Linux Loongarch Documentation

The kernel development community

Jan 15, 2023

CONTENTS

CHAPTER

ONE

INTRODUCTION TO LOONGARCH

LoongArch is a new RISC ISA, which is a bit like MIPS or RISC-V. There are currently 3 variants: a reduced 32-bit version (LA32R), a standard 32-bit version (LA32S) and a 64-bit version (LA64). There are 4 privilege levels (PLVs) defined in LoongArch: PLV0~PLV3, from high to low. Kernel runs at PLV0 while applications run at PLV3. This document introduces the registers, basic instruction set, virtual memory and some other topics of LoongArch.

1.1 Registers

LoongArch registers include general purpose registers (GPRs), floating point registers (FPRs), vector registers (VRs) and control status registers (CSRs) used in privileged mode (PLV0).

1.1.1 GPRs

LoongArch has 32 GPRs (`$r0` ~ `$r31`); each one is 32-bit wide in LA32 and 64-bit wide in LA64. `$r0` is hard-wired to zero, and the other registers are not architecturally special. (Except `$r1`, which is hard-wired as the link register of the BL instruction.)

The kernel uses a variant of the LoongArch register convention, as described in the LoongArch ELF psABI spec, in *References*:

Name	Alias	Usage	Preserved across calls
`$r0`	`$zero`	Constant zero	Unused
`$r1`	`$ra`	Return address	No
`$r2`	`$tp`	TLS/Thread pointer	Unused
`$r3`	`$sp`	Stack pointer	Yes
`$r4`-`$r11`	`$a0`-`$a7`	Argument registers	No
`$r4`-`$r5`	`$v0`-`$v1`	Return value	No
`$r12`-`$r20`	`$t0`-`$t8`	Temp registers	No
`$r21`	`$u0`	Percpu base address	Unused
`$r22`	`$fp`	Frame pointer	Yes
`$r23`-`$r31`	`$s0`-`$s8`	Static registers	Yes

Note: The register `$r21` is reserved in the ELF psABI, but used by the Linux kernel for storing the percpu base address. It normally has no ABI name, but is called `$u0` in the kernel. You may also see `$v0` or `$v1` in some old code,however they are deprecated aliases of `$a0` and `$a1` respectively.

1.1.2 FPRs

LoongArch has 32 FPRs (`$f0` ~ `$f31`) when FPU is present. Each one is 64-bit wide on the LA64 cores.

The floating-point register convention is the same as described in the LoongArch ELF psABI spec:

Name	Alias	Usage	Preserved across calls
`$f0-$f7`	`$fa0-$fa7`	Argument registers	No
`$f0-$f1`	`$fv0-$fv1`	Return value	No
`$f8-$f23`	`$ft0-$ft15`	Temp registers	No
`$f24-$f31`	`$fs0-$fs7`	Static registers	Yes

Note: You may see `$fv0` or `$fv1` in some old code, however they are deprecated aliases of `$fa0` and `$fa1` respectively.

1.1.3 VRs

There are currently 2 vector extensions to LoongArch:

- LSX (Loongson SIMD eXtension) with 128-bit vectors,
- LASX (Loongson Advanced SIMD eXtension) with 256-bit vectors.

LSX brings `$v0` ~ `$v31` while LASX brings `$x0` ~ `$x31` as the vector registers.

The VRs overlap with FPRs: for example, on a core implementing LSX and LASX, the lower 128 bits of `$x0` is shared with `$v0`, and the lower 64 bits of `$v0` is shared with `$f0`; same with all other VRs.

1.1.4 CSRs

CSRs can only be accessed from privileged mode (PLV0):

Address	Full Name	Abbrev Name
0x0	Current Mode Information	CRMD
0x1	Pre-exception Mode Information	PRMD
0x2	Extension Unit Enable	EUEN
0x3	Miscellaneous Control	MISC
0x4	Exception Configuration	ECFG
0x5	Exception Status	ESTAT
0x6	Exception Return Address	ERA
0x7	Bad (Faulting) Virtual Address	BADV

