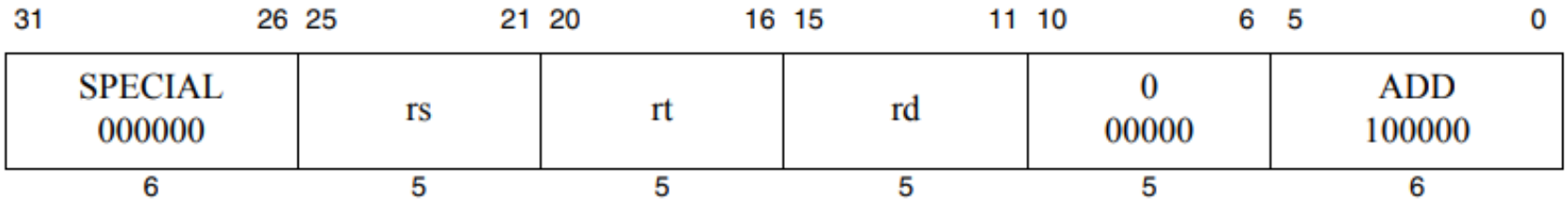
VMIPS reference v1.0

▼ 1.1 Understanding of Instruction field

1.1.1 Instruction format



constant field & opcode 는 대문자로 표기 : SPECIAL & ADD

SPECIAL은

```
SPECIAL = inst[31:26]
ADD = inst[5:0]
```

모든 variables 는 소문자로 표기 : rs, rt, rd

```
rs = inst[25:21]
rt = inst[20:16]
rd = inst[15:11]
```

0을 포함하는 field는 not named, unused이고 0으로 채워져야한다.

0이 아닌 값을 포함하면 processor가 예측불가능한 동작을 한다!

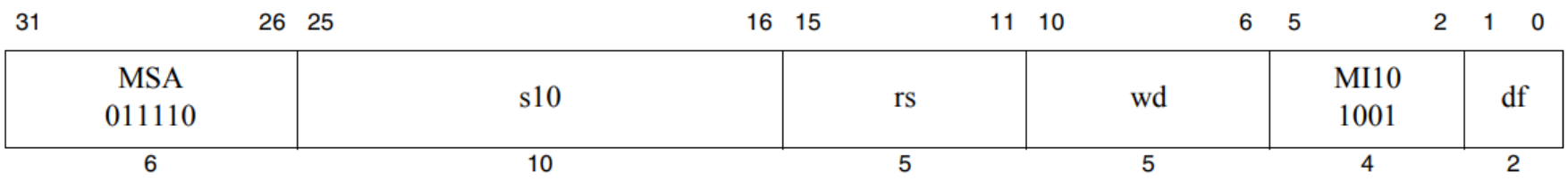
MSA는 **MIPS SIMD Architecture** 의 약자

모든 MSA 명령어는 MSA major opcode space로 encoding 된다!

RTL설계에 당장 필요한 Instructions

- **ADDV, ADDVI : Vector Add - reference p.88, 89**
- **SUBV, SUBVI : Vector Substract - reference p.314, 315**
- **LD, LDI : Vector Load - reference p.228**
- **ST : Vector Store - reference p.304**

Vector store

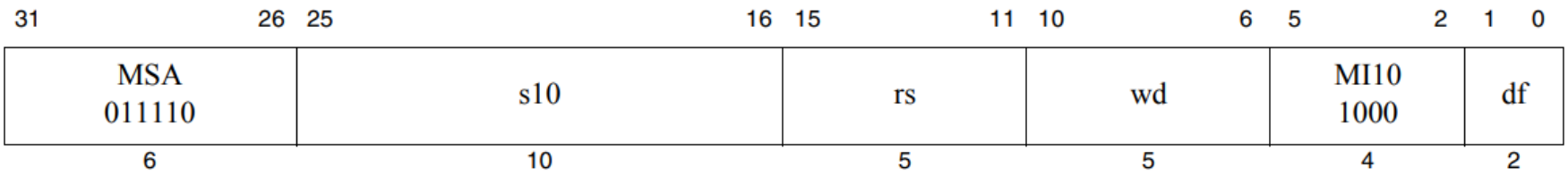


MI10(opcode의 일종인듯? funct code?)를 제외하고는 LD와 완전히 같다.

