ANDES TECHNOLOGY

DEPT. OF ELECTRONICS ENGINEERING & INST. OF ELECTRONICS NCTU

Andes Instruction Set

Agenda

- Andes Instruction Set Architecture
  - Introduction
  - 32-bit Baseline Instructions
  - 16-bit Baseline Instructions
  - Extension Instructions
- Andes System Privileged Architecture
  - Memory Management Unit
  - Interruption Architecture
  - Performance Monitoring Mechanism
  - Local Memory Interface
  - High Speed Memory Port Support
  - System Registers

back

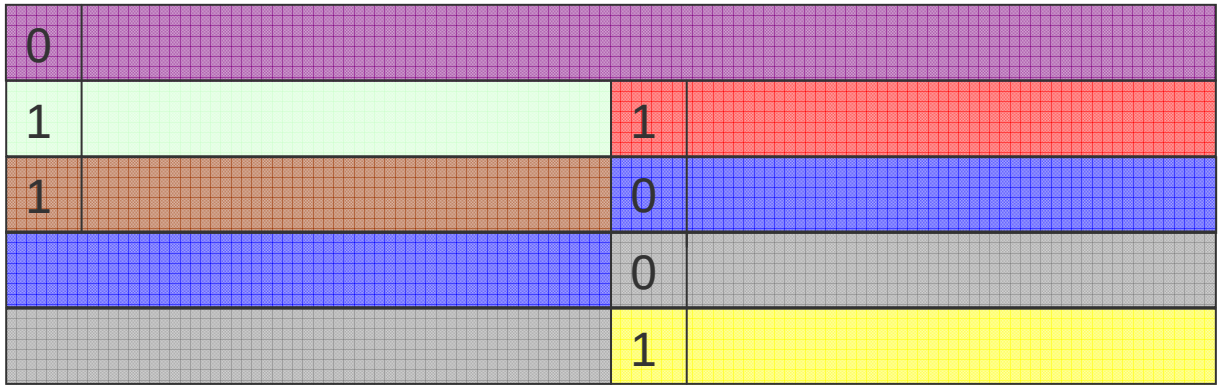
Agenda

- Andes Instruction Set Architecture
  - Introduction
  - 32-bit Baseline Instructions
  - 16-bit Baseline Instructions
  - Extension Instructions
- Andes System Privileged Architecture
  - Memory Management Unit
  - Interruption Architecture
  - Performance Monitoring Mechanism
  - Local Memory Interface
  - High Speed Memory Port Support
  - System Registers

back

ISA Features

- 32/16-bit mixed-length RISC ISA
  - 16-bit is a frequently used subset.
  - 32/16-bit can be mixed in a program.
  - No 32/16-bit mode switching is required.
- Endian
  - Instruction: big-endian format
  - Data: mixed-endian format
- Privilege
  - User / Super-user.
- Instruction Length
  - Bit [31] of a 32-bit instruction and Bit [15] of a 16-bit instruction are used to determine instruction length.



ISA Features

- Semantics
  - ALU: op **rt**, ra, rb_or_imm
    - **rt**= ra op rb_or_imm
  - load **rt**, [address]
    - **rt**= memory[address]
  - store **rt**, [address]
    - memory[address]= **rt**

Data Types

- Data Types
  - Bit (1-bit, b)
  - Byte (8-bit, B)
  - Halfword (16-bit, H)
  - Word (32-bit, W)

General Purpose Registers

32-bit General Purpose Registers

Register	32/16-bit (5)	16-bit (4)	16-bit (3)	Comments
r0-r4	a0-a4	h0-h4	o0-o4	
r5	a5	h5	o5	Implied register for beqs38 and bnes38
r6-r7	s0-s1	h6-h7	o6-o7	Saved by callee
r8-r11	s2-s5	h8-h11		Saved by callee
r12-r14	s6-s8			Saved by callee
r15	ta			Implied register for slt(s|i)45, b[eq|ne]zs8
r16-r19	t0-t3	h12-h15		Saved by caller
r20-r25	t4-t9			Saved by caller
r26-r27	p0-p1			Reserved for Privileged-mode use.
r28	s9/fp			Frame Pointer
r29	gp			Global Pointer
r30	lp			Link Pointer
r31	sp			Stack Pointer

General Purpose Registers - Reduced

- 16/32-bit General Purpose Registers

Register	32/16-bit (5)	16-bit (4)	16-bit (3)	Comments
r0-r4	a0-a4	h0-h4	o0-o4	
r5	a5	h5	o5	Implied register for beqs38 and bnes38
r6-r7	s0-s1	h6-h7	o6-o7	Saved by callee
r8-r10	s2-s4	h8-h10		Saved by callee
r15	ta			Implied register for slt(s|i)45, b[eq|ne]zs8
r28	fp			Frame Pointer / Saved by callee
r29	gp			Global Pointer
r30	lp			Link Pointer
r31	sp			Stack Pointer

User Special and System Registers

- User Special Register

Register	32-bit Instr.	16-bit Instr.	Comments
D0	d0.{hi, lo}	N/A	For multiplication and division related instructions (64-bit)
D1	d1.{hi, lo}	N/A	For multiplication and division related instructions (64-bit)

- System Registers
  - Configuration System Registers
  - Interruption System Registers
  - MMU System Registers
  - Other System Registers

Agenda

- Andes Instruction Set Architecture
  - Introduction
  - 32-bit Baseline Instructions
  - 16-bit Baseline Instructions
  - Extension Instructions
- Andes System Privileged Architecture
  - Memory Management Unit
  - Interruption Architecture
  - Performance Monitoring Mechanism
  - Local Memory Interface
  - High Speed Memory Port Support
  - System Registers

back

32-bit Baseline Instruction

- The minimum set of Andes ISA.
  - A complete set of instructions which can be used to construct a fully-functional Andes system or program.
- Instruction Catalog
  - Data Processing
  - Load and Store
  - Flow Control (Jump and Branch)
  - Privileged Resource Access
  - Miscellaneous

32-bit Baseline – data processing

- **ALU Instructions with Immediate**
  - OP rt_5, ra_5, imm_15
    - ADDI, SUBRI, ANDI, ORI, XORI
    - SLTI, SLTSI: set rt_5 if ra_5 < imm_15 (unsigned or signed comparison)
  - OP rt_5, imm_20
    - SETHI: set low or high 20 bits of rt_5 (the rest bits are set to 0).
- **ALU Instructions without Immediate**
  - OP rt_5, ra_5, rb_5
    - ADD, SUB, AND, NOR, OR, XOR, SLT, SLTS
    - SVA, SVS: set if overflow on add/sub
  - OP rt_5, ra_5
    - SEB, SEH, ZEB, WSBH: sign- or zero-extension byte/half, word-swap bytes.
- **Shift and rotate instructions**
  - OP rt_5, ra_5, imm_5
    - SLLI, SRLI, SRAI, ROTRI
  - OP rt_5, ra_5, rb_5
    - SLL, SRL, SRA, ROTR

link

32-bit Baseline – data processing

- **Multiplication-related Instructions**
  - OP rt_5, ra_5, rb_5 ➔ 32-bit results of ra_5 x rb_5 to rt_5
    - MUL
  - OP d_1, ra_5, rb_5 ➔ 32-bit or 64-bit results to “d” registers
    - MULTS64, MULT64, MADDS64, MADD64, MSUBS64, MSUB64
    - MULT32, MADD32, MSUB32 : lower 32-bit result is written to d.lo
  - OP rt_5, d_1.{hi, lo}
    - MFUSR, MTUSR: move-from or move-to a USR register

Example:

```
mult64  d0, r0, r1
mfusr   r2, d0.hi
mfusr   r3, d0.lo
```

32-bit Baseline – load / store

- Load/Store Single
  - Immediate value is in the unit of access size.
    - OP rt_5, [ra_5+imm_15]
      - LWI, LHI, LHSI, LBI, LBSI, SWI, SHI, SBI
    - OP rt_5, [ra_5], imm_15 : with post update
      - LWI.bi, LHI.bi, LHSI.bi, LBI.bi, LBSI.bi, SWI.bi, SHI.bi, SBI.bi
  - Index register is left-shifted by 0,1,2,3 bits by si
    - OP rt_5, [ra_5+rb_5<< si]
      - LW, LH, LHS, LB, LBS, SW, SH, SB
    - OP rt_5, [ra_5], rb_5<<si : with post update
      - LW.bi, LH.bi, LHS.bi, LB.bi, LBS.bi, SW.bi, SH.bi, SB.bi

32-bit Baseline – load / store

Examples

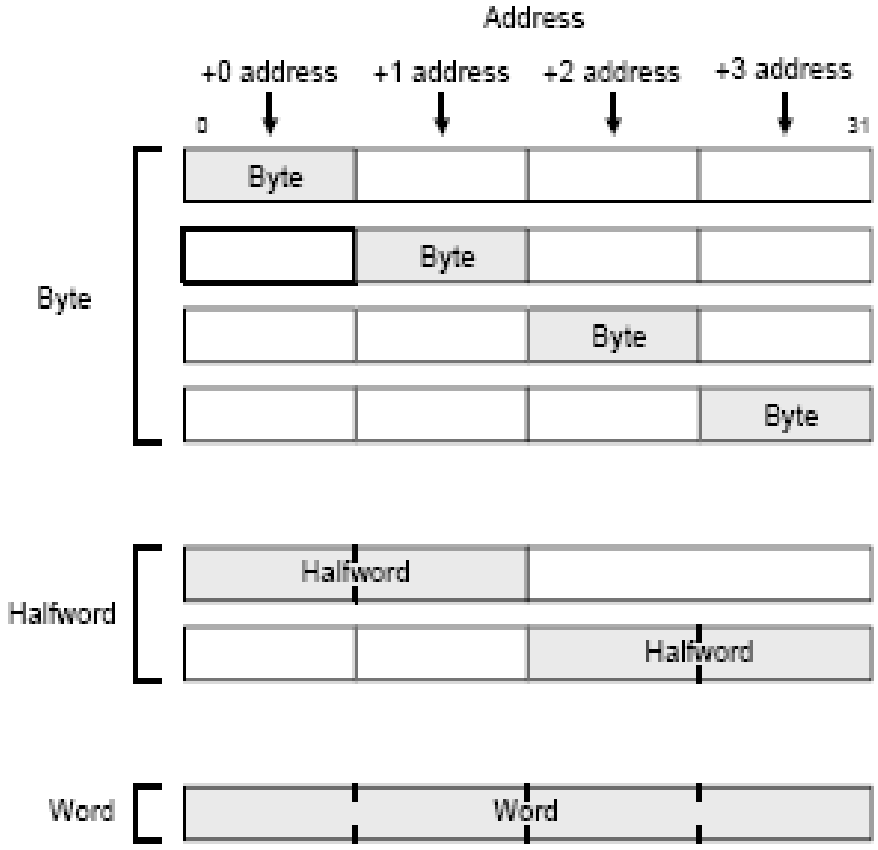
```
lbi.bi    r1, [r0], 1
lbi.bi    r2, [r0], 1
lbi.bi    r3, [r0], 1
lbi.bi    r4, [r0], 1

lhi.bi    r1, [r0], 2
lhi.bi    r2, [r0], 2
```

- Half word addr alignment check!

```
lwi.bi    r1, [r0], 4
```

- word addr alignment check!