continues on next page

Table 1 – continued from previous page

Address	Full Name	Abbrev Name
0x8	Bad (Faulting) Instruction Word	BADI
0xC	Exception Entrypoint Address	EENTRY
0x10	TLB Index	TLBIDX
0x11	TLB Entry High-order Bits	TLBEHI
0x12	TLB Entry Low-order Bits 0	TLBELO0
0x13	TLB Entry Low-order Bits 1	TLBELO1
0x18	Address Space Identifier	ASID
0x19	Page Global Directory Address for Lower-half Address Space	PGDL
0x1A	Page Global Directory Address for Higher-half Address Space	PGDH
0x1B	Page Global Directory Address	PGD
0x1C	Page Walk Control for Lower- half Address Space	PWCL
0x1D	Page Walk Control for Higher- half Address Space	PWCH
0x1E	STLB Page Size	STLBPS
0x1F	Reduced Virtual Address Configuration	RVACFG
0x20	CPU Identifier	CPUID
0x21	Privileged Resource Configuration 1	PRCFG1
0x22	Privileged Resource Configuration 2	PRCFG2
0x23	Privileged Resource Configuration 3	PRCFG3
0x30+n (0≤n≤15)	Saved Data register	SAVEn
0x40	Timer Identifier	TID
0x41	Timer Configuration	TCFG
0x42	Timer Value	TVAL
0x43	Compensation of Timer Count	CNTC
0x44	Timer Interrupt Clearing	TICLR
0x60	LLBit Control	LLBCTL
0x80	Implementation-specific Control 1	IMPCTL1
0x81	Implementation-specific Control 2	IMPCTL2
0x88	TLB Refill Exception Entrypoint Address	TLBRENTRY
0x89	TLB Refill Exception BAD (Faulting) Virtual Address	TLBRBADV
0x8A	TLB Refill Exception Return Address	TLBRERA
0x8B	TLB Refill Exception Saved Data Register	TLBRSAVE
0x8C	TLB Refill Exception Entry Low-order Bits 0	TLBRELO0
0x8D	TLB Refill Exception Entry Low-order Bits 1	TLBRELO1
0x8E	TLB Refill Exception Entry High-order Bits	TLBEHI
0x8F	TLB Refill Exception Pre-exception Mode Information	TLBRPRMD
0x90	Machine Error Control	MERRCTL
0x91	Machine Error Information 1	MERRINFO1
0x92	Machine Error Information 2	MERRINFO2
0x93	Machine Error Exception Entrypoint Address	MERRENTRY
0x94	Machine Error Exception Return Address	MERRERA
0x95	Machine Error Exception Saved Data Register	MERRSAVE
0x98	Cache TAGs	CTAG
0x180+n (0≤n≤3)	Direct Mapping Configuration Window n	DMWn
0x200+2n (0≤n≤31)	Performance Monitor Configuration n	PMCFGn
0x201+2n (0≤n≤31)	Performance Monitor Overall Counter n	PMCNTn
0x300	Memory Load/Store WatchPoint Overall Control	MWPC
0x301	Memory Load/Store WatchPoint Overall Status	MWPS

continues on next page

Table 1 – continued from previous page

Address	Full Name	Abbrev Name
0x310+8n (0≤n≤7)	Memory Load/Store WatchPoint n Configuration 1	MWPnCFG1
0x311+8n (0≤n≤7)	Memory Load/Store WatchPoint n Configuration 2	MWPnCFG2
0x312+8n (0≤n≤7)	Memory Load/Store WatchPoint n Configuration 3	MWPnCFG3
0x313+8n (0≤n≤7)	Memory Load/Store WatchPoint n Configuration 4	MWPnCFG4
0x380	Instruction Fetch WatchPoint Overall Control	FWPC
0x381	Instruction Fetch WatchPoint Overall Status	FWPS
0x390+8n (0≤n≤7)	Instruction Fetch WatchPoint n Configuration 1	FWPnCFG1
0x391+8n (0≤n≤7)	Instruction Fetch WatchPoint n Configuration 2	FWPnCFG2
0x392+8n (0≤n≤7)	Instruction Fetch WatchPoint n Configuration 3	FWPnCFG3
0x393+8n (0≤n≤7)	Instruction Fetch WatchPoint n Configuration 4	FWPnCFG4
0x500	Debug Register	DBG
0x501	Debug Exception Return Address	DERA
0x502	Debug Exception Saved Data Register	DSAVE

ERA, TLBRERA, MERRERA and DERA are sometimes also known as EPC, TLBREPC, MERREPC and DEPC respectively.

