending on other enabling mechanisms	R/W	1
NMIE	3	Non-Maskable Interrupt (NMI) enable for Non-Debug Mode: 0: NMI disabled 1: NMI enabled	R/W	1
NMIpend	2	Indication for pending NMI: 0: No NMI pending 1: NMI pending	R	0
SRstE	1	Unused in the 20Kc processor.	R	0
ProbEn	0	Indicates value of the ProbEn value in the *ECR* register: 0: No access should occur to dmseg 1: Probe services accesses to dmseg Bit is read-only (R) and reads as zero if Test Access Port (TAP) is not implemented.	R	Same value as ProbEn in the *ECR*. See Section 11.6.5.6, "EJTAGBOOT Indication Determines Reset Value of EjtagBrk, ProbTrap, and ProbEn" for more information.
0	63:30, 28:18, 15:5	Must be written as zeros; return zeros on reads.	0	0

11.5 Hardware Breakpoints

The hardware breakpoints compare addresses and data of executed instructions, including data load/store accesses. Instruction breakpoints can be set on addresses even in ROM areas, and data breakpoints can cause debug exceptions on a specific data access. Instruction and data hardware breakpoints are alike in many aspects, and are described in parallel

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

in the following sections. The term “hardware” is assumed to modify the word “breakpoint” in the following text, and is used only explicitly when required to distinguish it from a software breakpoint.

11.5.1 Introduction

There are four instruction hardware breakpoints and two data breakpoints implemented in the 20Kc processor. These breakpoints provide the following key features:

- Instruction hardware breakpoints are provided to cause debug exceptions on executed instructions, both in ROM and RAM. Bit masking is provided for virtual address compares, and masking of compares with ASID (optional) is also provided.
- Data hardware breakpoints are provided to cause debug exceptions on data accesses. Bit masking is provided for virtual address compares, masking of compares with ASID is provided, data value compares allow masking at byte level, and qualification on byte access and access type is possible.
- Registers for setup and control are memory mapped in drseg, accessible in Debug Mode only.

Details of instruction and data breakpoints are provided in the following.

11.5.1.1 Instruction Breakpoint Feature

Figure 11-3 shows an overview of the instruction hardware breakpoint feature. The feature compares the virtual address (PC) and the ASID of the executed instructions with each instruction breakpoint, applying masking on address and ASID. When an enabled instruction breakpoint matches the PC and ASID, a debug exception and/or a trigger is generated, and an internal bit in an instruction breakpoint register is set to indicate that a match occurred.

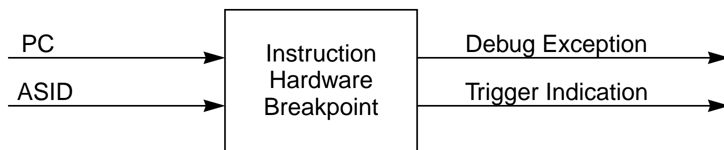


Figure 11-3 Instruction Hardware Breakpoint Overview

11.5.1.2 Data Breakpoint Feature

Figure 11-4 shows an overview of the data hardware breakpoint feature. The feature compares the load or store access type (TYPE), the virtual address of the access (ADDR), the ASID, the accessed bytes (BYTELANE), and data value (DATA) with each data breakpoint, applying masks and/or qualifications on the access properties.

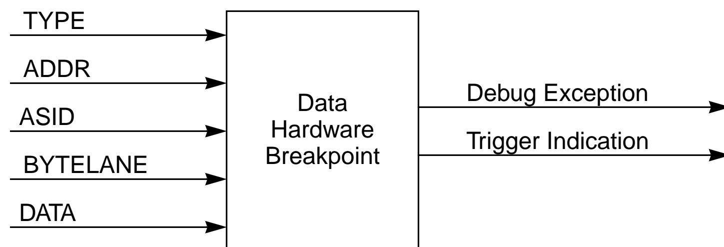


Figure 11-4 Data Hardware Breakpoint Overview

When an enabled data breakpoint matches, a debug breakpoint exception and/or a trigger is generated, and an internal bit in a data breakpoint register is set to indicate that a match occurred. The match is either precise (the debug exception or trigger occurs on the instruction that caused the breakpoint to match) or imprecise (the debug exception or trigger occurs later in the program flow).

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

11.5.2 Overview of Instruction and Data Breakpoint Registers

The InstBrk and DataBrk bits in the DCR register indicate whether hardware breakpoints are implemented or not. If no hardware breakpoints are implemented, then none of the registers associated with the hardware breakpoint feature are implemented, otherwise the registers shown below are implemented.

Section 11.5.2.1, "Overview of Instruction Breakpoint Registers" and Section 11.5.2.2, "Overview of Data Breakpoint Registers" provide overviews of the instruction and data breakpoint registers, respectively. All registers are memory mapped in the drseg segment. All registers are 64 bits wide.

11.5.2.1 Overview of Instruction Breakpoint Registers

Table 11-15 lists the Instruction Breakpoint registers. The Instruction Breakpoint Status register provides implementation indication and status for instruction breakpoints in general. The four implemented instruction breakpoints are numbered 0 to 3 for registers and breakpoints. The specific breakpoint number is indicated by “n”.

Table 11-15 Instruction Breakpoint Register Summary

Register Mnemonic	Register Name and Description	Reference
IBS	Instruction Breakpoint Status	See Section 11.5.6.1, "Instruction Breakpoint Status (IBS) Register"
IBAn	Instruction Breakpoint Address n	See Section 11.5.6.2, "Instruction Breakpoint Address n (IBAn) Register"
IBMn	Instruction Breakpoint Address Mask n	See Section 11.5.6.3, "Instruction Breakpoint Address Mask n (IBMn) Register"
IBASIDn	Instruction Breakpoint ASID n	See Section 11.5.6.4, "Instruction Breakpoint ASID n (IBASIDn) Register"
IBCn	Instruction Breakpoint Control n	See Section 11.5.6.5, "Instruction Breakpoint Control n (IBCn) Register"

Register addresses are shown in Section 11.5.6, "Instruction Breakpoint Registers".

11.5.2.2 Overview of Data Breakpoint Registers

Table 11-16 lists the Data Breakpoint Registers. The Data Breakpoint Status register provides implementation indication and status for data breakpoints in general. The two implemented data breakpoints are numbered 0 and 1 for registers and breakpoints. The specific breakpoint number is indicated by “n”.

Table 11-16 Data Breakpoint Register Summary

Register Mnemonic	Register Name and Description	Reference
DBS	Data Breakpoint Status	See Section 11.5.7.1, "Data Breakpoint Status (DBS) Register"
DBAn	Data Breakpoint Address n	See Section 11.5.7.2, "Data Breakpoint Address n (DBAn) Register"
DBMn	Data Breakpoint Address Mask n	See Section 11.5.7.3, "Data Breakpoint Address Mask n (DBMn) Register"
DBASIDn	Data Breakpoint ASID n	See Section 11.5.7.4, "Data Breakpoint ASID n (DBASIDn) Register"
DBCn	Data Breakpoint Control n	See Section 11.5.7.5, "Data Breakpoint Control n (DBCn) Register"
DBVn	Data Breakpoint Value n	See Section 11.5.7.6, "Data Breakpoint Value n (DBVn) Register"

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Register addresses are shown in Section 11.5.7, "Data Breakpoint Registers".

11.5.3 Conditions for Matching Breakpoints

A number of conditions must be fulfilled in order for a breakpoint to match on an executed instruction or a data access. These conditions are described in the following subsections. A breakpoint only matches for instructions executed in Non-Debug Mode, never due to instructions executed in Debug Mode.

The match of an enabled breakpoint generates a debug exception as described in Section 11.5.4, "Debug Exceptions from Breakpoints" and a trigger indication as described in Section 11.5.5, "Breakpoints Used as Triggerpoints". The BE and TE bits in the *IBCn* or *DBCn* registers enable the breakpoints for breaks and triggers, respectively.

11.5.3.1 Conditions for Matching Instruction Breakpoints

When an instruction breakpoint is enabled, that breakpoint is evaluated in Non-Debug Mode with the instruction boundary address (the lowest address of a byte in the instruction) of every executed instruction. The instruction breakpoint is also evaluated on addresses usually causing Address Error exception, TLB exception, or other exceptions. It is thereby possible to cause a Debug Instruction Break exception on the destination address of a jump, even if jump to that address would otherwise cause an Address Error exception. The breakpoint is not evaluated on instructions from speculative fetches or execution.

A match of an instruction breakpoint depends on a number of parameters, shown in Table 11-17. The fields in the instruction breakpoint registers are in the form REG_{FIELD}.

Table 11-17 Instruction Breakpoint Condition Parameters

Parameter	Description	Width
ASID	ASID field in the *EntryHi* CP0 register.	8 bits
$IBCn_{ASIDuse}$	Use ASID value in compare for instruction breakpoint n: 0: Do not use ASID value in compare 1: Use ASID value in compare	1 bit
$IBASIDn_{ASID}$	Conditional Instruction breakpoint n ASID value for comparing.	8 bits
PC	Virtual address of instruction boundary or target for jump/branch.	64 bits
$IBAn_{IBA}$	Instruction breakpoint n address for compare with conditions.	64 bits
$IBMn_{IBM}$	Instruction breakpoint n address mask condition: 0: Corresponding address bit compared 1: Corresponding address bit masked	64 bits

The equation that determines the match is shown below with “C” like operators. In the equation, 0 means all bits are 0’s, and ~0 means all bits are 1’s. The widths are similar to the widths of the parameters. The match equation is IB_match:

$$IB_match = (!IBCn_{ASIDuse} || (ASID == IBASIDn_{ASID})) \&\& ((IBMn_{IBM} | \sim(PC \ \hat{} \ IBAn_{IBA})) == \sim 0)$$

The IB_match equation also applies to 64-bit processors running in 32-bit addressing mode, in which case all 64 bits are compared between the PC and the $IBAn_{IBA}$ register.

The match indication for instruction breakpoints is always precise; that is, it is indicated on the instruction causing the IB_match to be true.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

11.5.3.2 Conditions for Matching Data Breakpoints

When a data breakpoint is enabled, that breakpoint is evaluated in Non-Debug Mode with the access address of every data access due to load/store instructions (including loads/stores of coprocessor registers) and address causing address errors at data access. Data breakpoints are not evaluated with addresses from PREF (prefetch) or CACHE instructions.

A match of the data breakpoint depends on a number of parameters, shown in Table 11-18. The fields in the data breakpoint registers are in the form REG_{FIELD}.

Table 11-18 Data Breakpoint Condition Parameters

Reference	Description	Width
TYPE	Data access type is either load or store.	(no width)
$DBCn_{NoSB}$	Controls whether condition for data breakpoint is fulfilled on a store access: 0: Condition can be fulfilled on store access 1: Condition is never fulfilled on store access	1 bit
$DBCn_{NoLB}$	Controls whether condition for data breakpoint is fulfilled on a load access: 0: Condition can be fulfilled on load access 1: Condition is never fulfilled on load access	1 bit
ASID	ASID field in the *EntryHi* CP0 register.	8 bits
$DBCn_{ASIDuse}$	ASID value used in compare for data breakpoint n: 0: Do not use ASID value in compare 1: Use ASID value in compare	1 bit
$DBASIDn_{ASID}$	Conditional Data breakpoint n ASID value for comparison	8 bits
ADDR	Virtual address of data access.	64 bits
$DBAn_{DBA}$	Data breakpoint n address for compare with conditions	64 bits
$DBMn_{DBM}$	Conditional Data breakpoint n address mask: 0: Corresponding address bit compared 1: Corresponding address bit masked	64 bits
BYTELANE	Byte lane access indication, where BYTELANE[0] is 1 only if the byte at bits [7:0] on the data bus is accessed, BYTELANE[1] is 1 only if the byte at bits [15:8] on the data bus is accessed, etc.	8 bits
$DBCn_{BAI}$	Determines whether access is ignored to specific bytes. BAI[0] ignores access to byte at bits [7:0] of the data bus, BAI[1] ignores access to byte at bits [15:8] of the data bus, etc. 0: Condition depends on access to corresponding byte 1: Access for corresponding byte is ignored	8 bits
DATA	Data value from the data bus.	64 bits
$DBVn_{DBV}$	Conditional Data breakpoint n data value for compare	64 bits
$DBCn_{BLM}$	Conditional Byte lane mask for value compare on data breakpoint. BLM[0] masks byte at bits [7:0] of the data bus, BLM[1] masks byte at bits [15:8], etc.: 0: Compare corresponding byte lane 1: Mask corresponding byte lane	8 bits

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

The match equations are shown below with "C" like operators. In the equation, 0 means all bits are 0's, and ~0 means all bits are 1's. The bit widths are similar to the widths of the parameters.

DB_match is the overall match equation (the DB_addr_match, DB_no_value_compare, and DB_value_match equations in the DB_match equation are defined below):

```
DB_match =
(((TYPE==load)&&!DBCn_{NoLB})||((TYPE==store)&&!DBCn_{NoSB}))&&
DB_addr_match&&(DB_no_value_compare||DB_value_match)
```

DB_addr_match is defined as:

```
DB_addr_match =
(!DBCn_{ASIDuse}||(ASID==DBASIDn_{ASID}))&&
((DBMn_{DBM}|~(ADDR^DBAn_{DBA}))==~0)&&
((~DBCn_{BAI}&BYTELANE)!=0)
```

The DB_addr_match equation also applies to 64-bit processors running in 32-bit addressing mode, in which case all 64 bits are compared between the ADDR and the $DBAn_{DBA}$ field.

DB_no_value_compare is defined as:

```
DB_no_value_compare =
((DBCn_{BLM}|DBCn_{BAI}|~BYTELANE)==~0)
```

If a data value compare is indicated on a breakpoint then DB_no_value_compare is false, and if there is no data value compare then DB_no_value_compare is true. Note that a data value compare is a run-time property of the breakpoint if ($DBCn_{BLM}$ | $DBCn_{BAI}$) is not ~0, because DB_no_value_compare then depends on BYTELANE provided by the load/store instructions.

If a data value compare is required, then the data value from the data bus is compared and masked with the registers for the data breakpoint, as shown in the DB_value_match equation:

```
DB_value_match =
((DATA[7:0]==DBVn_{DBV[7:0]})||!BYTELANE[0]||DBCn_{BLM[0]}||DBCn_{BAI[0]} )&&
((DATA[15:8]==DBVn_{DBV[15:8]})||!BYTELANE[1]||DBCn_{BLM[1]}||DBCn_{BAI[1]})&&
......
((DATA[63:56]==DBVn_{DBV[63:56]})||
!BYTELANE[7]||DBCn_{BLM[7]}||DBCn_{BAI[7]})
```

Data breakpoints depend on endianness, because values on the byte lanes are used in the equations. Thus it is required that the debug software programs the breakpoints accordingly to endianness.

Precise breakpoint exceptions only occur on stores and loads if there is no data value compare. An imprecise breakpoint exception only occurs on a data breakpoint on stores and loads with data value compare.

If a data value compare is required to evaluate a data breakpoint, (the DB_no_value_compare equation is false), but a bus or cache error occurs on the load, then there is no valid data to use in the compare, and there will be no match in this case.

11.5.4 Debug Exceptions from Breakpoints

This subsection describes how to set up instruction and data breakpoints to generate debug exceptions when the match conditions are true.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

11.5.4.1 Debug Exception Caused by Instruction Breakpoint

When the BE bit in the *IBCn* register is set, instruction breakpoints are enabled. A Debug Instruction Break exception occurs when the IB_match equation is true (see Section 11.5.3.1, "Conditions for Matching Instruction Breakpoints"). The corresponding BSn bit in the *IBS* register is set when the breakpoint generates the debug exception.

The Debug Instruction Break exception is precise, so the *DEPC* register and DBD bit in the *Debug* register point to the instruction that caused the IB_match equation to be true. Refer to Section 11.5.4.2, "Precise Debug Exception Caused by Data Breakpoint"

The instruction receiving the debug exception only updates the debug related registers. That instruction does not cause any loads/stores to occur. Thus a debug exception from a data breakpoint cannot occur at the same time an instruction receives a Debug Instruction Break exception.

The debug handler usually returns to the instruction causing the Debug Instruction Break exception, whereby the instruction is executed. Debug software must disable the breakpoint when returning to the instruction, otherwise the Debug Instruction Break exception reoccurs. An alternative is for debug software to emulate the instructions in software and change the *DEPC* accordingly.

11.5.4.2 Precise Debug Exception Caused by Data Breakpoint

The BE bit in the *DBCn* register must be set for data breakpoints to be enabled. A debug exception occurs when the DB_match condition is true (see Section 11.5.3.2, "Conditions for Matching Data Breakpoints").

A Debug Data Break Load/Store exception occurs when a data breakpoint indicates a precise match. In this case, the DEPC register and DBD bit in the Debug register point to the load/store instruction that caused the DB_match equation (see Section 11.5.3.2, "Conditions for Matching Data Breakpoints") to be true, and the corresponding BSn bit in the *DBS* register is set. For the 20Kc processor, these precise debug exceptions only occur for data breakpoints without data value compare on load and store instructions. Details about behavior of the instruction causing the debug exception is shown in Table 11-19.

Table 11-19 Behavior on Precise Exceptions from Data Breakpoints

Data Breakpoint and Instruction	Load/Store Instruction Execution	Destination Register	External Memory System Access
Store without value match	Not completed	Not updated[1]	Store to memory is not committed
Load without value match		Not updated[2]	Load from memory does not occur

1. This applies to the Store Conditional Word (SC) instruction
2. This includes side effects like for the Load Linked Word (LL) instruction

The rules shown in Table 11-20 describe update of the BSn bits when several data breakpoints match the same access and generate a debug exception.

Table 11-20 Rules for Update of BSn Bits on Precise Exceptions from Data Breakpoints

Instruction	Breakpoints That Would Match		Data Breakpoints and Update of BSn Bits	
	Without Value Compare	With Value Compare	Without Value Compare	With Value Compare
Load / Store	One or more	None	BSn bits set for all	Unchanged BSn bits

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 11-20 Rules for Update of BSn Bits on Precise Exceptions from Data Breakpoints (Continued)

Instruction	Breakpoints That Would Match		Data Breakpoints and Update of BSn Bits	
	Without Value Compare	With Value Compare	Without Value Compare	With Value Compare
Load	One or more	One or more	BSn bits set for all	Unchanged BSn bits since load of data value does not occur, so match of the breakpoint cannot be determined
Load / Store	None	One or more	Does not apply since these breakpoints are imprecise.	
Store	One or more	One or more	BSn bits set for all	Optional to either set BSn bits for all, or change none of the BSn bits

Any BSn bit set prior to the match and debug exception is kept set, because only debug software can clear the BSn bits.

The debug handler usually returns to the instruction that caused the Debug Data Break Load/Store exception, whereby the instruction is re-executed. Debug software must disable breakpoints when returning to the instruction, otherwise the Debug Data Break Load/Store exception will reoccur.

11.5.4.3 Imprecise Debug Exception Caused by Data Breakpoint

The BE bit in the *DBCn* register must be set for data breakpoints to be enabled. A debug exception occurs when the DB_match condition is true (see Section 11.5.3.2, "Conditions for Matching Data Breakpoints").

A Debug Data Break Load/Store Imprecise exception occurs when a data breakpoint indicates an imprecise match. In this case, the DEPC register and DBD bit in the Debug register point to an instruction later in the execution flow rather than at the load/store that caused the DB_match equation to be true. For the 20Kc processor, these imprecise debug exceptions only occur for data breakpoints with data value compare on load and store instructions.

The load/store instruction causing the Debug Data Break Load/Store Imprecise exception always updates the destination register and finalizes the access to the external memory system. Therefore this load/store instruction is not re-executed at return from the debug handler, because the DEPC register and DBD bit do not point to that instruction.

Several imprecise data breakpoints can be pending at a given time, and the breakpoints are then evaluated as the accesses finalize, but a Debug Data Break Load/Store Imprecise exception is generated only for the first one matching. Both the first and succeeding matches cause corresponding BSn bits and DDBLImpr/DDBSImpr bit to be set, but no debug exception is generated for succeeding matches because the processor is already in Debug Mode. Similarly, if a debug exception had already occurred at the time of the first match (for example, due to a precise debug exception), then all matches cause the corresponding BSn bits and DDBLImpr/DDBSImpr bit to be set, but no debug exception is generated because the processor is already in Debug Mode.

The debug handler is required to execute the SYNC instruction, followed by two cycles spacing (for example, using two SSNOP instructions), before the BSn bits and DDBLImpr/DDBSImpr bit are accessed for read or write. This delay ensures that these bits are fully updated.

Any BSn bit set prior to the match and debug exception are kept set, because only debug software can clear the BSn bits.

11.5.5 Breakpoints Used as Triggerpoints

Software can set up both instruction and data hardware breakpoints such that a matching breakpoint does not generate a debug exception, but sends an indication through the BSn bit only. The TE bit in the *IBCn* or *DBCn* register controls

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

whether an instruction or data breakpoint, respectively, is used as a triggerpoint. Triggerpoints are evaluated for matches under the same criteria as breakpoints.

The BSn bit in the *IBS* or *DBS* register is set for a triggerpoint when the respective IB_match condition (see Section 11.5.3.1, "Conditions for Matching Instruction Breakpoints") or DB_match condition (see Section 11.5.3.2, "Conditions for Matching Data Breakpoints") is true.

For the BSn bit to be set for an instruction triggerpoint, either the instruction must be fully executed or an exception must occur on the instruction itself.

For the BSn bit to be set for a data triggerpoint it requires initiation of the data access, which includes accesses causing an exception for example TLB or Bus Error exception. However, data triggerpoints with value compare requires the data value to be valid for the BSn bit to be set, which is not the case for a load instruction on which a TLB or Bus Error exception occurs. Exceptions earlier in the pipe might inhibit the data access whereby the triggerpoint has no effect. The rules for update of the BSn bits are shown in Table 11-21.

Table 11-21 Rules for Update of BSn Bits on Data Triggerpoints

Instruction	Without/With Value Compare	BSn Bit Update for Triggerpoint
Load / Store	Without value compare	BSn bits are set if the data access is initiated, even if the data access causes an exception.
Load	With value compare	BSn bits are set only if the load data value is returned and thus not set if an exception causes the load access not to occur.
Store	With value compare	BSn bits are set if the data access occurs. The BSn bits are not set if the data access causes a higher priority exception.

Data breakpoints with imprecise matches generate imprecise triggers when enabled by the TE bit.

11.5.6 Instruction Breakpoint Registers

This subsection describes the instruction breakpoint registers. These registers provide status and control for the instruction breakpoints. All registers are in drseg. The four implemented breakpoints are numbered 0 to 3, respectively, for registers and breakpoints. The specific breakpoint number is indicated by “n”. The registers and their respective addresses offsets are shown in Table 11-22.

Table 11-22 Instruction Breakpoint Register Mapping

Offset in drseg	Register Mnemonic	Register Name and Description
0x1000	IBS	Instruction Breakpoint Status
0x1100 + 0x100 * n	IBAn	Instruction Breakpoint Address n
0x1108 + 0x100 * n	IBMn	Instruction Breakpoint Address Mask n
0x1110 + 0x100 * n	IBASIDn	Instruction Breakpoint ASID n
0x1118 + 0x100 * n	IBCn	Instruction Breakpoint Control n

11.5.6.1 Instruction Breakpoint Status (IBS) Register

The Instruction Breakpoint Status (*IBS*) register holds implementation and status information about the instruction breakpoints. It is located at drseg offset 0x1000. The ASIDsup bit applies to all instruction breakpoints. Figure 11-5 shows the format of the *IBS* register; Table 11-23 describes the *IBS* register fields.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

63 31	30	29 28	27 24	23 4	3 0
0	ASIDsup	0	BCN	0	BS[3:0]

Figure 11-5 IBS Register Format

Table 11-23 IBS Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	**Bits**			
ASIDsup	30	Indicates if ASID compare is supported in instruction breakpoints: 0: No ASID compare 1: ASID compare (*IBASIDn* register implemented) ASID support indication does not guarantee a TLB-type MMU, because the same hardware breakpoint implementation can be used with processors having all different types of MMUs.	R	1
BCN	27:24	Number of instruction breakpoints implemented: 0: Reserved 1-13: Number of instructions breakpoints	R	4
BS[3:0]	3:0	Break status for breakpoint n is at BSn, where n is 0 to 3. A bit is set to 1 when the condition for its corresponding breakpoint has matched. The number of BS bits corresponds to the number of breakpoints indicated by the BCN field. Debug software is expected to clear the bits before use, because reset does not clear these bits.	R/W0	Undefined
0	63:31, 29:28, 23:4	Must be written as zeros; return zeros on read.	0	0

11.5.6.2 Instruction Breakpoint Address n (IBAn) Register

The Instruction Breakpoint Address n (*IBAn*) register has the address used in the condition for instruction breakpoint n. It is located at drseg offset 0x1100 + 0x100 * n. Figure 11-6 shows the format of the *IBAn* register; Table 11-24 describes the *IBAn* register field.

63 0
IBAn

Figure 11-6 IBAn Register Format

Table 11-24 IBAn Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	**Bits**			
IBA	63:0	Instruction breakpoint address for condition.	R/W	Undefined

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

11.5.6.3 Instruction Breakpoint Address Mask n (IBMn) Register

The Instruction Breakpoint Address Mask n (*IBMn*) register has the address compare mask used in the condition for instruction breakpoint n. The address that is masked is in the *IBAn* register. The *IBMn* register is located at drseg offset 0x1108 + 0x100 * n. Figure 11-7 shows the format of the *IBMn* register; Table 11-25 describes the *IBMn* register field.