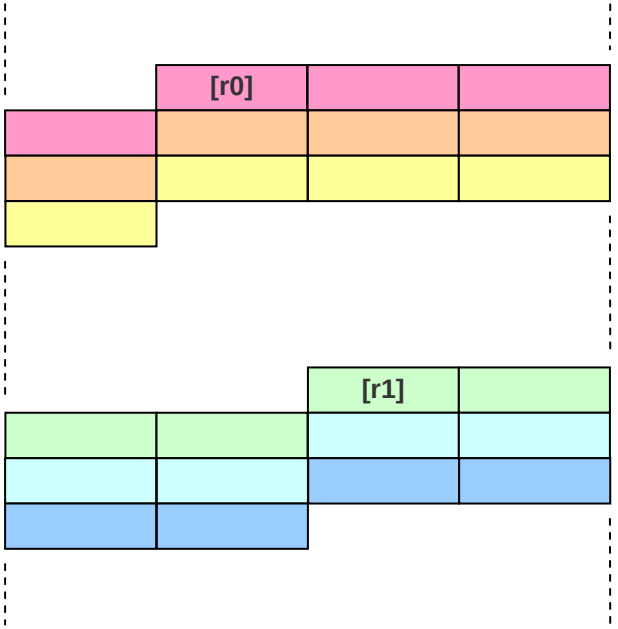


32-bit Baseline – load / store

- Load/Store Multiple Words:
  - **LMW.{b,a}{i,d}{m?} Rb, [Ra], Re, Enable4**
  - **SMW.{b,a}{i,d}{m?} Rb, [Ra], Re, Enable4**
    - **Register list**: Rb to Re, and Enable4, which is a mask selecting r28-r31 respectively.
    - **Base address**: Ra
    - {b,a}: do load or store **before/after** advancing the address from Ra.
    - {i,d} generates **increasing/decreasing** addresses from Ra.
    - If {m} is specified, the base address register will be **updated** to the last address generated.

32-bit Baseline – load / store

- Load/Store Multiple Words:
  - Aligned/unaligned address access
  - smw.adm r0, [sp], r5, 0b1100
    - Push r0-r5, fp, gp into the stack (pointed by sp).
    - sp is decremented by 32
  - Use for unaligned load/store
    - lmw.bim r8, [r0] , r15, 0
    - smw.bim r8, [r1], r15, 0
    - Can greatly decrease code size



32-bit Baseline – flow control

- Jump Instruction
  - OP imm_24
    - J: unconditional direct branch
    - JAL: direct function call
  - OP rb_5
    - JR: unconditional indirect branch
    - RET: return
    - JRAL: indirect function call
- Branch Instruction
  - OP rt_5, ra_5, imm_14
    - BEQ, BNE
  - OP rt_5, imm_16
    - BEQZ, BNEZ, BGEZ, BLTZ, BGTZ, BLEZ

link

32-bit Baseline – privilege

- Privileged Resource Access
  - System and Special Register
    - MFSR/MTSR, MFUSR/MTUSR
    - SETEND, SETGIE
    - JRAL.xTON, JR.xTOFF, RET.xTOFF
  - Cache Management
    - CCTL
  - TLB Management
    - TLBOP

    tlbop $r0,TargetRead

link

32-bit Baseline – miscellaneous

- Miscellaneous
  - Data Prefetch : DPREF, DPREFI.
  - Interruption : BREAK, SYSCALL, TRAP, TEQZ, TNEZ, IRET.
  - Serialization : ISB, DSB.
  - Synchronization : ISYNC, MSYNC.
  - Standby : STANDBY.
  - No Operation : NOP.

link

Agenda

- Andes Instruction Set Architecture
  - Introduction
  - 32-bit Baseline Instructions
  - 16-bit Baseline Instructions
  - Extension Instructions
- Andes System Privileged Architecture
  - Memory Management Unit
  - Interruption Architecture
  - Performance Monitoring Mechanism
  - Local Memory Interface
  - High Speed Memory Port Support
  - System Registers

back

16-bit Baseline

- 1-on-1 mapping with 32-bit instruction
  - 16-bit instructions are a frequently used subset of 32-bit instructions.
  - All 16-bit instructions can be mapped to the corresponding 32-bit instructions.

```
addi45  $r0, 1          <=>     addi $r0, $r0, 1
swi37   $r0, [$fp+12]   <=>     swi  $r0, [$fp + 12]
```

- Register Indexing
  - 5-bit index
  - 4/3-bit index
  - Implied register

link

16-bit Baseline

- Data Processing
  - MOVI55, MOV55, ADDI45, ADDI333, SUBI45…
- Load/Store:
  - LWI450, LWI333, LWI333.bi, LHI333, LBI333, SWI450…
- Flow Control
  - BEQS38, BNES38, BEQZ38, BNEZ38, J8, JR5, RET5, JRAL5…
- Misc
  - Break16, NOP16

Agenda

- Andes Instruction Set Architecture
  - Introduction
  - 32-bit Baseline Instructions
  - 16-bit Baseline Instructions
  - Extension Instructions
- Andes System Privileged Architecture
  - Memory Management Unit
  - Interruption Architecture
  - Performance Monitoring Mechanism
  - Local Memory Interface
  - High Speed Memory Port Support
  - System Registers

back

32-bit Performance Extension

- Performance extension
  - Arithmetic :ABS, AVE, MAX, MIN
  - Bit operation :BSET, BCLR, BTGL, BTST
  - Saturation :CLIPS, CLIP
  - Counting :CLZ, CLO

link

32-bit String Extension

- String extension
  - String Search : FFB, FFBI
  - String Compare : FFMISM, FLMISM

link

Agenda

- Andes Instruction Set Architecture
  - Introduction
  - 32-bit Baseline Instructions
  - 16-bit Baseline Instructions
  - Extension Instructions
- Andes System Privileged Architecture
  - Memory Management Unit
  - Interruption Architecture
  - Performance Monitoring Mechanism
  - Local Memory Interface
  - High Speed Memory Port Support
  - System Registers

back

Memory Management Unit (MMU)

- To supports virtual memory and paging by translating virtual addresses into physical addresses.
- Provide address space protection and cache control capabilities.
- Main component: Translation Look-aside Buffer (TLB)
- Managed by software (TLBOPs) or hardware (Hardware page table walker).
- Provides large page and multiple page size support.
- Provides TLB locking support.

Memory Management Unit (MMU)

- Virtual Address (VA): virtual/logical address space. 32-bit wide, thus 4 GB space.
- Physical Address (PA): The real address used to access system bus. Typically, managed by superuser mode processes.
- When VA->PA translation process is disabled, VA is equivalent to PA.

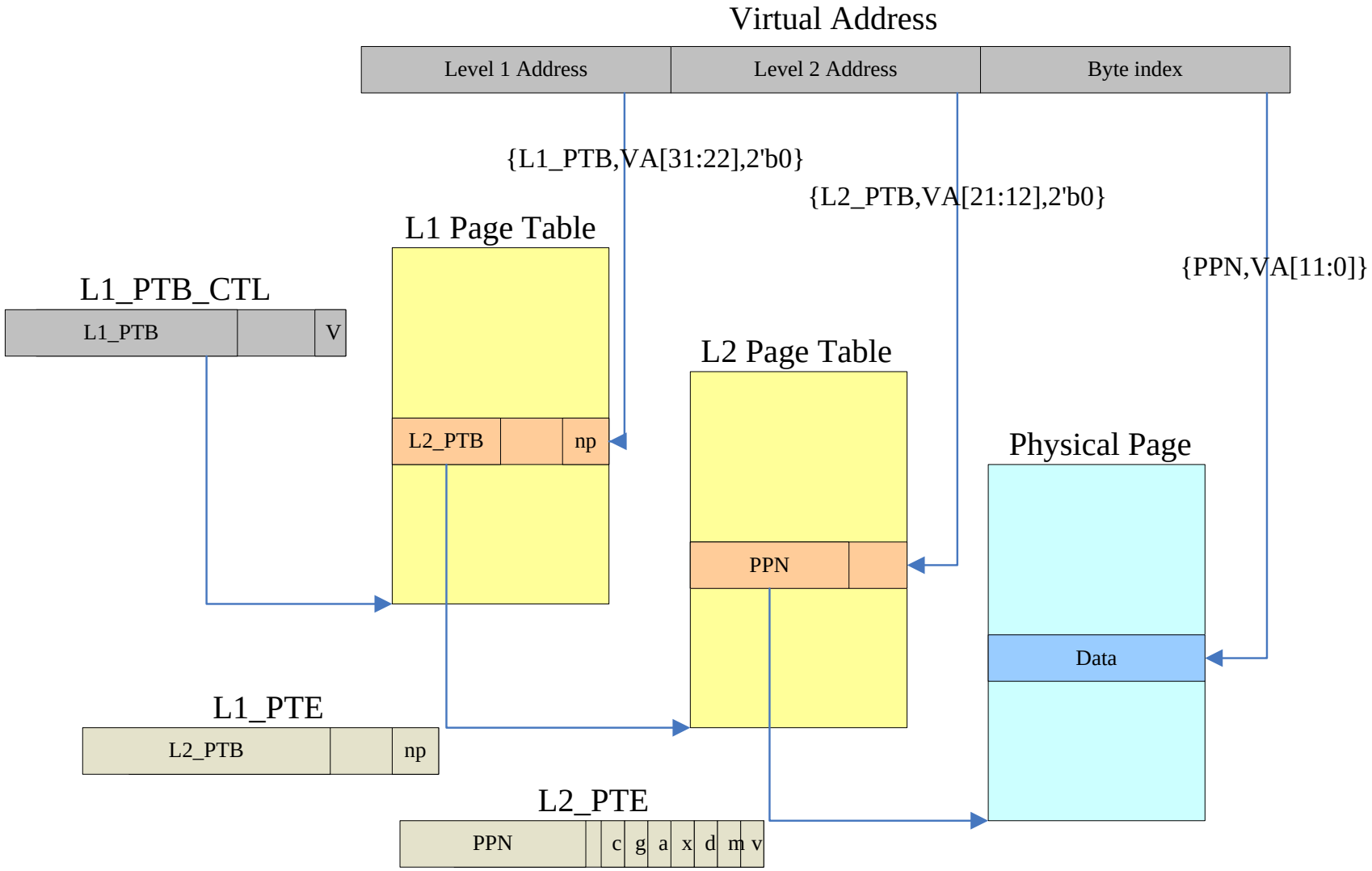
Address space attributes

- For access control, and sharing of program or data.
- Defines various properties
  - Cachebility requirement/hint
  - Access/execute permissions
- Translated: defined in Page Table Entry.
- Non-translated: defined in system register, selected by PA index.

