Dominik Salvet, 2016

Limen Alpha
Programmer's manual

Table of Contents

1 A brief description of Limen Alpha

Limen Alpha is processor architecture with following characteristics:

- 16-bit width,
- Von Neumann type,
- RISC type,
- dual-core variant,
- can address up to 64 kDB.

2 Memory model

It's used physically addressed flat memory model - every address used in instruction is equal address that will appear on the address bus. Memory address space also contains mapped I/O device and is shared for both processor cores.

3 Register file

Architecture defines eight general purpose registers. However, register R0 represents always zero value, write to this register is ignored. There is also IP register that holds address of actual executed instruction and some control registers.

4 List of instructions

- SLU - set if less unsigned
- SL - set if less
- ADD - addition
- LL - load linked
- LD - load
- SC - store conditional
- ST - store
- OR - logical disjunction
- ORN - logical disjunction not
- AND - logical conjunction
- ANDN - logical conjunction not
- XOR - exclusive logical disjunction
- SLL - shift left logical
- SRL - shift right logical
- SRA - shift right arithmetic
- SUB - subtraction
- RTC - register to control
- CTR - control to register
- LI - load immediate
- LIS - load immediate scaled
- LIL - load immediate low
- LIH - load immediate high
- JNE - jump if not equal
- JE - jump if equal
- JL - jump if less
- JLE - jump if less or equal
- JG - jump if greater
- JGE - jump if greater or equal
- JWL - jump with link

5 Instruction formats

Bits with indexes 13 to 15 of every instruction word determine its instruction format. Instructions that are in the same instruction format are often executed very similarly.

5.1 Introducing

The following concept `[s]imm<x>`, where `<x>` is natural number, represents numbers:

- `imm<x>` - from 0 to 2^{x} - 1,
- `simm<x>` - from $-2^{(x - 1)}$ to $2^{(x - 1)}$ - 1.

`imm<x>` format and natural numbers of `simm<x>` format can be expressed in various radix:

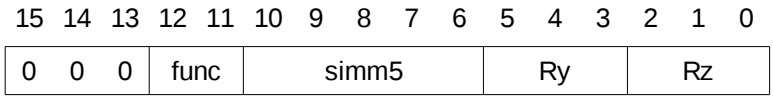
- Decimal - <digits(0-9)> or <digits(0-9)>D or <digits(0-9)>d,
- Hexadecimal - <digits(0-F or 0-f)>H or <digits(0-F or 0-f)>h,
- Octal - <digits(0-7)>O or <digits(0-7)>o,
- Binary - <digits(0,1)>B or <digits(0,1)>b.

Negative numbers must be expressed only in one of decimal radix formats.

5.2 Arithmetic with immediate value

Instructions of arithmetic type with 5-bit signed immediate value. This value is in two's complement format.

Instruction word structure



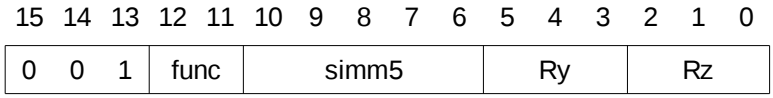
List of instructions

func		mnemonic	C language
0	0	SLU Rz, Ry, simm5	Rz = Ry < simm5;
0	1	SL Rz, Ry, simm5	Rz = sRy < simm5;
1	0	X	X
1	1	ADD Rz, Ry, simm5	Rz = Ry + simm5;

5.3 Transfer with immediate displacement

The only format that allows access to memory and perform synchronization matters. The final memory address is result of sum register value and 5-bit signed immediate value. This value represents displacement from register value.