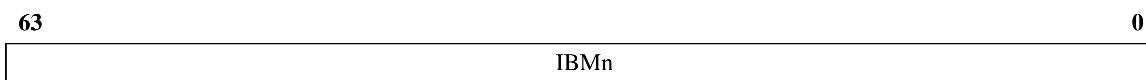


Figure 11-7 IBMn Register Format

Table 11-25 IBMn Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	Bits			
IBM	63:0	Instruction breakpoint address mask for condition: 0: Corresponding address bit compared 1: Corresponding address bit masked	R/W	Undefined

11.5.6.4 Instruction Breakpoint ASID n (IBASIDn) Register

The Instruction Breakpoint ASID n (*IBASIDn*) register has the ASID value used in the compare for instruction breakpoint n. It is located at drseg offset 0x1110 + 0x100 * n.

Figure 11-8 shows the format of the *IBASIDn* register; Table 11-26 describes the *IBASIDn* register fields.

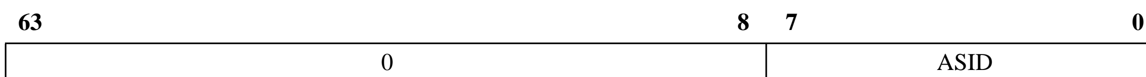


Figure 11-8 IBASIDn Register Format

Table 11-26 IBASIDn Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	Bits			
ASID	7:0	Instruction breakpoint ASID value for compare.	R/W	Undefined
0	63:8	Must be written as zeros; return zeros on read.	0	0

11.5.6.5 Instruction Breakpoint Control n (IBCn) Register

The Instruction Breakpoint Control n (*IBCn*) register determines what constitutes instruction breakpoint n: triggerpoint, breakpoint, ASID value inclusion. This register is located at drseg offset 0x1118 + 0x100 * n. Figure 11-9 shows the format of the *IBCn* register; Table 11-26 describes the *IBCn* register fields.

Figure 11-9 IBCn Register Format

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 11-27 IBCn Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	Bits			
ASIDuse	23	Use ASID value in compare for instruction breakpoint n: 0: Do not use ASID value in compare 1: Use ASID value in compare Debug software should only set the ASIDuse if a TLB in the implementation is used by the application software.	R/W	Undefined
TE	2	Use instruction breakpoint n as triggerpoint: 0: Do not use it as triggerpoint 1: Use it as triggerpoint	R/W	0
BE	0	Use instruction breakpoint n as breakpoint: 0: Do not use it as breakpoint 1: Use it as breakpoint	R/W	0
0	63:24, 22:3, 1	Must be written as zeros; return zeros on read.	0	0

11.5.7 Data Breakpoint Registers

This section describes the data breakpoint registers. These registers provide status and control for the data breakpoints. All registers are in drseg. The two implemented breakpoints are numbered 0 and 1, respectively, for registers and breakpoints. The specific breakpoint number is indicated by “n”. The registers and their respective address offsets are shown in Table 11-28.

Table 11-28 Data Breakpoint Register Mapping

Offset in drseg	Register Mnemonic	Register Name and Description
0x2000	*DBS*	Data Breakpoint Status
0x2100 + 0x100 * n	*DBAn*	Data Breakpoint Address n
0x2108 + 0x100 * n	*DBMn*	Data Breakpoint Address Mask n
0x2110 + 0x100 * n	*DBASIDn*	Data Breakpoint ASID n
0x2118 + 0x100 * n	*DBCn*	Data Breakpoint Control n
0x2120 + 0x100 * n	*DBVn*	Data Breakpoint Value n

To remove hazards when updating data hardware breakpoint registers, the debug handler must execute the SYNC instruction followed by at least two cycles in Debug Mode after writing to the data breakpoint registers (for example, using two SSNOPs). This procedure ensures that the registers are fully updated for Non-Debug Mode, otherwise behavior of the processor is UNDEFINED.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

11.5.7.1 Data Breakpoint Status (DBS) Register

The Data Breakpoint Status (*DBS*) register holds implementation and status information about the data breakpoints. It is located at drseg offset 0x2000. The ASIDsup, NoSVmatch, and NoLVmatch fields apply to all data breakpoints. Figure 11-10 shows the format of the *DBS* register; Table 11-29 describes the *DBS* register fields.

63 31	30	29	28	27 24	23 2	1 0
0	ASIDsup	NoSVmatch	NoLVmatch	BCN	0	BS[1:0]

Figure 11-10 DBS Register Format

Table 11-29 DBS Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	**Bits**			
ASIDsup	30	Indicates if ASID compare is supported in data breakpoints: 0: No ASID compare 1: ASID compare (*DBASIDn* register implemented) ASID support indication does not guarantee a TLB-type MMU, because the same hardware breakpoint implementation can be used with processors having all different types of MMUs.	R	1
NoSVmatch	29	Indicates if a value compare on a store is supported in data breakpoints: 0: Data value and address in condition on store 1: Address compare only in condition on store	R	0
NoLVmatch	28	Indicates if a value compare on a load is supported in data breakpoints: 0: Data value and address in condition on load 1: Address compare only in condition on load	R	0
BCN	27:24	Number of data breakpoints implemented: 0: Reserved 1-13: Number of data breakpoints	R	2
BS[1:0]	1:0	Break status for breakpoint n is at BSn, where n is 0 or 1. The bit is set to 1 when the condition for its corresponding breakpoint has matched. The number of BS bits implemented corresponds to the number of breakpoints indicated by the BCN bit. Debug software is expected to clear the bits before use, since these are not cleared by reset.	R/W0	Undefined
0	63:31, 23:2	Must be written as zeros; return zeros on read.	0	0

11.5.7.2 Data Breakpoint Address n (DBAn) Register

The Data Breakpoint Address n (*DBAn*) register has the address used in the condition for data breakpoint n. This register is located at drseg offset 0x2100 + 0x100 * n. Figure 11-11 shows the format of the *DBAn* register; Table 11-30 describes the *DBAn* register field.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Figure 11-11 DBAn Register Format

Table 11-30 DBAn Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	Bits			
DBA	63:0	Data breakpoint address for condition	R/W	Undefined

11.5.7.3 Data Breakpoint Address Mask n (DBMn) Register

The Data Breakpoint Address Mask n (*DBMn*) register has the address compare mask used in the condition for data breakpoint n. The address that is masked is in the *DBAn* register. The *DBMn* register is located at drseg offset 0x2108 + 0x100 * n. Figure 11-12 shows the format of the *DBMn* register; Table 11-31 describes the *DBMn* register field.

Figure 11-12 DBMn Register Format

Table 11-31 DBMn Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	Bits			
DBMn	63:0	Data breakpoint address mask for condition: 0: Corresponding address bit compared 1: Corresponding address bit masked	R/W	Undefined

11.5.7.4 Data Breakpoint ASID n (DBASIDn) Register

The Data Breakpoint ASID n (*DBASIDn*) register has the ASID value used in the compare for data breakpoint n. It is located at drseg offset 0x2110 + 0x100 * n.

Figure 11-13 shows the format of the *DBASIDn* register; Table 11-32 describes the *DBASIDn* register fields.

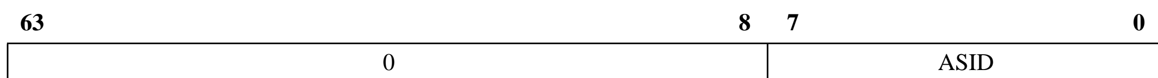


Figure 11-13 DBASIDn Register Format

Table 11-32 DBASIDn Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	Bits			
ASID	7:0	Data breakpoint ASID value for compare.	R/W	Undefined
0	63:8	Must be written as zeros; return zeros on read.	0	0

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

11.5.7.5 Data Breakpoint Control n (DBCn) Register

The Data Breakpoint Control n (*DBCn*) register determines what constitutes data breakpoint n: triggerpoint, breakpoint, ASID value inclusion, load/store access fulfillment, ignore byte access, byte lane mask. This register is located at drseg offset 0x2118 + 0x100 * n. Figure 11-14 shows the format of the *DBCn* register; Table 11-33 describes the *DBCn* register fields.

63 — 24	23	22	21 — 14	13	12	11 — 4	3	2	1	0
0	ASIDuse	0	BAI[7:0]	NoSB	NoLB	BLM[7:0]	0	TE	0	BE

Figure 11-14 DBCn Register Format

Table 11-33 DBCn Register Field Descriptions

Fields: Name	Fields: Bits	Description	Read/ Write	Reset State
ASIDuse	23	Use ASID value in compare for data breakpoint n: 0: Do not use ASID value in compare 1: Use ASID value in compare Debug software should only set the ASIDuse if a TLB in the implementation is used by the application software.	R/W	Undefined
BAI[7:0]	21:14	Byte access ignore. Controls ignore of access to specific bytes. BAI[0] ignores access to byte at bits [7:0] of the data bus, BAI[1] ignores access to byte at bits [15:8], etc.: 0: Condition depends on access to corresponding byte 1: Access for corresponding byte is ignored Debug software must adjust for endianness when programming this field.	R/W	Undefined
NoSB	13	Controls whether condition for data breakpoint is ever fulfilled on a store access: 0: Condition can be fulfilled on store access 1: Condition is never fulfilled on store access	R/W	Undefined
NoLB	12	Controls whether condition for data breakpoint is ever fulfilled on a load access: 0: Condition can be fulfilled on load access 1: Condition is never fulfilled on load access	R/W	Undefined
BLM[7:0]	11:4	Byte lane mask for value compare on data breakpoint. BLM[0] masks byte at bits [7:0] of the data bus, BLM[1] masks byte at bits [15:8], etc.: 0: Compare corresponding byte lane 1: Mask corresponding byte lane Debug software must adjust for endianness when programming this field.	R/W	Undefined
TE	2	Use data breakpoint n as triggerpoint: 0: Do not use it as triggerpoint 1: Use it as triggerpoint	R/W	0

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 11-33 DBCn Register Field Descriptions (Continued)

Fields		Description	Read/ Write	Reset State
Name	Bits			
BE	0	Use data breakpoint n as breakpoint: 0: Do not use it as breakpoint 1: Use it as breakpoint	R/W	0
0	63:24, 22, 3, 1	Must be written as zeros; return zeros on read.	0	0

11.5.7.6 Data Breakpoint Value n (DBVn) Register

The Data Breakpoint Value n (*DBVn*) register has the value used in the condition for data breakpoint n. It is located at drseg offset 0x2120 + 0x100 * n. Figure 11-15 shows the format of the *DBVn* register; Table 11-34 describes the *DBVn* register field.

Figure 11-15 DBVn Register Format

Table 11-34 DBVn Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	Bits			
DBV	63:0	Data breakpoint data value for condition. Debug software must adjust for endianness when programming this field.	R/W	Undefined

11.6 EJTAG Test Access Port

This section describes the EJTAG features provided with the EJTAG Test Access Port (TAP).

The overall features of the EJTAG Test Access Port (TAP) are:

- Identification of device and EJTAG debug features accessed through the TAP
- EJTAG memory in dseg, which provides a memory-mapped area handled by the probe through processor accesses, whereby the processor can execute a debug handler not present in the system memory
- Reset handling allows debug exception immediately after reset
- Debug interrupt request from probe
- Low-power mode indications

Figure 11-16 shows an overview of the elements in the TAP.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

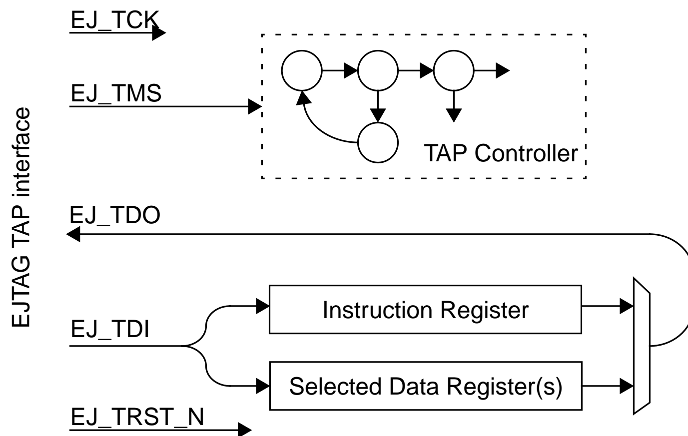


Figure 11-16 Test Access Port (TAP) Overview

The TAP consists of the following signals: Test Clock (*EJ_TCK*), Test Mode (*EJ_TMS*), Test Data In (*EJ_TDI*), Test Data Out (*EJ_TDO*), and the Test Reset (*EJ_TRST_N*). *EJ_TCK* and *EJ_TMS* control the state of the TAP controller, which controls access to the Instruction or selected data register(s). The Instruction register controls selection of data registers. Access to the Instruction and data register(s) occurs serially through *EJ_TDI* and *EJ_TDO*. The *EJ_TRST_N* is an asynchronous reset signal to the TAP.

Access through the TAP does not interfere with the operation of the processor, unless features specifically described to do so are used.

11.6.1 TAP Signals

The signals *EJ_TCK*, *EJ_TMS*, *EJ_TDI*, *EJ_TDO*, and the *EJ_TRST_N* make up the interface for the TAP. These signals are described in detail below.

11.6.1.1 Test Clock Input (EJ_TCK)

EJ_TCK is the clock that controls the updating of the TAP controller and the shifting of data through the Instruction or selected data register(s).

EJ_TCK is independent of the processor clock, with respect to both frequency and phase.

11.6.1.2 Test Mode Select Input (EJ_TMS)

EJ_TMS is the control signal for the TAP controller. This signal is sampled on the rising edge of *EJ_TCK*.

11.6.1.3 Test Data Input (EJ_TDI)

EJ_TDI is the test data input to the Instruction or selected data register(s). This signal is sampled on the rising edge of *EJ_TCK* for some TAP controller states.

11.6.1.4 Test Data Output (EJ_TDO)

EJ_TDO is the test data output from the Instruction or data register(s). This signal changes on the falling edge of *EJ_TCK*. *EJ_TDO* is always driven but the *EJ_TDOzstate* output is provided to indicate when *EJ_TDO* is shifting data out. For more information on using the *EJ_TDOzstate* port, refer to Chapter 4 in the *MIPS64™ 20Kc™ Processor Core Integrator's Guide*.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

11.6.1.5 Test Reset Input (EJ_TRST_N)

EJ_TRST_N is the test reset input that asynchronously resets the TAP, with the following immediate effects:

- The TAP controller is put into the Test-Logic-Reset state
- The *Instruction* register is loaded with the IDCODE instruction
- Any EJTAGBOOT indication is cleared
- The *EJ_TDOzstate* port goes high

EJ_TRST_N does not reset any other part of the TAP or processor. Thus this type of reset does not affect the processor, and the processor reset is not allowed to have any effect on the above parts of the TAP.

The TAP reset takes effect when the *EJ_TRST_N* input is low, asynchronous to clocking with *EJ_TCK*.

11.6.2 TAP Controller

The TAP controller is a state machine whose active state controls TAP reset and access to Instruction and data registers.

The state transitions in the TAP controller occur on the rising edge of *EJ_TCK* or when *EJ_TRST_N* goes low. The *EJ_TMS* signal determines the transition at the rising edge of *EJ_TCK*. Figure 11-17 shows the state diagram for the TAP controller.

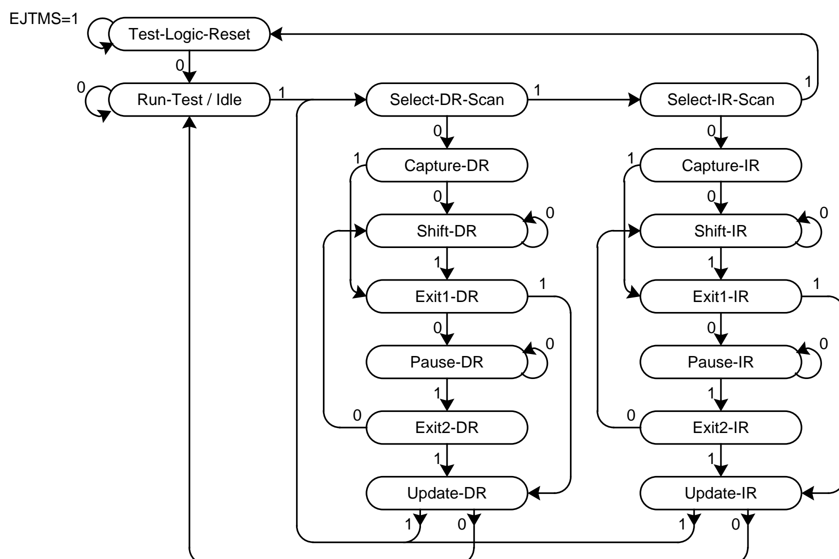


Figure 11-17 TAP Controller State Diagram

The behavior of the functional states shown in the figure is described below. The non-functional states are intermediate states in which no registers in the TAP change, and are not described here.

Events in the following subsections are described with relation to the rising and falling edge of *EJ_TCK*. The described events take place when the TAP controller is in the corresponding state when the clock changes.

The TAP controller is forced into the Test-Logic-Reset state at power-up by a low value on *EJ_TRST_N*.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

11.6.3 Test-Logic-Reset State

When this state is entered, the *Instruction* register is loaded with the IDCODE instruction, and any EJTAGBOOT indication is cleared. This state ensures that the TAP does not interfere with the normal operation of the CPU core.

The TAP controller always reaches this state after five rising edges on *EJ_TCK* when *EJ_TMS* is set to 1.

A low value on *EJ_TRST_N* immediately places the TAP controller in this state asynchronous to *EJ_TCK*.

11.6.3.1 Capture-IR State

In the Capture-IR state, the *Instruction* register is loaded with the IDCODE instruction on the rising edge of *EJ_TCK*.

11.6.3.2 Shift-IR State

In the Shift-IR state, the LSB of the *Instruction* register is output on *EJ_TDO* on the falling edge of *EJ_TCK*. The *Instruction* register is shifted one position from MSB to LSB on the rising edge of *EJ_TCK*, with the MSB shifted in from *EJ_TDI*. The value in the *Instruction* register does not take effect until the Update-IR state. Figure 11-18 shows the shifting direction for the *Instruction* register.

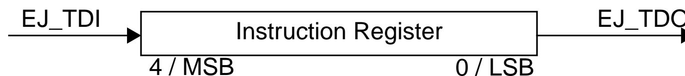


Figure 11-18 Shifting of the Instruction Register During the Shift IR State

The length of the *Instruction* register is five bits.

The value loaded in the Capture-IR state is used as the initial value for the *Instruction* register when shifting starts; thus it is not possible to read out the previous value of the *Instruction* register.

11.6.3.3 Update-IR State

The value in the *Instruction* register takes effect on the rising edge of *EJ_TCK*.

11.6.3.4 Capture-DR State

The value of the selected data register(s) is captured to a parallel register on the rising edge of *EJ_TCK* for shifting out in the Shift-DR state. The capture to a parallel register is performed so as not to affect normal processor operation. The Capture-DR state reads the data in order to output the read value in the Shift-DR state.

The *Instruction* register controls the selection of the following data register(s): Bypass, Device ID, Implementation, EJTAG Control, Address, Data register(s), and Fast Data.

11.6.3.5 Shift-DR State

The LSB of the selected data register(s) is output on *EJ_TDO* on the falling edge of *EJ_TCK*. The selected data register(s) is shifted one position from MSB to LSB on the rising edge of *EJ_TCK*, with *EJ_TDI* shifted in at the MSB. The value(s) shifted into the register(s) does not take effect until the Update-DR state. Figure 11-19 shows the shifting direction for the selected data register.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Figure 11-19 Shifting of the Instruction Register During the Shift DR State

The length of the shift path depends on the selected data register(s).

11.6.3.6 Update-DR State

The update of the selected data register(s) with the value from the Shift-DR state occurs on the falling edge of *EJ_TCK*. This update writes the selected register(s).

11.6.4 Instruction Register and Special Instructions

The *Instruction* register controls selection of accessed data register(s), and controls the setting and clearing of the EJTAGBOOT indication.

The *Instruction* register is five bits wide. Table 11-35 shows the allocation of the TAP instruction.

Table 11-35 TAP Instruction Overview

Code	Instruction	Function
0x00	EXTEST	Selects the EXTEST mode for boundary scan
0x01	IDCODE	Selects *Device Identification (ID)* register
0x03	IMPCODE	Selects *Implementation* register
0x08	ADDRESS	Selects *Address* register
0x09	DATA	Selects *Data* register
0x0A	CONTROL	Selects *EJTAG Control* register
0x0B	ALL	Selects the *Address*, *Data*, and *EJTAG Control* registers
0x0C	EJTAGBOOT	Makes the processor take a debug exception after reset
0x0D	NORMALBOOT	Makes the processor execute the reset handler after reset
0x0E	FASTDATA	Provides a one-bit register whose value is tagged to the front of the *Data* register to capture the value of the processor access pending (PrAcc) bit in the *EJTAG Control* register.
0x1C	SAMPPRELOAD	Selects the Sample/Preload mode for boundary scan
0x1E	Reserved	Reserved
0x1F	BYPASS	Selects the *Bypass* register

The instructions IDCODE, IMPCODE, ADDRESS, DATA, CONTROL, and BYPASS select a single data register, as described in the table. The unused instructions select the *Bypass* register. The ALL, EJTAGBOOT, NORMALBOOT, and FASTDATA instructions are described in the following subsections.

Any EJTAGBOOT indication is cleared at power-up either by a low value on the *EJ_TRST_N* or by a power-up reset circuit, and the Instruction register is loaded with the IDCODE instruction.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

11.6.4.1 ALL Instruction

The *Address*, *Data*, and *EJTAG Control* data registers are selected at once with the ALL instruction, as shown in Figure 11-20.

Figure 11-20 Selecting Registers Using the ALL Instruction

11.6.4.2 EJTAGBOOT and NORMALBOOT Instructions

The EJTAGBOOT and NORMALBOOT instructions control whether the processor takes a Debug Interrupt exception after reset with execution of the debug handler from the probe, or if it executes the reset handler as usual. An internal EJTAGBOOT indication holds information on the action to take at a processor reset, which applies to both reset and soft reset.

The internal EJTAGBOOT indication is set when the EJTAGBOOT instruction takes effect in the Update-IR state. The indication is cleared when the NORMALBOOT instruction takes effect in the Update-IR state, or when the Test-Logic-Reset state is entered, for example, when *EJ_TRST_N* is asserted low. The requirement of clearing the internal EJTAGBOOT indication when the Test-Logic-Reset state is entered, and not on a *EJ_TCK* clock when in the state, ensures that the indication can be cleared with five clocks on *EJ_TCK* when *EJ_TMS* is high.

The internal EJTAGBOOT indication is cleared at power-up either by a low value on the *EJ_TRST_N* or by a power-up reset circuit. Thus the processor executes the reset handler after power-up unless the EJTAGBOOT instruction is given through the TAP.

The *Bypass* register is selected when the EJTAGBOOT or NORMALBOOT instruction is given.

The EjtagBrk, ProbEn, and ProbTrap bits in the *EJTAG Control* register follow the internal EJTAGBOOT indication. They are all set at processor reset if a Debug Interrupt exception is to be generated, with execution of the debug handler from the probe.

When an EJTAGBOOT instruction is indicated at reset, then it must be possible to take the debug exception and execute the debug handler from the probe even if no instructions can be fetched from the reset handler. This condition guarantees that the system will not hang in this type of case.

11.6.4.3 FASTDATA Instruction

The FASTDATA instruction selects the Data and the Fastdata registers at the same time as shown in Figure 11-21.

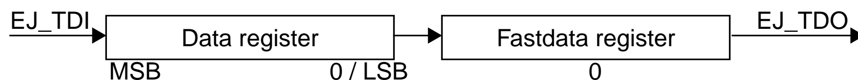


Figure 11-21 EJ_TDI to EJ_TDO Path when in Shift-DR State and FASTDATA Instruction is Selected

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

11.6.5 Data Registers

Table 11-36 summarizes the data registers in the TAP. Complete descriptions of these registers are located in the following subsections.

Table 11-36 EJTAG TAP Data Registers

Instruction Used to Access Register	Register Name	Function	Reference
IDCODE	Device ID	Identifies device and accessed processor in the device.	See Section 11.6.5.1, "Device Identification (ID) Register (TAP Instruction IDCODE)"
IMPCODE	Implementation	Identifies main debug features implemented and accessible through the TAP.	See Section 11.6.5.2, "Implementation Register (TAP Instruction IMPCODE)"
DATA or ALL	Data	Data register for processor access.	See Section 11.6.5.3, "Data Register (TAP Instruction DATA or ALL)"
ADDRESS or ALL	Address	Address register for processor access.	See Section 11.6.5.4, "Address Register (TAP Instruction ADDRESS or ALL)"
CONTROL or ALL	EJTAG Control	Control register for most EJTAG features used through the TAP.	See Section 11.6.5.5, "EJTAG Control Register (ECR) (TAP Instruction CONTROL or ALL)"
FASTDATA	Fastdata	Provides a one-bit register whose value is tagged to the front of the *Data* register to capture the value of the processor access pending (PrAcc) bit in the *EJTAG Control* register.	See Section 11.6.4.3, "FASTDATA Instruction"
BYPASS, EJTAGBOOT, NORMALBOOT or unused EJTAG instructions	Bypass	Provides a one-bit shift path through the TAP.	See Section 11.6.4.2, "EJTAGBOOT and NORMALBOOT Instructions"

A read of a data register corresponds only to the Capture-DR state of the TAP controller, and a write of the data register corresponds to the Update-DR state only.