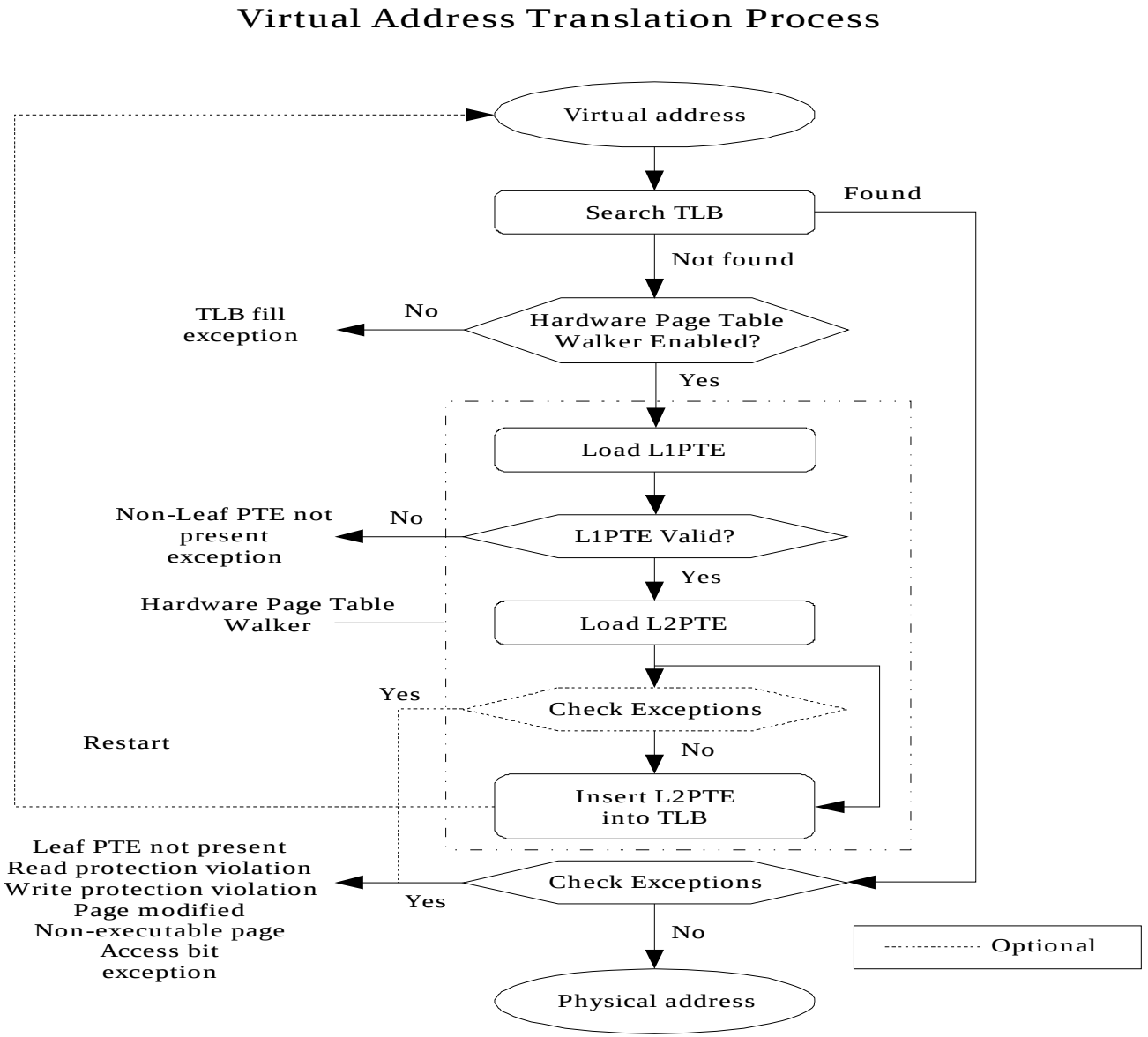
Hardware Page Table Walker

- Responsible for reading TLB entry located in system memory under Main TLB miss condition.
- Less flexible than software, handles only one or may be two page table format. But it speeds up the TLB refill time
- 2 Level address for looking up PTE in external memory
- Use physical address to access memory

Hardware Page Table Walker – 4KB



Virtual Address translation process



Agenda

- Andes Instruction Set Architecture
  - Introduction
  - 32-bit Baseline Instructions
  - 16-bit Baseline Instructions
  - Extension Instructions
- Andes System Privileged Architecture
  - Memory Management Unit
  - Interruption Architecture
  - Performance Monitoring Mechanism
  - Local Memory Interface
  - High Speed Memory Port Support
  - System Registers

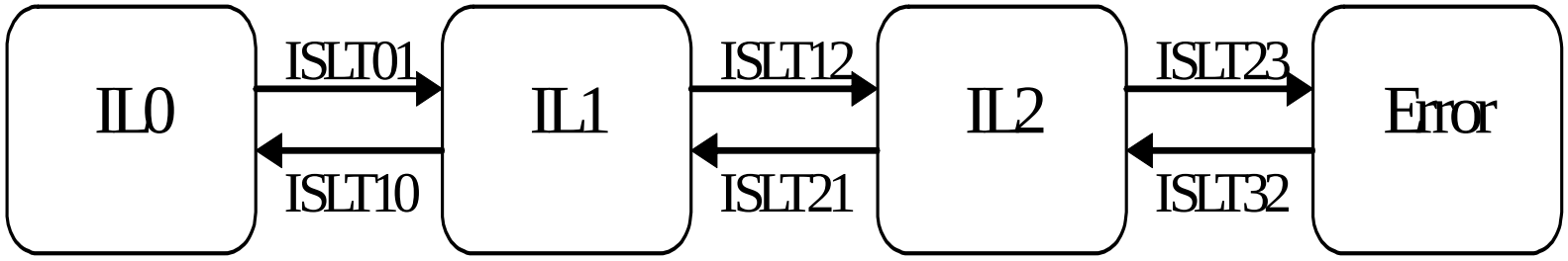
back

Interruption Introduction

- An Interruption is a control flow change of normal instruction execution generated by an Interrupt or an Exception
- An *interrupt* is a control flow change event generated by an asynchronous internal or external source
- An *exception* is a control flow change event generated as a by-product of instruction execution
- Two Interruption stack level : 2-level or 3-level

Interruption Stack Level

- 4 “Interruption Stack Level” (ISL), ISL0 means no interruption.
- Hardware update interruption states on “Interruption Stack Level Transition” (ISLT).
- ISLT01, ISLT12 updates Interruption Register Stack.
- ISLT23 updates limited states for server error condition.
- Stack level can only be up to level 2 in 2-level configured interruption



Interruption Stack

- Updates performed on interruption level transition 0/1 and 1/2:
  - New value ⇒ PC ⇔ IPC ⇔ P_IPC
  - New value ⇒ PSW ⇔ IPSW ⇔ P_IPSW
  - VA ⇒ EVA ⇔ P_EVA
  - Interruption ⇒ ITYPE ⇔ P_ITYPE
  - P0 ⇔ P_P0
  - P1 ⇔ P_P1
- Updates performed on interruption level transition 2/3:
  - New value ⇒ PC ⇔ O_IPC

Interruption Behavior

- Transition to Superuser mode
- Disable interrupt
- Disable I/D address translation
- Use default Endian
- ISLT0→1/ISLT1→2 achieves these behavior by updating corresponding PSW states while ISLT2→3 achieves these behavior by assumption.

