



Building a Multi-Issue Vector DSP with Configurable-Processor Technology

September 30, 2004

**Steven Leibson & Himanshu A. Sanghavi
Tensilica Inc.**

- “Order of magnitude” more bandwidth!
- “Order of magnitude” more computation throughput!
- Lower power consumption!
- Programmability a *must* (complex algorithms, evolving standards, short product cycle)

Video Example

	90's	Now	Change
<i>Resolution</i>	Full D1	HD	10X data
<i>Compression</i>	MPEG2	H.264	30X compute
<i>Power consumption</i>	Line	Line & battery	Same or lower

	PRO	CON
<i>General DSP processor cores</i>	Programmable	Inefficient for application specific algorithms
<i>Hardwired digital signal processing blocks</i>	Highly efficient implementation of algorithms	Rigid, Non-programmable

	PRO	CON
<i>General DSP processor cores</i>	Programmable	Inefficient for application specific algorithms
<i>Hardwired digital signal processing blocks</i>	Highly efficient implementation of algorithms	Rigid, Non-programmable



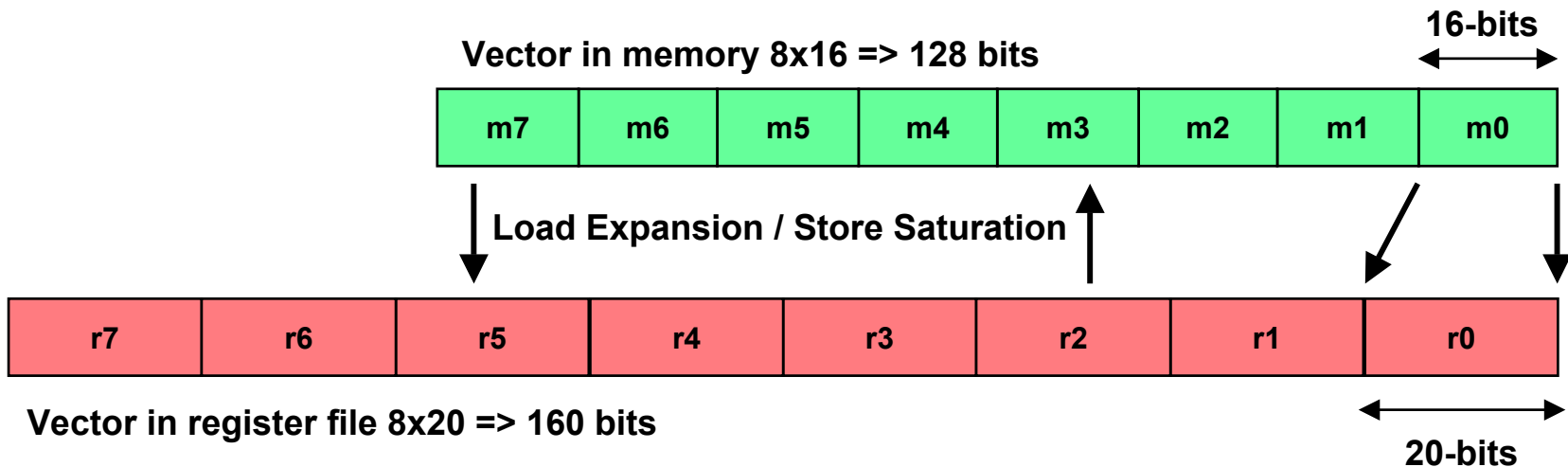
Requirements !