1.2 Basic Instruction Set

1.2.1 Instruction formats

LoongArch instructions are 32 bits wide, belonging to 9 basic instruction formats (and variants of them):

Format name	Composition
2R	Opcode + Rj + Rd
3R	Opcode + Rk + Rj + Rd
4R	Opcode + Ra + Rk + Rj + Rd
2RI8	Opcode + I8 + Rj + Rd
2RI12	Opcode + I12 + Rj + Rd
2RI14	Opcode + I14 + Rj + Rd
2RI16	Opcode + I16 + Rj + Rd
1RI21	Opcode + I21L + Rj + I21H
I26	Opcode + I26L + I26H

Rd is the destination register operand, while Rj, Rk and Ra (“a” stands for “additional”) are the source register operands. I8/I12/I16/I21/I26 are immediate operands of respective width. The longer I21 and I26 are stored in separate higher and lower parts in the instruction word, denoted by the “L” and “H” suffixes.

1.2.2 List of Instructions

For brevity, only instruction names (mnemonics) are listed here; please see the *References* for details.

1. Arithmetic Instructions:

```
ADD.W SUB.W ADDI.W ADD.D SUB.D ADDI.D
SLT SLTU SLTI SLTUI
AND OR NOR XOR ANDN ORN ANDI ORI XORI
MUL.W MULH.W MULH.WU DIV.W DIV.WU MOD.W MOD.WU
MUL.D MULH.D MULH.DU DIV.D DIV.DU MOD.D MOD.DU
PCADDI PCADDU12I PCADDU18I
LU12I.W LU32I.D LU52I.D ADDU16I.D
```

2. Bit-shift Instructions:

```
SLL.W SRL.W SRA.W ROTR.W SLLI.W SRLI.W SRAI.W ROTRI.W
SLL.D SRL.D SRA.D ROTR.D SLLI.D SRLI.D SRAI.D ROTRI.D
```

3. Bit-manipulation Instructions:

```
EXT.W.B EXT.W.H CLO.W CLO.D SLZ.W CLZ.D CTO.W CTO.D CTZ.W CTZ.D
BYTEPICK.W BYTEPICK.D BSTRINS.W BSTRINS.D BSTRPICK.W BSTRPICK.D
REVB.2H REVB.4H REVB.2W REVB.D REVH.2W REVH.D BITREV.4B BITREV.8B BITREV.W␣
↪BITREV.D
MASKEQZ MASKNEZ
```

4. Branch Instructions:

```
BEQ BNE BLT BGE BLTU BGEU BEQZ BNEZ B BL JIRL
```

5. Load/Store Instructions:

```
LD.B LD.BU LD.H LD.HU LD.W LD.WU LD.D ST.B ST.H ST.W ST.D
LDX.B LDX.BU LDX.H LDX.HU LDX.W LDX.WU LDX.D STX.B STX.H STX.W STX.D
LDPTR.W LDPTR.D STPTR.W STPTR.D
PRELD PRELDX
```

6. Atomic Operation Instructions:

```
LL.W SC.W LL.D SC.D
AMSWAP.W AMSWAP.D AMADD.W AMADD.D AMAND.W AMAND.D AMOR.W AMOR.D AMXOR.W␣
↪AMXOR.D
AMMAX.W AMMAX.D AMMIN.W AMMIN.D
```

7. Barrier Instructions:

```
IBAR DBAR
```

8. Special Instructions:

```
SYSCALL BREAK CPUCFG NOP IDLE ERTN(ERET) DBCL(DBGCALL) RDTIMEL.W RDTIMEH.W␣
↪RDTIME.D
ASRTLE.D ASRTGT.D
```

9. Privileged Instructions:

```
CSRRD CSRWR CSRXCHG
IOCSRRD.B IOCSRRD.H IOCSRRD.W IOCSRRD.D IOCSRWR.B IOCSRWR.H IOCSRWR.W␣
↪IOCSRWR.D
CACOP TLBP(TLBSRCH) TLBRD TLBWR TLBFILL TLBCLR TLBFLUSH INVTLB LDDIR LDPTE
```

1.3 Virtual Memory

LoongArch supports direct-mapped virtual memory and page-mapped virtual memory.

Direct-mapped virtual memory is configured by CSR.DMWn (n=0~3), it has a simple relationship between virtual address (VA) and physical address (PA):

```
VA = PA + FixedOffset
```

Page-mapped virtual memory has arbitrary relationship between VA and PA, which is recorded in TLB and page tables. LoongArch's TLB includes a fully-associative MTLB (Multiple Page Size TLB) and set-associative STLB (Single Page Size TLB).