Types of Interruption

- **Reset/NMI**: Cold Reset, Warn Reset, Non-Maskable Interrupt (NMI)
- **Interrupt**: External, Performance counter, Software interrupt
- **Debug exception**: Instruction Address break, Data address & value break, Other debug exceptions
- **MMU related exception**:
  - TLB fill exception (I/D)
  - Non-Leaf PTE not present (I/D), Leaf PTE not present (I/D)
  - Read protection violation (D), Write protection violation (D)
  - Page modified (D), Non-executable page (I), Access bit (I/D)
- **Syscall exception**
- **General exception**:
  - Trap, Arithmetic, Reserved instruction/value exception
  - Privileged instruction, mis-Alignment (I/D), Bus error (I/D)
  - Nonexistence local memory address (I/D), MPZIU control
  - Coprocessor N not-usable exception, Coprocessor N-related exception
- **Machine error exception:**
  - Cache/TLB errors

Priority table

Vectored Entry Point

- IVB register defines the base address and the size of offset.
- 2 interrupt controller modes (configurable)
  - Internal VIC mode
  - External VIC mode
- Interruption sub-type in ITYPE register is defined based on individual entry point

Entry Point for IVIC/EVIC

- Internal VIC mode
  - 16 entry points (9 exception + 7 interrupt)
  - 4 bits index

Offset	Entry point
0	Reset/NMI
1	TLB fill
2	PTE not present
3	TLB misc
4	Reserved
5	Machine Error
6	Debug related
7	General exception
8	Syscall
9 -14	HW 0- 5
15	SW 0

- External VIC
  - 73 entry points (9 exception + 64 interrupt)
  - 7 bits index

Offset	Entry point
0	Reset/NMI
1	TLB fill
2	PTE not present
3	TLB misc
4	Reserved
5	Machine Error
6	Debug related
7	General exception
8	Syscall
9-72	VEP 0-63

Agenda

- Andes Instruction Set Architecture
  - Introduction
  - 32-bit Baseline Instructions
  - 16-bit Baseline Instructions
  - Extension Instructions
- Andes System Privileged Architecture
  - Memory Management Unit
  - Interruption Architecture
  - Performance Monitoring Mechanism
  - Local Memory Interface
  - High Speed Memory Port Support
  - System Registers

back

Performance Monitor

- Mechanism to obtain program running statistics and performance data.
- Performance monitoring features:
  - One cycle/instruction counter
  - Two event counters
  - Individual controls:
    - Counter enable
    - Interrupt enable
    - Overflow status/clear bit
    - Counting event selection among maximum of 64 events
    - Superuser and user mode event counting filters.
  - Readable/Writable in privileged mode
  - Continuing counting after overflow

Counting Events

- Instruction counts
  - Branch, RET, JAL,PREF, ISB/DSB...
- Program flow events
  - taken branches, mispredict, interrupt, exception...
- MMU events
  - MTLB/uITLB/uDTLB access/misss, HPTWK cycles...
- Cache events
  - Load/Store I/D cache access/miss...
- Other events
  - ILM/DLM access, DMA cycles...

Agenda

- Andes Instruction Set Architecture
  - Introduction
  - 32-bit Baseline Instructions
  - 16-bit Baseline Instructions
  - Extension Instructions
- Andes System Privileged Architecture
  - Memory Management Unit
  - Interruption Architecture
  - Performance Monitoring Mechanism
  - Local Memory Interface
  - High Speed Memory Port Support
  - System Registers

back

Local Memory and Local Memory DMA

- Local memory is on-chip memories that are as fast as L1 caches.
- The local memory DMA is provided to transfer blocks of data between Andes core LM and external, off-core, memory in parallel with the Andes core execution pipeline.
- To transfer large amounts of data to or from LM, it is more efficient to use the DMA engine instead of the Andes core load/store instructions.

Local Memory (LM)

- The base address is aligned on 1MB boundary.
- The size is implementation depended.
  - 4KB, 8KB, 16KB ~ 1024KB
- The DLM support ***Double-buffer*** mode
  - DLM is divided into two banks.
  - One for processor and the other for DMA.
  - Processor and DMA can access the same address simultaneously.
- Both Internal LM and External LM are supported

Local Memory DMA

- Two channels
- Programmed using physical addressing
- Software takes care of address translation and permission check
- Only accessed by privileged mode
- For both instruction and data local memory
- External address can be incremented with stride
- Aligned internal address
- Aligned external address for basic configuration
- Un-aligned External Address (UNEA) feature which allows un-aligned external address for element transfer
- 2-D Element Transfer (2DET) feature which provides an easy way to transfer two-dimensional blocks from external memory

Agenda

- Andes Instruction Set Architecture
  - Introduction
  - 32-bit Baseline Instructions
  - 16-bit Baseline Instructions
  - Extension Instructions
- Andes System Privileged Architecture
  - Memory Management Unit
  - Interruption Architecture
  - Performance Monitoring Mechanism
  - Local Memory Interface
  - High Speed Memory Port Support
  - System Registers

back

High Speed Memory Port (HSMP)

- In addition to AMBA2.0 AHB-style system interface, to reduce memory access delays and thus increase system performance, N12 provide a *high speed memory port interface* which has *higher bus protocol efficiency* and can run at a *higher frequency* to connect to a memory controller.
- The high speed memory port is AMBA3.0 (AXI) protocol compliant, but with reduced I/O requirements.

Agenda

- Andes Instruction Set Architecture
  - Introduction
  - 32-bit Baseline Instructions
  - 16-bit Baseline Instructions
  - Extension Instructions
- Andes System Privileged Architecture
  - Memory Management Unit
  - Interruption Architecture
  - Performance Monitoring Mechanism
  - Local Memory Interface
  - High Speed Memory Port Support
  - System Registers

back

System Registers

- System registers control the behaviors of the mentioned mechanisms.
- System Register Access
  - MFSR $rt, $sr
  - MTSR $rt, $sr
- Side effect Barrier Instruction
  - DSB :Data Serialization Barrier
  - ISB :Instruction Serialization Barrier

Configuration System Registers

Register	Type	Description
CPU_VER	RO	CPU configuration, version and CPU ID
ICM_CFG	RO	Instruction Cache/Memory size and type
DCM_CFG	RO	Data Cache/Memory size and types
MMU_CFG	RO	MMU size and configurations
MSC_CFG	RO	EDM,LMDMA,PFM,HSMP,TRACE,DIV and MAC configurations
CORE_ID	RO	Core Identification Number

MMU System Register

Register	Type	Description
MMU_CTL	R/W	page size and attributes for non-translation.
L1_PPTB	R/W	HPTWK base address (physical address)
TLB_VPN	R/W	Virtual Page Number for TLB Access
TLB_DATA	R/W	Page table Entry for TLB
TLB_MISC	R/W	Miscellaneous Data for TLB Access
CACHE_CTL	R/W	Cache configuration

Interruption System Register

Register	Type	Description
PSW	R/W	Processor Status Control/Status
IPSW	R/W	Level 1 PSW stack
P_IPSW	R/W	Level 2 PSW stack
IVB	R/W	Base address of the interruption vector table
EVA	RO	Virtual address which causes the exception
P_EVA	R/W	Level 2 EVA stack
ITYPE	RO	Type of Interruption
P_ITYPE	R/W	Level 2 ITPE stack
MERR	R/W	Machine Error log
IPC	R/W	Level 1 PC stack
P_IPC	R/W	Level 2 PC stack
OIPC	R/W	Overflow PC used for ILT23
P_P0 - P_P1	R/W	Level 2 P0 - 1 stack
INT_MASK	R/W	Interruption Masking
INT_PEND	R/W	Interruption Pending

Other System Register

Register	Type	Description
ILMB	R/W	ILM Base Physical address
DLMB	R/W	DLM Base Physical address

Register	Type	Description
PFM0-PFM2	R/W	Performance Event Counter 0-2
PFM_CTL	R/W	Performance Monitor Configuration

Register	Type	Description
HSMP_SADDR	R/W	High Speed Memory Port Starting Address
HSMP_EADDR	R/W	High Speed Memory Port Starting Address

ANDES TECHNOLOGY

DEPT. OF ELECTRONICS ENGINEERING & INST. OF ELECTRONICS NCTU

Thank You!!!

The following are spare slides…

Instruction Details 1

- 32-bit Baseline Instructions
  - Data Processing
  - Load and Store
    - LWUP, SWUP, LMW, SMW, LLW, SCW.
  - Flow Control
  - Privileged Resource Access
    - CCTL, TLBOP.
  - Miscellaneous
    - ISB, DSB, ISYNC, MSYNC, STANDBY.
- 16-bit Baseline Mapping

Instruction Details 2

- Audio Instructions
- Floating-point Instructions

Data Processing 1

Set Constant Instructions

Mnemonic	Instruction	Operation
MOVI rt5, imm20s	Move Immediate	rt5 = SE(imm20s)
SETHI rt5, imm20u	Set High Immediate	rt5 = {imm20u, 12'b0}

Data Processing 2

ALU Instructions

Mnemonic	Instruction	Operation
ADDI rt5, ra5, imm15s	Add Immediate	rt5 = ra5 + SE(imm15s)
SUBRI rt5, ra5, imm15s	Subtract Reverse Immediate	rt5 = SE(imm15s) – ra5
ANDI rt5, ra5, imm15u	And Immediate	rt5 = ra5 && ZE(imm15u)
ORI rt5, ra5, imm15u	Or Immediate	rt5 = ra5 || ZE(imm15u)
XORI rt5, ra5, imm15u	Exclusive Or Immediate	rt5 = ra5 ^ ZE(imm15u)
ADD rt5, ra5, rb5	Add	rt5 = ra5 + rb5
SUB rt5, ra5, rb5	Subtract	rt5 = ra5 - rb5
AND rt5, ra5, rb5	And	rt5 = ra5 && rb5
NOR rt5, ra5, rb5	Nor	rt5 = ~(ra5 || rb5)
OR rt5, ra5, rb5	Or	rt5 = ra5 || rb5
XOR rt5, ra5, rb5	Exclusive Or	rt5 = ra5 ^ rb5

Data Processing 3

Shift/Rotate Instructions

Mnemonic	Instruction	Operation
SLLI rt5, ra5, imm5u	Shift Left Logical Immediate	rt5 = ra5 << imm5u
SRLI rt5, ra5, imm5u	Shift Right Logical Immediate	rt5 = ra5 (logic)>> imm5u
SRAI rt5, ra5, imm5u	Shift Right Arithmetic Immediate	rt5 = ra5 (arith)>> imm5u
ROTRI rt5, ra5, imm5u	Rotate Right Immediate	rt5 = ra5 >>| imm5u
SLL rt5, ra5, rb5	Shift Left Logical	rt5 = ra5 << rb5(4,0)
SRL rt5, ra5, rb5	Shift Right Logical	rt5 = ra5 (logic)>> rb5(4,0)
SRA rt5, ra5, rb5	Shift Right Arithmetic	rt5 = ra5 (arith)>> rb5(4,0)
ROTR rt5, ra5, rb5	Rotate Right	rt5 = ra5 >>| rb5(4,0)

Data Processing 4

Compare Instructions

Mnemonic	Instruction	Operation
SLTI rt5, ra5, imm15s	Set on Less Than Immediate	rt5 = (ra5 (unsigned)< SE(imm15s)) ? 1 : 0
SLTSI rt5, ra5, imm15s	Set on Less Than Signed Immediate	rt5 = (ra5 (signed)< SE(imm15s)) ? 1 : 0
SLT rt5, ra5, rb5	Set on Less Than	rt5 = (ra5 (unsigned)< rb5) ? 1 : 0
SLTS rt5, ra5, rb5	Set on Less Than Signed	rt5 = (ra5 (signed)< rb5) ? 1 : 0