Purpose : Vector Store

element-by-element로 base register(rs) + offset 주소값에 vector를 저장한다.

Vector Load



Purpose : Vector Load (from main memory to vector registers)

Description : wd[i] ← memory [rs + (s10 + i) * sizeof(wd[i])]

즉, wd vector의 i th element는 main memory에서 rs(vector시작 주소) + (s10 + i) * **size of element**

data를 element by element로 가져온다. 그래서 element를 load하는 과정은 atomic operation이어야 한다!!(element를 가져오는 과정에서 다른 operation이 끼어들면 안됨)

관례상, assembly syntax에서 모든 offset이 byte 단위이며, df(data format)의 배수여야 한다.

→ s10은 byte offset을 df의 size로 나눠서 결정한다!!

일반적으로 어셈블리 언어 구문에서는 모든 오프셋이 바이트 단위이며 데이터 크기의 배수여야 합니다. 포맷 df 어셈블러는 바이트 오프셋을 데이터 형식 df의 크기로 나눈 s10 비트 필드 값을 결정한다.

```
inst[31:26] = MSA = 6'b011_110
inst[25:16] = s10 // 10bit for immediate offset value
inst[15:11] = rs // 5bit for scalar register address
```

▼ 2.1 MSA Vector Registers

MSA operates on 32 128-bit wide vector register(이하 VR)!

→ **128bit wide register가 32개?**

2.1.1 Data foramt

MSA VR에는 4가지 data format이 있다.

→ **Byte(8bit), half-word(16bit), word(32bit), double-word(64bit)**

→ISA에 2bit wide인 df라는 영역이 있는데, 이 2bit를 통해서 4가지 data format을 결정하는듯

Data format에 따라, vector register는 여러 element로 구성된다!

예를 들어, Data format이 다음과 같은 table을 따른다면 각 vector register의 element는 다음과 같이 구성된다!

df[1:0] 예시입니다!!	Data format	Element mapping
2’b00	Byte	128bit / 8bit = 16 elements
2’b01	half-word	128bit / 16bit = 8 elements
2’b10	word	128bit / 32bit = 4 elements
2’b11	double-word	128bit / 64bit = 2 elements