Vectra LX: High Performance DSP Engine

- **Configuration option to synthesizable Xtensa LX processor.**
 - Click button GUI selection, fully integrated with Xtensa core
- **Based on FLIX “Flexible Length Instruction Xtension” Technology**
 - General DSP instruction set >210 instructions: ALU, MAC and load/store
 - 64b instructions modelessly intermix with 16b/24b Xtensa core instructions
 - 3 vector operations per instruction bundle. Dual load/store units.
- **SIMD machine – register operands are vectors**
 - 4 or 8 scalar elements per vector
- **Very Low Power**
 - Fine grain clock gating of every functional element
- **User Extensible: instruction extensions can be added**
 - Built using Tensilica Instruction Extension (TIE) methodology
 - Application specific accelerations can be easily added

- ▀ **Fundamental “memory” data-type is 16-bits wide**
 - Eight 16-bit scalars per vector => 128-bit wide vector in memory
- ▀ **Fundamental “register” data-type is 20-bits wide**
 - 4 guard bits per scalar to increase intermediate precision
- ▀ **Double-width data type for “accumulator”**
 - 40 bits in register – 4 double-width scalars per vector





Vectra LX: Instruction Format

63		46	45		28	27		4	3		0	
ALU & 2 nd Ld/St			MAC & Select			Load/Store & Core			1	1	1	0