Data Processing 5

Special Instructions

Mnemonic	Instruction	Operation
SVA rt5, ra5, rb5	Set on Overflow Add	rt5 = ((ra5 + rb5) overflow)? 1 : 0
SVS rt5, ra5, rb5	Set on Overflow Subtract	rt5 = ((ra5 - rb5) overflow))? 1 : 0
SEB rt5, ra5	Sign Extend Byte	rt5 = SE(ra5[7:0])
SEH rt5, ra5	Sign Extend Halfword	rt5 = SE(ra5[15:0])
ZEB rt5, ra5	Zero Extend Byte	rt5 = ZE(ra5[7:0]) (alias of ANDI rt5, ra5, 0xFF)
ZEH rt5, ra5	Zero Extend Halfword	rt5 = ZE(ra5[15:0])
WSBH rt5, ra5	Word Swap Byte within Halfword	rt5 = {ra5[23:16], ra5[31:24], ra5[7:0], ra5[15:8]}

Data Processing 6

Conditional Move Instructions

Mnemonic	Instruction	Operation
CMOVZ rt5, ra5, rb5	Conditional Move on Zero	rt5 = ra5 if (rb5 == 0)
CMOVN rt5, ra5, rb5	Conditional Move on Not Zero	rt5 = ra5 if (rb5 != 0)

Data Processing 7

MFUSR and MTUSR (Group 0)

Mnemonic	Instruction	Operation
MFUSR rt5, USR	Move From User Special Register	rt5 = USReg[USR]
MTUSR rt5, USR	Move To User Special Register	USReg[USR] = rt5

Group	USR value	User Special Register
0	0	D0.LO
0	1	D0.HI
0	2	D1.LO
0	3	D1.HI
0	30-4	Reserved
0	31	PC (The PC value of this instruction)

Load and Store 1

Mnemonic	Instruction	Operation
LWI rt5, [ra5 + (imm15s << 2)]	Load Word Immediate	address = ra5 + SE(imm15s << 2) rt5 = Word-memory(address)
LHI rt5, [ra5 + (imm15s << 1)]	Load Halfword Immediate	address = ra5 + SE(imm15s << 1) rt5 = ZE(Halfword-memory(address))
LHSI rt5, [ra5 + (imm15s << 1)]	Load Halfword Signed Immediate	address = ra5 + SE(imm15s << 1) rt5 = SE(Halfword-memory(address))
LBI rt5, [ra5 + imm15s]	Load Byte Immediate	address = ra5 + SE(imm15s) rt5 = ZE(Byte-memory(address))
LBSI rt5, [ra5 + imm15s]	Load Byte Signed Immediate	address = ra5 + SE(imm15s) rt5 = SE(Byte-memory(address))
SWI rt5, [ra5 + (imm15s << 2)]	Store Word Immediate	address = ra5 + SE(imm15s << 2) Word-memory(address) = rt5
SHI rt5, [ra5 + (imm15s << 1)]	Store Halfword Immediate	address = ra5 + SE(imm15s << 1) Halfword-memory(address) = rt5[15:0]
SBI rt5, [ra5 + imm15s]	Store Byte Immediate	address = ra5 + SE(imm15s) Byte-memory(address) = rt5[7:0]

Load and Store 2

Mnemonic	Instruction	Operation
LWI.bi rt5, [ra5], (imm15s << 2)	Load Word Immediate with Post Increment	rt5 = Word-memory(ra5) ra5 = ra5 + SE(imm15s << 2)
LHI.bi rt5, [ra5], (imm15s << 1)	Load Halfword Immediate with Post Increment	rt5 = ZE(Halfword-memory(ra5)) ra5 = ra5 + SE(imm15s << 1)
LHSI.bi rt5, [ra5], (imm15s << 1)	Load Halfword Signed Immediate with Post Increment	rt5 = SE(Halfword-memory(ra5)) ra5 = ra5 + SE(imm15s << 1)
LBI.bi rt5, [ra5], imm15s	Load Byte Immediate with Post Increment	rt5 = ZE(Byte-memory(ra5)) ra5 = ra5 + SE(imm15s)
LBSI.bi rt5, [ra5], imm15s	Load Byte Signed Immediate with Post Increment	rt5 = SE(Byte-memory(ra5)) ra5 = ra5 + SE(imm15s)
SWI.bi rt5, [ra5], (imm15s << 2)	Store Word Immediate with Post Increment	Word-memory(ra5) = rt5 ra5 = ra5 + SE(imm15s << 2)
SHI.bi rt5, [ra5], (imm15s << 1)	Store Halfword Immediate with Post Increment	Halfword-memory(ra5) = rt5[15:0] ra5 = ra5 + SE(imm15s << 1)
SBI.bi rt5, [ra5], imm15s	Store Byte Immediate with Post Increment	Byte-memory(ra5) = rt5[7:0] ra5 = ra5 + SE(imm15s)

Load and Store 3

Mnemonic	Instruction	Operation
LW rt5, [ra5 + (rb5 << sv)]	Load Word	address = ra5 + (rb5 << sv) rt5 = Word-memory(address)
LH rt5, [ra5 + (rb5 << sv)]	Load Halfword	address = ra5 + (rb5 << sv) rt5 = ZE(Halfword-memory(address))
LHS rt5, [ra5 + (rb5 << sv)]	Load Halfword Signed	address = ra5 + (rb5 << sv) rt5 = SE(Halfword-memory(address))
LB rt5, [ra5 + (rb5 << sv)]	Load Byte	address = ra5 + (rb5 << sv) rt5 = ZE(Byte-memory(address))
LBS rt5, [ra5 + (rb5 << sv)]	Load Byte Signed	address = ra5 + (rb5 << sv) rt5 = SE(Byte-memory(address))
SW rt5, [ra5 + (rb5 << sv)]	Store Word	address = ra5 + (rb5 << sv) Word-memory(address) = rt5
SH rt5, [ra5 + (rb5 << sv)]	Store Halfword	address = ra5 + (rb5 << sv) Halfword-memory(address) = rt5[15:0]
SB rt5, [ra5 + (rb5 << sv)]	Store Byte	address = ra5 + (rb5 << sv) Byte-memory(address) = rt5[7:0]

Load and Store 4

Mnemonic	Instruction	Operation
LW.bi rt5, [ra5], rb5<<sv	Load Word with Post Increment	rt5 = Word-memory(ra5) ra5 = ra5 + (rb5 << sv)
LH.bi rt5, [ra5], rb5<<sv	Load Halfword with Post Increment	rt5 = ZE(Halfword-memory(ra5)) ra5 = ra5 + (rb5 << sv)
LHS.bi rt5, [ra5], rb5<<sv	Load Halfword Signed with Post Increment	rt5 = SE(Halfword-memory(ra5)) ra5 = ra5 + (rb5 << sv)
LB.bi rt5, [ra5], rb5<<sv	Load Byte with Post Increment	rt5 = ZE(Byte-memory(ra5)) ra5 = ra5 + (rb5 << sv)
LBS.bi rt5, [ra5], rb5<<sv	Load Byte Signed with Post Increment	rt5 = SE(Byte-memory(ra5)) ra5 = ra5 + (rb5 << sv)
SW.bi rt5, [ra5], rb5<<sv	Store Word with Post Increment	Word-memory(ra5) = rt5 ra5 = ra5 + (rb5 << sv)
SH.bi rt5, [ra5], rb5<<sv	Store Halfword with Post Increment	Halfword-memory(ra5) = rt5[15:0] ra5 = ra5 + (rb5 << sv)
SB.bi rt5, [ra5], rb5<<sv	Store Byte with Post Increment	Byte-memory(ra5) = rt5[7:0] ra5 = ra5 + (rb5 << sv)

LWUP and SWUP

- Usage
  - To access a 32-bit word between memory and GPR with the user mode privilege address translation.
  - To use the user-mode privilege address translation regardless of the current processor operation mode (PSW.POM) and the current data address translation state (PSW.DT).

back

LMW and SMW

- Usage
  - To load/store multiple 32-bit words from/to sequential memory locations.

31	30 25	24 20	19 15	14 10	9 6	5	4	3	2	1 0
0	LSMW 011101	Rb	Ra	Re	Enable4	LMW 0	b:0 a:1	i:0 d:1	m	00
0	LSMW 011101	Rb	Ra	Re	Enable4	SMW 1	b:0 a:1	i:0 d:1	m	00

LLW and SCW

- Usage
  - To implement a primitive to perform atomic read-modify-write operations.
    - LLW sets a lock flag while loading data.
    - SCW checks the lock flags before storing data and returns the status bit and write to GPR.
  - Example

    LLW Rx

    … Modifying Rx

    SCW Rx

    BEQZ Rx

back

Flow Control 1

Mnemonic	Instruction	Operation
J imm24s	Jump	PC = PC + SE(imm24s << 1)
JAL imm24s	Jump and Link	LP = next sequential PC (PC + 4); PC = PC + SE(imm24s << 1)
JR rb5	Jump Register	PC = rb5
RET rb5	Return from Register	PC = rb5
JRAL rb5 JRAL rt5, rb5	Jump Register and Link	LP = next sequential PC (PC + 4); PC = rb5; rt5 = next sequential PC (PC + 4); PC = rb5;