11.6.5.1 Device Identification (ID) Register (TAP Instruction IDCODE)

The *Device ID* register is a 32-bit read-only register that identifies the specific device implementing EJTAG. This register is also defined in IEEE 1149.1. The register holds a unique number among different devices with EJTAG compliant processors implemented. Figure 11-22 shows the format of the *Device ID* register; Table 11-37 describes the *Device ID* register fields.

Figure 11-22 Device ID Register Format

Table 11-37 Device ID Register Field Descriptions

Fields		Description	Read/ Write	Power-up State
Name	Bits			
Version	31:28	Identifies the version of a specific device.	R	Preset to the state of *EJ_Version[3:0]*

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 11-37 Device ID Register Field Descriptions (Continued)

Fields		Description	Read/ Write	Power-up State
Name	**Bits**			
PartNumber	27:12	Identifies the part number of a specific device.	R	Bits 27:20 are preset to 0x05, and bits 19:16 preset value depends on technology and process. Bits 15:12 are preset to the state of *EJ_PartNumber[3:0]*
ManufID	11:1	Identifies the manufacturer identity code of a specific device. Hardwired to the compressed MIPS JEDEC code.	R	0x127
1	0	Ignored on write; returns one on read.	R	1

11.6.5.2 Implementation Register (TAP Instruction IMPCODE)

The *Implementation* register is a 32-bit read-only register that identifies features implemented in this EJTAG-compliant processor, mainly those accessible from the TAP. Figure 11-23 shows the format of the *Implementation* register; Table 11-38 describes the *Implementation* register fields.

31–29	28	27–25	24	23	22–21	20–17	16	15	14	13–1	0
EJTAGver	R4k/ R3k	0	DINT sup	0	ASID size	0	MIPS 16	0	No DMA	0	MIPS 32/64

Figure 11-23 Implementation Register Format

Table 11-38 Implementation Register Field Descriptions

Fields		Description	Read/ Write	Power-up State
Name	**Bits**			
EJTAGver	31:29	Indicates the EJTAG version: 0: Version 1 and 2.0 1: Version 2.5 2: Version 2.6 3-7: Reserved	R	2
R4k/R3k	28	Indicates R4k or R3k privileged environment: 0: R4k privileged environment 1: R3k privileged environment	R	0
DINTsup	24	Indicates support for *DINT* signal from probe: 0: *DINT* signal from the probe is not supported by this chip 1: Probe can use *DINT* signal to make debug interrupt on this chip	R	1

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 11-38 Implementation Register Field Descriptions (Continued)

Fields		Description	Read/ Write	Power-up State
Name	Bits			
ASIDsize	22:21	Indicates size of the ASID field: 0: No ASID in implementation 1: 6-bit ASID 2: 8-bit ASID 3: Reserved	R	2
MIPS16	16	Indicates MIPS16™ ASE support in the processor: 0: No MIPS16 ASE support 1: MIPS16 ASE is supported	R	0
NoDMA	14	Indicates no EJTAG DMA support: 0: Reserved 1: No EJTAG DMA support	R	1
MIPS32/64	0	Indicates 32-bit or 64-bit processor: 0: 32-bit processor 1: 64-bit processor See the R4k/R3k bit for indication of privileged environment.	R	1
0	27:25, 23, 20:17, 15, 13:1	Ignored on writes; return zeros on reads.	R	0

11.6.5.3 Data Register (TAP Instruction DATA or ALL)

The read/write *Data* register is used for opcode and data transfers during processor accesses. The width of the *Data* register is 64 bits.

The value read in the *Data* register is valid only if a processor access for a write is pending, in which case the data register holds the store value. The value written to the *Data* register is only used if a processor access for a pending read is finished afterwards, in which case the data value written is the value for the fetch or load. This behavior implies that the *Data* register is not a memory location where a previously written value can be read afterwards.

Figure 11-24 shows the format of the *Data* register; Table 11-39 describes the *Data* register field.

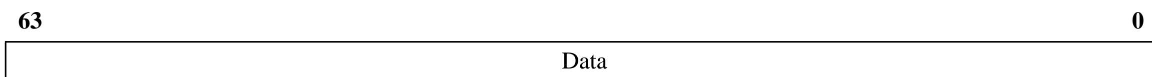


Figure 11-24 Data Register Format

Table 11-39 Data Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	Bits			
Data	63:0	Data used by processor access.	R/W	Undefined

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

The contents of the *Data* register are not aligned but hold data as it is seen on a data bus for an external memory system. Thus the bytes are positioned in the *Data* register based on access size, address, and endianness. The fetch/load/store shifter or other alignment mechanism in the processor performs the required alignment between the *Data* register and the processor; the contents of the *Data* register follow the format as on an external data bus.

The bytes not accessed for a processor access write are undefined, and the bytes not accessed for a processor access read must be written to 0s when the probe shifting the value in provides the *Data* register value.

Table 11-40 shows the byte positioning in which case the three LSBs of the Address register are used together with the Psz field from the *EJTAG Control* register. Byte 0 refers to bits [7:0], byte 1 refers to bits [15:8], and so on up to byte 7, which refers to bits [63:56], independent of endianness.

Table 11-40 Data Register Contents

Psz from ECR	**Size**	**Address[2:0]**	**Little Endian**								**Big Endian**							
			7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	Byte	000_2								■	■							
		001_2							■			■						
		010_2						■					■					
		011_2					■							■				
		100_2				■									■			
		101_2			■											■		
		110_2		■													■	
		111_2	■															■
1	Halfword	000_2							■	■	■	■						
		010_2					■	■					■	■				
		100_2			■	■									■	■		
		110_2	■	■													■	■
2	Word	000_2					■	■	■	■	■	■	■	■				
	5-byte/Quinti	000_2				■	■	■	■	■	■	■	■	■	■			
	6-byte/Sexti	000_2			■	■	■	■	■	■	■	■	■	■	■	■		
	7-byte/Septi	000_2		■	■	■	■	■	■	■	■	■	■	■	■	■	■	
	Word	100_2	■	■	■	■									■	■	■	■
	5-byte/Quinti	011_2	■	■	■	■	■							■	■	■	■	■
	6-byte/Sexti	010_2	■	■	■	■	■	■					■	■	■	■	■	■
	7-byte/Septi	001_2	■	■	■	■	■	■	■			■	■	■	■	■	■	■

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 11-40 Data Register Contents (Continued)

Psz from ECR	Size	Address[2:0]	Little Endian								Big Endian							
			7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
3	Triple	000_2																
		001_2																
		100_2																
		101_2																
	Doubleword	000_2																
Reserved			n.a.								n.a.							

11.6.5.4 Address Register (TAP Instruction ADDRESS or ALL)

The read-only *Address* register provides the address for a processor access. The width of the register corresponds to the size of the physical address in the processor implementation, which is 36 bits.

The value read in the register is valid if a processor access is pending, otherwise the value is undefined.

The three LSBs of the register are used with the Psz field from the EJTAG Control register to indicate the size and data position of the pending processor access transfer. These bits are not taken directly from the address referenced by the load/store. See Section 11.6.5.3, "Data Register (TAP Instruction DATA or ALL)" on page 222 for more details. Figure 11-25 shows the format of the *Address* register; Table 11-41 describes the *Address* register field.

Figure 11-25 Address Register Format

Table 11-41 Address Register Field Descriptions

Fields		Description	Read/ Write	Reset State
Name	Bits			
Address	35:0	Address used by processor access.	R	Undefined

11.6.5.5 EJTAG Control Register (ECR) (TAP Instruction CONTROL or ALL)

The 32-bit EJTAG Control Register (*ECR*) handles processor reset and soft reset indication, Debug Mode indication, access start, finish, and size and read/write indication. The *ECR* also:

- controls debug vector location and indication of serviced processor accesses
- allows debug interrupt request
- indicates processor low-power mode
- allows implementation dependent processor and peripheral reset

The *ECR* is not updated/written in the Update-DR state unless the Reset occurred; that is Rocc (bit 31) is either already 0 or is written to 0 at the same time. This condition ensures proper handling of processor accesses after a reset.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Bits that are R/W in the register return their written value on a subsequent read, unless other behavior is defined. Internal synchronization hardware thus ensures that a written value is updated for reading immediately afterwards, even when the TAP controller takes the shortest path from the Update-DR to Capture-DR state.

Reset of the processor can be indicated in the *EJ_TCK* domain a number of *EJ_TCK* cycles after it is removed in the processor clock domain in order to allow for proper synchronization between the two clock domains.

Figure 11-26 shows the format of the *ECR*; Table 11-42 describes the *ECR* fields.

31	30 29	28 23	22	21	20	19	18	17	16	15	14	13	12	11 4	3	2 0
Rocc	Psz	0	Doze	Halt	PerRst	PRnW	PrAcc	0	PrRst	ProbEn	ProbTrap	0	EjtagBrk	0	DM	0

Figure 11-26 EJTAG Control Register Format

Table 11-42 EJTAG Control Register Field Descriptions

Fields				
Name	**Bits**	**Description**	**Read/ Write**	**Reset State**
Rocc	31	Indicates if a processor reset or soft reset has occurred since the bit was cleared: 0: No reset occurred 1: Reset occurred The Rocc bit stays set as long as reset is applied. This bit must be cleared to acknowledge that the reset was detected. The EJTAG Control register is not updated in the Update-DR state unless Rocc is 0 or written to 0 at the same time. This is in order to ensure correct handling of the processor access after reset.	R/W0	1
Psz	30:29	Indicates the size of a pending processor access in combination with the Address register: 0: Byte 1: Halfword 2: Word, 5-7 bytes 3: Triple, Doubleword A full description is located in Section 11.6.5.3, "Data Register (TAP Instruction DATA or ALL)", including reserved combinations with Address register bits. This field is valid only when a processor access is pending, otherwise the read value is undefined.	R	Undefined
Doze	22	Indicates if the processor is in low-power mode: 0: Processor is not in low-power mode 1: Processor is in low-power mode Doze indicates Reduced Power (RP) and WAIT low-power modes.	R	0
Halt	21	Indicates if the internal system bus clock is running: 0: Internal system bus clock is running 1: Internal system bus clock is stopped Halt indicates a WAIT event in the system.	R	0
PerRst	20	Unused in the 20Kc processor.	R	0

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 11-42 EJTAG Control Register Field Descriptions (Continued)

Fields		Description	Read/ Write	Reset State
Name	Bits			
PRnW	19	Indicates read or write of a pending processor access: 0: Read processor access, for a fetch/load access 1: Write processor access, for a store access This value is defined only when a processor access is pending.	R	Undefined
PrAcc	18	Indicates a pending processor access and controls finishing of a pending processor access. When read: 0: No pending processor access 1: Pending processor access A write of 0 finishes a processor access if pending; otherwise operation of the processor is UNDEFINED if the bit is written to 0 when no processor access is pending. A write of 1 is ignored.	R/W0	0
PrRst	16	Unused in the 20Kc processor.	R	0
ProbEn	15	Controls indication to the processor of whether the probe expects to handle accesses to EJTAG memory through servicing of processors accesses: 0: Probe does not service processors accesses 1: Probe does service processor accesses The ProbEn bit is reflected as a read-only bit in the Debug Control Register (*DCR*) bit 0. When this bit is changed, then it is guaranteed that the new value has taken effect in the *DCR* when it can be read back here. This handshake mechanism ensures that the setting from the *EJ_TCK* clock domain takes effect in the processor clock domain. However, a change of the ProbEn prior to setting the EjtagBrk bit is ensured to affect execution of the debug handler due to the debug exception. Not all combinations of ProbEn and ProbTrap are allowed, see Table 11-43.	R/W	See Section 11.6.5.6, "EJTAGBOOT Indication Determines Reset Value of EjtagBrk, ProbTrap, and ProbEn" for more information.
ProbTrap	14	Controls location of the debug exception vector: 0: Normal memory 0xFFFF FFFF BFC0 0480 1: EJTAG memory 0xFFFF FFFF FF20 0200 When this bit is changed, then it is guaranteed that the new value is indicated to the processor when it can be read back here. This handshake mechanism ensures that the setting from the *EJ_TCK* clock domain takes effect in the processor clock domain. However, a change of the ProbTrap prior to setting the EjtagBrk bit is ensured to affect execution of the debug handler due to the debug exception.	R/W	See Section 11.6.5.6, "EJTAGBOOT Indication Determines Reset Value of EjtagBrk, ProbTrap, and ProbEn" for more information.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 11-42 EJTAG Control Register Field Descriptions (Continued)

Fields		Description	Read/ Write	Reset State
Name	Bits			
EjtagBrk	12	Requests a Debug Interrupt exception to the processor when this bit is written as 1. The debug exception request is ignored if the processor is already in debug at the time of the request. A write of 0 is ignored. The debug request restarts the processor clock if the processor was in a low-power mode, which stopped the processor clock. The read value indicates a pending Debug Interrupt exception requested through this bit: 0: No pending Debug Interrupt exception requested through this bit 1: Pending Debug Interrupt exception This bit is cleared by hardware when the processor enters Debug Mode.	R/W1	See Section 11.6.5.6, "EJTAGBOOT Indication Determines Reset Value of EjtagBrk, ProbTrap, and ProbEn" for more information.
DM	3	Indicates if the processor is in Debug Mode: 0: Processor is in Non-Debug Mode 1: Processor is in Debug Mode	R	0
0	28:23, 17, 13, 11:4, 2:0	Must be written as zeros; return zeros on reads.	0	0

11.6.5.6 EJTAGBOOT Indication Determines Reset Value of EjtagBrk, ProbTrap, and ProbEn

The reset value of the EjtagBrk, ProbTrap, and ProbEn bits follows the setting of the internal EJTAGBOOT indication. If the EJTAGBOOT instruction has been given, and the internal EJTAGBOOT indication is active, then the reset value of the three bits is set (1), otherwise the reset value is clear (0).

The results of setting these bits are:

- A Debug Interrupt exception is requested right after reset because EjtagBrk is set
- The debug handler is executed from the EJTAG memory because ProbTrap is set to indicate debug vector in EJTAG memory at 0xFFFF FFFF FF20 0200
- Service of the processor access is indicated because ProbEn is set

Thus it is possible to execute the debug handler right after reset, without executing any instructions from the normal reset handler.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

11.6.5.7 Combinations of ProbTrap and ProbEn

Use of ProbTrap and ProbEn allows independent specification of the debug exception vector location and availability of EJTAG memory. Behavior for the different combinations is shown in Table 11-43. Note that not all combinations are allowed.

Table 11-43 Combinations of ProbTrap and ProbEn

ProbTrap	ProbEn	Debug Exception Vector	Processor Accesses
0	0	Normal memory at 0xFFFF FFFF BFC0 0480	Not serviced by probe
0	1		Serviced by probe
1	0	If these two bits are changed to this state the operation of the processor is UNDEFINED. This would indicate that the debug exception vector is in EJTAG memory, but the probe will nevertheless not service processor accesses.	
1	1	EJTAG memory at 0xFFFF FFFF FF20 0200	Serviced by probe

11.6.5.8 Fastdata Register (TAP Instruction FASTDATA)

The width of the *Fastdata* register is one bit. During a Fastdata access, the *Fastdata* register is written and read; that is, a bit is shifted in and a bit is shifted out. During a Fastdata access, the *Fastdata* register value shifted in specifies whether the Fastdata access should be completed or not. The value shifted out is a flag that indicates whether the Fastdata access was successful or not (if completion was requested).

Figure 11-27 Fastdata Register Format

Table 11-44 Fastdata Register Field Description

Fields		Description	Read/ Write	Power-Up State
Name	Bits			
SPrAcc	0	Shifting in a zero value requests completion of the Fastdata access. The PrAcc bit in the EJTAG Control register is overwritten with zero when the access succeeds. (The access succeeds if PrAcc is one and the operation address is in the legal dmseg Fastdata area.) When successful, a one is shifted out. Shifting out a zero indicates a Fastdata access failure. Shifting in a one does not complete the Fastdata access and the PrAcc bit is unchanged. Shifting out a one indicates that the access would have been successful if allowed to complete, and a zero indicates the access would not have successfully completed.	R/W	Undefined

The FASTDATA access is used for efficient block transfers between dmseg (on the probe) and target memory (on the processor). An “upload” is defined as a sequence of processor loads from target memory and stores to dmseg. A “download” is a sequence of processor loads from dmseg and stores to target memory. The “Fastdata area” specifies the legal range of dmseg addresses (0xF..F20.0000 – 0xF..F20.000F) that can be used for uploads and downloads. The *Data* + *Fastdata* registers (selected with the FASTDATA instruction) allow efficient completion of pending Fastdata area accesses.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

During Fastdata uploads and downloads, the processor stalls on accesses to the Fastdata area. The PrAcc (processor access pending bit) is 1, indicating the probe is required to complete the access. Both upload and download accesses are attempted by shifting in a zero SPrAcc value (to request access completion) and shifting out SprAcc to see if the attempt is successful (that is, there was an access pending and a legal Fastdata area address was used). Downloads also shift in the data to be used to satisfy the load from dmseg's Fastdata area, while uploads shift out the data being stored to dmseg's Fastdata area.

As noted above, two conditions must be true for the Fastdata access to succeed:

- PrAcc must be 1 (there must be a pending processor access)
- The Fastdata operation must use a valid Fastdata area address in dmseg (0xF..F20.0000 – 0xF..F20.0000F)

Table 11-45 shows the values of the PrAcc and SPrAcc bits and the results of a Fastdata access.

Table 11-45 Operation of the FASTDATA Access

Probe Operation	Address Match Check	PrAcc in the Control Register	LSB (SPrAcc) Shifted In	Action in the Data Register	PrAcc Changes to	LSB Shifted Out	Data Shifted Out
Download using FASTDATA	Fails	x	x	None	Unchanged	0	Invalid
	Passes	1	1	None	Unchanged	1	Invalid
		1	0	Write Data	0 (SPrAcc)	1	Valid (Previous) Data
		0	x	None	Unchanged	0	Invalid
Upload using FASTDATA	Fails	x	x	None	Unchanged	0	Invalid
	Passes	1	1	None	Unchanged	1	Invalid
		1	0	Read Data	0 (SPrAcc)	1	Valid Data
		0	x	None	Unchanged	0	Invalid

There is no restriction on the contents of the *Data* register. It is expected that the transfer size is negotiated between the download/upload transfer code and the probe software. Note that the most efficient transfer sizes are word and doubleword for 32-bit and 64-bit processors, respectively.

The Rocc bit of the *Control* register is not used for the FASTDATA operation.

11.6.5.9 Bypass Register (TAP Instruction BYPASS, EJTAGBOOT, NORMALBOOT or Unused)

The *Bypass* register is a one-bit read-only register, which provides a minimum shift path through the TAP. This register is also defined in IEEE 1149.1. Figure 11-28 shows the format of the *Bypass* register; Table 11-46 describes the *Bypass* register field.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

0
0

Figure 11-28 Bypass Register Format

Table 11-46 Bypass Register Field Description

Fields		Description	Read/ Write	Power-up State
Name	Bits			
0	0	Ignored on writes; returns zero on reads.	R	0

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

20Kc Test Features

This chapter describes the Enhanced Joint Test Action Group (EJTAG) debug features supported by the 20Kc processor and contains the following sections:

- Section 12.1, "Cache Test Mode"
- Section 12.2, "PLL Bypass Mode"
- Section 12.3, "BIST (Built-In Self Test)"

12.1 Cache Test Mode

The 20Kc processor provides a cache test mode that can be used during manufacturing test and debug to directly access the following internal RAM arrays:

- Data Cache (DCache) data array
- Data Cache (DCache) tag array
- Instruction Cache (ICache) data array
- Instruction Cache (ICache) tag array

12.1.1 Cache Test Mode Interface Signals

Cache test mode is accessed by using a subset of the system interface signals. The following system interface signals are used during cache test mode:

- *EB_SysAD[31:0]*
- *EB_SysADP[3:0]*
- *EB_PrcAD[31:0*
- *EB_PrcADP[3:0]*
- *EB_SysVld*
- *SI_PLLBypass*
- *TS_TCmd[3:0]*
- *SI_Reset*
- *SI_ColdReset*

12.1.2 System Interface Clock Divisor and Mode

Cache Test Mode is only supported when the PLL is bypassed. In this mode, the *SI_ClkIn* to CPU clock ratio is effectively 1:1. In order to bypass the PLL, the *SI_PLLBypass* input pin must be asserted. In order to distinguish from ordinary PLL Bypass mode operation, the *TS_TCmd[3:0]* bits are used. A setting of 0010 on the *TS_TCmd[3:0]* inputs indicates Cache Test Mode.

Cache Test Mode operation is only guaranteed up to a maximum frequency of 100Mhz.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Cache Test Mode is only operational in synchronous mode and cannot be used in source synchronous double data rate mode.

12.1.3 Entering Cache Test Mode

Entry into Cache Test Mode happens through a mechanism similar to the Power-On Reset Sequence or a Cold Reset sequence. During this sequence the *SI_PLLBypass* input pin must be asserted and the *TS_TCmd[3:0]* pins must be set to 0010. One key difference from the Power-On Reset Sequence or Cold Reset Sequence is that the *SI_Reset* and *SI_ColdReset* signals are kept asserted through the Cache Test Mode operation.

If the entry into Cache Test Mode is via the beginning of a Power-On sequence, the following sequence is applied:

1. All Input pins are held low.
2. *Vdd* and *VddP* are applied to the part.
3. *SI_ClkIn* is applied concurrent with or after *Vdd*.
4. *SI_PLLBypass* and *TS_TCmd[1]* are asserted concurrent with or after *Vdd*.
5. Once *Vdd*, *VddP*, and *SI_SysClkIn* are stable and 5 *SI_ClkIn* cycles after *SI_PLLBypass* and *TS_TCmd[1]* are stable, *SI_VddOk* is asserted.
6. 100 ns after *SI_VddOk* is asserted, the first Cache Test Mode command can be issued.

In order to stay in Cache Test Mode, the external testing agent, must keep *SI_Reset* and *SI_ColdReset* asserted. The different cache arrays may then be tested by appropriately controlling the *EB_SysAD[31:0], EB_SysADP[3:0]* and *EB_SysVld* pins. Any results are observed on the *EB_PrcAD[31:0]* and *EB_PrcADP[3:0]* pins.