Figure 3-3 MSA Vector Register Byte Elements

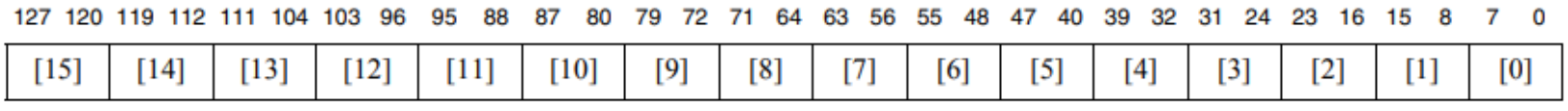


Figure 3-4 MSA Vector Register Halfword Elements

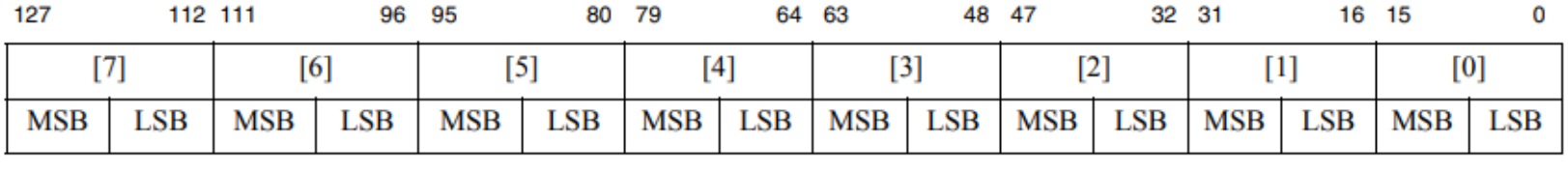


Figure 3-5 MSA Vector Register Word Elements

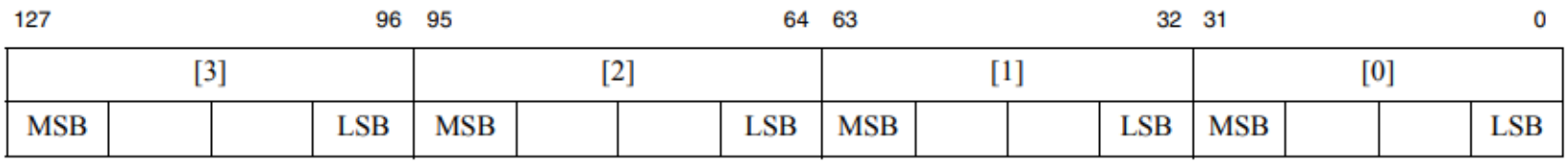
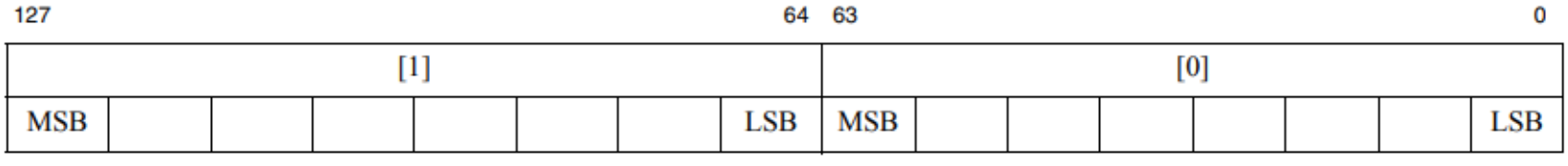


Figure 3-6 MSA Vector Register Doubleword Elements



2.1.2 Vector register layout

slide instruction(SLD, SLDI)에 쓰이는 VR은 2 dimension byte array layout을 가진다.

slide inst는 당장 사용할 필요는 없으니, 추후에 필요하다면 reference p.37 참고

2.1.3 How vector is stored in memory?

MSA에서 vector는 **0번째 element**부터 시작해서 memory의 **lowest byte address**에 저장된다.

각 element가 Big-endian을 따를지 Little-endian을 따를지는 CP0 Config register(special register의 일종인듯)의 BE bit에 의해 결정된다!

가령 예를들어, 다음 표는 word data format을 가지는 MSA vector가 memory에 저장되는 방식을 나타낸다.

Table 3.1 Word Vector Memory Representation

Word Vector Element		Little-Endian Byte Address Offset	Big-Endian Byte Address Offset
Word [0]	Byte [0] / LSB	0	3
	Byte [1]	1	2
	Byte [2]	2	1
	Byte [3] / MSB	3	0
Word [1]	Byte [0] / LSB	4	7
	Byte [1]	5	6
	Byte [2]	6	5
	Byte [3] / MSB	7	4
Word [2]	Byte [0] / LSB	8	11
	Byte [1]	9	10
	Byte [2]	10	9
	Byte [3] / MSB	11	8
Word [3]	Byte [0] / LSB	12	15
	Byte [1]	13	14
	Byte [2]	14	13
	Byte [3] / MSB	15	12

each element가 직관적으로 표현되는 Little-endian을 따르는게 좋을듯?

2.1.4 Floating point Register

아직 FP을 다루기는 시기상조인듯 하다!! 일단 int type으로 먼저 설계해보고 추후에 추가!