Instruction word structure



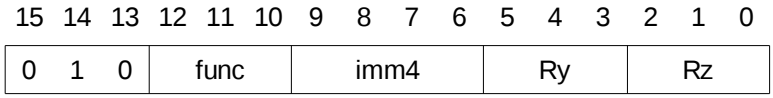
List of instructions

func		mnemonic	C language
0	0	LL Rz, Ry, simm5	Rz = *(Ry + simm5);
			LC = AC;
			LB = 1;
0	1	LD Rz, Ry, simm5	Rz = *(Ry + simm5);
1	0	SC Rz, Ry, simm5	if(LB && AC == LC) {
			*(Ry + simm5) = Rz;
			Rz = 1;
			LB = 0; }
			else {
			Rz = 0; }
1	1	ST Rz, Ry, simm5	*(Ry + simm5) = Rz;

5.4 Logic with immediate value

This instruction format can perform logic operations with 4-bit immediate value, which is first of all extended with zeros to 16-bit width. Instruction format also includes logical and arithmetic shift instructions.

Instruction word structure



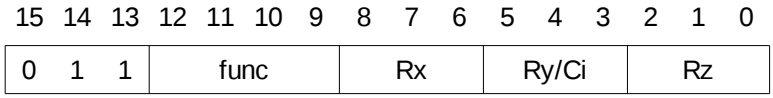
List of instructions

func			mnemonic	C language
0	0	0	OR Rz, Ry, imm4	Rz = Ry | imm4;
0	0	1	ORN Rz, Ry, imm4	Rz = Ry | ~imm4;
0	1	0	AND Rz, Ry, imm4	Rz = Ry & imm4;
0	1	1	ANDN Rz, Ry, imm4	Rz = Ry & ~imm4;
1	0	0	XOR Rz, Ry, imm4	Rz = Ry ^ imm4;
1	0	1	SLL Rz, Ry, imm4	Rz = Ry << imm4;
1	1	0	SRL Rz, Ry, imm4	Rz = Ry >> imm4;
1	1	1	SRA Rz, Ry, imm4	Rz = sRy >> imm4;

5.5 Arithmetic, logic and control with registers

Instruction format can perform all previous logical and arithmetic instructions. Moreover there is a subtraction. This instruction format uses only registers as operands. There are also instructions for access to control registers.*

Instruction word structure



List of instructions

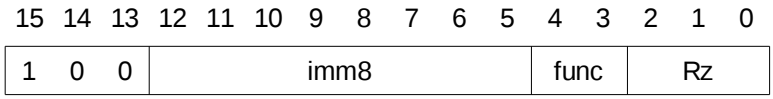
func				mnemonic	C language
0	0	0	0	OR Rz, Ry, Rx	Rz = Ry | Rx;
0	0	0	1	ORN Rz, Ry, Rx	Rz = Ry | ~Rx;
0	0	1	0	AND Rz, Ry, Rx	Rz = Ry & Rx;
0	0	1	1	ANDN Rz, Ry, Rx	Rz = Ry & ~Rx;
0	1	0	0	XOR Rz, Ry, Rx	Rz = Ry ^ Rx;
0	1	0	1	SLL Rz, Ry, Rx	Rz = Ry << (Rx & 0x000F);
0	1	1	0	SRL Rz, Ry, Rx	Rz = Ry >> (Rx & 0x000F);
0	1	1	1	SRA Rz, Ry, Rx	Rz = sRy >> (Rx & 0x000F);
1	0	0	0	SLU Rz, Ry, Rx	Rz = Ry < Rx;
1	0	0	1	SL Rz, Ry, Rx	Rz = sRy < sRx;
1	0	1	0	SUB Rz, Ry, Rx	Rz = Ry - Rx;
1	0	1	1	ADD Rz, Ry, Rx	Rz = Ry + Rx;
1	1	0	0	X	X
1	1	0	1	X	X
1	1	1	0	RTC Rx, Ci	Ci = Rx
1	1	1	1	CTR Rz, Ci	Rz = Ci

* When accessing to control registers, unused register index must have zero value.

5.6 Load immediate value

Instructions that load 8-bit unsigned immediate to registers. There are two mods: 1) load 8-bit immediate direct/scaled to register, 2) merge 8-bit immediate with lower/higher part of register.