Figure 12-1 shows an entry into Cache Test Mode via the beginning of a Power-On Reset sequence.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

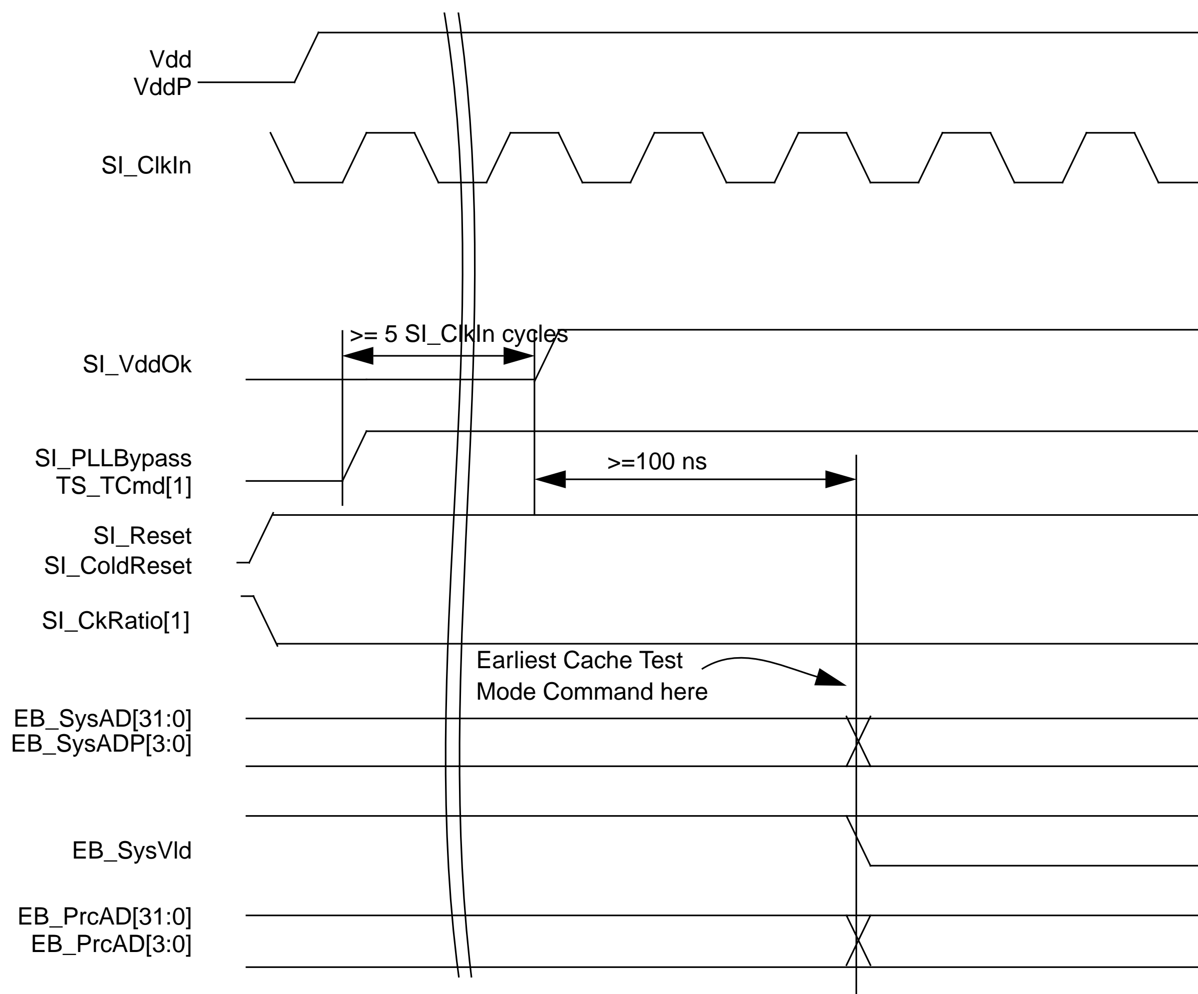


Figure 12-1 Entering Cache Test Mode After a Power-On Reset Sequence

If the entry into Cache Test Mode is via the beginning of a ColdReset sequence the following sequence is applied:

1. *SI_ColdReset* and *SI_Reset* are asserted.
2. *SI_VddOk* is deasserted after *SI_ColdReset* and *Reset* are asserted.
3. *SI_PLLBypass* is asserted and *TS_TCmd[3:0]* is set to 0010.
4. A minimum of 100 ns after *SI_PLLBypass* and *TS_TCmd[3:0]* are stable, *SI_VddOk* is asserted.
5. The first Cache test mode command can be presented at the pins five *SI_ClkIn* cycles after *SI_VddOk* is asserted.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

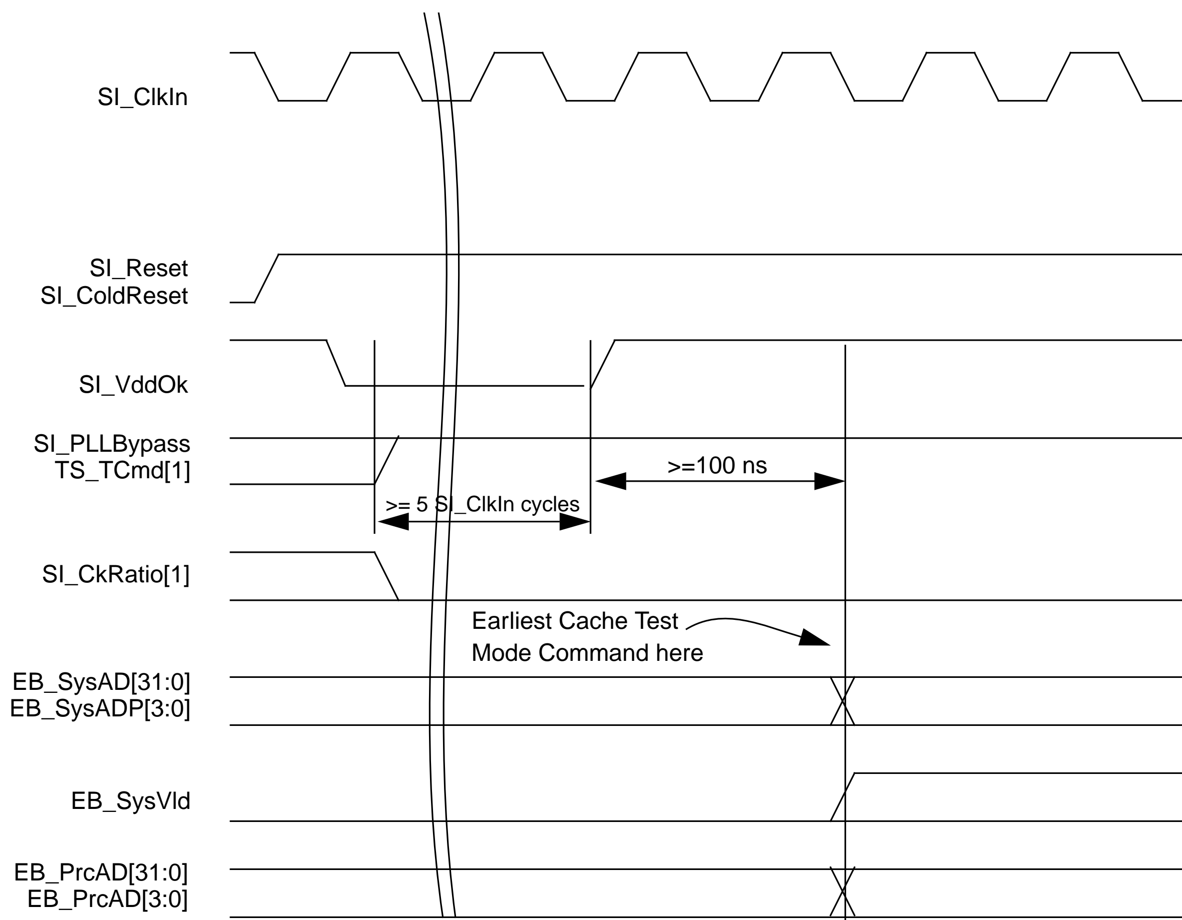


Figure 12-2 Entry into Cache Test Mode during a ColdReset Sequence

12.1.4 Exit from Cache Test Mode

It is possible to exit Cache Test Mode and resume normal operation by following the steps for a Cold Reset sequence as described in Chapter 9, “Reset and Initialization.” The following steps summarize the sequence of events required:

- *SI_VddOK* is deasserted.
- The configuration signals *SI_PLLBypass* and *TS_TCmd[3:0]* are reset to their desired state.
- Once *SI_PLLBypass* and *TS_TCmd[3:0]* have been stable for at least five *SI_ClkIn* cycles, *SI_VddOK* is asserted.
- *SI_Reset* and *SI_ColdReset* are then deasserted no sooner than 120 microseconds after the assertion of *SI_VddOK*, after which normal operation is resumed.

12.1.5 Cache Test Mode Commands

There are three supported cache test mode commands:

- **Normal Read:** This command can be used to read any one of the four different cache arrays. The
- **Normal Write:** This command can be used to write any one of the four different cache arrays.
- **Write Same Data**: This command can be used to write any one of the four different cache arrays.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

12.1.6 Read/Write Granularity

The read/write granularity depends on the operation being performed.

12.1.6.1 Read Granularity

For the purpose of read operations, the cache arrays are treated as word-addressed memories. A read operation can read a maximum of 36 bits for each command.

In the case of the ICache and DCache data arrays, this corresponds to a word of data and its associated 4-bit, byte parity field. The actual word within a 32-byte cache line that is read is determined by bits [4:2] of the address provided.

Since each entry in the DCache Tag is only 128 bits wide, the read operation on the DCache Tag results in a read quantum of 28 bits. The line whose corresponding tag field is read is determined by bits [12:5] of the address provided.

The ICache Tag is treated in a special way. Since each entry of the ICache Tag contains 46 bits, it is treated as a doubleword where a read of the lower word, as specified by setting address bit [4] to zero provides the lower word of the tag, whereas a read of the upper word, which is specified by setting address bit [4] to 1 provides the remaining eight bits of the tag.

12.1.6.2 Write Granularity

A write of both the data tag and data arrays write an entire cache line worth of data. Thus the write granularity of the DCache and ICache Data Arrays is equal to 32 bytes of data and its associated four bytes of parity.

The write granularity of the DCache Tag Array is therefore 28 bits. The write granularity of the ICache Tag Array is therefore 46 bits.

12.1.7 Encodings

The encoding of the signals involved in the transactions depends on the type of bus cycle. There are two types of bus cycles:

- Address/Command Cycle - These are used to transfer the actual command.
- Data Cycles - These are used to transfer read/write data.

Table 12-1 SysAD[31:0] Encoding for Address/Command Cycles

<table>
<tr><th>EB_SysAD Bit</th><th>DCache Data Array
Read</th><th>DCache Tag Array
Read</th><th>ICache Data Array
Read</th><th>ICache Tag Array
Read</th><th>DCache Data Array
Write</th><th>DCache Tag Array
Write</th><th>ICache Data Array
Write</th><th>ICache Tag Array
Write</th></tr>
<tr><td>1:0</td><td>Unused</td><td rowspan="4">Unused</td><td>Unused</td><td rowspan="3">Unused</td><td rowspan="4">Unused</td><td rowspan="4">Unused</td><td rowspan="4">Unused</td><td rowspan="4">Unused</td></tr>
<tr><td>2</td><td rowspan="4">Address</td><td rowspan="4">Address</td></tr>
<tr><td>3</td></tr>
<tr><td>4</td><td rowspan="2">Address</td></tr>
<tr><td>12:5</td><td>Address</td><td>Address</td><td>Address</td><td>Address</td><td>Address</td></tr>
<tr><td>14:13</td><td>Way</td><td>Way</td><td>Way</td><td>Way</td><td>Way</td><td>Way</td><td>Way</td><td>Way</td></tr>
<tr><td>15</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td></tr>
</table>

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 12-1 SysAD[31:0] Encoding for Address/Command Cycles (Continued)

EB_SysAD Bit	DCache Data Array Read	DCache Tag Array Read	ICache Data Array Read	ICache Tag Array Read	DCache Data Array Write	DCache Tag Array Write	ICache Data Array Write	ICache Tag Array Write
17:16	00	10	01	11	00	10	01	11
18	Unused	Unused	Unused	Unused	Write Same Data	Write Same Data	Write Same Data	Write Same Data
31:19					Unused	Unused	Unused	Unused

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 12-2 SysAD[31:0] Encoding for Write Data Cycles

<table>
<tr><th>EB_SysAD bit</th><th>D/I Cache Data Array
Write Cycle 0-7</th><th>DCache Tag Array
Write Cycle 0</th><th>DCache Tag Array
Write Cycle 1</th><th>ICache Tag Array
Write Cycle 0</th><th>ICache Tag Array
Write Cycle 1</th></tr>
<tr><td>2:0</td><td rowspan="22">Data</td><td rowspan="13">Tag</td><td rowspan="22">Unused</td><td rowspan="17">Tag</td><td>Asid[7:5]</td></tr>
<tr><td>3</td><td>Global</td></tr>
<tr><td>4</td><td>Valid</td></tr>
<tr><td>5</td><td>LRF</td></tr>
<tr><td>6</td><td>Lock</td></tr>
<tr><td>7</td><td>Endian</td></tr>
<tr><td>8</td><td>FetchSeg[63]</td></tr>
<tr><td>9</td><td>FetchSeg[62]</td></tr>
<tr><td>10</td><td>FetchSeg[61]</td></tr>
<tr><td>11</td><td>FetchSeg[60]</td></tr>
<tr><td>12</td><td>FetchSeg[59]</td></tr>
<tr><td>13</td><td>Parity</td></tr>
<tr><td>22:14</td><td rowspan="10">Unused</td></tr>
<tr><td>23</td><td>Valid</td></tr>
<tr><td>24</td><td>Dirty</td></tr>
<tr><td>25</td><td>LRF</td></tr>
<tr><td>26</td><td>Lock</td></tr>
<tr><td>27</td><td>Parity</td><td rowspan="5">Asid[4:0]</td></tr>
<tr><td>28</td><td rowspan="4">Unused</td></tr>
<tr><td>29</td></tr>
<tr><td>30</td></tr>
<tr><td>31</td></tr>
</table>

Table 12-3 EB_SysADP[3:0] Encoding for Write Data Cycles

<table>
<tr><th>EB_SysADP bit</th><th>D/ICache Data Array Write Cycle 0-7</th><th>D/ICache Tag Array Write Cycle 0-1</th></tr>
<tr><td>0</td><td>Parity for EB_SysAD[7:0]</td><td rowspan="4">Unused</td></tr>
<tr><td>1</td><td>Parity for EB_SysAD[15:8]</td></tr>
<tr><td>2</td><td>Parity for EB_SysAD[23:16]</td></tr>
<tr><td>3</td><td>Parity for EB_SysAD[31:24]</td></tr>
</table>

Note: The *SysADP[3:0]* signals represent the data that is written into the parity fields of the caches. However, it is not required that the data transferred on this bus during Cache Test Mode represent actual calculated parity.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 12-4 EB_PrcAD[31:0] Encoding for Read Data Cycles

<table>
<tr><th>EB_PrcAD Bit</th><th>D/ICache Data Array</th><th>DCache Tag Array</th><th>ICache Tag Array (Address bit 4 = 0)</th><th>ICache Tag Array (Address bit 4 = 1)</th></tr>
<tr><td>2:0</td><td rowspan="21">Data</td><td rowspan="13">Tag</td><td rowspan="17">Tag</td><td>Asid[7:5]</td></tr>
<tr><td>3</td><td>Global</td></tr>
<tr><td>4</td><td>Valid</td></tr>
<tr><td>5</td><td>LRF</td></tr>
<tr><td>6</td><td>Lock</td></tr>
<tr><td>7</td><td>Endian</td></tr>
<tr><td>8</td><td>FetchSeg[63]</td></tr>
<tr><td>9</td><td>FetchSeg[62]</td></tr>
<tr><td>10</td><td>FetchSeg[61]</td></tr>
<tr><td>11</td><td>FetchSeg[60]</td></tr>
<tr><td>12</td><td>FetchSeg[59]</td></tr>
<tr><td>13</td><td>Parity</td></tr>
<tr><td>22:14</td><td rowspan="9">Unused</td></tr>
<tr><td>23</td><td>Valid</td></tr>
<tr><td>24</td><td>Dirty</td></tr>
<tr><td>25</td><td>LRF</td></tr>
<tr><td>26</td><td>Lock</td></tr>
<tr><td>27</td><td>Parity</td><td rowspan="5">Asid[4:0]</td></tr>
<tr><td>28</td><td rowspan="4">Unused</td></tr>
<tr><td>29</td></tr>
<tr><td>30</td></tr>
<tr><td>31</td></tr>
</table>

Table 12-5 EB_PrcADP[3:0] Encoding for Read Cycles

EB_PrcADP bit	D/ICache Data Array Read Cycles
0	Parity for EB_PrcAD[7:0]
1	Parity for EB_PrcAD[15:8]
2	Parity for EB_PrcAD[23:16]
3	Parity for EB_PrcAD[31:24]

Note: The *EB_PrcADP[3:0]* signals represent the data that is written into the parity fields of the caches. However, it is not required that the data transferred on this bus during Cache Test Mode represent actual calculated parity.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

12.1.8 Protocols

12.1.8.1 Normal Read

This command is used to read any one of the four different cache arrays, by applying an address and the appropriate command on the *EB_SysAD[31:0]* pins. The *EB_SysVld* signal is asserted to indicate a valid Command/Address cycle.

The read data can then be observed for a single cycle on the *EB_PrcAD[31:0]* and *EB_PrcADP[3:0]* pins a certain number of cycles later. The read latency is variable and depends on the array being read. Table 12-6 shows the latencies for the different cache arrays. The *EB_SysAD[3:0]* bus is unused during a normal read cycle.

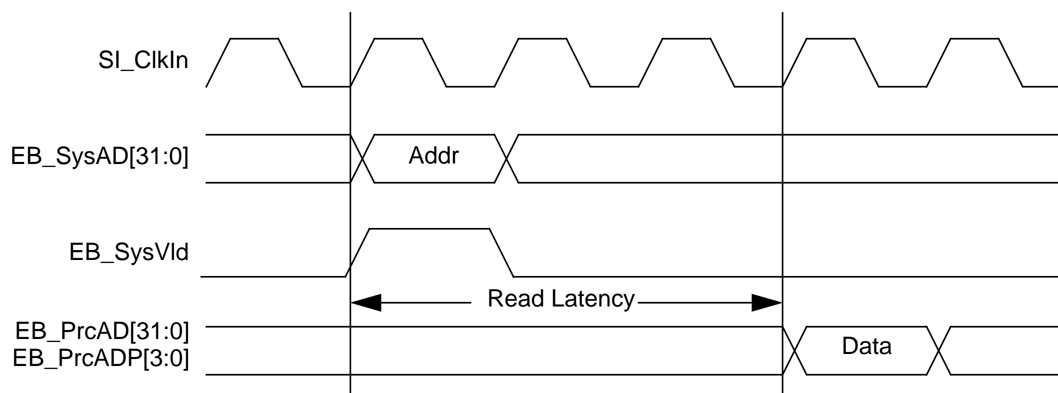


Figure 12-3 Normal Read Cycle

Table 12-6 Cache Test Mode Read Latency

Cache Array	Read Latency in System Clocks
Data Cache Data Array	8
Data Cache Tag Array	9
Instruction Cache Data Array	8
Instruction Cache Tag Array	8

12.1.8.2 Normal Write

This command can be used to write any one of the four different cache arrays, by applying an address and the appropriate command on the *EB_SysAD[31:0]* pins and by providing the data to be written in subsequent cycles on the *EB_SysAD[31:0]* and *EB_SysADP[3:0]* pins. *EB_SysVld* is asserted during the Address/Command cycle, but is not asserted during subsequent data cycles. The last data cycle must be followed by an empty cycle, where no where no new command is issued.The number of data cycles is dependent on the array being written. Table 12-7 indicates the number of data cycles required for each array.

Table 12-7 Normal Write Command

Cache Array	Number of Data Cycles	Rep Rate Cycles
Data Cache Data Array	8	10
Data Cache Tag Array	2	4
Instruction Cache Data Array	8	10

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 12-7 Normal Write Command (Continued)

Cache Array	Number of Data Cycles	Rep Rate Cycles
Instruction Cache Data Array	2	4

Figure 12-4 shows the Normal Write sequence used for Data and Instruction Cache Tag Write commands. The equivalent figure for Data and Instruction Cache Data Writes can be extrapolated by adding the appropriate number of data cycles.

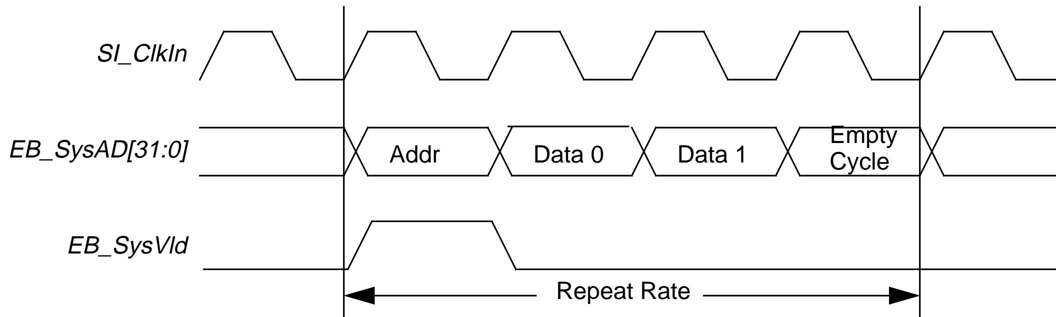


Figure 12-4 Data/Instruction Cache Tag Normal Write

12.1.8.3 Write Same Data

This command can be used to write any one of the four different cache arrays, by applying an address and the appropriate command on the *EB_SysAD[31:0]* pins. The *EB_SysVld* signal is asserted to indicate a valid Command/Address cycle. The data to be written into the arrays is obtained from the last Normal Write Command. The Write Same Data command must be preceded by either a Normal Write Command or another Write Same Data Command. The cache array referenced by the Write Same Data command cannot be different from the one referenced by the prior Normal Write Command or Write Same Data Command.

The Write Same Data Command is identical for all four arrays.

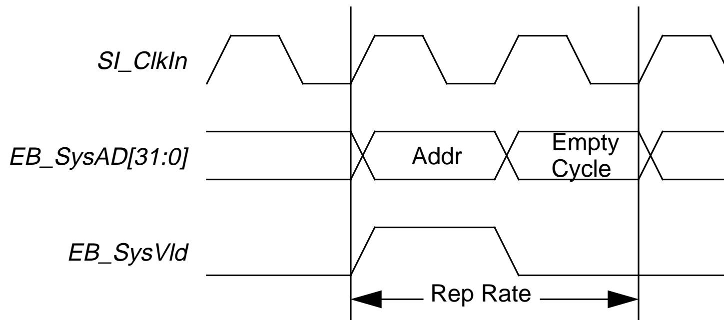


Figure 12-5 Write Same Data Command

12.2 PLL Bypass Mode

PLL Bypass Mode can be entered via a Power-On Reset Sequence or a Cold Reset Sequence. These sequences are described in the chapter on Reset and Initialization. To put the 20Kc processor in PLL Bypass Mode, the *SI_PLLBypass* pin must be asserted and the *SI_CkRatio[2:0]* pins must be set to the value of 000. As shown in the above mentioned sequences, these configuration pins can be toggled only when *SI_VddOk* is deasserted and the earliest that *SI_VddOk* can be reasserted is 100 ns after one of these inputs has toggled.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

12.3 BIST (Built-In Self Test)

This section describes the implementation of Built-In Self Test (BIST) for the Cache Memory Arrays on the 20Kc core. The BIST approach described here is for at-speed testing of Tag and Data RAM arrays in the Instruction and Data Cache on the 20Kc core. This approach is not intended for smaller memory arrays on the 20Kc core, such as register files or TLB.

12.3.1 Overview

Figure 12-6 shows a high-level block diagram of the BIST architecture. The BIST module applies a series of patterns to the Memory Module and then compares the outputs against a set of expected responses. The patterns are based on the March family of algorithms and are highly regular and deterministic. This allows the use of a simple comparator to flag a failure. The signals from the BIST module to the Memory Module are muxed with the normal input signals to the Memory. The location of these muxes has been chosen to minimize the impact on the timing of the normal signals as well as the existing layout of the caches.

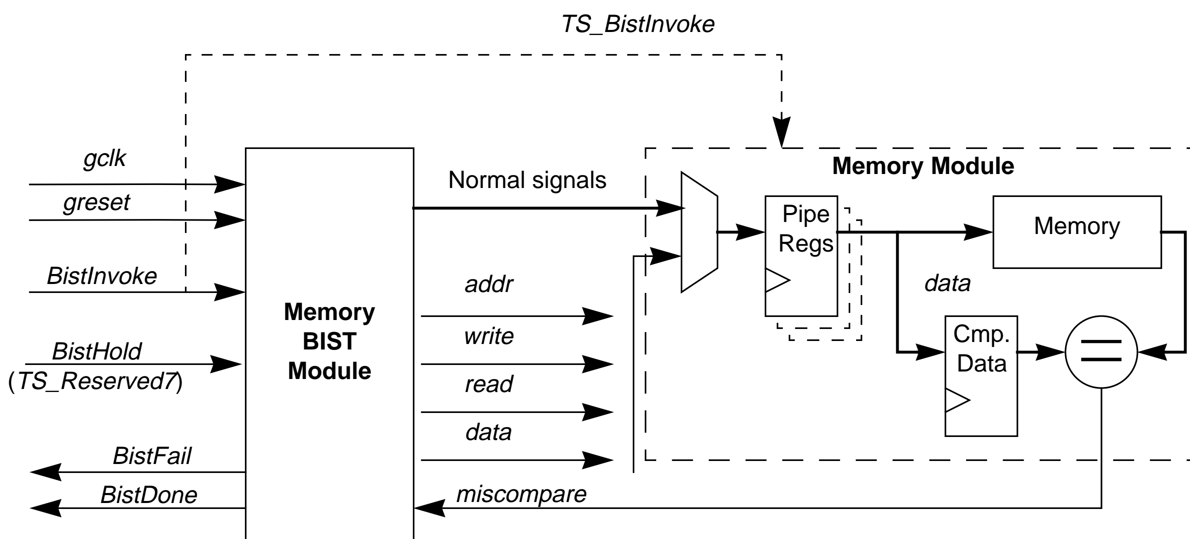


Figure 12-6 Integration of BIST with Memory

The current implementation of memory BIST provides only a Pass/Fail indication on a device. An enhancement to provide failure diagnostics at the very first failure can be implemented. This can be done by latching the necessary signals at the first failure point. These signals could include the address of the memory location, the data output from the memory and the actual operation within the test algorithm being applied. This latched data can then be scanned out serially.

Several memories can be tested in parallel with a single memory BIST module, in which case the address, data, and control must be distributed to all the tested memories, and a single compare result must be the generated for the miscompare signal. Parallel testing can reduce test time. On the other hand, since many memory arrays are active at the same time, parallel testing also consumes more power, and in some cases, this might cause BIST mode to exceed the power specification of the chip.

On the 20Kc processor, the Data and Instruction Caches are tested in parallel by independent BIST modules and a single miscompare result is generated for the miscompare signal. The Data and Instruction Caches are organized in multiple ways and banks, which also can be accessed in parallel. However, bank or way parallel testing is not possible without significant redesign because the output data from multiple banks is muxed onto a common output data bus. For this reason, the intention is to test the banks within each cache sequentially.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

12.3.1.1 Interface Signals

The chip-level interface signals needed for Memory BIST are listed in Table 12-8.

Table 12-8 Chip-Level Memory BIST Interface for the 20Kc Processor

Signal Name	Dir	Description
TS_BistInvoke	I	BIST testing is initiated when this signal is asserted. BIST is terminated when this signal is deasserted.
TS_BistHold	1	1) Selects BIST algorithm in the first cycle the *TS_BistInvoke* is asserted. 2) Controls the retention test duration during the delay period of the IFA-13 algorithm
TS_BistFail	O	Asserted to indicate that the test has failed.
TS_BistDone	O	Asserted when the test is finished, either successfully or with fail.

BIST testing is done while reset is applied to the processor. The *TS_BistInvoke* signal must be asserted and deasserted while reset is applied. The test is finished when *TS_BistDone* is asserted. Once *TS_BistDone* is asserted, *TS_BistDone* and *TS_BistFail* maintain their status as long as TS_BistInvoke is asserted. The *TS_BistFail* and *TS_BistDone* signals are deasserted a few clock cycles after the deassertion of *TS_BistInvoke*. *TS_BistHold* selects the BIST algorithm and extends the delay periods of the IFA-13 algorithm. If *TS_BistHold* is asserted with *TS_BistInvoke*, the IFA-13 BIST algorithm is selected; otherwise the March C+ algorithm is selected. The algorithms are described in Section 12.3.2, "Algorithms for Memory Test".