By default, the whole virtual address space of LA32 is configured like this:

Name	Address Range	Attributes
UVRANGE	0x00000000 - 0x7FFFFFFF	Page-mapped, Cached, PLV0~3
KPRANGE0	0x80000000 - 0x9FFFFFFF	Direct-mapped, Uncached, PLV0
KPRANGE1	0xA0000000 - 0xBFFFFFFF	Direct-mapped, Cached, PLV0
KVRANGE	0xC0000000 - 0xFFFFFFFF	Page-mapped, Cached, PLV0

User mode (PLV3) can only access UVRANGE. For direct-mapped KPRANGE0 and KPRANGE1, PA is equal to VA with bit30~31 cleared. For example, the uncached direct-mapped VA of 0x00001000 is 0x80001000, and the cached direct-mapped VA of 0x00001000 is 0xA0001000.

By default, the whole virtual address space of LA64 is configured like this:

Name	Address Range	Attributes
XUVRANGE	0x0000000000000000 - 0x3FFFFFFFFFFFFFFF	Page-mapped, Cached, PLV0~3
XSPRANGE	0x4000000000000000 - 0x7FFFFFFFFFFFFFFF	Direct-mapped, Cached / Uncached, PLV0
XKPRANGE	0x8000000000000000 - 0xBFFFFFFFFFFFFFFF	Direct-mapped, Cached / Uncached, PLV0
XKVRANGE	0xC000000000000000 - 0xFFFFFFFFFFFFFFFF	Page-mapped, Cached, PLV0

User mode (PLV3) can only access XUVRANGE. For direct-mapped XSPRANGE and XKPRANGE, PA is equal to VA with bits 60~63 cleared, and the cache attribute is configured by bits 60~61 in VA: 0 is for strongly-ordered uncached, 1 is for coherent cached, and 2 is for weakly-ordered uncached.

Currently we only use XKPRANGE for direct mapping and XSPRANGE is reserved.

To put this in action: the strongly-ordered uncached direct-mapped VA (in XKPRANGE) of 0x00000000_00001000 is 0x80000000_00001000, the coherent cached direct-mapped VA (in XKPRANGE) of 0x00000000_00001000 is 0x90000000_00001000, and the weakly-ordered uncached direct-mapped VA (in XKPRANGE) of 0x00000000 _00001000 is 0xA0000000_00001000.

1.4 Relationship of Loongson and LoongArch

LoongArch is a RISC ISA which is different from any other existing ones, while Loongson is a family of processors. Loongson includes 3 series: Loongson-1 is the 32-bit processor series, Loongson-2 is the low-end 64-bit processor series, and Loongson-3 is the high-end 64-bit processor series. Old Loongson is based on MIPS, while New Loongson is based on LoongArch. Take Loongson-3 as an example: Loongson-3A1000/3B1500/3A2000/3A3000/3A4000 are MIPS-compatible, while Loongson- 3A5000 (and future revisions) are all based on LoongArch.

1.5 References

Official web site of Loongson Technology Corp. Ltd.:

  http://www.loongson.cn/

Developer web site of Loongson and LoongArch (Software and Documentation):

  http://www.loongnix.cn/

  https://github.com/loongson/

  https://loongson.github.io/LoongArch-Documentation/

Documentation of LoongArch ISA:

  https://github.com/loongson/LoongArch-Documentation/releases/latest/download/LoongArch-Vol1-v1.00-CN.pdf (in Chinese)

  https://github.com/loongson/LoongArch-Documentation/releases/latest/download/LoongArch-Vol1-v1.00-EN.pdf (in English)

Documentation of LoongArch ELF psABI:

  https://github.com/loongson/LoongArch-Documentation/releases/latest/download/LoongArch-ELF-ABI-v1.00-CN.pdf (in Chinese)

  https://github.com/loongson/LoongArch-Documentation/releases/latest/download/LoongArch-ELF-ABI-v1.00-EN.pdf (in English)

Linux kernel repository of Loongson and LoongArch:

  https://git.kernel.org/pub/scm/linux/kernel/git/chenhuacai/linux-loongson.git

CHAPTER

TWO

IRQ CHIP MODEL (HIERARCHY) OF LOONGARCH

Currently, LoongArch based processors (e.g. Loongson-3A5000) can only work together with LS7A chipsets. The irq chips in LoongArch computers include CPUINTC (CPU Core Interrupt Controller), LIOINTC (Legacy I/O Interrupt Controller), EIOINTC (Extended I/O Interrupt Controller), HTVECINTC (Hyper-Transport Vector Interrupt Controller), PCH-PIC (Main Interrupt Controller in LS7A chipset), PCH-LPC (LPC Interrupt Controller in LS7A chipset) and PCH-MSI (MSI Interrupt Controller).