Instruction word structure



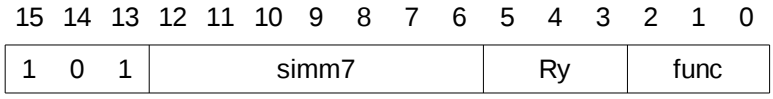
List of instructions

func		mnemonic	C language
0	0	LI Rz, imm8	Rz = imm8;
0	1	LIS Rz, imm8	Rz = imm8 << 8;
1	0	LIL Rz, imm8	Rz = (Rz & 0xFF00) + imm8;
1	1	LIH Rz, imm8	Rz = (imm8 << 8) + (Rz & 0x00FF);

5.7 Conditional jump with immediate displacement

Provides the only way to realize conditional branching. The final address of success jump is displacement of actual instruction address (IP register value). For this reason, instruction word contains 7-bit signed immediate value. Conditional branch is always result of compare register value and zero.

Instruction word structure



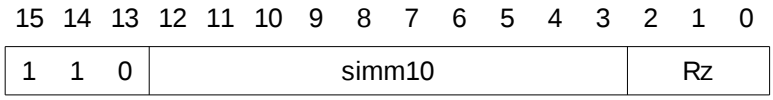
List of instructions

	func		mnemonic	C language
0	0	0	X	X
0	0	1	X	X
0	1	0	JNE Ry, simm7	if(Ry != 0) IP += simm7;
0	1	1	JE Ry, simm7	if(Ry == 0) IP += simm7;
1	0	0	JL Ry, simm7	if(sRy < 0) IP += simm7;
1	0	1	JLE Ry, simm7	if(sRy <= 0) IP += simm7;
1	1	0	JG Ry, simm7	if(sRy > 0) IP += simm7;
1	1	1	JGE Ry, simm7	if(sRy >= 0) IP += simm7;

5.8 Unconditional jump with immediate displacement

Includes only one instruction - instruction for realize unconditional jumps and calling functions in program. The first operand is register into which is saved return address (incremented IP register value). Using register R0 if don't needed to save return address. The second operand is 10-bit signed immediate in which range can be jump performed.

Instruction word structure



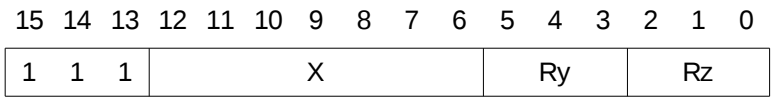
List of instructions

mnemonic	C language
JWL Rz, simm10	Rz = ++IP; IP += simm10;

5.9 Unconditional jump with registers

Another format for implement unconditional jumps and calling functions. However, opposed to previous format, the jump address is register value that is used as a pointer, save return address is also possible.