12.3.1.2 External Signal Behavior

The Figure 12-7 show the waveform to select IFA-13 and March C+ algorithm. The Figure 12-8 shows an overall waveform for BIST testing using March C+.

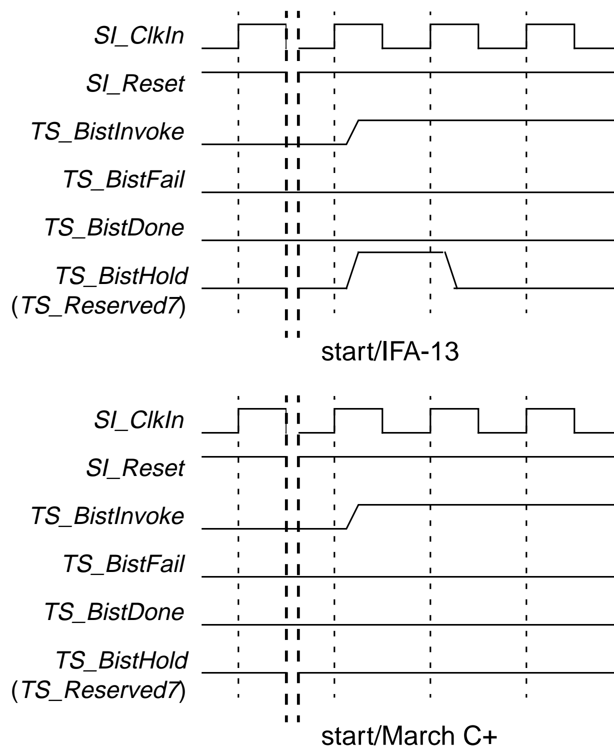


Figure 12-7 BIST algorithm selection

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

An example of external signal behavior for a memory test is shown in Figure 12-8.

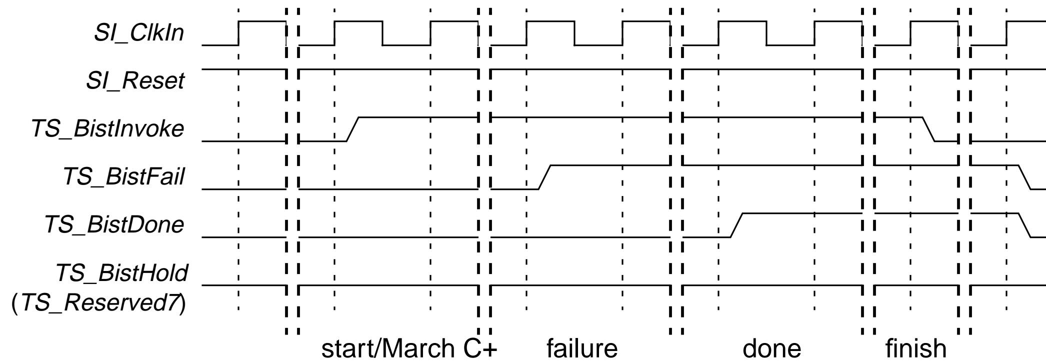


Figure 12-8 External Signal Behavior for a Memory Test

The following notation describes how TS_BistHold(TS_Reserved7) needs to be asserted for retention testing.

Legend :

w : wait with *TS_BistHold*(*TS_Reserved7*) = 0
hzero : assert *TS_BistHold*(*TS_Reserved7*) as required for retention 0 testing
hone : assert *TS_BistHold*(*TS_Reserved7*) as required for retention 1 testing

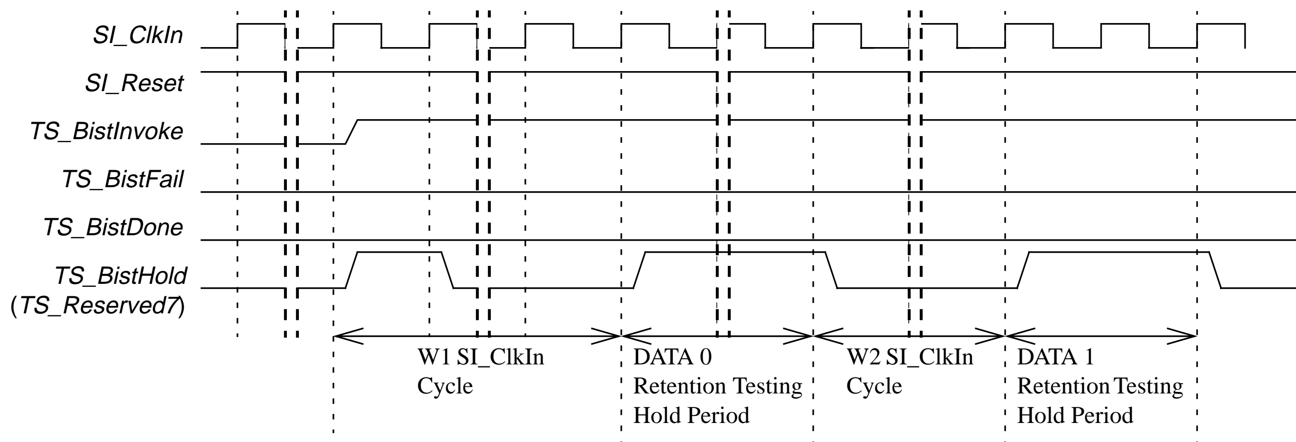
Numbers are in internal clock cycles. In order to find the numbers in sys clock(*SI_ClkIn*) cycles, divide them with the system ratio. The waveform starts when BistInvoke is asserted.

<6681 w><hzero><1027 w><hone><7208 w><hzero><1027 w><hone><7208 w><hzero><1027 w><hone>
<7208 w><hzero><1027 w><hone><7208 w><hzero><1027 w><hone><7208 w><hzero><1027 w><hone>
<7208 w><hzero><1027 w><hone><7208 w><hzero><1027 w><hone>
<3886 w><hzero><515 w><hone><3624 w><hzero><515 w><hone><3624 w><hzero><515 w><hone>
<3624 w><hzero><515 w><hone><3624 w><hzero><515 w><hone><3624 w><hzero><515 w><hone>
<3624 w><hzero><515 w><hone><3624 w><hzero><515 w><hone>

Note: The above description does not include the assertion of *TS_BistHold*(*TS_Reserved7*) for IFA-13 Algorithm selection when *TS_BistInvoke* is first asserted.

Figure 12-9 shows the retention testing waveform

Copyright © 2002 MIPS Technologies Inc. All rights reserved.



W1 = 6681 *SI_ClkIn* Cycles for PLL Bypass Mode
= 3340 *SI_ClkIn* Cycles for Sys Ratio of 2
= 2227 *SI_ClkIn* Cycles for Sys Ratio of 3
W2 = 1027 *SI_ClkIn* Cycles For PLL ByPass Mode
= 513 *SI_ClkIn* Cycles For Sys Ratio of 2
= 342 *SI_ClkIn* Cycles For Sys Ratio of 3

Figure 12-9 Retention Testing Example Waveform

The *BistFail* and *BistDone* signals indicate the progress and status of the test when the test is ongoing as shown in Table 12-9.

Table 12-9 Status and Progress Indications

BistFail	BistDone	Description
0	0	Test is ongoing, and no fail is detected yet.
1	0	Test is ongoing, and fail was detected.
0	1	Test is done, and no fail was detected during the test.
1	1	Test is done, and fail was detected during the test.

A passing test, with no compare errors, is not indicated on the *BistFail* signal until the *BistDone* signal is asserted; however, the *BistFail* signal is sticky so a failing test can be seen by an asserted *BistFail* signal as soon as the miscompare is detected.

12.3.2 Algorithms for Memory Test

For a hard core design the algorithm used for BIST is embedded within the design and once the BIST hardware is generated, the algorithm is fixed and unchangeable. For this reason the BIST algorithm needs to be carefully selected to cover the most commonly found defects in memory arrays in the target fabrication facility.

The March family of algorithms is commonly used for memory BIST.

The notation for these algorithms uses a number of March elements each with a set of operations.

The operations in each march element is either a bit write of 0, w0, or a bit write of 1, w1, or a bit read expected to be either 0, r0, or expected to be 1, r1.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

The march test notation prescribes that the memory is accessed bitwise.

An example of a march test is shown in Figure 12-10.

$$\{ \Uparrow(w0); \Downarrow(r0,w1) \}$$
$$M_0 \quad M_1$$

Figure 12-10 Example of March Test Written in March Test Notation

The test in the example has two march elements, M_0 and M_1, and requires three operations per memory bit, first doing one operation per bit for increasing addresses, and then doing two operations per bit for decreasing addresses.

12.3.2.1 March C+ Algorithm

The March C+ covers the following faults:

- Address faults
- Stuck at faults
- Transition faults
- Coupling faults

The March C+ algorithm requires 14 operations for each memory bit. Figure 12-11 shows the test using March test notation.

$$\{ \Uparrow(w0); \Uparrow(r0,w1,r1); \Uparrow(r1,w0,r0); \Downarrow(r0,w1,r1); \Downarrow(r1,w0,r0); \Downarrow(r0) \}$$
$$M_0 \quad M_1 \quad M_2 \quad M_3 \quad M_4 \quad M_5$$

Figure 12-11 March C+ Algorithm in March Test Notation

12.3.2.2 IFA-13 Algorithm

The IFA-13 algorithm requires 16 operations for each memory location and has two delays in the algorithm. The algorithm tests the same faults as the March C+ algorithm and additionally tests for data retention faults.

The IFA-13 algorithm is similar to the March C+ algorithm with additional delays and march elements, thus requiring 16 operations for each memory bit and additional cycles for the retention delay. The algorithm is shown in Figure 12-12 using March test notation.

$$\{ \Uparrow(w0); \Uparrow(r0,w1,r1); \Uparrow(r1,w0,r0); \Downarrow(r0,w1,r1); \Downarrow(r1,w0,r0); delay; \Downarrow(r0); \Downarrow(w1); delay; \Downarrow(r1); \}$$
$$M_0 \quad M_1 \quad M_2 \quad M_3 \quad M_4 \quad M_5 \quad M_6 \quad M_7$$

Figure 12-12 IFA-13 Algorithm in March Test Notation

The BIST controllers on the 20Kc core apply to either the March C+ or IFA-13 algorithms for BIST. If *TS_BistHold* is asserted in the first cycle that *TS_BistInvoke* is asserted, the IFA-13 algorithm is selected; otherwise the March C+ algorithm is selected. After the BIST algorithm selection, *TS_BistHold* can extend the one cycle delay periods of the IFA-13 algorithm. In particular when *TS_BistHold* is asserted during a delay period, it pauses both BIST controllers of the 20Kc core. During any other period, *TS_BistHold* is ignored.

12.3.3 BIST Integration on 20Kc Cache Memories

For BIST purposes, the 20Kc cache memories can be partitioned into four memory modules:

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

- Data Cache RAM Array: 32 KBytes in size
- Data Cache Tag Array: 4 x 256 x 28 bits in size
- Instruction Cache RAM Array: 32 KBytes in size
- Instruction Cache Tag Array: 4 x 256 x 46 bits in size

12.3.4 Cycles for Memory BIST Testing

The Instruction Cache is tested in parallel with the Data Cache. However, within each cache, the RAM and Tag arrays are tested sequentially. The read and write accesses during BIST are done along these granularities for the four memory arrays:

- Data Cache RAM Array: dword (8 bytes or 72 bits)
- Data Cache Tag Array: 28 bits
- Instruction Cache RAM Array: half line (16 bytes or 144 bits)
- Instruction Cache Tag Array: 46 bits

The total number of memory units tested is:

- DCache_Ram = 4096
- ICache_Ram = 2048
- DCache_Tag = (4 * 256) =1024
- ICache_Tag = (4 * 256) = 1024

The total number of cycles for a March C+ test using 14 operations per memory unit and running the DCache and ICache in parallel is then:

$Cycle_{March\ C+} = 14 * (4096+1024) = 71680$ cycles

The number of cycles for an IFA-13 test using 16 operations per bit plus delay is then:

$Cycles_{IFA-13} = 81920$ cycles + 4 * 2 * 8 delay cycles

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Instruction Set Architecture

The 20Kc ISA is fully compliant with the MIPS64 instruction set. The MIPS64 ISA includes the following instructions:

- The MIPS V CPU instructions
- The MIPS V FPU instructions
- A set of new instructions targeted at embedded applications.
- The instructions that act as the ISA interface to the MIPS Privileged Resource Architecture

This specification does not describe many of these instructions in any detail because it is assumed that the reader also has access to the most recent copy of the two-volume set entitled MIPS RISC Architecture. Only differences and additions are described in this document.

This chapter contains the following sections:

- Section 13.1, "CPU Architecture"
- Section 13.2, "FPU Architecture"
- Section 13.3, "Coprocessor Architecture"
- Section 13.4, "Privileged Instruction Set Architecture"
- Section 13.5, "EJTAG Support Instructions"
- Section 13.6, "Instruction Bit Encoding"
- Section 13.7, "MIPS64 Instruction Descriptions"

13.1 CPU Architecture

13.1.1 CPU Register Overview

The MIPS64 Architecture defines the following CPU registers:

- 32 64-bit general purpose registers (GPRs)
- a pair of special-purpose registers to hold the results of integer multiply, divide, and multiply-accumulate operations (HI and LO)
- a special-purpose program counter (PC), which is affected only indirectly by certain instructions—it is not an architecturally visible register.

A MIPS64 processor always produces a 64-bit result, even for those instructions which are architecturally defined to operate on 32 bits. Such instructions typically sign-extend the 32-bit result into 64 bits. In so doing, 32-bit programs work as expected, even though the registers are actually 64 bits wide rather than 32.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

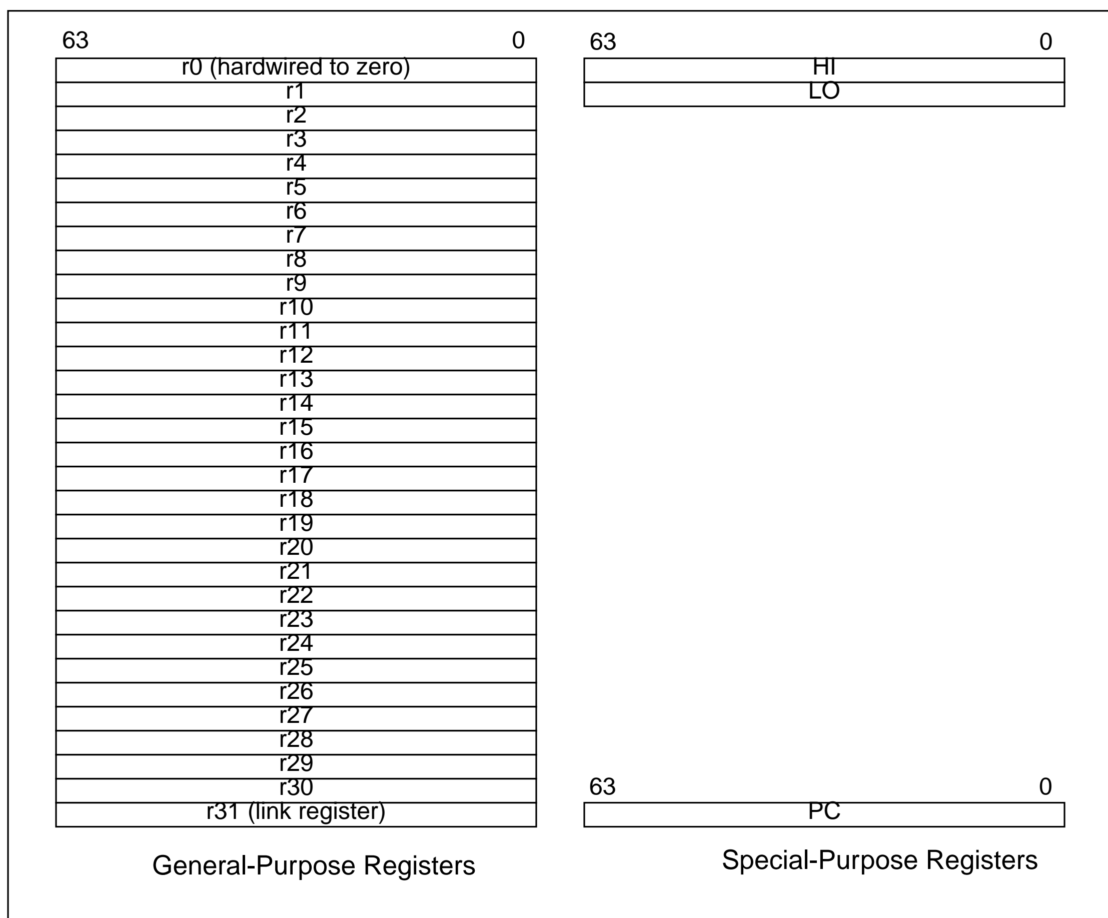


Figure 13-1 CPU Registers in MIPS64 Native Mode

13.1.2 Endianness

Compliant implementations of the MIPS64 Architecture must be bi-endian. That is, they must be capable of running in either a big-endian or a little-endian byte order, as selected by an implementation-specific power-up sequence. The BE bit in the *Config* register, set as part of the power-up sequence, indicates the endian mode in which the processor is running. It is implementation-dependent whether reverse-endian mode is implemented.

The 20Kc processor is bi-endian. The endianness mode is selected by appropriately setting the BigEndian input pin of the processor. The 20Kc processor also supports the reverse endian mode through the RE bit in the Status Register.

13.1.3 CPU Instruction Overview

Table 13-1 through Table 13-9 list the CPU instructions that are part of the MIPS64 ISA. If 64-bit operations are not enabled, certain instructions, as described in the Instruction Bit Encoding tables, are not legal and result in a Coprocessor Unusable Exception or Reserved Instruction Exception, as appropriate to the type of instruction.

Note: Although the Branch Likely instructions are included in this specification, software is strongly encouraged to avoid use of the Branch Likely instructions because they will be removed from a future revision of the MIPS64 architecture. The Branch Likely instructions were added to the ISA at a time when processor implementations were much simpler. Since that time, implementation of the Branch Likely instructions has been shown to be increasingly difficult and costly on processors with aggressive branch prediction. Continued use by software results in serious

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

performance issues as such processor designs penetrate the embedded market. The Branch Likely instructions are listed in Table 13-8 and Table 13-15.

Table 13-1 CPU Load, Store, and Memory Control Instructions

Mnemonic	Instruction	Original MIPS ISA Level
LB	Load Byte	I
LBU	Load Byte Unsigned	I
LH	Load Halfword	I
LHU	Load Halfword Unsigned	I
LW	Load Word	I
LWL	Load Word Left	I
LWR	Load Word Right	I
SB	Store Byte	I
SH	Store Halfword	I
SW	Store Word	I
SWL	Store Word Left	I
SWR	Store Word Right	I
LL	Load Linked Word	II
SC	Store Conditional Word	II
SYNC	Synchronize Memory Operations	II
LD	Load Doubleword	III
LDL	Load Doubleword Left	III
LDR	Load Doubleword Right	III
LLD	Load Linked Doubleword	III
LWU	Load Word Unsigned	III
SCD	Store Conditional Doubleword	III
SD	Store Doubleword	III
SDL	Store Doubleword Left	III
SDR	Store Doubleword Right	III
PREF	Prefetch Memory Data	IV
PREFX	Prefetch Memory Data Indexed	IV

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 13-2 CPU Arithmetic Instructions

Mnemonic	Instruction	Original MIPS ISA Level
ADD	Add Word	I
ADDI	Add Immediate Word	I
ADDIU	Add Immediate Unsigned Word	I
ADDU	Add Unsigned Word	I
DIV	Divide Word	I
DIVU	Divide Unsigned Word	I
MULT	Multiply Word	I
MULTU	Multiply Unsigned Word	I
SLT	Set on Let Than	I
SLTI	Set on Less Than Immediate	I
SLTIU	Set on Less Than Immediate Unsigned	I
SLTU	Set on Less Than Unsigned	I
SUB	Subtract Word	I
SUBU	Subtract Unsigned Word	I
DADD	Add Doubleword	III
DADDI	Add Immediate Doubleword	III
DADDIU	Add Immediate Unsigned Doubleword	III
DADDU	Add Unsigned Doubleword	III
DDIV	Divide Doubleword	III
DDIVU	Divide Unsigned Doubleword	III
DMULT	Multiply Doubleword	III
DMULTU	Multiply Unsigned Doubleword	III
DSUB	Subtract Doubleword	III
DSUBU	Subtract Unsigned Doubleword	III

Table 13-3 CPU Logical Instructions

Mnemonic	Instruction	Original MIPS ISA Level
AND	Logical AND	I
ANDI	Logical AND Immediate	I
LUI	Load Upper Immediate	I
NOR	Logical NOR	I

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 13-3 CPU Logical Instructions

Mnemonic	Instruction	Original MIPS ISA Level
OR	Logical OR	I
ORI	Logical OR Immediate	I
XOR	Logical XOR	I
XORI	Logical XOR Immediate	I

Table 13-4 CPU Move Instructions

Mnemonic	Instruction	Original MIPS ISA Level
MFHI	Move from HI	I
MFLO	Move from LO	I
MTHI	Move to HI	I
MTLO	Move to LO	I
MOVF[1]	Move Conditional on Floating-Point False	IV
MOVN	Move Conditional on Not Zero	IV
MOVT[1]	Move Conditional on Floating-Point True	IV
MOVZ	Move Conditional on Zero	IV

1. These instructions require a floating-point unit and can be subsetted out if no floating-point unit is implemented.

Table 13-5 CPU Shift Instructions

Mnemonic	Instruction	Original MIPS ISA Level
SLL	Shift Word Left Logical	I
SLLV	Shift Word Left Logical Variable	I
SRA	Shift Word Right Arithmetic	I
SRAV	Shift Word Right Arithmetic Variable	I
SRL	Shift Word Right Logical	I
SRLV	Shift Word Right Logical Variable	I
DSLL	Shift Doubleword Left Logical	III
DSLL32	Shift Doubleword Left Logical + 32	III
DSLLV	Shift Doubleword Right Logical Variable	III
DSRA	Shift Doubleword Right Arithmetic	III
DSRA32	Shift Doubleword Right Arithmetic + 32	III

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 13-5 CPU Shift Instructions

Mnemonic	Instruction	Original MIPS ISA Level
DSRAV	Shift Doubleword Right Arithmetic Variable	III
DSRL	Shift Doubleword Right Logical	III
DSRL32	Shift Doubleword Right Logical + 32	III
DSRLV	Shift Doubleword Right Logical Variable	III

Table 13-6 CPU Branch and Jump Instructions

Mnemonic	Instruction	Original MIPS ISA Level
BEQ	Branch on Equal	I
BGEZ	Branch on Greater Than or Equal Zero	I
BGEZAL	Branch on Greater Than or Equal Zero and Link	I
BGTZ	Branch on Greater Than Zero	I
BLEZ	Branch on Less Than or Equal Zero	I
BLTZ	Branch on Less Than Zero	I
BLTZAL	Branch on Less Than Zero and Link	I
BNE	Branch on Not Equal	I
J	Jump	I
JAL	Jump and Link	I
JALR	Jump and Link Register	I
JR	Jump Register	I

Table 13-7 CPU Trap Instructions

Mnemonic	Instruction	Original MIPS ISA Level
BREAK	Breakpoint	I
SYSCALL	System Call	I
TEQ	Trap if Equal	II
TEQI	Trap if Equal Immediate	II
TGE	Trap if Greater Than or Equal	II
TGEI	Trap if Greater Than or Equal Immediate	II
TGEIU	Trap if Greater Than or Equal Immediate Unsigned	II
TGEU	Trap if Greater Than or Equal Unsigned	II

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 13-7 CPU Trap Instructions

Mnemonic	Instruction	Original MIPS ISA Level
TLT	Trap if Less Than	II
TLTI	Trap if Less Than Immediate	II
TLTIU	Trap if Less Than Immediate Unsigned	II
TLTU	Trap if Less Than Unsigned	II
TNE	Trap if Not Equal	II
TNEI	Trap if Not Equal Immediate	II

Table 13-8 Obsolete[1] Branch Instructions

Mnemonic	Instruction	Original MIPS ISA Level
BEQL	Branch on Equal Likely	II
BGEZALL	Branch on Greater Than or Equal Zero and Link Likely	II
BGEZL	Branch on Greater Than or Equal Zero Likely	II
BGTZL	Branch on Greater Than Zero Likely	II
BLEZL	Branch on Less Than or Equal Zero Likely	II
BLTZALL	Branch on Less Than Zero and Link Likely	II
BLTZL	Branch on Less Than Zero Likely	II
BNEL	Branch on Not Equal Likely	II

1. Software is strongly encouraged to avoid use of the Branch Likely instructions because they will be removed from a future revision of the MIPS64 architecture.

Table 13-9 Embedded Application Instructions

Mnemonic	Instruction	Original MIPS ISA Level
CLO	Count Leading Ones in Word	MIPS32
CLZ	Count Leading Zeros in Word	MIPS32
DCLO	Count Leading Ones in Doubleword	MIPS64
DCLZ	Count Leading Zeros in Doubleword	MIPS64
MADD	Multiply and Add Word	MIPS32
MADDU	Multiply and Add Unsigned Word	MIPS32
MSUB	Multiply and Subtract Word	MIPS32
MSUBU	Multiply and Subtract Unsigned Word	MIPS32

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 13-9 Embedded Application Instructions

Mnemonic	Instruction	Original MIPS ISA Level
MUL	Multiply Word to Register	MIPS32
SSNOP	Superscalar Inhibit NOP	MIPS32

13.2 FPU Architecture

13.2.1 FPU Register Overview

The MIPS64 Architecture defines the following FPU registers:

- 32 64-bit floating-point registers (FPRs)
- Five FPU control registers

For compatibility with MIPS32 processors, a MIPS64 processor can be configured to run in a mode in which the FPRs are treated as 32 32-bit registers, each of which is capable of storing only 32-bit data types. In this mode, the double-precision floating-point (type D) data type is stored in even-odd pairs of FPRs; the long-integer (type L) and paired single (type PS) data types are not supported.