Flow Control 2

Mnemonic	Instruction	Operation
BEQ rt5, ra5, imm14s	Branch on Equal (2 Register)	PC = (rt5 == ra5) ? (PC+SE(imm14s << 1)) : (PC+4)
BNE rt5, ra5, imm14s	Branch on Not Equal (2 Register)	PC = (rt5 =! ra5)? (PC + SE(imm14s << 1)) : (PC+4)
BEQZ rt5, imm16s	Branch on Equal Zero	PC = (rt5==0)? (PC + SE(imm16s << 1)) : (PC + 4)
BNEZ rt5, imm16s	Branch on Not Equal Zero	PC = (rt5 =! 0)? (PC + SE(imm16s << 1)) : (PC + 4)
BGEZ rt5, imm16s	Branch on Greater than or Equal to Zero	PC = (rt5 (signed)>= 0) ? (PC+SE(imm16s << 1)) : (PC+4)
BLTZ rt5, imm16s	Branch on Less than Zero	PC = (rt5 (signed)< 0)? (PC + SE(imm16s << 1)) : (PC + 4)
BGTZ rt5, imm16s	Branch on Greater than Zero	PC = (rt5 (signed)> 0)? (PC + SE(imm16s << 1)) : (PC + 4)
BLEZ rt5, imm16s	Branch on Less than or Equal to Zero	PC = (rt5 (signed)<= 0)? (PC + SE(imm16s << 1)) : (PC+4)
BGEZAL rt5, imm16s	Branch on Greater than or Equal to Zero and Link	LP = next sequential PC (PC + 4); PC = (rt5 (signed)>= 0)? (PC + SE(imm16s << 1)) : (PC+4);
BLTZAL rt5, imm16s	Branch on Less than Zero and Link	LP = next sequential PC (PC +4); PC = (rt5 (signed)< 0)? (PC + SE(imm16s << 1)) : (PC+4);

Privileged Resource Access 1

MFSR and MTSR

Mnemonic	Instruction	Operation
MFSR rt5, SRIDX	Move from System Register	rt5 = SR[SRIDX]
MTSR rt5, SRIDX	Move to System Register	SR[SRIDX] = rt5

- Example

```
MFSR    r5, IVB
update  r5
MTSR    r5, IVB
```

Privileged Resource Access 2

Mnemonic	Instruction	Operation

SETEND.B
SETEND.L
	Atomic set or clear of PSW.BE bit	PSW.BE = 1; // SETEND.B
		PSW.BE = 0; // SETEND.L

SETGIE.E
SETGIE.D
	Atomic set or clear of PSW.GIE bit	PSW.GIE = 1; // SETGIE.E
		PSW.GIE = 0; // SETGIE.D

JR.ITOFF rb5	Jump Register and Instruction Translation OFF	PC = rb5; PSW.IT = 0;
JR.TOFF rb5	Jump Register and Translation OFF	PC = rb5; PSW.IT = 0, PSW.DT = 0;
JRAL.ITON rb5 JRAL.ITON rt5, rb5	Jump Register and Link and Instruction Translation ON	LP = PC+4; PC = rb5; rt5 = PC+4; PC = rb5; PSW.IT = 1;
JRAL.TON rb5 JRAL.TON rt5, rb5	Jump Register and Link and Translation ON	LP = PC+4; PC = rb5; rt5 = PC+4; PC = rb5; PSW.IT = 1, PSW.DT = 1;
RET.ITOFF Rb5	Return and Instruction Translation OFF	PC = Rb5 ; PSW.IT = 0;
RET.TOFF Rb5	Return and Translation OFF	PC = Rb5; PSW.IT = 0; PSW.DT = 0;

MFUSR/MTUSR (Group 1/2)

- Accessing group 1/2 USR registers in user-level requires the access permission being enabled.

Group	USR value	User Special Register
2	0	PFMC0
2	1	PFMC1
2	2	PFMC2
2	3	Reserved
2	4	PFM_CTL
2	5-31	Reserved

Group	USR value	User Special Register
1	0	DMA_CFG
1	1	DMA_GCSW
1	2	DMA_CHNSEL
1	3	DMA_ACTION (Write only register, Read As Zero)
1	4	DMA_SETUP
1	5	DMA_ISADDR
1	6	DMA_ESADDR
1	7	DMA_TCNT
1	8	DMA_STATUS
1	9	DMA_2DSET
1	10-24	Reserved
1	25	DMA_2DSCTL
1	26-31	Reserved

Cache Management (CCTL)

- 20 operations for CCTL (Cache Control) instruction.
  - “CCTL Ra, Subtype”, “CCTL Rt, Ra, Subtype”, and “CCTL Rb, Ra, Subtype”
  - Index or VA invalidate L1 I/D cache (4)
  - Index or VA write back L1 D cache (2)
  - Index or VA write back & invalidate L1 D cache (2)
  - VA fill and lock L1 I/D cache (2)
  - VA unlock I/D cache (2)
  - Index read tag/data L1 I/D cache (4)
  - Index write tag/data L1 I/D cache (4)

CCTL (Data Cache)

Mnemonics	Operation	Ra Type	Rt?/Rb?	User Privilege	Compliance
L1D_IX_INVAL	Invalidate L1D cache	Index	-/-	-	Required
L1D_VA_INVAL	Invalidate L1D cache	VA	-/-	Yes	Required
L1D_IX_WB	Write Back L1D cache	Index	-/-	-	Required
L1D_VA_WB	Write Back L1D cache	VA	-/-	Yes	Required
L1D_IX_WBINVAL	Write Back & Invalidate L1Dcache	Index	-/-	-	Optional
L1D_VA_WBINVAL	Write Back & Invalidate L1D cache	VA	-/-	Yes	Optional
L1D_VA_FILLCK	Fill and Lock L1D cache	VA	-/-	-	Optional
L1D_VA_ULCK	unlock L1D cache	VA	-/-	-	Optional
L1D_IX_RTAG	Read tag L1D cache	Index	Yes/-	-	Optional
L1D_IX_RWD	Read word data L1D cache	Index/w	Yes/-	-	Optional
L1D_IX_WTAG	Write tag L1D cache	Index	-/Yes	-	Optional
L1D_IX_WWD	Write word data L1D cache	Index/w	-/Yes	-	Optional
L1D_INVALALL	Invalidate All L1D cache	N/A	-/-	-	Optional

CCTL (Instruction Cache)

Mnemonics	Operation	Ra Type	Rt?/Rb?	User Privilege	Compliance
L1I_VA_FILLCK	Fill and Lock L1I cache	VA	-/-	-	Optional
L1I_VA_ULCK	unlock L1I cache	VA	-/-	-	Optional
L1I_IX_INVAL	Invalidate L1I cache	Index	-/-	-	Required
L1I_VA_INVAL	Invalidate L1I cache	VA	-/-	Yes	Required
L1I_IX_RTAG	Read tag L1I cache	Index	Yes/-	-	Optional
L1I_IX_RWD	Read word data L1I cache	Index/w	Yes/-		Optional
L1I_IX_WTAG	Write tag L1I cache	Index	-/Yes	-	Optional
L1I_IX_WWD	Write word data L1I cache	Index/w	-/Yes	-	Optional

TLBOP

SubType	Mnemonics	Operation	Rt?/Ra?
0	TargetRead (TRD)	Read targeted TLB entry	-/Ra
1	TargetWrite (TWR)	Write targeted TLB entry	-/Ra
2	RWrite (RWR)	Write a hardware-determined TLB entry	-/Ra
3	RWriteLock (RWLK)	Write a hardware-determined TLB entry and lock	-/Ra
4	Unlock (UNLK)	Unlock a TLB entry	-/Ra
5	Probe (PB)	Probe TLB entry information	Rt/Ra
6	Invalidate (INV)	Invalidate TLB entries except locked entries	-/Ra
7	FlushAll (FLUA)	Flush all TLB entries except locked entries	-/-

Miscellaneous

Mnemonic	Instruction
NOP	No Operation (alias of SRLI R0, R0, 0)
DPREFI [ra5 + imm15s]	Data Prefetch Immediate
DPREF [ra5 + (rb5 << si)]	Data Prefetch
BREAK	Breakpoint
SYSCALL	System Call
TRAP	Trap Always
TEQZ	Trap on Equal Zero
TNEZ	Trap on Not Equal Zero
IRET	Interruption Return

ISB

- Usage
  - Instruction Serialization Barrier
  - To block the processor core from fetching any subsequent instructions until all of the previously modified architecture/hardware states can be observed by the subsequent dependent instruction fetching operations.
  - Example

```
MTSR   Ra, PSW
ISB
ADD    r5, r4, r3
```

DSB

- Usage
  - Data Serialization Barrier
  - To block the processor core from executing any subsequent instructions until all of the previously modified architecture/hardware states can be observed by the subsequent dependent data operations.
  - Example

```
MTSR   Ra, PSW
DSB
MFSR   Rb, PSW
```

ISYNC

- Usage
  - Instruction Synchronizer
  - To guarantee an instruction fetch event after an instruction serialization barrier instruction can properly observe previous instruction data updates.
  - Example

```
store   Ra, [Rb, 0]
isync   Rb
isb
jr  Rb
```

MSYNC

- Usage
  - Memory Synchronizer
  - To order load and store events on the synchronizing memories among multiple processor cores and/or Direct Memory Access agents.
- Subtype

SubType	Name
0	All
1	Store
2 - 7	Reserved

Data Cache Prefetch

Mnemonic	Instruction
DPREFI [ra5 + imm15s]	Data Cache Line Prefetch Immediate
DPREF [ra5 + (rb5 << si)]	Data Cache Line Prefetch

STANDBY 1

- Usage
  - To put a processor core entering an idle mode and waiting for external events to wakeup.
  - Can be utilized as power-saving mechanism.
    - Gated-clock, frequency scaling, etc...
- Behavior
  - The main pipeline will be halted and no new instructions will be fetched and executed.
  - All external memory and I/O accesses must be completed before entering standby.

STANDBY 2

- Subtype
  - User-level always gets subtype-0 effects.

SubType	Mnemonic	Wakeup Condition
0	no_wake_grant	The STANDBY instruction immediately monitors and accepts a wakeup event (e.g external interrupt) to leave the standby mode without waiting for a wakeup_consent notification from an external agent.
1	wake_grant	The STANDBY instruction waits for a wakeup_consent notification from an external agent (e.g. power management unit) before monitoring and accepting a wakeup_event (e.g. external interrupt) to leave the standby mode.
2	wait_done	The STANDBY instruction waits for a wakeup_consent notification from an external agent (e.g. power management unit). When the wakeup_consent notification arrives, the core leaves the standby mode immediately.