CPUINTC is a per-core controller (in CPU), LIOINTC/EIOINTC/HTVECINTC are per-package controllers (in CPU), while PCH-PIC/PCH-LPC/PCH-MSI are controllers out of CPU (i.e., in chipsets). These controllers (in other words, irqchips) are linked in a hierarchy, and there are two models of hierarchy (legacy model and extended model).

2.1 Legacy IRQ model

In this model, IPI (Inter-Processor Interrupt) and CPU Local Timer interrupt go to CPUINTC directly, CPU UARTS interrupts go to LIOINTC, while all other devices interrupts go to PCH-PIC/PCH-LPC/PCH-MSI and gathered by HTVECINTC, and then go to LIOINTC, and then CPUINTC:

```
+-----+     +---------+     +-------+
| IPI | --> | CPUINTC | <-- | Timer |
+-----+     +---------+     +-------+
                 ^
                 |
            +---------+     +-------+
            | LIOINTC | <-- | UARTs |
            +---------+     +-------+
                 ^
                 |
           +-----------+
           | HTVECINTC |
           +-----------+
            ^         ^
            |         |
      +---------+ +---------+
      | PCH-PIC | | PCH-MSI |
      +---------+ +---------+
        ^     ^           ^
        |     |            |
+---------+ +---------+ +---------+
| PCH-LPC | | Devices | | Devices |
+---------+ +---------+ +---------+
     ^
     |
+---------+
| Devices |
+---------+
```

2.2 Extended IRQ model

In this model, IPI (Inter-Processor Interrupt) and CPU Local Timer interrupt go to CPUINTC directly, CPU UARTS interrupts go to LIOINTC, while all other devices interrupts go to PCH-PIC/PCH-LPC/PCH-MSI and gathered by EIOINTC, and then go to to CPUINTC directly:

```
      +-----+     +---------+     +-------+
      | IPI | --> | CPUINTC | <-- | Timer |
      +-----+     +---------+     +-------+
                   ^       ^
                   |       |
            +---------+ +---------+     +-------+
            | EIOINTC | | LIOINTC | <-- | UARTs |
            +---------+ +---------+     +-------+
             ^       ^
             |       |
      +---------+ +---------+
      | PCH-PIC | | PCH-MSI |
      +---------+ +---------+
        ^     ^           ^
        |     |           |
+---------+ +---------+ +---------+
| PCH-LPC | | Devices | | Devices |
+---------+ +---------+ +---------+
     ^
     |
+---------+
| Devices |
+---------+
```

2.3 ACPI-related definitions

CPUINTC:

```
ACPI_MADT_TYPE_CORE_PIC;
struct acpi_madt_core_pic;
enum acpi_madt_core_pic_version;
```

LIOINTC:

```
ACPI_MADT_TYPE_LIO_PIC;
struct acpi_madt_lio_pic;
enum acpi_madt_lio_pic_version;
```

EIOINTC:

```
ACPI_MADT_TYPE_EIO_PIC;
struct acpi_madt_eio_pic;
enum acpi_madt_eio_pic_version;
```

HTVECINTC:

```
ACPI_MADT_TYPE_HT_PIC;
struct acpi_madt_ht_pic;
enum acpi_madt_ht_pic_version;
```

PCH-PIC:

```
ACPI_MADT_TYPE_BIO_PIC;
struct acpi_madt_bio_pic;
enum acpi_madt_bio_pic_version;
```

PCH-MSI:

```
ACPI_MADT_TYPE_MSI_PIC;
struct acpi_madt_msi_pic;
enum acpi_madt_msi_pic_version;
```

PCH-LPC:

```
ACPI_MADT_TYPE_LPC_PIC;
struct acpi_madt_lpc_pic;
enum acpi_madt_lpc_pic_version;
```

2.4 References

Documentation of Loongson-3A5000:

> https://github.com/loongson/LoongArch-Documentation/releases/latest/download/Loongson-3A5000-usermanual-1.02-CN.pdf (in Chinese)
>
> https://github.com/loongson/LoongArch-Documentation/releases/latest/download/Loongson-3A5000-usermanual-1.02-EN.pdf (in English)

Documentation of Loongson's LS7A chipset:

> https://github.com/loongson/LoongArch-Documentation/releases/latest/download/Loongson-7A1000-usermanual-2.00-CN.pdf (in Chinese)
>
> https://github.com/loongson/LoongArch-Documentation/releases/latest/download/Loongson-7A1000-usermanual-2.00-EN.pdf (in English)

Note:

- CPUINTC is CSR.ECFG/CSR.ESTAT and its interrupt controller described in Section 7.4 of "LoongArch Reference Manual, Vol 1";
- LIOINTC is "Legacy I/OInterrupts" described in Section 11.1 of "Loongson 3A5000 Processor Reference Manual";
- EIOINTC is "Extended I/O Interrupts" described in Section 11.2 of "Loongson 3A5000 Processor Reference Manual";
- HTVECINTC is "HyperTransport Interrupts" described in Section 14.3 of "Loongson 3A5000 Processor Reference Manual";
- PCH-PIC/PCH-MSI is "Interrupt Controller" described in Section 5 of "Loongson 7A1000 Bridge User Manual";
- PCH-LPC is "LPC Interrupts" described in Section 24.3 of "Loongson 7A1000 Bridge User Manual".

CHAPTER

THREE

FEATURE STATUS ON LOONGARCH ARCHITECTURE

Subsystem	Feature	Kconfig	Status	Desc
core	cBPF-JIT	HAVE_CBPF_JIT		arch
core	eBPF-JIT	HAVE_EBPF_JIT		arch
core	generic-idle-thread	GENERIC_SMP_IDLE_THREAD		arch
core	jump-labels	HAVE_ARCH_JUMP_LABEL		arch
core	thread-info-in-task	THREAD_INFO_IN_TASK		arch
core	tracehook	HAVE_ARCH_TRACEHOOK		arch
debug	debug-vm-pgtable	ARCH_HAS_DEBUG_VM_PGTABLE		arch
debug	gcov-profile-all	ARCH_HAS_GCOV_PROFILE_ALL		arch
debug	KASAN	HAVE_ARCH_KASAN		arch
debug	kcov	ARCH_HAS_KCOV		arch
debug	kgdb	HAVE_ARCH_KGDB		arch
debug	kmemleak	HAVE_DEBUG_KMEMLEAK		arch
debug	kprobes	HAVE_KPROBES		arch
debug	kprobes-on-ftrace	HAVE_KPROBES_ON_FTRACE		arch
debug	kretprobes	HAVE_KRETPROBES		arch
debug	optprobes	HAVE_OPTPROBES		arch
debug	stackprotector	HAVE_STACKPROTECTOR		arch
debug	uprobes	ARCH_SUPPORTS_UPROBES		arch
debug	user-ret-profiler	HAVE_USER_RETURN_NOTIFIER		arch
io	dma-contiguous	HAVE_DMA_CONTIGUOUS		arch
locking	cmpxchg-local	HAVE_CMPXCHG_LOCAL		arch
locking	lockdep	LOCKDEP_SUPPORT		arch
locking	queued-rwlocks	ARCH_USE_QUEUED_RWLOCKS		arch
locking	queued-spinlocks	ARCH_USE_QUEUED_SPINLOCKS		arch
perf	kprobes-event	HAVE_REGS_AND_STACK_ACCESS_API		arch
perf	perf-regs	HAVE_PERF_REGS		arch
perf	perf-stackdump	HAVE_PERF_USER_STACK_DUMP		arch
sched	membarrier-sync-core	ARCH_HAS_MEMBARRIER_SYNC_CORE		arch
sched	numa-balancing	ARCH_SUPPORTS_NUMA_BALANCING		arch
seccomp	seccomp-filter	HAVE_ARCH_SECCOMP_FILTER		arch
time	arch-tick-broadcast	ARCH_HAS_TICK_BROADCAST		arch
time	clockevents	!LEGACY_TIMER_TICK		arch
time	context-tracking	HAVE_CONTEXT_TRACKING		arch
time	irq-time-acct	HAVE_IRQ_TIME_ACCOUNTING		arch
time	virt-cpuacct	HAVE_VIRT_CPU_ACCOUNTING		arch

Table 1 – continued from p

Subsystem	Feature	Kconfig	Status	Desc
vm	batch-unmap-tlb-flush	ARCH_WANT_BATCHED_UNMAP_TLB_FLUSH		arch
vm	ELF-ASLR	ARCH_HAS_ELF_RANDOMIZE		arch
vm	huge-vmap	HAVE_ARCH_HUGE_VMAP		arch
vm	ioremap_prot	HAVE_IOREMAP_PROT		arch
vm	PG_uncached	ARCH_USES_PG_UNCACHED		arch
vm	pte_special	ARCH_HAS_PTE_SPECIAL		arch
vm	THP	HAVE_ARCH_TRANSPARENT_HUGEPAGE		arch