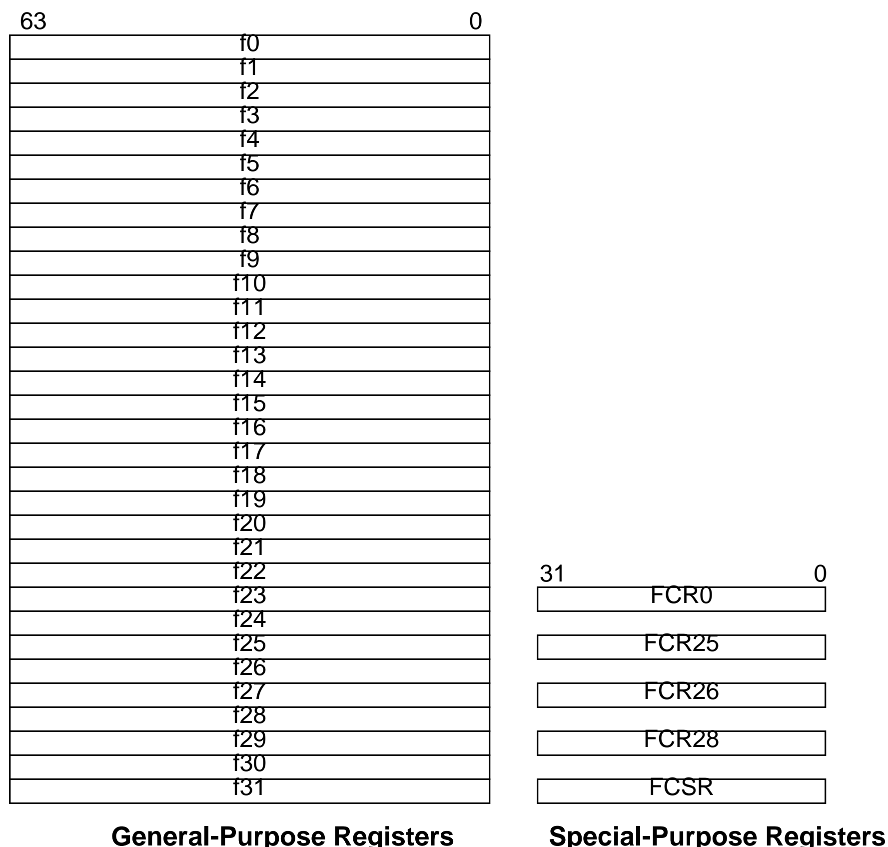


Figure 13-2 FPU Registers if $Status_{FR}$ is 1

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

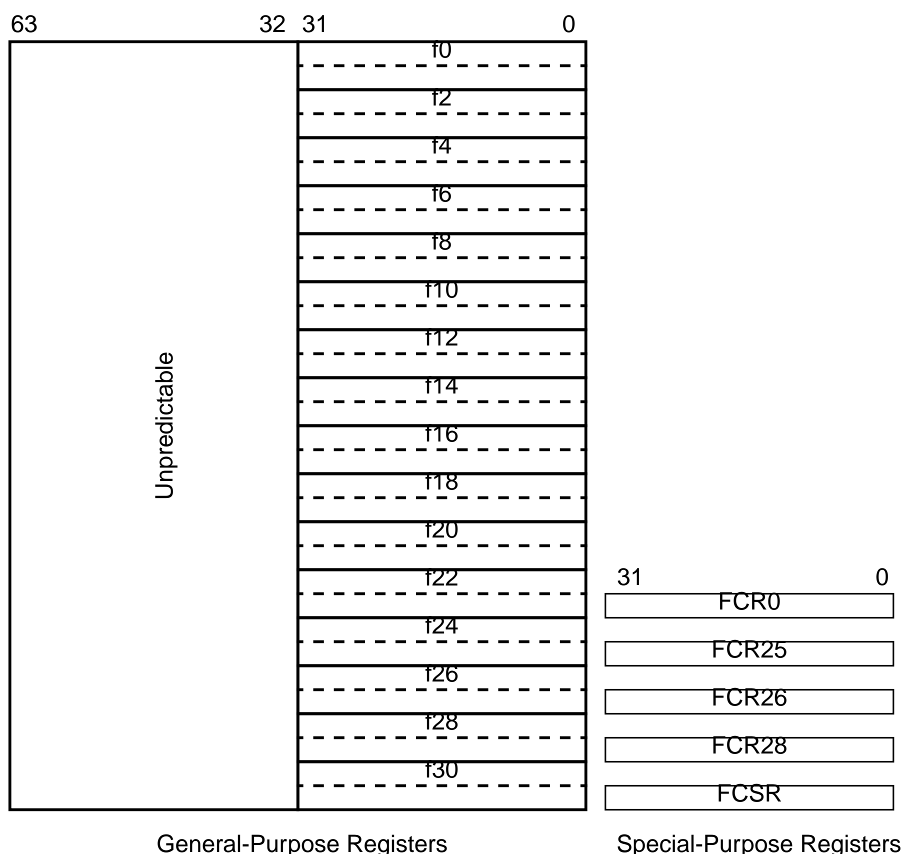


Figure 13-3 FPU Registers if $Status_{FR}$ is 0

13.2.2 FPU Instruction Overview

Table 13-10 through Table 13-15 list the FPU instructions that are part of the MIPS64 ISA. If the processor is configured to run in the mode providing backward compatibility to MIPS32 processors, certain instructions, as described in the Instruction Bit Encoding tables, are not legal and result in a Reserved Instruction exception. This includes those instructions that operate on paired single floating-point (type PS) and 64-bit fixed-point (type L) data types.

Table 13-10 FPU Load and Store Instructions

Mnemonic	Instruction	Original MIPS ISA Level
LWC1	Load Word to Floating-Point	I
SWC1	Store Word to Floating-Point	I
LDC1	Load Doubleword to Floating-Point	II
SDC1	Store Doubleword to Floating-Point	II
LDXC1	Load Doubleword Indexed to Floating-Point	IV
LWXC1	Load Word Indexed to Floating-Point	IV

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 13-10 FPU Load and Store Instructions

Mnemonic	Instruction	Original MIPS ISA Level
SDXC1	Store Doubleword Indexed to Floating-Point	IV
SWXC1	Store Word Indexed to Floating-Point	IV
LUXC1	Load Doubleword Indexed Unaligned to Floating-Point	V
SUXC1	Store Doubleword Indexed Unaligned to Floating-Point	V

Table 13-11 FPU Arithmetic Instructions

Mnemonic	Instruction	Original MIPS ISA Level
ABS.fmt	Floating-Point Absolute Value	I, V
ADD.fmt	Floating-Point Add	I, V
C.cond.fmt	Floating-Point Compare	I, V
DIV.fmt	Floating-Point Divide	I
MUL.fmt	Floating-Point Multiply	I, V
NEG.fmt	Floating-Point Negate	I, V
SUB.fmt	Floating-Point Subtract	I, V
SQRT.fmt	Floating-Point Square Root	II
MADD.fmt	Floating-Point Multiply Add	IV, V
MSUB.fmt	Floating-Point Multiply Subtract	IV, V
NMADD.fmt	Floating-Point Negative Multiply Add	IV, V
NMSUB.fmt	Floating-Point Negative Multiply Subtract	IV, V
RECIP.fmt	Floating-Point Reciprocal Approximation	IV
RSQRT.fmt	Floating-Point Reciprocal Square Root Approximation	IV

Table 13-12 FPU Move Instructions

Mnemonic	Instruction	Original MIPS ISA Level
CFC1	Copy Word from Floating-Point Control Register	I
CTC1	Copy Word to Floating-Point Control Register	I
MFC1	Move Word from FPR	I
MOV.fmt	Floating-Point Move	I
MTC1	Move Word to FPR	I
DMFC1	Move Doubleword from FPR	III

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 13-12 FPU Move Instructions

Mnemonic	Instruction	Original MIPS ISA Level
DMTC1	Move Doubleword to FPR	III
MOVF.fmt	Floating-Point Conditional Move on FP False	IV, V
MOVN.fmt	Floating-Point Conditional Move on Non-Zero	IV, V, MIPS64
MOVT.fmt	Floating-Point Conditional Move on FP True	IV, V
MOVZ.fmt	Floating-Point Conditional Move on Zero	IV, V, MIPS64

Table 13-13 FPU Convert Instructions

Mnemonic	Instruction	Original MIPS ISA Level
CVT.W.fmt	Floating-Point Convert to Word Fixed Point	I
CVT.D.fmt	Floating-Point Convert to Double Floating-Point	I, III
CVT.S.fmt	Floating-Point Convert to Single Floating-Point	I,III,V
CEIL.W.fmt	Floating-Point Ceiling to Word Fixed Point	II
FLOOR.W.fmt	Floating-Point Floor to Word Fixed Point	II
ROUND.W.fmt	Floating-Point Round to Word Fixed Point	II
TRUNC.W.fmt	Floating-Point Truncate to Word Fixed Point	II
CEIL.L.fmt	Floating-Point Ceiling to Long Fixed Point	III
CVT.L.fmt	Floating-Point Convert to Long Fixed Point	III
FLOOR.L.fmt	Floating-Point Floor to Long Fixed Point	III
ROUND.L.fmt	Floating-Point Round to Long Fixed Point	III
TRUNC.L.fmt	Floating-Point Truncate to Long Fixed Point	III
ALNV.PS	Floating-Point Align Variable	V
CVT.PS.S	Floating-Point Convert Pair to Pair Single	V
CVT.S.PL	Floating-Point Convert Pair Lower to Single	V
CVT.S.PU	Floating-Point Convert Pair Upper to Single	V
PLL.PS	Floating-Point Pair Lower Lower	V
PLU.PS	Floating-Point Pair Lower Upper	V
PUL.PS	Floating-Point Pair Upper Lower	V
PUU.PS	Floating-Point Pair Upper Upper	V

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 13-14 FPU Branch Instructions

Mnemonic	Instruction	Original MIPS ISA Level
BC1F	Branch on Floating-Point False	I, IV
BC1T	Branch on Floating-Point True	I, IV

Table 13-15 Obsolete[1] FPU Branch Instructions

Mnemonic	Instruction	Original MIPS ISA Level
BC1FL	Branch on Floating-Point False Likely	II, IV
BC1TL	Branch on Floating-Point True Likely	II, IV

1. Software is strongly encouraged to avoid use of the Branch Likely instructions because they will be removed from a future revision of the MIPS64 architecture.

13.3 Coprocessor Architecture

The MIPS64 architecture supports the use of an additional coprocessor to perform application-specific tasks. The current version of the 20Kc processor does not support this additional coprocessor.

13.4 Privileged Instruction Set Architecture

13.4.1 Privileged Register Overview

The MIPS64 architecture defines a set of privileged registers as described in Chapter 6, “Coprocessor Registers.”

13.4.2 Privileged Instruction Overview

Table 13-16 lists the privileged instructions, which act as the ISA interface to the MIPS Privileged Resource Architecture.

Table 13-16 Privileged Instructions

Mnemonic	Instruction
CACHE	Perform Cache Operation
DMFC0	Move Doubleword From Coprocessor Zero
DMTC0	Move Doubleword To Coprocessor Zero
ERET	Exception Return
MFC0	Move Word From Coprocessor Zero
MTC0	Move Word To Coprocessor Zero
TLBP	Translation Look Aside Buffer Probe

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 13-16 Privileged Instructions

Mnemonic	Instruction
TLBR	Translation Look Aside Buffer Read
TLBWI	Translation Look Aside Buffer Write Indexed
TLBWR	Translation Look Aside Buffer Write Random
WAIT	Enter Standby Mode

13.5 EJTAG Support Instructions

Table 13-17 lists the EJTAG support instructions that are supported by the MIPS64 architecture. Refer to Ref [1] for more information about these instructions.

Table 13-17 EJTAG Support Instructions

Mnemonic	Instruction
DERET	Debug Exception Return
SDBBP	Software Debug Breakpoint

13.6 Instruction Bit Encoding

Table 13-19 through Table 13-33 describe the encoding used for the MIPS64 ISA. Table 13-18 describes the meaning of the symbols used in the tables.

Table 13-18 Symbols Used in the Instruction Encoding Tables

Symbol	Meaning
*	Operation or field codes marked with this symbol are reserved for future use. Executing such an instruction must cause a Reserved Instruction Exception.
δ	(Also *italic* field name.) Operation or field codes marked with this symbol denotes a field class. The instruction word must be further decoded by examining additional tables that show values for another instruction field.
β	Operation or field codes marked with this symbol represent a valid encoding for a higher-order MIPS ISA level. Executing such an instruction must cause a Reserved Instruction Exception.
⊥	Operation or field codes marked with this symbol represent instructions which are not legal if the processor is configured to be backward compatible with MIPS32 processors. If the processor is executing in Kernel Mode, Debug Mode, or 64-bit instructions are enabled, execution proceeds normally. In other cases, executing such an instruction must cause a Reserved Instruction Exception (non-coprocessor encodings or coprocessor instruction encodings for a coprocessor to which access is allowed) or a Coprocessor Unusable Exception (coprocessor instruction encodings for a coprocessor to which access is not allowed).
θ	Operation or field codes marked with this symbol are available to licensed MIPS partners. To avoid multiple conflicting instruction definitions, the partner must notify MIPS Technologies, Inc. when one of these encodings is used. If no instruction is encoded with this value, executing such an instruction must cause a Reserved Instruction Exception (*SPECIAL2* encodings or coprocessor instruction encodings for a coprocessor to which access is allowed) or a Coprocessor Unusable Exception (coprocessor instruction encodings for a coprocessor to which access is not allowed).
σ	Field codes marked with this symbol represent an EJTAG support instruction and implementation of this encoding is optional for each implementation. If the encoding is not implemented, executing such an instruction must cause a Reserved Instruction Exception. If the encoding is implemented, it must match the instruction encoding as shown in the table.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 13-18 Symbols Used in the Instruction Encoding Tables

Symbol	Meaning
ε	Operation or field codes marked with this symbol are reserved for MIPS Application Specific Extensions. If the ASE is not implemented, executing such an instruction must cause a Reserved Instruction Exception.
ϕ	Operation or field codes marked with this symbol are obsolete and will be removed from a future revision of the MIPS64 ISA. Software must avoid using these operation or field codes.

Table 13-19 MIPS64 Encoding of the Opcode Field

opcode		*bits 28..26*							
		0	1	2	3	4	5	6	7
bits 31..29		000	001	010	011	100	101	110	111
0	000	SPECIAL δ	REGIMM δ	J	JAL	BEQ	BNE	BLEZ	BGTZ
1	001	ADDI	ADDIU	SLTI	SLTIU	ANDI	ORI	XORI	LUI
2	010	*COP0* δ	*COP1* δ	*COP2* θδ	*COP1X* δ⊥	BEQL ϕ	BNEL ϕ	BLEZL ϕ	BGTZL ϕ
3	011	DADDI ⊥	DADDIU ⊥	LDL ⊥	LDR ⊥	SPECIAL2 δ	JALX ε	*MDMX* εδ	*
4	100	LB	LH	LWL	LW	LBU	LHU	LWR	LWU ⊥
5	101	SB	SH	SWL	SW	SDL ⊥	SDR ⊥	SWR	CACHE
6	110	LL	LWC1	LWC2 θ	PREF	LLD ⊥	LDC1	LDC2 θ	LD ⊥
7	111	SC	SWC1	SWC2 θ	*	SCD ⊥	SDC1	SDC2 θ	SD ⊥

Table 13-20 MIPS64 *SPECIAL* Opcode Encoding of Function Field

function		*bits 2..0*							
		0	1	2	3	4	5	6	7
bits 5..3		000	001	010	011	100	101	110	111
0	000	SLL	MOVCI δ	SRL	SRA	SLLV	*	SRLV	SRAV
1	001	JR	JALR	MOVZ	MOVN	SYSCALL	BREAK	*	SYNC
2	010	MFHI	MTHI	MFLO	MTLO	DSLLV ⊥	*	DSRLV ⊥	DSRAV ⊥
3	011	MULT	MULTU	DIV	DIVU	DMULT ⊥	DMULTU ⊥	DDIV ⊥	DDIVU ⊥
4	100	ADD	ADDU	SUB	SUBU	AND	OR	XOR	NOR
5	101	*	*	SLT	SLTU	DADD ⊥	DADDU ⊥	DSUB ⊥	DSUBU ⊥
6	110	TGE	TGEU	TLT	TLTU	TEQ	*	TNE	*
7	111	DSLL ⊥	*	DSRL ⊥	DSRA ⊥	DSLL32 ⊥	*	DSRL32 ⊥	DSRA32 ⊥

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 13-21 MIPS64 *REGIMM* Encoding of rt Field

rt		*bits 18..16*							
		0	1	2	3	4	5	6	7
bits 20..19		000	001	010	011	100	101	110	111
0	00	BLTZ	BGEZ	BLTZL ϕ	BGEZL ϕ	*	*	*	*
1	01	TGEI	TGEIU	TLTI	TLTIU	TEQI	*	TNEI	*
2	10	BLTZAL	BGEZAL	*BLTZALL* ϕ	*BGEZALL* ϕ	*	*	*	*
3	11	*	*	*	*	*	*	*	*

Table 13-22 MIPS64 *SPECIAL2* Encoding of Function Field

function		*bits 2..0*							
		0	1	2	3	4	5	6	7
bits 5..3		000	001	010	011	100	101	110	111
0	000	MADD	MADDU	MUL	θ	MSUB	MSUBU	θ	θ
1	001	θ	θ	θ	θ	θ	θ	θ	θ
2	010	θ	θ	θ	θ	θ	θ	θ	θ
3	011	θ	θ	θ	θ	θ	θ	θ	θ
4	100	CLZ	CLO	θ	θ	DCLZ ⊥	DCLO ⊥	θ	θ
5	101	θ	θ	θ	θ	θ	θ	θ	θ
6	110	θ	θ	θ	θ	θ	θ	θ	θ
7	111	θ	θ	θ	θ	θ	θ	θ	SDBBP σ

Table 13-23 MIPS64 *MOVCI* Encoding of tf Bit

tf	*bit 16*	
	0	1
	MOVF	MOVT

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 13-24 MIPS64 *COPz* Encoding of rs Field

rs		*bits 23..21*							
		0	1	2	3	4	5	6	7
bits 25..24		000	001	010	011	100	101	110	111
0	00	MFCz	DMFCz ⊥	CFCz	*	MTCz	DMTCz ⊥	CTCz	*
1	01	BCz δ	*	*	*	*	*	*	*
2	10	*CO* δ							
3	11								

Table 13-25 MIPS64 COPz Encoding of rt Field When rs=*BCz*

rt	*bit 16*	
bit 17	0	1
0	BCzF	BCzT
1	BCzFL ϕ	BCzTL ϕ

Table 13-26 MIPS64 *COP0* Encoding of rs Field

rs		*bits 23..21*							
		0	1	2	3	4	5	6	7
bits 25..24		000	001	010	011	100	101	110	111
0	00	MFC0	DMFC0 ⊥	*	*	MTC0	DMTC0 ⊥	*	*
1	01	*	*	*	*	*	*	*	*
2	10	*CO* δ							
3	11								

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 13-27 MIPS64 *COP0* Encoding of Function Field When rs=*CO*

function		*bits 2..0*							
		0	1	2	3	4	5	6	7
bits 5..3		000	001	010	011	100	101	110	111
0	000	*	TLBR	TLBWI	*	*	*	TLBWR	*
1	001	TLBP	*	*	*	*	*	*	*
2	010	*	*	*	*	*	*	*	*
3	011	ERET	*	*	*	*	*	*	DERET σ
4	100	WAIT	*	*	*	*	*	*	*
5	101	*	*	*	*	*	*	*	*
6	110	*	*	*	*	*	*	*	*
7	111	*	*	*	*	*	*	*	*

Table 13-28 MIPS64 *COP1* Encoding of rs Field

rs		*bits 23..21*							
		0	1	2	3	4	5	6	7
bits 25..24		000	001	010	011	100	101	110	111
0	00	MFC1	DMFC1 ⊥	CFC1	*	MTC1	DMTC1 ⊥	CTC1	*
1	01	BC1 δ	*BC1ANY2* δε⊥	BC1ANY4 δε⊥	*	*	*	*	*
2	10	S δ	D δ	*	*	W δ	L δ⊥	PS δ⊥	*
3	11	*	*	*	*	*	*	*	*

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 13-29 MIPS64 *COP1* Encoding of Function Field When rs=*S*

function		*bits 2..0*							
		0	1	2	3	4	5	6	7
bits 5..3		000	001	010	011	100	101	110	111
0	000	ADD	SUB	MUL	DIV	SQRT	ABS	MOV	NEG
1	001	ROUND.L ⊥	TRUNC.L ⊥	CEIL.L ⊥	FLOOR.L ⊥	ROUND.W	TRUNC.W	CEIL.W	FLOOR.W
2	010	*	MOVCF δ	MOVZ	MOVN	*	RECIP ⊥	RSQRT ⊥	*
3	011	*	*	*	*	RECIP2 ε⊥	RECIP1 ε⊥	RSQRT1 ε⊥	RSQRT2 ε⊥
4	100	*	CVT.D	*	*	CVT.W	CVT.L ⊥	CVT.PS ⊥	*
5	101	*	*	*	*	*	*	*	*
6	110	C.F CABS.F ε⊥	C.UN CABS.UN ε⊥	C.EQ CABS.EQ ε⊥	C.UEQ CABS.UEQ ε⊥	C.OLT CABS.OLT ε⊥	C.ULT CABS.ULT ε⊥	C.OLE CABS.OLE ε⊥	C.ULE CABS.ULE ε⊥
7	111	C.SF CABS.SF ε⊥	C.NGLE CABS.NGLE ε⊥	C.SEQ CABS.SEQ ε⊥	C.NGL CABS.NGL ε⊥	C.LT CABS.LT ε⊥	C.NGE CABS.NGE ε⊥	C.LE CABS.LE ε⊥	C.NGT CABS.NGT ε⊥

Table 13-30 MIPS64 *COP1* Encoding of Function Field When rs=*D*

function		*bits 2..0*							
		0	1	2	3	4	5	6	7
bits 5..3		000	001	010	011	100	101	110	111
0	000	ADD	SUB	MUL	DIV	SQRT	ABS	MOV	NEG
1	001	ROUND.L ⊥	TRUNC.L ⊥	CEIL.L ⊥	FLOOR.L ⊥	ROUND.W	TRUNC.W	CEIL.W	FLOOR.W
2	010	*	MOVCF δ	MOVZ	MOVN	*	RECIP ⊥	RSQRT ⊥	*
3	011	*	*	*	*	RECIP2 ε⊥	RECIP1 ε⊥	RSQRT1 ε⊥	RSQRT2 ε⊥
4	100	CVT.S	*	*	*	CVT.W	CVT.L ⊥	*	*
5	101	*	*	*	*	*	*	*	*
6	110	C.F CABS.F ε⊥	C.UN CABS.UN ε⊥	C.EQ CABS.EQ ε⊥	C.UEQ| CABS.UEQ ε⊥	C.OLT CABS.OLT ε⊥	C.ULT CABS.ULT ε⊥	C.OLE CABS.OLE ε⊥	C.ULE CABS.ULE ε⊥
7	111	C.SF CABS.SF ε⊥	C.NGLE CABS.NGLE ε⊥	C.SEQ CABS.SEQ ε⊥	C.NGL CABS.NGL ε⊥	C.LT CABS.LT ε⊥	C.NGE CABS.NGE ε⊥	C.LE CABS.LE ε⊥	C.NGT CABS.NGT ε⊥

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 13-31 MIPS64 *COP1* Encoding of Function Field When rs=*W or L*[1]

function		bits 2..0							
		0	1	2	3	4	5	6	7
bits 5..3		000	001	010	011	100	101	110	111
0	000	*	*	*	*	*	*	*	*
1	001	*	*	*	*	*	*	*	*
2	010	*	*	*	*	*	*	*	*
3	011	*	*	*	*	*	*	*	*
4	100	CVT.S	CVT.D	*	*	*	*	CVT.PS.PW ε⊥	*
5	101	*	*	*	*	*	*	*	*
6	110	*	*	*	*	*	*	*	*
7	111	*	*	*	*	*	*	*	*

1. Format type L is legal only if 64-bit operations are enabled.

Table 13-32 MIPS64 *COP1* Encoding of Function Field When rs=P*S*[1]

function		bits 2..0							
		0	1	2	3	4	5	6	7
bits 5..3		000	001	010	011	100	101	110	111
0	000	ADD	SUB	MUL	*	*	ABS	MOV	NEG
1	001	*	*	*	*	*	*	*	*
2	010	*	MOVCF δ	MOVZ	MOVN	*	*	*	*
3	011	ADDR ε	*	MULR e	*	RECIP2 ε	RECIP1 ε	RSQRT1 ε	RSQRT2 ε
4	100	CVT.S.PU	*	*	*	CVT.PW.PS ε	*	*	*
5	101	CVT.S.PL	*	*	*	PLL.PS	PLU.PS	PUL.PS	PUU.PS
6	110	C.F CABS.F ε	C.UN CABS.UN ε	C.EQ CABS.EQ ε	C.UEQ CABS.UEQ ε	C.OLT CABS.OLT ε	C.ULT CABS.ULT ε	C.OLE CABS.OLE ε	C.ULE CABS.ULE ε
7	111	C.SF CABS.SF ε	C.NGLE CABS.NGLE ε	C.SEQ CABS.SEQ ε	C.NGL CABS.NGL ε	C.LT CABS.LT ε	C.NGE CABS.NGE ε	C.LE CABS.LE ε	C.NGT CABS.NGT ε

1. Format type PS is legal only if 64-bit operations are enabled.

Table 13-33 MIPS64 *COP1* Encoding of tf Bit When rs=*S, D, or PS,* Function=*MOVCF*

tf	bit 16	
	0	1
	MOVF.fmt	MOVT.fmt

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 13-34 MIPS64 *COP1X* Encoding of Function Field[1]

function		*bits 2..0*							
		0	1	2	3	4	5	6	7
bits 5..3		000	001	010	011	100	101	110	111
0	000	LWXC1	LDXC1	*	*	*	LUXC1	*	*
1	001	SWXC1	SDXC1	*	*	*	SUXC1	*	PREFX
2	010	*	*	*	*	*	*	*	*
3	011	*	*	*	*	*	*	ALNV.PS	*
4	100	MADD.S	MADD.D	*	*	*	*	MADD.PS	*
5	101	MSUB.S	MSUB.D	*	*	*	*	MSUB.PS	*
6	110	NMADD.S	NMADD.D	*	*	*	*	NMADD.PS	*
7	111	NMSUB.S	NMSUB.D	*	*	*	*	NMSUB.PS	*

1. COP1X instructions are legal only if 64-bit operations are enabled.

13.7 MIPS64 Instruction Descriptions

As described earlier, this specification does not include instruction descriptions for all instructions that are in the MIPS64 ISA. Rather, it includes by reference the MIPS RISC Architecture document for the majority of the instructions. The following subsections describe only those ISA-related features and any instructions that are new or modified by inclusion of them in the MIPS64 ISA.