Instruction word structure

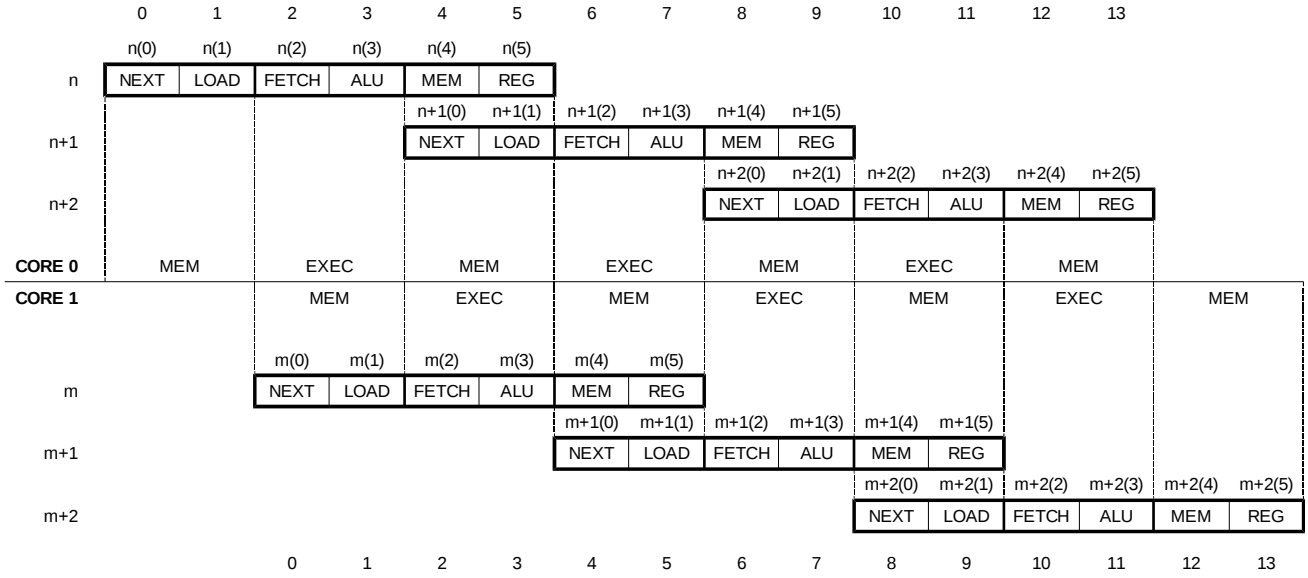


List of instructions

mnemonic	C language
JWL Rz, Ry	Rz = ++IP; IP = Ry;

6 Instruction processing

Each instruction takes six clock signals and is overlapped in two phases with a following instruction. So every four clock signals is completed one instruction on the processor core. A pair of processor cores are running together without any stalls.



6.1 Instruction phases

- NEXT - write new instruction address to IP register
- LOAD - read instruction from memory
- FETCH - fetch instruction from memory bus
- ALU - perform arithmetic or logic operation
- MEM - access to memory / control registers
- REG - write result to register

7 Control registers

Control registers are auxiliary core registers that provides way to use some extra functionality of core.

index	mnemonic	access	internal structure
0	MSR	RW	ID
1	SIP	RW	addr16
2	PRNG	R	rand16
3 - 7	X		

7.1 MSR - Machine Status Register

ID - Interrupt Disable

Reset value

ID - 1, others - 0

7.2 SIP - Saved Instruction Pointer

When interrupt occurs, the return address is saved here.

7.3 PRNG - Pseudorandom number generator

Every clock is generated a new pseudorandom number, which can be read from this register. All program writes to this register are ignored.