16-bit Baseline Mapping 1

Mnemonic	Instruction	32-Bit Operation
MOVI55 rt5, imm5s	Move Immediate	MOVI rt5, SE(imm5s)
MOV55 rt5, ra5	Move	ADDI/ORI rt5, ra5, 0
ADDI45 rt4, imm5u	Add Word Immediate	ADDI rt5, rt5, ZE(imm5u)
ADDI333 rt3, ra3, imm3u	Add Word Immediate	ADDI rt5, ra5, ZE(imm3u)
SUBI45 rt4, imm5u	Subtract Word Immediate	ADDI rt5, rt5, NEG(imm5u)
SUBI333 rt3, ra3, imm3u	Subtract Word Immediate	ADDI rt5, ra5, NEG(imm3u)
ADD45 rt4, rb5	Add Word	ADD rt5, rt5, rb5
ADD333 rt3, ra3, rb3	Add Word	ADD rt5, ra5, rb5
SUB45 rt4, rb5	Subtract Word	SUB rt5, rt5, rb5
SUB333 rt3, ra3, rb3	Subtract Word	SUB rt5, ra5, rb5
SRAI45 rt4, imm5u	Shift Right Arithmetic Immediate	SRAI rt5, rt5, imm5u
SRLI45 rt4, imm5u	Shift Right Logical Immediate	SRLI rt5, rt5, imm5u
SLLI333 rt3, ra3, imm3u	Shift Left Logical Immediate	SLLI rt5, ra5, ZE(imm3u)

16-bit Baseline Mapping 2

Mnemonic	Instruction	32-Bit Operation
BFMI333 rt3, ra3, imm3u	Bit Field Mask Immediate	
ZEB33 rt3, ra3 (BFMI333 rt3, ra3, 0)	Zero Extend Byte	ZEB rt5, ra5
ZEH33 rt3, ra3 (BFMI333 rt3, ra3, 1)	Zero Extend Halfword	ZEH rt5, ra5
SEB33 rt3, ra3 (BFMI333 rt3, ra3, 2)	Sign Extend Byte	SEB rt5, ra5
SEH33 rt3, ra3 (BFMI333 rt3, ra3, 3)	Sign Extend Halfword	SEH rt5, ra5
XLSB33 rt3, ra3 (BFMI333 rt3, ra3, 4)	Extract LSB	ANDI rt5, ra5, 1
X11B33 rt3, ra3 (BFMI333 rt3, ra3, 5)	Extract the least 11 Bits	ANDI rt5, ra5, 0x7ff

16-bit Baseline Mapping 3

Mnemonic	Instruction	32-Bit Operation
LWI450 rt4, [ra5]	Load Word Immediate	LWI rt5, [ra5+0]
LWI333 rt3, [ra_3+ imm3u]	Load Word Immediate	LWI rt5, [ra5+ZE(imm3u)]
LWI333.bi rt3, [ra_3], imm3u	Load Word Immediate with Post-increment	LWI.bi rt5, [ra5], ZE(imm3u)
LHI333 rt3, [ra_3+ imm3u]	Load Halfword Immediate	LHI rt5, [ra5+ZE(imm3u)]
LBI333 rt3, [ra_3+ imm3u]	Load Byte Immediate	LBI rt5, [ra5+ZE(imm3u)]
SWI450 rt4, [ra5]	Store Word Immediate	SWI rt5, [ra5+0]
SWI333 rt3, [ra_3+ imm3u]	Store Word Immediate	SWI rt5, [ra5+ZE(imm3u)]
SWI333.bi rt3, [ra_3], imm3u	Store Word Immediate with Post-increment	SWI.bi rt5, [ra5], ZE(imm3u)
SHI333 rt3, [ra_3+ imm3u]	Store Halfword Immediate	SHI rt5, [ra5+ZE(imm3u)]
SBI333 rt3, [ra_3+ imm3u]	Store Byte Immediate	SBI rt5, [ra5+ZE(imm3u)]
LWI37 rt3, [fp+imm7u]	Load Word with Implied FP	LWI rt5, [fp+ZE(imm7u)]
SWI37 rt3, [fp+imm7u]	Store Word with Implied FP	SWI rt5, [fp+ZE(imm7u)]

16-bit Baseline Mapping 4

Mnemonic	Instruction	32-Bit Operation
BEQS38 rt3, imm8s	Branch on Equal (implied r5)	BEQ rt5, r5, SE(imm8s) (next sequential PC = PC + 2)
BNES38 rt3, imm8s	Branch on Not Equal (implied r5)	BNE rt5, r5, SE(imm8s) (next sequential PC = PC + 2)
BEQZ38 rt3, imm8s	Branch on Equal Zero	BEQZ rt5, SE(imm8s) (next sequential PC = PC + 2)
BNEZ38 rt3, imm8s	Branch on Not Equal Zero	BNEZ rt5, SE(imm8s) (next sequential PC = PC + 2)
J8 imm8s	Jump Immediate	J SE(imm8s)
JR5 rb5	Jump Register	JR rb5
RET5 rb5	Return from Register	RET rb5
JRAL5 rb5	Jump Register and Link	JRAL rb5

16-bit Baseline Mapping 5

Mnemonic	Instruction	32-Bit Operation
SLTI45 ra4, imm5u	Set on Less Than Unsigned Immediate	SLTI r15, ra5, ZE(imm5u)
SLTSI45 ra4, imm5u	Set on Less Than Signed Immediate	SLTSI r15, ra5, ZE(imm5u)
SLT45 ra4, rb5	Set on Less Than Unsigned	SLT r15, ra5, rb5
SLTS45 ra4, rb5	Set on Less Than Signed	SLTS r15, ra5, rb5
BEQZS8 imm8s	Branch on Equal Zero (implied r15)	BEQZ r15, SE(imm8s) (next sequential PC = PC + 2)
BNEZS8 imm8s	Branch on Not Equal Zero (implied r15)	BNEZ r15, SE(imm8s) (next sequential PC = PC + 2)

Mnemonic	Instruction	32-Bit Operation
BREAK16	Breakpoint	BREAK
NOP16 (alias of SRLI45 R0,0)	No Operation	NOP

Cachebility & Bufferability

- device space
- device space, write bufferable/coalescable
- non-cacheable memory
- cacheable, write-back, write-allocate, shared memory
- cacheable, write-through, no-write-allocate, shared memory
- cacheable, write-back, write allocate, non-shared memory
- cacheable, write-through, no-write-allocate, non-shared memory

Basic Non-Translated Cachebility

- Cacheable/write-back
- Cacheable/write-through
- Non-cacheable/coalesable
- Non-cacheable/non-coalesable

4KB Page Hardware Table Walker Load Addresses

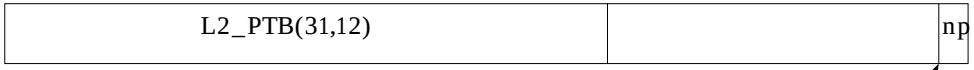
L1_PTB_CTL (31,0) - L1 Page Table Base Control Register

L1_PTB is physical address.

V: HTW enable bit

L1PTE load address = L1_PTB(31,12).VA(31,22).00

L1_PTE(31,0) format



np: L1_PTE not present bit

L2PTE load address = L2_PTB(31,12).VA(21,12).00

L2_PTE(31,0) format

v: L2_PTE valid bit

PA(31,0) = PPN(31,12).VA(11,0)

Priority of Interruptions

Interruption Name (highest to lowest)	Vectored Entry Point
cold reset/warm reset	Reset/NMI
external debug request (debug interrupt)	Debug
Hardware single-stepping	Debug
NMI	Reset/NMI
Debug Data value watchpoint - imprecise	Debug
Interrupt	(Based on interrupt priority)
Debug Instruction breakpoint	Debug
Instruction alignment check	General exception
ITLB multiple hit	Machine error
ITLB fill	TLB fill
ITLB VLPT miss	TLB VLPT miss
IPTE not present (Non-leaf)	PTE not present
IPTE not present (Leaf)	PTE not present
Reserved IPTE attribute	TLB misc
Instruction MPZIU control	General exception
ITLB non-execute page	TLB misc
IAccess bit	TLB misc
Instruction Machine Error	Machine error
Instruction nonexistent local memory address	General exception

Priority of Interruptions (Conti...)

Instruction bus error (precise)	General exception
Reserved instruction	General exception
Privileged instruction	General exception
Reserved value	General exception
Unimplemented page size error	Machine error
Trap/syscall	General exception
Break	Debug
Coprocessor	General exception
Arithmetic	General exception
Debug Data address watchpoint	Debug
Data alignment check	General exception
DTLB multiple hit	Machine error
DTLB fill	TLB fill
DTLB VLPT miss	TLB VLPT miss
DPTE not present (Non-leaf)	PTE not present
DPTE not present (Leaf)	PTE not present
Reserved DPTE attribute	TLB misc
Data MPZIU control	General exception
DTLB permission (R/W)	TLB misc
Dpage modified	TLB misc
DAccess bit	TLB misc
Data Machine Error	Machine error
Data nonexistent local memory address	General exception
Debug Data value watchpoint - precise	Debug
Data bus error - precise	General exception
Data bus error - imprecise (e.g. HW prefetch)	General exception
Instruction bus error - imprecise (e.g. HW prefetch)	General exception

Basic Counting Events

SEL	C1	C2
0	total cycles	
1	completed instructions	
2	conditional branch	conditional branch mispredict
3	taken conditional branches	taken conditional branch mispredict
4	prefetch insts	prefetch with $ hit
5	RET inst	RET mispredict
6	JR (non-RET) inst	immeidate J insts (exclude JAL)
7	JAL/JRAL insts	multiply insts
8	NOP instruction	16-bit instructions
9	SCW instruction	failed SCW
10	ISB/DSB instruction	ld-after-st conflict replays
11	CCTL instruction	-
12	taken interrupt	exceptions taken
13	loads completed (LMW count as 1)	stores completed (SMW count as 1)
14	uITLB access	uITLB miss
15	uDTLB access	uDTLB miss