13.7.1 UNPREDICTABLE and UNDEFINED

The terms **UNPREDICTABLE** and **UNDEFINED** are used throughout this specification to describe the behavior of the processor in certain cases. **UNDEFINED** behavior or operations can occur only as the result of executing instructions in a privileged mode (i.e., in Kernel Mode or Debug Mode, or with the CP0 usable bit set in the Status register). Unprivileged software can never cause **UNDEFINED** behavior or operations. Conversely, both privileged and unprivileged software can cause **UNPREDICTABLE** results or operations.

13.7.1.1 UNPREDICTABLE

UNPREDICTABLE results may vary from implementation to implementation, instruction to instruction, or as a function of time on the same implementation or instruction. Software can never depend on results that are **UNPREDICTABLE**. **UNPREDICTABLE** operations can cause a result to be generated or not. If a result is generated, it is **UNPREDICTABLE**. **UNPREDICTABLE** operations can cause arbitrary exceptions.

UNPREDICTABLE results or operations have several implementation restrictions:

- **UNPREDICTABLE** results must not depend on any data source (memory or internal state) which are inaccessible in the current processor mode.
- **UNPREDICTABLE** operations must not read, write, or modify the contents of memory or internal state which is inaccessible in the current processor mode. For example, **UNPREDICTABLE** operations executed in user mode must not access memory or internal state that is only accessible in Kernel Mode or Debug Mode or in another process.
- **UNPREDICTABLE** operations must not halt or hang the processor.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

13.7.1.2 UNDEFINED

UNDEFINED operations or behavior can vary from implementation to implementation, instruction to instruction, or as a function of time on the same implementation or instruction. **UNDEFINED** operations or behavior can vary from nothing to creating an environment in which execution can no longer continue. **UNDEFINED** operations or behavior might cause data loss.

There is one implementation restriction with **UNDEFINED** operations or behavior:

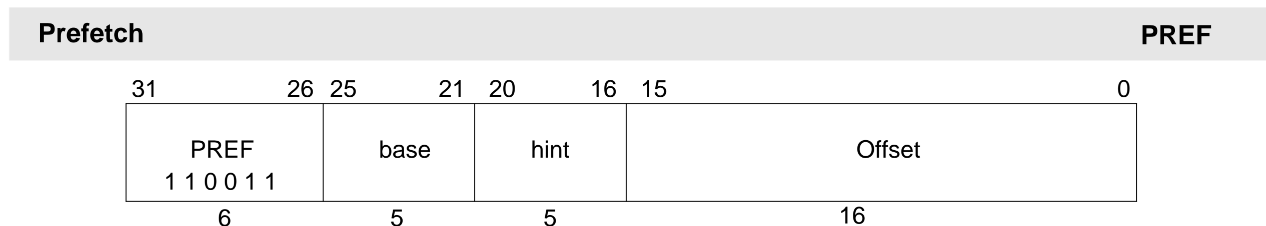
- **UNDEFINED** operations or behavior must not cause the processor to hang (that is, enter a state from which there is no exit other than powering down the processor). The assertion of any of the reset signals must restore the processor to an operational state.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

13.7.2 Unprivileged Instructions

13.7.2.1 The PREF Instruction

Prefetch **PREF**



Format:

`PREF hint, offset(base)` **MIPS IV**

Purpose:

To move data between memory and cache.

Description:

PREF adds the 16-bit signed *offset* to the contents of GPR *base* to form an effective byte address. The *hint* field supplies information about how the addressed data is to be manipulated.

PREF enables the processor to take some action as specified by the *hint* field, to improve program performance. The action taken for a specific PREF instruction is both system and context dependent. Any action, including doing nothing, is permitted as long as it does not change architecturally visible state or alter the meaning of a program. Implementations are expected either to do nothing, or take an action that increases the performance of the program.

PREF does not cause addressing-related exceptions. If it does raise an exception condition, the exception condition is ignored. If an addressing-related exception condition is raised and ignored, no data movement occurs.

PREF never generates a memory operation for a location with an uncached memory access type.

For a cached location, the expected and useful action for the processor is to move a block of data between cache and the memory hierarchy. The size of the block transferred is implementation dependent, but software may assume that it is at least one cache block.

For the 20Kc processor, the size of the block transferred is always equal to one cache line.

Table 13-35 defines the hint field values.

Table 13-35 PREF Hint Field Encodings

Value	Name	Data Use and Desired PREF Action
0	load	Use: Prefetched data is expected to be read (not modified) Action: Fetch data as if for a load.
1	store	Use: Prefetched data is expected to be stored or modified Action: Fetch data as if for a store.
2-3	Reserved	Reserved for future use - not available to implementations.
4	load_streamed	Use: Prefetched data is expected to be read (not modified) but not reused extensively; it “streams” through the cache Action: Fetch data as if for a load and place it in the cache so that it does not displace data prefetched as “retained”

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 13-35 PREF Hint Field Encodings (Continued)

Value	Name	Data Use and Desired PREF Action
5	store_streamed	Use: Prefetched data is expected to be stored or modified but not reused extensively; it “streams” through the cache Action: Fetch data as if for a store and place it in the cache so that it does not displace data prefetched as “retained”
6	load_retained	Use: Prefetched data is expected to be read (not modified) and reused extensively; it should be “retained” in the cache Action: Fetch data as if for a load and place it in the cache so that it is not displaced by data prefetched as “streamed”
7	store_retained	Use: Prefetched data is expected to be stored or modified and reused extensively; it should be “retained” in the cache Action: Fetch data as if for a store and place it in the cache so that it is not displaced by data prefetched as “streamed”
8-24	Reserved	Reserved for future use - not available to implementations.
25	writeback_invalidate (also known as nudge)	Use: Data is no longer to be expected to be used Action: For a writeback cache, schedule a writeback of any dirty data. At the completion of the writeback, mark as invalid the state of any cache lines written back.
26-31	Implementation Dependent	Unassigned by the Architecture - available for implementation dependent use This field is unused in the 20Kc core.

Restrictions:

None

Operation:

```
vAddr ← GPR[base] + sign_extend(offset)
(pAddr, CCA) ← AddressTranslation(vAddr, DATA, LOAD)
Prefetch(CCA, pAddr, vAddr, DATA, hint)
```

Exceptions:

Prefetch does not take any TLB-related or address-related exceptions under any circumstances.

Programming Notes:

Prefetch cannot access a mapped location unless the translation for that location is present in the TLB. Locations in memory pages that have not been accessed recently might not have translations in the TLB, so prefetch might not be effective for such locations.

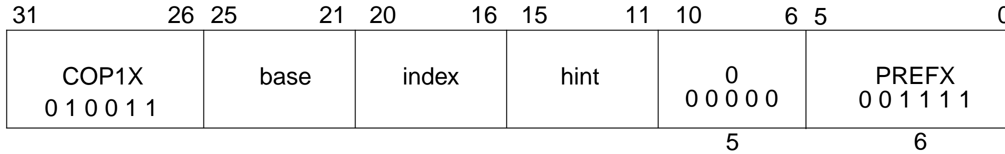
Prefetch does not cause addressing exceptions. It does not cause an exception to prefetch using a pointer before the validity of the pointer is determined.

Hint field encodings whose function is described as “streamed” or “retained” convey usage intent from software to hardware. Software should not assume that hardware always prefetches data in an optimal way. If data is to be truly retained, software should use the Cache instruction to lock data into the cache.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

13.7.2.2 The PREFX Instruction

Prefetch Indexed **PREFX**



Format:

`PREFX hint, index(base)` **MIPS IV**

Purpose:

To move data between memory and cache.

Description:

PREFX adds the contents of GPR index to the contents of GPR base to form an effective byte address. The *hint* field supplies information about how the addressed data is to be manipulated.

The only functional difference between the PREF and PREFX instructions is the addressing mode implemented by the two. Refer to the PREF instruction description for all other details, including the encoding of the *hint* field.

Note, however, that the PREFX instruction is only available on processors that implement floating point, and should only be generated by compilers in situations in which the corresponding load and store indexed floating-point instructions are generated.

Restrictions:

None

Operation:

```
if (Status_CU1 = 0) then
    InitiateCoprocessorUnusableException(1)
endif
vAddr <- GPR[base] + GPR[index]
(pAddr, CCA) <- AddressTranslation(vAddr, DATA, LOAD)
Prefetch(CCA, pAddr, vAddr, DATA, hint)
```

Exceptions:

Coprocessor Unusable Exception

Prefetch does not take any TLB-related or address-related exceptions under any circumstances.

Programming Notes:

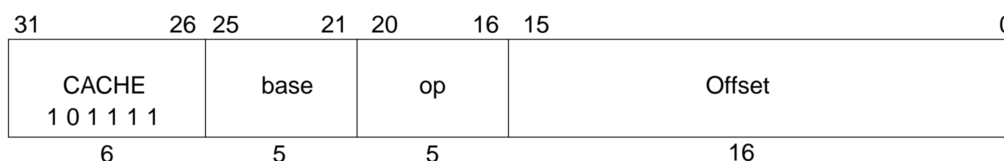
Refer to the corresponding section in the PREF instruction description.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

13.7.3 Privileged Instructions

13.7.3.1 CACHE Instruction

Perform Cache Operation **CACHE**



Format:

`CACHE    op, offset(base)` **MIPS64**

Purpose:

To perform the cache operation specified by *op*.

Description:

The 16-bit offset is sign-extended and added to the contents of the *base* register to form an effective address. The effective address is used in one of three ways based on the operation to be performed and the type of cache as described in Table 13-36.

Table 13-36 Usage of Effective Address

Operation Requires an	Type of Cache	Usage of Effective Address
Address	Virtual	The effective address is used to address the cache. It is implementation dependent whether an address translation is performed on the effective address (with the possibility that a TLB Refill or TLB Invalid exception might occur). The Instruction Cache in the 20Kc processor is virtual. Cache instructions referencing the Instruction Cache perform address translation only if they require transactions to go off-chip. The two Cache instruction types that fall under this category are *Fill* and *Fetch and Lock*.
Address	Physical	The effective address is translated by the MMU to a physical address. The physical address is then used to address the cache.
Index	N/A	The effective address can be translated by the MMU to a physical address. It is implementation dependent whether the effective address or the translated physical address are used to index the cache. The 20Kc processor uses the translated physical address to index the physical Data Cache. The Instruction Cache is virtual and therefore always uses the effective address. Assuming that the total cache size in bytes is CS, the associativity is A, and the number of bytes per tag is BPT, the following calculations give the fields of the address which specify the way and the index: OffsetBit <- Log2(BPT) IndexBit <- Log2(CS / A) WayBit <- IndexBit + Ceiling(Log2(A)) Way <- $Addr_{WayBit-1..IndexBit}$ Index <- $Addr_{IndexBit-1..OffsetBit}$ For a direct-mapped cache, the Way calculation is ignored and the Index value fully specifies the cache tag. This is shown symbolically in Figure 13-4.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

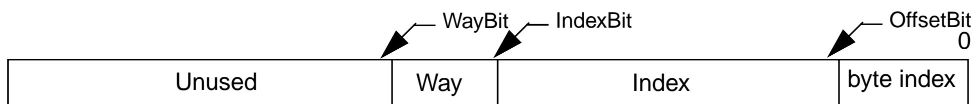


Figure 13-4 Usage of Address Fields to Select Index and Way

A TLB Refill and TLB Invalid (both with cause code equal TLBL) exception can occur on any operation. For index operations (where the address is used to index the cache but need not match the cache tag), software should use unmapped addresses to avoid TLB exceptions. This instruction never causes TLB Modified exceptions or TLB Refill exceptions with a cause code of TLBS nor data Watch exceptions.

A Cache Error exception might occur as a byproduct of some operations performed by this instruction. For example, if a Writeback operation detects a cache or bus error during the processing of the operation, that error is reported via a Cache Error exception. The Index Load Tag and Index Store Tag CACHE instructions do not trigger cache error exceptions; however, the software must not rely on this behavior. Future processors might have different behaviors regarding whether these instructions incur cache error exceptions. The reasons for this are:

- The Index Store Tag CACHE instruction is the way to initialize the Instruction cache in the first place.
- The Index Load Tag CACHE instruction is a diagnostic instruction that may be used to probe the cache on a cache error exception.

An Address Error Exception (with cause code equal AdEL) can occur if the effective address references a portion of the kernel address space that would normally result in such an exception. It is implementation dependent whether such an exception does occur.

The 20Kcprocessor takes Address Error exceptions on Cache instructions when any of the following conditions are met:

1. It references the Instruction Cache and is a *Fill* or *Fetch and Lock* type Cache Instruction.
2. It references the Data Cache.

It is implementation dependent whether a data watch is triggered by a cache instruction whose address matches the Watch register address match conditions. The preferred implementation is not to match on the cache instruction.

The 20Kc processor does not take Watch Exceptions on Cache instructions.

Bits [17:16] of the instruction specify the cache on which to perform the operation, as follows:

Table 13-37 Encoding of Bits[17:16] of CACHE Instruction

Code	Name	Cache
0 0	I	Primary Instruction
0 1	D	Primary Data or Unified Primary
1 0	T	Tertiary
1 1	S	Secondary

Bits [20:18] of the instruction specify the operation to perform. To provide software with a consistent base of cache operations, certain encodings must be supported on all processors. The remaining encodings are recommended.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 13-38 Encoding of Bits [20:18] of the CACHE Instruction

Code	Caches	Name	Effective Address Operand Type	Operation	Compliance
0 0 0	I	Index Invalidate	Index	Set the state of the cache block at the specified index to invalid. This required encoding can be used by software to invalidate the entire instruction cache by stepping through all valid indices.	Required
	D	Index Writeback Invalidate / Index Invalidate	Index	For a write-back cache: If the state of the cache block at the specified index is valid and dirty, write the block back to the memory address specified by the cache tag. After that operation is completed, set the state of the cache block to invalid. If the block is valid but not dirty, set the state of the block to invalid. For a write-through cache: Set the state of the cache block at the specified index to invalid. This required encoding can be used by software to invalidate the entire data cache by stepping through all valid indices.	Required
0 0 1	I, D	Index Load Tag	Index	Read the tag for the cache block at the specified index into the TagLo and TagHi COP0 registers. If the DataLo and DataHi registers are implemented, also read the data corresponding to the byte index into the DataLo and DataHi registers. The 20Kc processor implements separate DataLo and DataHi registers for the Instruction and Data Caches, termed as IDataLo/IDataHi and DDataLo and DDataHi respectively. A doubleword of data is read into these registers. The double word accessed is determined by bits <4:3> of the address/index. The 20Kc processor produces valid values in the IDataLo and IDataHi registers only if the CACHE instruction and the instruction following it were fetched from uncached memory.	Recommended
0 1 0	I, D	Index Store Tag	Index	Write the tag for the cache block at the specified index from the TagLo and TagHi COP0 registers. This required encoding can be used by software to initialize the entire instruction of data caches by stepping through all valid indices. Doing so requires that the *TagLo* and *TagHi* registers associated with the cache be initialized first.	Required
0 1 1	D	Create Dirty Exclusive		Available for implementation-dependent operation. The 20Kc processor uses this encoding to define the Create Dirty Exclusive operation, which is used to avoid loading data needlessly from memory when writing new contents into an entire cache block. If the cache block at the index given by the Cache instruction, does not contain the specified address, and the block is dirty, it is written back to memory. The cache block tag is then set to the address specified by the Cache instruction, and the state is set to Dirty Exclusive.	Optional

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 13-38 Encoding of Bits [20:18] of the CACHE Instruction (Continued)

Code	Caches	Name	Effective Address Operand Type	Operation	Compliance
1 0 0	I, D	Hit Invalidate	Address	If the cache block contains the specified address, set the state of the cache block to invalid. This required encoding may be used by software to invalidate a range of addresses from the instruction cache by stepping through the address range by the line size of the cache.	Required (Instruction Cache Encoding Only)
1 0 1	I	Fill	Address	Fill the cache from the specified address	Recommended
	D	Hit Writeback Invalidate / Hit Invalidate	Address	For a write-back cache: If the cache block contains the specified address and it is valid and dirty, write the contents back to memory. After that operation is completed, set the state of the cache block to invalid. If the block is valid but not dirty, set the state of the block to invalid. For a write-through cache: If the cache block contains the specified address, set the state of the cache block to invalid. This required encoding can be used by software to invalidate a range of addresses from the data cache by stepping through the address range by the line size of the cache.	Required
1 1 0	D	Hit Writeback	Address	If the cache block contains the specified address and it is valid and dirty, write the contents back to memory. After the operation is completed, leave the state of the line valid, but clear the dirty state. For a write-through cache, this operation may be treated as a nop.	Recommended

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 13-38 Encoding of Bits [20:18] of the CACHE Instruction (Continued)

Code	Caches	Name	Effective Address Operand Type	Operation	Compliance
1 1 1	I, D	Fetch and Lock	Address	If the cache does not contain the specified address, fill it from memory, performing a writeback if required, and set the state to valid and locked. If the cache already contains the specified address, set the state to locked. In set-associative or fully-associative caches, the way selected on a fill from memory is implementation specific. In the 20Kc processor, the way selected on a fill from memory is determined by an LRF (Least Recently Filled) cache replacement algorithm. The lock state may be cleared by executing an Index Invalidate, Index Writeback Invalidate, Hit Invalidate, or Hit Writeback Invalidate operation to the locked line, or via an Index Store Tag operation to the line that clears the lock bit. Note that clearing the lock state via Index Store Tag is dependent on the implementation-dependent cache tag and cache line organization, and that Index and Index Writeback Invalidate operations are dependent on cache line organization. Only Hit and Hit Writeback Invalidate operations are generally portable across implementations. It is implementation dependent whether a locked line is displaced as the result of an external invalidate or intervention that hits on the locked line. Software must not depend on the locked line remaining in the cache if an external invalidate or intervention would invalidate the line if it were not locked. In the 20Kc processor an external invalidate or intervention causes a locked line to be displaced.	Recommended

Restrictions:

Execution of this instruction is legal only if the processor is operating in Kernel Mode or Debug Mode, or if the CP0 enable bit is set in the *Status* register. In other circumstances, a Coprocessor Unusable Exception is taken.

The operation of this instruction is **UNDEFINED** for any operation/cache combination that is not implemented. The operation of this instruction is **UNDEFINED** for uncacheable addresses.

Operation:

```
vAddr <- GPR[base] + sign_extend(offset)
(pAddr, uncached) <- AddressTranslation(vAddr, DataReadReference)
CacheOp(op, vAddr, pAddr)
```

Exceptions:

TLB Refill Exception

TLB Invalid Exception

Coprocessor Unusable Exception

Address Error Exception

Cache Error Exception

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Programming Notes:

Software must adhere to the following standard mechanism to handle self-modifying code properly (this mechanism applies to software that copies code to a memory location for subsequent execution):

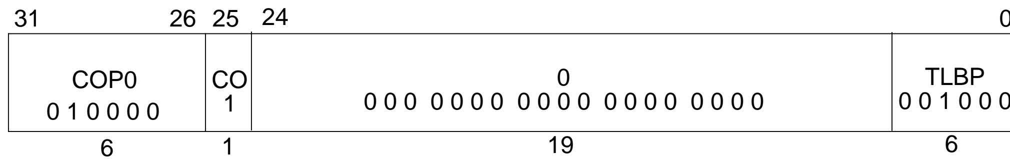
1. Write new instructions to “memory” with store instructions.
2. Do a writeback or writeback invalidate on the D-Cache lines that might hold the new instructions.
3. Execute SYNC.
4. Do an invalidate on the I-Cache lines that might hold the old instructions.
5. Jump to the code with an ERET.

The above steps are the standard mechanism that handles self-modifying code on MIPS processors. Predictable behavior is not guaranteed for software that does not follow this mechanism when writing code.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

13.7.3.2 The TLBP Instruction

Probe TLB for Matching Entry **TLBP**



Format:

`TLBP` **MIPS64**

Purpose:

Find a matching entry in the TLB.

Description:

The *Index* register is loaded with the index of the TLB entry whose contents match the contents of the *EntryHi* register. If no TLB entry matches, the high-order bit of the Index register is set.

Restrictions:

This instruction is legal only if the processor is in Kernel Mode or Debug Mode, or if the CP0 usable bit is set in the *Status* register. In other circumstances, execution of this instruction results in a Coprocessor Unusable Exception.

For processors that do not include the standard TLB MMU, the operation of this instruction is **UNDEFINED**. However, the preferred implementation is a Reserved Instruction Exception.

The 20Kc processor implements a standard TLB MMU, so this is not an issue.

Operation:

```
Index <- 1 || UNPREDICTABLE^31
for i in 0...TLBEntries-1
    if (TLB[i]_R = EntryHi_R) and
        ((TLB[i]_VPN2 and not (TLB[i]_Mask)) =
         (EntryHi_VPN2 and not (TLB[i]_Mask))) and
        (TLB[i]_G or (TLB[i]_ASID = EntryHi_ASID)) then
         Index <- i
    endif
endfor
```

Exceptions:

Coprocessor Unusable Exception

Reserved Instruction Exception (if not implemented)

Machine Check (if implemented and a TLB shutdown condition is detected on a TLB read)

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

13.7.3.3 The TLBR Instruction

Read Indexed TLB Entry **TLBR**

31–26	25	24–6	5–0
COP0 0 1 0 0 0 0	CO 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	TLBR 0 0 0 0 0 1
6	1	19	6

Format:

`TLBR` **MIPS64**

Purpose:

Read an entry from the TLB.

Description:

The *EntryHi*, *EntryLo0*, *EntryLo1*, and *PageMask* registers are loaded with the contents of the TLB entry pointed to by the *Index* register. Note that the value written to the *EntryHi*, *EntryLo0*, and *EntryLo1* registers can be different from that originally written to the TLB via these registers as follows:

- The value returned in the VPN2 field of the *EntryHi* register can have those bits reset to zero corresponding to the one bits in the Mask field of the TLB entry (the least significant bit of VPN2 corresponds to the least significant bit of the Mask field). The 20Kc processor preserves these bits on a TLBR. However, these bits are zeroed out on TLBWI and TLBWR instructions, and therefore to the user they appear to be zeroed out on a read.
- The value returned in the PFN field of the *EntryLo0* and *EntryLo1* registers may have those bits set to zero corresponding to the one bits in the Mask field of the TLB entry (the least significant bit of PFN corresponds to the least significant bit of the Mask field). The 20Kc processor preserves these bits on a TLBR. However, these bits are zeroed out on TLBWI and TLBWR instructions, and therefore to the user they appear to be zeroed out on a read.
- The value returned in the G bit in both the *EntryLo0* and *EntryLo1* registers comes from the single G bit in the TLB entry. Recall that this bit was set from the logical AND of the two G bits in *EntryLo0* and *EntryLo1* when the TLB was written.

Restrictions:

This instruction is legal only if the processor is in Kernel Mode or Debug Mode, or if the CP0 usable bit is set in the Status register. In other circumstances, execution of this instruction results in a Coprocessor Unusable Exception.

The operation is UNDEFINED if the contents of the *Index* register are greater than or equal to the number of TLB entries in the processor.

For processors that do not include the standard TLB MMU, the operation of this instruction is **UNDEFINED**. However, the preferred implementation is a Reserved Instruction Exception. The 20Kc processor includes a standard TLB MMU and therefore this is not an issue.