Reset value

Core 0 - 0xFFFF, Core 1 - 0xDDD4

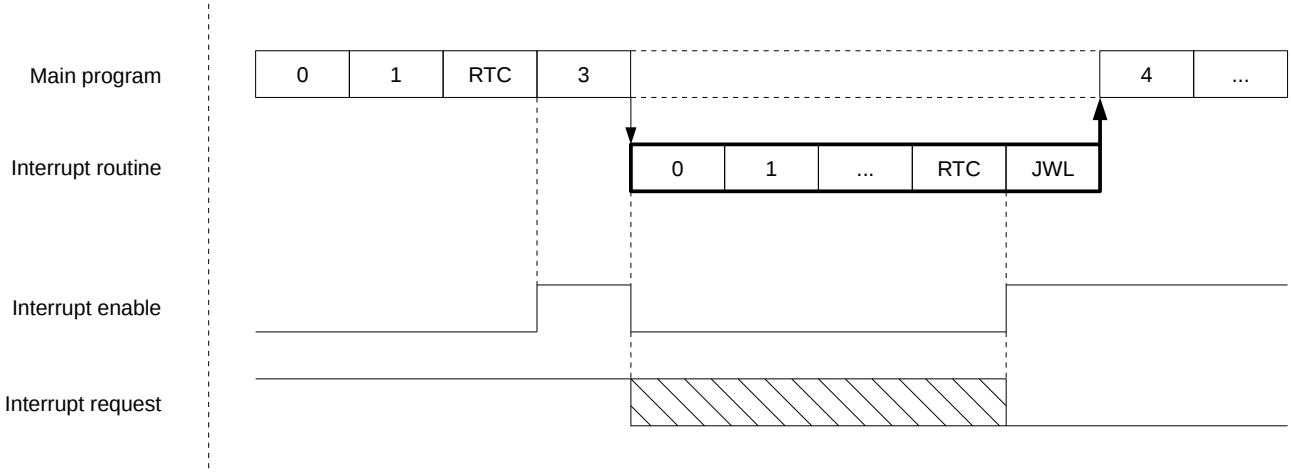
Algorithm

For generate pseudorandom numbers is used linear-feedback shift register Galois type.

Polynomial used: $x^{16}+x^{15}+x^{14}+x^{12}+x^{11}+x^{9}+x^{8}+x^{7}+x^{5}+x^{2}+1$

8 Interrupt processing

One instruction that follows RTC instruction can not be interrupted and always will be executed. After accept interrupt, further interrupts are automatically disabled.



9 Cores synchronization

Synchronization mechanism is used when core requires exclusive access to shared data. Before program gets exclusive access to this data, must use some form of synchronization primitives. The very access to this data is done in “critical section”.

9.1 Mechanism to achieve synchronization

To realize synchronization of access to shared data, there are two instructions LL and SC. Indeed, this instructions provide a way to implement higher level constructions for synchronization.

9.2 Behavior of LL and SC instructions

In the processor there are two bit registers called Link bit and Link core. When LL instruction is executed, Link bit is set to one and Link core is set to core's index which executed this instruction. When SC instruction is executed, it will be indicate whether it was successful or not with write value to register (1 - success, 0 - fail). SC will be successful only if Link bit is set to one and Link core is equal to index of core that is executing this instruction. If this condition is met, write to memory is approved and performed and Link bit is set to zero. Otherwise write to memory is dismissed and Link bit doesn't change.

9.3 Lock implementation example

This example uses Test and test-and-set method with active waiting.

```
; R1 - lock address

DEF unlock_st 0
DEF lock_st 1

test:          LD R2, R1, 0
               JNE R2, test

test_and_set:  LI R2, lock_st
               LL R3, R1, 0
               JNE R3, test
               SC R2, R1, 0
               JE R2, test_and_set

locked:        …
```

10 Program example

```
; == Call procedure ==
; test call procedure via register value as pointer

;address  mnemonic           binary             C like
                                                ; void main() {
                                                ; do {
 0   :    LI R4, 0           1000000000000100   ; R4 = 0;
 1   :    LIH R4, 4          1000000010011100   ; R4 = 1024;
 2   :    SRL R5, R4, 3      0101100011100101   ; R5 = R4 >> 3;
 3   :    JWL R6, R5         1110000000101110   ; <R5>(); //128
 4   :    LIL R4, 0          1000000000010100   ; R4 &= 0xFF00;
 5   :    JWL R0, R4         1110000000100000   ; } while(1);
                                                ; void 128() {
 128 :    SUB R4, R4, R5     0111010101100100   ; R4 -= R5;
 129 :    SRL R4, R4, 1      0101100001100100   ; R4 >>= 1;
 130 :    LIH R4, 0          1000000000011100   ; R4 &= 0x00FF;
 131 :    JWL R0, R4         1110000000100000   ; //“JMP R4” (192)
                                                ;
 192 :    JWL R0, R6         1110000000110000   ; return;
                                                ; }
                                                ; }
```

The program is only for demonstrate the possibilities. Used algorithm is not intended to be optimized.

List of Symbols and Abbreviations

DB	Double byte
IP	Instruction pointer
Rz, Ry, Rx	General purpose register indexes
LC	Link core
AC	Actual core
LB	Link bit
Ci	Control register index
DEF	Define