Basic Counting Events

SEL	C1	C2
16	MTLB access	MTLB miss
17	I$ access	I$ miss
18	data dependency stall cycles (when it is the sole cause of stall)	no inst from IQ stall cycles
19	D$ miss stall cycles	D$ writebacks
20	D$ access (LMW/SMW count as many, not including CCTL)	
21	D$ miss	
22	LD D$ access (LMW count as many, not including CCTL)	LD D$ miss (LMW count as many, not including CCTL)
23	ST D$ access (SMW count as many, not including CCTL)	ST D$ miss (LMW count as many, not including CCTL)
24	ILM access	DLM access
25	LSU BIU cycles (D$ fill, noncacheable, write-back, write-through)	LSU BIU request (D$ fill, noncachable, write-back, write-through)
26	HPTWK BIU cycles	HPTWK BIU request
27	DMA BIU cycles	DMA BIU request
28	I$ fill BIU cycles	I$ fill BIU request
29	legal unaligned D$ access	external event (implementation-dep)

back

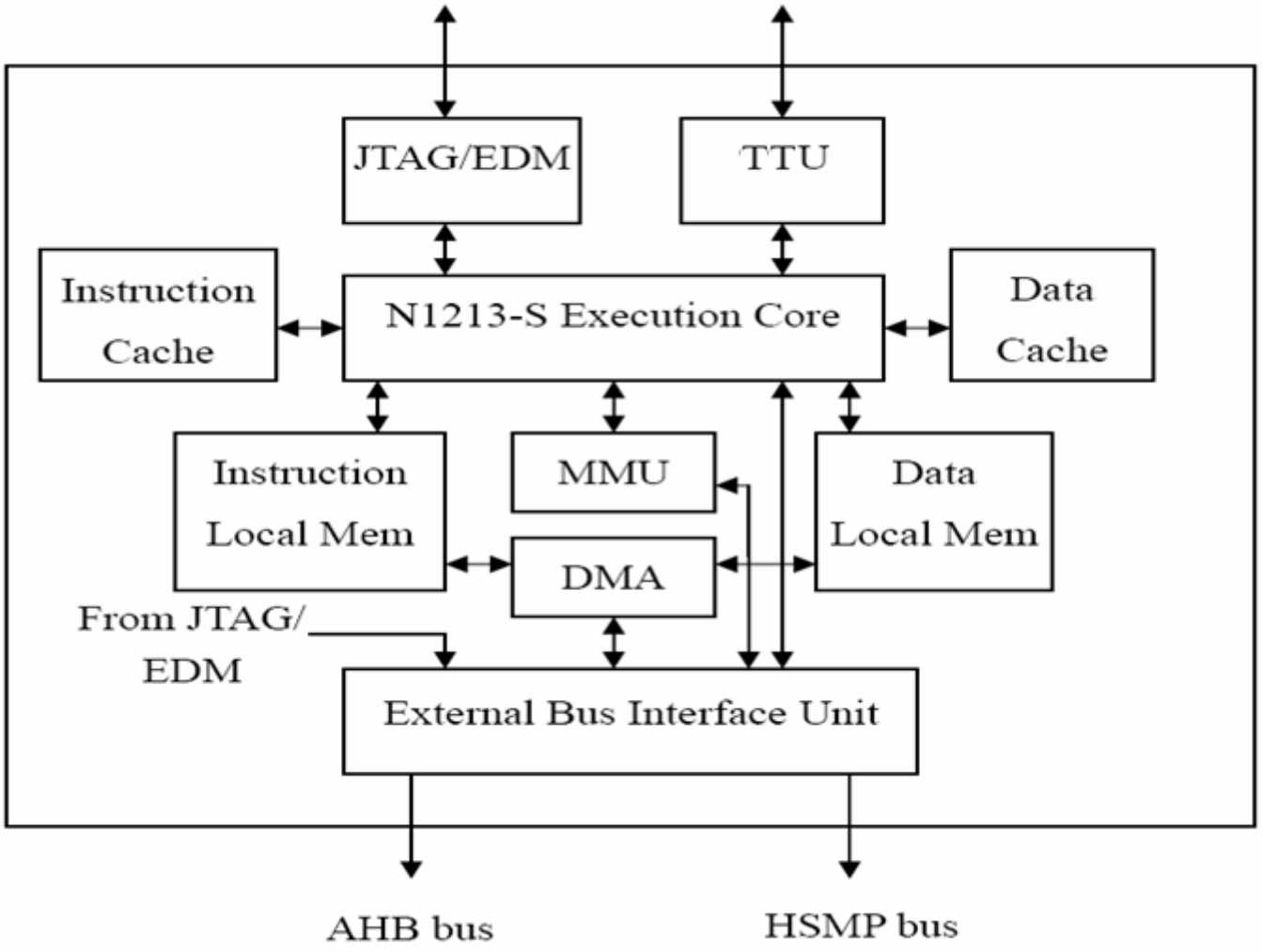
Performance Extension Instruction

Mnemonic	Instruction	Operation
ABS rt5, ra5	Absolute with Register	rt5 = | ra5 |
AVE rt5, ra5, rb5	Average two signed integers with rounding	rt5 = (ra5 + rb5 + 1) (arith) >> 1 (page 206)
MAX rt5, ra5, rb5	Return the Larger	rt5 = signed-max (ra5, rb5)
MIN rt5, ra5, rb5	Return the Smaller	rt5 = signed-min (ra5, rb5)
BSET rt5, ra5, imm_5	Bit Set	rt5 = ra5 || (1 << imm_5)
BCLR rt5, ra5, imm_5	Bit Clear	rt5 = (ra5 && ~(1 << imm_5))
BTGL rt5, ra5, imm_5	Bit Toggle	rt5 = ra5 ^ (1 << imm_5)
BTST rt5, ra5, imm_5	Bit Test	rt5 = (ra5 && (1 << imm_5))? 1 : 0
CLIPS rt5, ra5, imm_5	Clip Value Signed	rt5 = (ra5 > 2^{imm5}-1)? 2^{imm5}-1 : ((ra5 < -2^{imm5})? -2^{imm5} : ra5)
CLIP rt5, ra5, imm_5	Clip Value	rt5 = (ra5 > 2^{imm5}-1)? 2^{imm5}-1 : ((ra5 < 0)? 0 : ra5)
CLZ rt5, ra5	Counting Leading Zeros in Word	rt5 = COUNT_ZERO_FROM_MSB(ra5)
CLO rt5, ra5	Counting Leading Ones in Word	rt5 = COUNT_ONE_FROM_MSB(ra5)

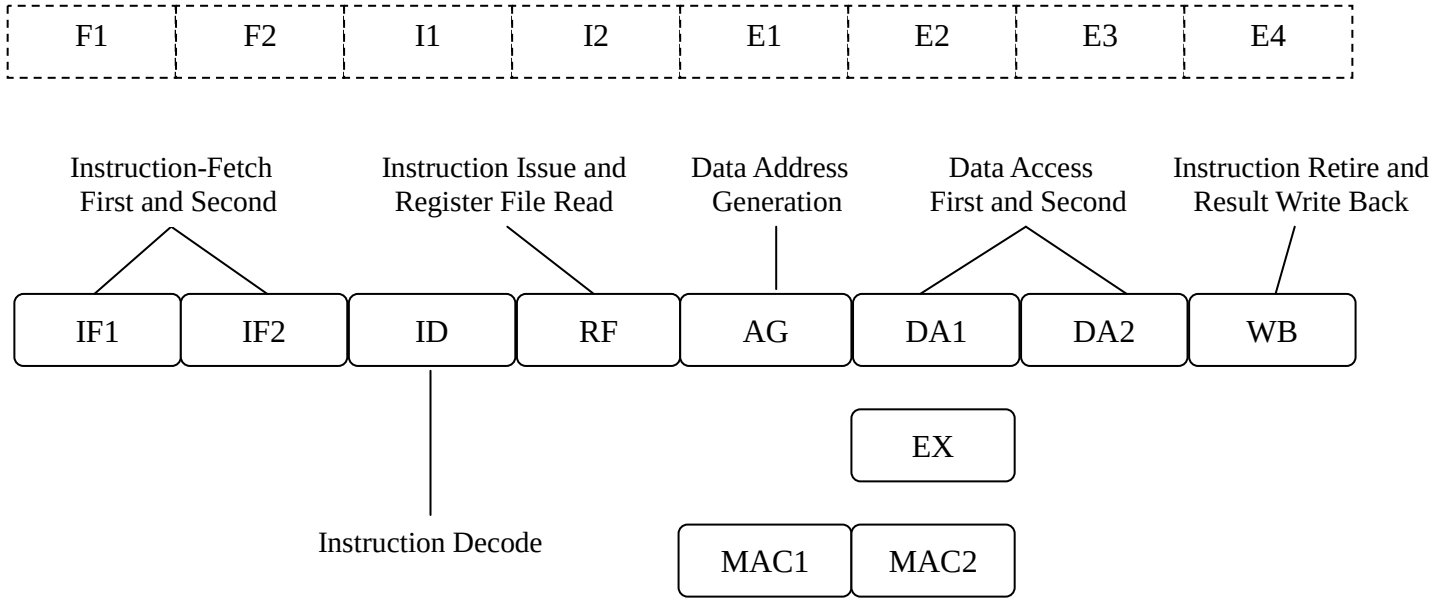
String Extension Instruction

Mnemonic	Instruction	Operation
FFB rt5, ra5, rb5	Find First Byte	rt5 = Find_First_Byte(ra5, rb5) Please see page 218 for details.
FFBI rt5, ra5, imm8	Find First Byte Immediate	rt5 = Find_First_Byte(ra5, imm8) Please see page 220 for details.
FFMISM rt5, ra5, rb5	Find First Mis-Match	rt5 = Find_First_Mismatch(ra5, rb5) Please see page 222 for details.
FLMISM rt5, ra5, rb5	Find Last Mis-Match	rt5 = Find_Last_Mismatch(ra5, rb5) Please see page 224 for details.

N1213 Block diagram



8-stage pipeline



4KB Page Hardware Table Walker Load Addresses

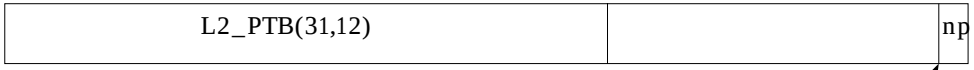
L1_PTB_CTL (31,0) - L1 Page Table Base Control Register

L1_PTB is physical address.

V: HTW enable bit

L1PTE load address = L1_PTB(31,12).VA(31,22).00

L1_PTE(31,0) format



np: L1_PTE not present bit

L2PTE load address = L2_PTB(31,12).VA(21,12).00

L2_PTE(31,0) format

v: L2_PTE valid bit

PA(31,0) = PPN(31,12).VA(11,0)

ANDES TECHNOLOGY

DEPT. OF ELECTRONICS ENGINEERING & INST. OF ELECTRONICS NCTU

Thank You!!!