Operation:

```
i <- Index
          if i > TLBEntries -1 then
             UNDEFINED
          endif
          PageMask_Mask <- TLB[i]_Mask
          EntryHi <- TLB[i]_R || 0^Fill ||
                    (TLB[i]_VPN2 and not TLB[i]_Mask) ||
                     0^5 || TLB[i]_ASID
          EntryLo1 <- 0^Fill || (TLB[i]_PFN1 and not TLB[i]_Mask) ||
```


Copyright © 2002 MIPS Technologies Inc. All rights reserved.

```
                  TLB[i]C1 || TLB[i]D1 || TLB[i]V1 || TLB[i]G
EntryLo0 <- 0Fill || (TLB[i]PFN0 and not TLB[i]Mask) ||
                  TLB[i]C0 || TLB[i]D0 || TLB[i]V0 || TLB[i]G
```

Exceptions:

Coprocessor Unusable Exception

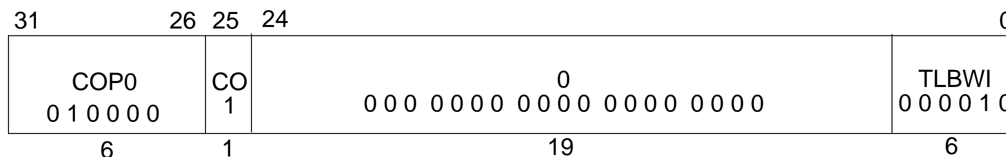
Reserved Instruction Exception (if not implemented)

Machine Check (if implemented and a TLB shutdown condition is detected on a TLB read)

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

13.7.3.4 The TLBWI Instruction

Write Indexed TLB Entry **TLBWI**



Format:

`TLBWI` **MIPS64**

Purpose:

Write a TLB entry indexed by the *Index* register.

Description:

The TLB entry pointed to by the *Index* register is written from the contents of the *EntryHi*, *EntryLo0*, *EntryLo1*, and *PageMask* registers. The information written to the TLB entry might be different from that in the *EntryHi*, *EntryLo0*, and *EntryLo1* registers in that:

- The value written to the VPN2 field of the TLB entry might have those bits set to zero corresponding to the one bits in the Mask field of the *PageMask* register (the least significant bit of VPN2 corresponds to the least significant bit of the Mask field). The 20Kc processor zeros out these bits on a write.
- The value written to the PFN0 and PFN1 fields of the TLB entry might have those bits set to zero corresponding to the one bits in the Mask field of the *PageMask* register (the least significant bit of PFN corresponds to the least significant bit of the Mask field). The 20Kc processor zeros out these bits on a write.
- The single G bit in the TLB entry is set from the logical AND of the G bits in the *EntryLo0* and *EntryLo1* registers.

Restrictions:

This instruction is legal only if the processor is in Kernel Mode or Debug Mode, or if the CP0 usable bit is set in the *Status* register. In other circumstances, execution of this instruction results in a Coprocessor Unusable Exception.

The operation is UNDEFINED if the contents of the *Index* register are greater than or equal to the number of TLB entries in the processor.

For processors that do not include the standard TLB MMU, the operation of this instruction is **UNDEFINED**. However, the preferred implementation is a Reserved Instruction Exception. The 20Kc processor includes a standard TLB MMU, therefore this is not an issue.

Operation:

```
i <- Index
        if i > TLBEntries -1 then
           UNDEFINED
        endif
        TLB[i]Mask <- PageMaskMask
        TLB[i]R <- EntryHiR
        TLB[i]VPN2 <- EntryHiVPN2 and not PageMaskMask
        TLB[i]ASID <- EntryHiASID
        TLB[i]G <- EntryLo1G and EntryLo0G
        TLB[i]PFN1 <- EntryLo1PFN and not PageMaskMask
        TLB[i]C1 <- EntryLo1C
        TLB[i]D1 <- EntryLo1D
        TLB[i]V1 <- EntryLo1V
```


Copyright © 2002 MIPS Technologies Inc. All rights reserved.

```
TLB[i]PFN0 <- EntryLo0PFN and not PageMaskMask
TLB[i]C0 <- EntryLo0C
TLB[i]D0 <- EntryLo0D
TLB[i]V0 <- EntryLo0V
```

Exceptions:

Coprocessor Unusable Exception

Reserved Instruction Exception (if not implemented)

Machine Check (if implemented and a TLB shutdown condition is detected on a TLB write)

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

13.7.3.5 The TLBWR Instruction

Write Random TLB Entry **TLBWR**

31–26	25	24–1	0
COP0 0 1 0 0 0 0	CO 1	0 000 0000 0000 0000 0000	TLBWR 0 0 0 1 1 0
6	1	19	6

Format:

`TLBWR` **MIPS64**

Purpose:

Write a TLB entry indexed by the *Random* register.

Description:

The TLB entry pointed to by the *Random* register is written from the contents of the *EntryHi*, *EntryLo0*, *EntryLo1*, and *PageMask* registers. The information written to the TLB entry may be different from that in the *EntryHi*, *EntryLo0*, and *EntryLo1* registers in that:

- The value written to the VPN2 field of the TLB entry may have those bits set to zero corresponding to the one bits in the Mask field of the *PageMask* register (the least significant bit of VPN2 corresponds to the least significant bit of the Mask field). The 20Kc processor zeros out these bits on a write.
- The value written to the PFN0 and PFN1 fields of the TLB entry may have those bits set to zero corresponding to the one bits in the Mask field of the *PageMask* register (the least significant bit of PFN corresponds to the least significant bit of the Mask field). The 20Kc processor zeros out these bits on a write.
- The single G bit in the TLB entry is set from the logical AND of the G bits in the *EntryLo0* and *EntryLo1* registers.

Restrictions:

This instruction is legal only if the processor is in Kernel Mode or Debug Mode, or if the CP0 usable bit is set in the Status register. In other circumstances, execution of this instruction results in a Coprocessor Unusable Exception.

For processors that do not include the standard TLB MMU, the operation of this instruction is **UNDEFINED**. However, the preferred implementation is a Reserved Instruction Exception. The 20Kc processor includes a standard TLB MMU, therefore this is not an issue.

Operation:

```
i <- Random
            TLB[i]Mask <- PageMaskMask
            TLB[i]R <- EntryHiR
            TLB[i]VPN2 <- EntryHiVPN2 and not PageMaskMask
            TLB[i]ASID <- EntryHiASID
            TLB[i]G <- EntryLo1G and EntryLo0G
            TLB[i]PFN1 <- EntryLo1PFN and not PageMaskMask
            TLB[i]C1 <- EntryLo1C
            TLB[i]D1 <- EntryLo1D
            TLB[i]V1 <- EntryLo1V
            TLB[i]PFN0 <- EntryLo0PFN and not PageMaskMask
            TLB[i]C0 <- EntryLo0C
            TLB[i]D0 <- EntryLo0D
            TLB[i]V0 <- EntryLo0V
```


Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Exceptions:

Coprocessor Unusable Exception

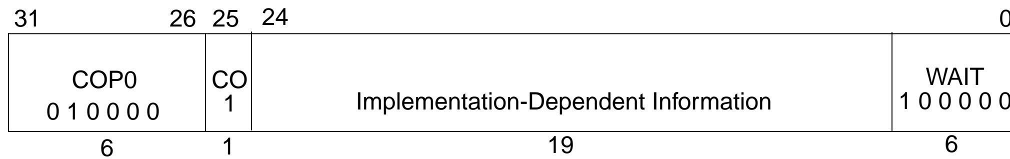
Reserved Instruction Exception (if not implemented)

Machine Check (if implemented and a TLB shutdown condition is detected on a TLB write)

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

13.7.3.6 The WAIT Instruction

Enter Standby Mode **WAIT**



Format:

`WAIT` **MIPS64**

Purpose:

Wait for Event.

Description:

The WAIT instruction initiates a power down sequence where the internal processor clock is divided by 16.

The WAIT instruction is implemented by stalling the pipeline at the completion of the instruction and entering a lower power mode. The pipeline is restarted when an external event, such as an interrupt or external request occurs, and execution continues with the instruction following the WAIT instruction. It is implementation-dependent whether the pipeline restarts when a non-enabled interrupt is requested. In this case, software must poll for the cause of the restart. If the pipeline restarts as the result of an enabled interrupt, that interrupt is taken between the WAIT instruction and the following instruction (EPC for the interrupt points at the instruction following the WAIT instruction).

The assertion of any reset or NMI signal (if not masked by EJTAG) must restart the pipeline, and the corresponding exception must be taken.

Upon restart the processor must wait until the internal clock PLL has locked before continuing instruction execution.

Restrictions:

The operation of the processor is **UNDEFINED** if a wait instruction is placed in the delay slot of a branch or a jump.

This instruction is legal only if the processor is in Kernel Mode or Debug Mode, or if the CP0 usable bit is set in the *Status* register. In other circumstances, execution of this instruction results in a Coprocessor Unusable Exception.

Operation:

```
Enter implementation dependent lower power mode
```

Exceptions:

Coprocessor Unusable Exception

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Floating-Point Unit

This chapter describes the 20Kc Floating-Point Unit (FPU) behavior not specifically covered by the MIPS64 and MIPS-3D ASE architectural definitions and includes the following sections:

- Section 14.1, "Special FCSR Bits"
- Section 14.2, "FCSR Cause Bit Update Flow"
- Section 14.3, "Denormal Handling"
- Section 14.4, "Reciprocal and Reciprocal Square Root"
- Section 14.5, "Single-Precision Result or Single-Word Load"
- Section 14.6, "QNaN Priority"
- Section 14.7, "Convert Ranges"

For more information on the MIPS floating-point architecture, refer to Section 13.2, "FPU Architecture".

14.1 Special FCSR Bits

The 20Kc FPU offers three bits in the Floating-Point Control and Status register (*FCSR*) to improve performance and accuracy when handling tiny numbers: flush-to-zero (FS), flush override (FO), and flush-to-nearest (FN). A tiny number is defined as a non-zero number that is smaller in magnitude than the minimum normal number (MinNorm).

14.1.1 Flush-to-Zero (FS)

Since denormal operands and tiny results can degrade the performance of an FPU, a fast non-IEEE-compliant flush-to-zero mode is provided by the MIPS architecture. This flush-to-zero mode is recommended for use in all applications not specifically requiring strict IEEE compliance with respect to denormal and tiny numbers.

When the *FCSR* bit 24 (FS) is set, denormal input operands are flushed to zero. Tiny results are flushed to either zero or MinNorm depending on the rounding mode settings. Also note that all flushing actions set the Inexact cause bit in the *FCSR*.

Table 14-1 shows the effect of rounding mode on tiny result flushing.

Table 14-1 Flushing of Results

Rounding Mode	Negative Tiny	Positive Tiny
RN (RM=0)	-Zero	+Zero
RZ (RM=1)	-Zero	+Zero
RP (RM=2)	-Zero	+MinNorm
RM (RM=3)	-MinNorm	+Zero

The flushing of results is based on an intermediate result that is computed by rounding the mantissa using an unbounded exponent range; that is, tiny numbers are not normalized into the supported exponent range prior to rounding.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

14.1.2 Flush-Override (FO)

When the *FCSR* bit 22 (FO) is set, a tiny intermediate multiply result of any multiply-add type instruction, including RECIP2.fmt and RSQRT2.fmt, is not flushed according to the FS bit. The intermediate result is maintained in an internal format that has a larger exponent range to improve accuracy. The primary intention is to increase accuracy when using MIPS-3D instructions to pipeline reciprocal or reciprocal square root operations.

14.1.3 Flush-to-Nearest (FN)

When the *FCSR* bit 21 (FN) is set and the rounding mode is round-to-nearest (RN), a tiny final result is flushed to zero or MinNorm, whichever is nearer. If a tiny number is strictly below MinNorm/2, then the result is flushed to zero; otherwise, it is flushed to MinNorm. The flushed result has the same sign as the result prior to flushing. Note that the FN bit takes precedence over the FS bit for result flushing. For rounding modes other than round-to-nearest (RN), setting the FN bit will cause final results to be flushed to zero or MinNorm as if the FS bit were set. As with the FO bit, the primary intention is to increase accuracy when using MIPS-3D instructions to pipeline reciprocal or reciprocal square root operations.

14.1.4 Summary of FS, FO and FN Bits

Table 14-2 summarizes the flushing behavior for all settings of FS, FO and FN.

Table 14-2 Denorm/Tiny Handling for All Combinations of FS/FO/FN

FS	FO	FN	Denorm Input	Tiny Intermediate MADD Result	Tiny Final Result	Remarks
0	0	0	E[1]	E	E	IEEE-compliant mode
0	0	1	E	E	UI(N)[2, 3, 4]	
0	1	0	E	-	E	
0	1	1	E	-	UI(N)	
1	0	0	I(0)[5]	UI(0)	UI(0)	Regular MIPS64 embedded application
1	0	1	I(0)	UI(0)	UI(N)	
1	1	0	I(0)	-	UI(0)	
1	1	1	I(0)	-	UI(N)	20Kc highest accuracy and performance configuration

1. E: Unimplemented exception
2. U: Underflow cause set
3. I: Inexact cause set
4. (N): If rounding mode is round-to-nearest, flush-to-nearest mode takes effect; otherwise flush-to-zero mode takes effect
5. (0): Flush-to-zero mode takes effect

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

14.2 FCSR Cause Bit Update Flow

14.2.1 Exceptions Triggered by CTC1

Regardless of the targeted floating-point control register, the CTC1 instruction inspects the enable and cause bits in the *FCSR* to determine if an exception has occurred.

14.2.2 Generic Cause Bit Update Flow

Floating-point computations are performed in two steps:

1. Compute rounded mantissa with unbounded exponent range.
2. Flush to default result if result from the step above overflows or is tiny (no flushing happens on denorms for instructions supporting denorm results).

The cause bits in the *FCSR* are updated after each of these two steps. Any enabled exceptions detected in each of these two steps cause a trap, and subsequent steps do not update the cause bits.

Step #1 can set any of the cause bits: I (inexact), U (underflow), O (overflow), Z (divide-by-zero), V (invalid), and E (unimplemented). E has priority over V, V has priority over Z, and Z has priority over U and O. When E, V, or Z is set in Step #1, no other cause bits can be set. Note, however, that I can be set with V if a denormal operand was flushed (FS = 1). I, U, and O can be set alone or in pairs: I, U or I, O. U and O can never both be set in Step #1. U and O are set if the computed unbounded exponent is outside the exponent range supported by the normalized IEEE format.

Step #2 can set I if the default result is generated.

14.2.3 Multiply-Add Cause Bit Update Flow

For multiply-add type instructions, the computation is extended with two additional steps:

1. Compute rounded mantissa with unbounded exponent range for multiply.
2. Flush to default result if the result from Step #1 is overflow or tiny (no flushing happens on tiny results if FO = 1).
3. Compute rounded mantissa with unbounded exponent range for add.
4. Flush to default result if the result from Step #3 overflows or is tiny.

The cause bits are updated after each of these four steps. Any enabled exceptions detected in each of these four steps causes a trap and subsequent steps do not update the cause bits.

Step #1 and Step #3 can set the cause bits as described for Step #1 in Section 14.2.2, "Generic Cause Bit Update Flow".

Step #2 and Step #4 can set I if the default result is generated.

Although U and O can never both be set in Step #1 or Step #3, both U and O might be set after the multiply-add has executed, because U might be set in Step #1, and O might be set in Step #3.

14.2.4 Cause Update Flow for Operands

Exceptional conditions directly related to the operand (for example, Cause V due to an SNaN operand) are detected in the step where the operand is used. For example, in multiply-add type instructions, exceptional conditions caused by the operand *fr* are detected in Step #3.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Denormal operands to Step #1 or Step #3 always set cause I when FS=1.

14.2.5 Cause Update Flow for Paired Single

For paired-single instructions, each of the two or four steps described above now contains two parallel computations for each of the two halves. Any cause bits set by either half in a given step are ORed together for the given step. Thus, for paired-single instructions, both U and O can be set in Step #1 or Step #3, because one half might set U while the other half sets O.

14.2.6 Cause Update Flow for Unimplemented Operation

The behavior of cause E (unimplemented) is special; it clears any previous cause updates from previous steps. For example, if Step #3 sets cause E, any updates done in Step #1 or Step #2 are cleared. Only E is set in the *FCSR* when an Unimplemented Operation trap is taken.

14.3 Denormal Handling

The 20Kc FPU handles denormal numbers for the following arithmetic operations:

- ABS.fmt
- NEG.fmt
- All compare instructions

Denormal operands or results in combination with any arithmetic instructions not listed above could cause an unimplemented exception, depending upon the settings of FS and FN (see Table 14-2). When the FS bit has been set, all denormal operands and results are flushed to zero or MinNorm, regardless of the selected operation. Specifically, this implies that when comparing denormal numbers in flush-to-zero mode, the operands are flushed to zero before comparison.

Table 14-3 summarizes 20Kc denormal operand handling for various instruction types:

Table 14-3 Denormal Operand Handling

FS bit of FCSR	Instruction Type	Behavior
FS=1	Abs/Neg	Flush to zero
	Compare	Flush to zero before comparison
	All other arithmetic	Flush to zero
FS=0	Abs/Neg	Handle denormal
	Compare	Handle denormal
	All other arithmetic	Unimplemented exception

Data movement instructions such as MOV.fmt do not interpret their data and are thus unaffected by denormal operands and the FS bit.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

14.4 Reciprocal and Reciprocal Square Root

14.4.1 Forced Inexact

Four approximation instructions force inexact on a computed result (the Inexact cause bit in the *FCSR* is set). These instructions are:

- RECIP.fmt
- RECIP1.fmt
- RSQRT.fmt
- RSQRT1.fmt

Because these instructions are by nature approximations, the results do not comply with the IEEE standard for accuracy and thus are always considered inexact.

14.4.2 Forced Round-to-Nearest

Four MIPS-3D instructions, RECIP1.fmt, RSQRT1.fmt, RECIP2.fmt, and RSQRT2.fmt, do not use the current rounding mode in the *FCSR*. Instead they always operate in the round-to-nearest rounding mode. Because these instructions are used as part of a pipelined reciprocal or reciprocal square root approximation sequence, round-to-nearest is used to increase accuracy.

14.4.3 Forced Flush-to-Nearest

Two MIPS-3D instructions, RECIP1.fmt and RSQRT1.fmt, do not use the value of FN (flush-to-nearest) in the *FCSR*. Instead they always operate in flush-to-nearest mode to increase accuracy. This implies that if the result of RECIP1.fmt and RSQRT1.fmt is tiny, an unimplemented exception is never taken even if FS and FN are zero in the *FCSR*.

14.4.4 Special Results

Table 14-4 lists the default answers generated for certain MIPS-3D ASE instructions.

Table 14-4 Default Answers for RECIPx, RSQRTx ASE Instructions

Input	RECIP1	RECIP2	RSQRT1	RSQRT2
+Zero	+MaxNorm	+Zero	+MaxNorm	+Zero
-Zero	-MaxNorm	+Zero	-MaxNorm	+Zero
+Inf	+Zero	+Zero	+Zero	+Zero
-Inf	-Zero	+Zero	QNaN	+Zero

The purpose of these default answers is to avoid infinite results when using MIPS-3D instructions in graphics applications.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

14.4.5 RSQRT2 Implementation

The MIPS3D instruction RSQRT2.fmt is a step in a pipelined reciprocal square root operation, consisting of the following operation:

```
fd = -(fs * ft - 1.0) / 2
```

The 20Kc processor implements the RSQRT2.fmt instruction as a special type of NMSUB.fmt with the following equivalent operation:

```
fd = - ((fs * ft) / 2 - 0.5)
```

The division by two is merged with the multiplication and is performed before any overflow or underflow conditions are detected.

Although normal use of RSQRT2.fmt should never result in overflow or underflow, the two equations above do have slightly different behavior with respect to exceptional conditions.

14.5 Single-Precision Result or Single-Word Load

In MIPS64 mode, all 32-bit results from a single-precision floating-point operation, single-word load, or single-word move are replicated to both lower 32-bit and upper 32-bit halves of a 64-bit FPR destination entry. The intention is to speed up graphic applications by avoiding execution of the CVT.PS.S instruction.

This behavior is a 20Kc specific implementation feature that is allowed by the MIPS architecture, since the upper 32 bits are declared as undefined for these cases.

14.6 QNaN Priority

In case of one or more QNaN operands (no SNaN operand), the QNaN operand is propagated from one of the input operands in this order of priority: *fs*, *ft*, *fr*. However, if the multiply result of a MADD-type instruction is a QNaN operand, this is prioritized over a QNaN in *fr*, because the MADD.fmt instruction is considered to be equivalent to a MUL.fmt followed by an ADD.fmt.

14.7 Convert Ranges

For certain format conversion operations, the 20Kc processor only supports a range of the legal input operands. If the input operand is out of range, an unimplemented exception occurs. Table 14-5 and Table 14-6 specify the minimum and maximum operand ranges supported in hardware.

14.7.1 Convert Integer to Float

Table 14-5 Convert Integer to Float: CVT.[DS].[WL]

Instruction	Minimum	Maximum
CVT.S.W	0xFF80.0000	0x007F.FFFF
CVT.S.L	0xFFFF.FFFF.FF80.0000	0x0000.0000.007F.FFFF
CVT.D.L	0xFFF8.0000.0000.0000	0x0007.FFFF.FFFF.FFFF

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Table 14-5 Convert Integer to Float: CVT.[DS].[WL]

Instruction	Minimum	Maximum
CVT.D.W	0x8000.0000	0x7FFF.FFFF

14.7.2 Convert Float to Integer

Table 14-6 Convert Float to Int: CVT/ROUND/CEIL/FLOOR/TRUNC/.[WL].[DS]

Instruction	Minimum	Maximum
convert.W.S	0xCAFF.FFFF	0x4AFF.FFFF
convert.W.D	0xC1CF.FFFF.FFFF.FFFF	0x41CF.FFFF.FFFF.FFFF
convert.L.S	0xCAFF.FFFF	0x4AFF.FFFF
convert.L.D	0x32F.FFFF.FFFF.FFFF	0x432F.FFFF.FFFF

14.7.3 Convert Double to Single Precision: CVT.S.D

Conversion from double to single precision is supported for all ranges. Values that when rounded are outside the full single-precision range will result in an overflow or underflow.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Revision History

In the left hand page margins of this document you may find vertical change bars to note the location of significant changes to this document since its last release. Significant changes are defined as those which you should take note of as you use the MIPS IP. Changes to correct grammar, spelling errors or similar may or may not be noted with change bars. Change bars will be removed for changes which are more than one revision old.

Revision	Date	Description
01.00	March 21, 2001	• Converted entire manual to latest template.
01.01	March 31, 2001	• Added miscellaneous hard copy edits to manual.
01.02	June 4, 2001	• Converted manual to latest template
01.03	June 6, 2001	• Minor edits and revisions made
01.04	August 31, 2001	• Edited titles, trademarks and revision number. Added Ch. 14 to manual.
01.05	September 12, 2001	• Made additional minor edits to Ch. 14 FPU. Minor edits made in Chapter 7 Caches
01.06	September 17, 2001	• Regenerates TOC, LOT and LOF.
1.07	December 14, 2001	• Updated template and removed non-template fonts. • Added new sections to Chapter 14, FPU, on FCSR Cause bit, Forced Flush-to-Nearest, and RSQRT2 Operation. • Updated Table 14-4. • Added VI bit to Config register. • WAIT instruction changes. • SR (Soft Reset) bit clarified. • Compatability segment decoding clarified. • Removed process-specific information. • Added programming notes to CACHE Instruction description in Chapter 13. • Updated latency for DIV and DDIV instructions in Table 3-1. • Made global edits to the document.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Revision	Date	Description
1.08	March 28, 2002	• ITagLo register definition corrected in Chapter 6. • Corrected name of register 26 in Table 6-1. • Minor writing style changes. • Description update for Table 6-15 ERL bit. • BIST testability description updated. Bist Algorithm selection and retention timing information have been introduced in Chapter 12 • Added more details and examples on coherency conflict checking, and added subsection on flow control implications for SOC controller design in Chapter 8. • Added additional information on Virtual Icache and implications to software programmer in Chapter 7. • Added description of Instruction recode algorithm and its effect on IDataLo/IDataHi register content to Chapter 6. • Added more software programming guidelines for uncached accelerated stores in Chapter 7.
01.09	June 21, 2002	• Updated Power-On Reset Sequence (9.3.1) #1 and#2 deleted. • Updated PRID Register Field Descriptions "Revision" field description in Table 6-19. • Updated Table 6-35 DTagLo Register Field Descriptions. PState[1] description is changed from Valid to Reserved. • Added explanation of change bars to the readers above the revision history table • Changed *TS_BistInvoke* to *TS_BistHold* in the last 2 sentences of 12.3.2.2.
01.10	Sept 28, 2002	• Updated Table 3-1 and Table 3-2. • Updated section 4.8, Figure 4-12 and Figure 4-13. • Updated section 5.7.9. • Updated Table 6-19 and Table 6-35. • Updated section 8.1.5 and Figure 8-4 • Updated Address field of Data Register Contents in Table 11-40. • Updated Figure 12-11 and Figure 12-12.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

References

[1] *EJTAG Specification, Revision 2.6*, Document Number MD00047

[2] *MIPS64™ 20Kc™ TSMC CL013LV Processor Core Datasheet*, Document Number MD00125

[3] *MIPS64™ 20Kc™ TSMC CL018G Processor Core Datasheet*, Document Number MD00207

[4] IEEE Std. 1149.1-1990, *IEEE Standard Test Access Port and Boundary-Scan Architecture*

Copyright © 2002 MIPS Technologies Inc. All rights reserved.

Copyright © 2002 MIPS Technologies Inc. All rights reserved.