Instructions

Arithmetic
Logic
Shift
Special ALU

2nd Load/Store

Instructions

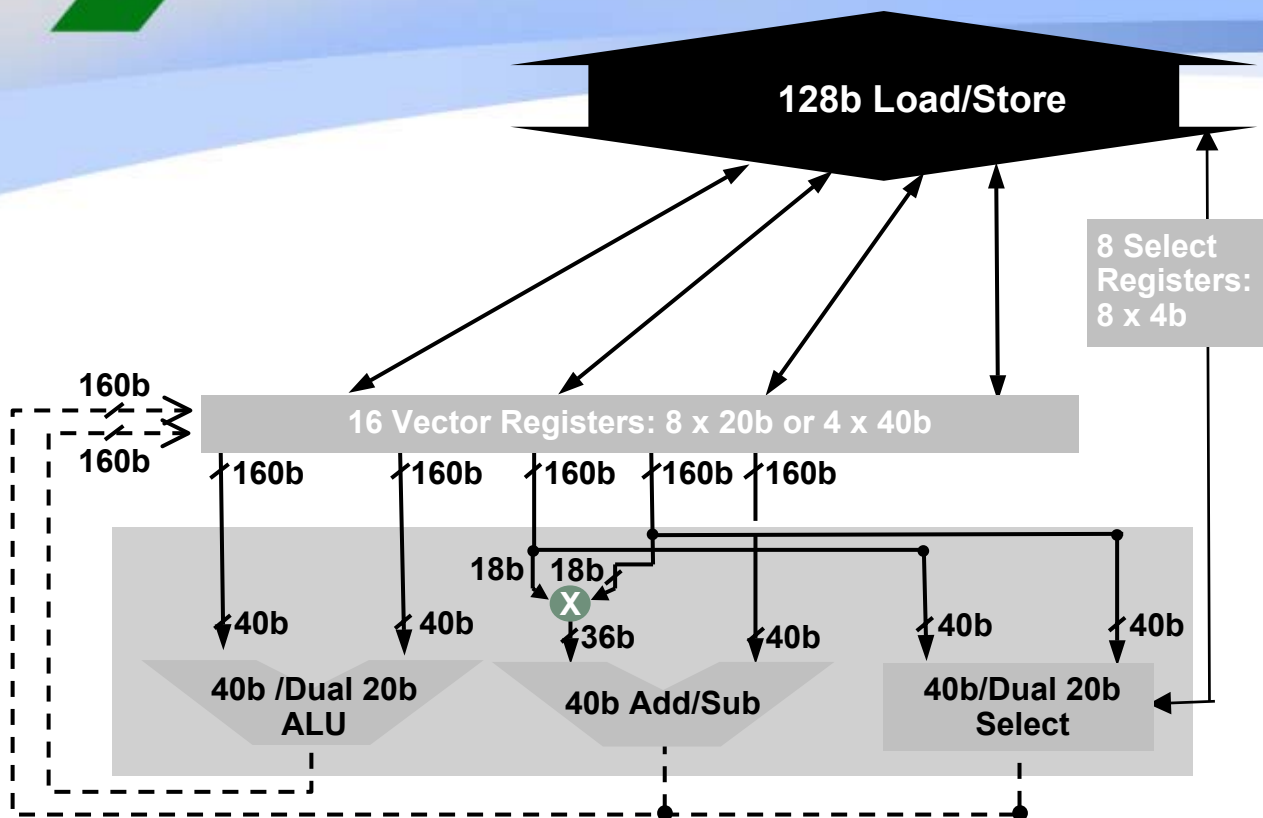
MAC
Complex MAC
Select

Instructions

Vector Load/Store
Scalar Load/Store
Unaligned Load/Store

Xtensa Core

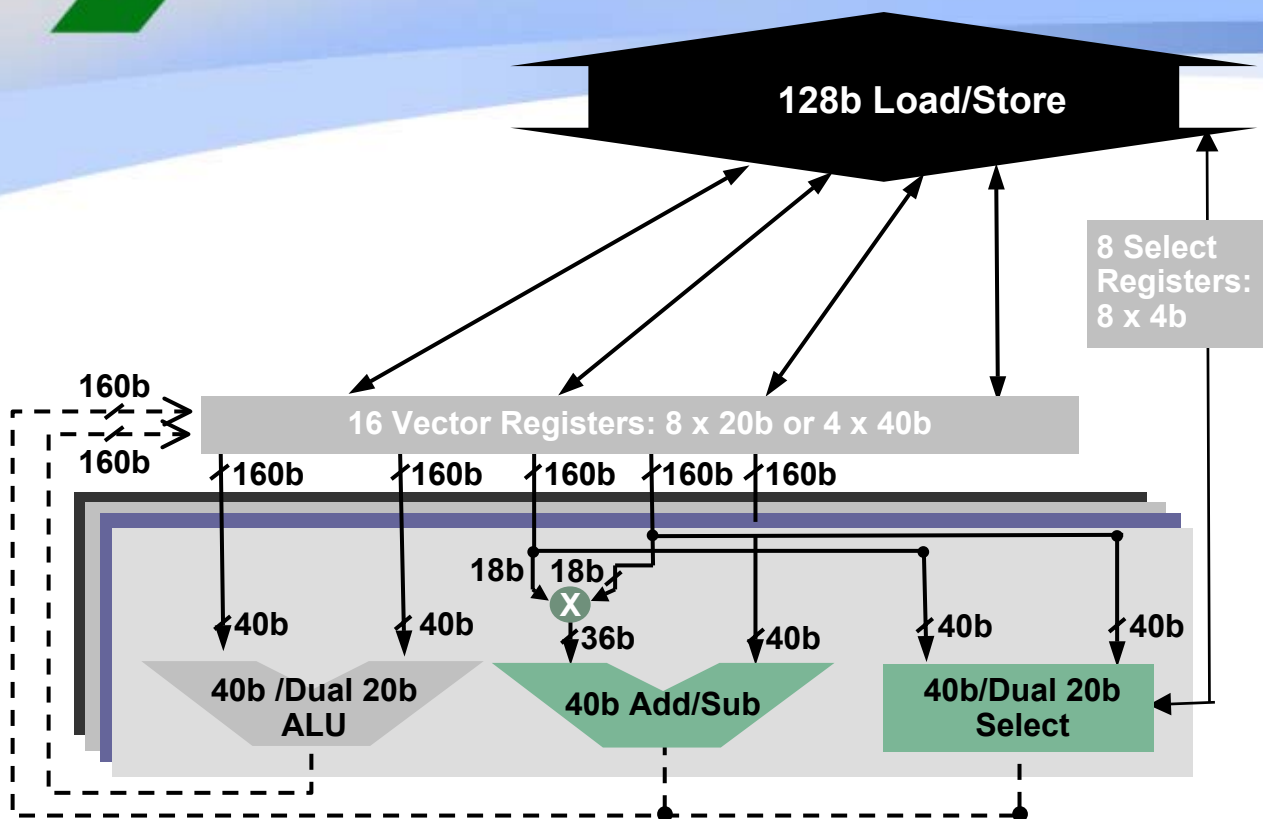
Parallel Operations



Issue 3 vector operations per cycle.

- 128b Load / Store or Core Instruction

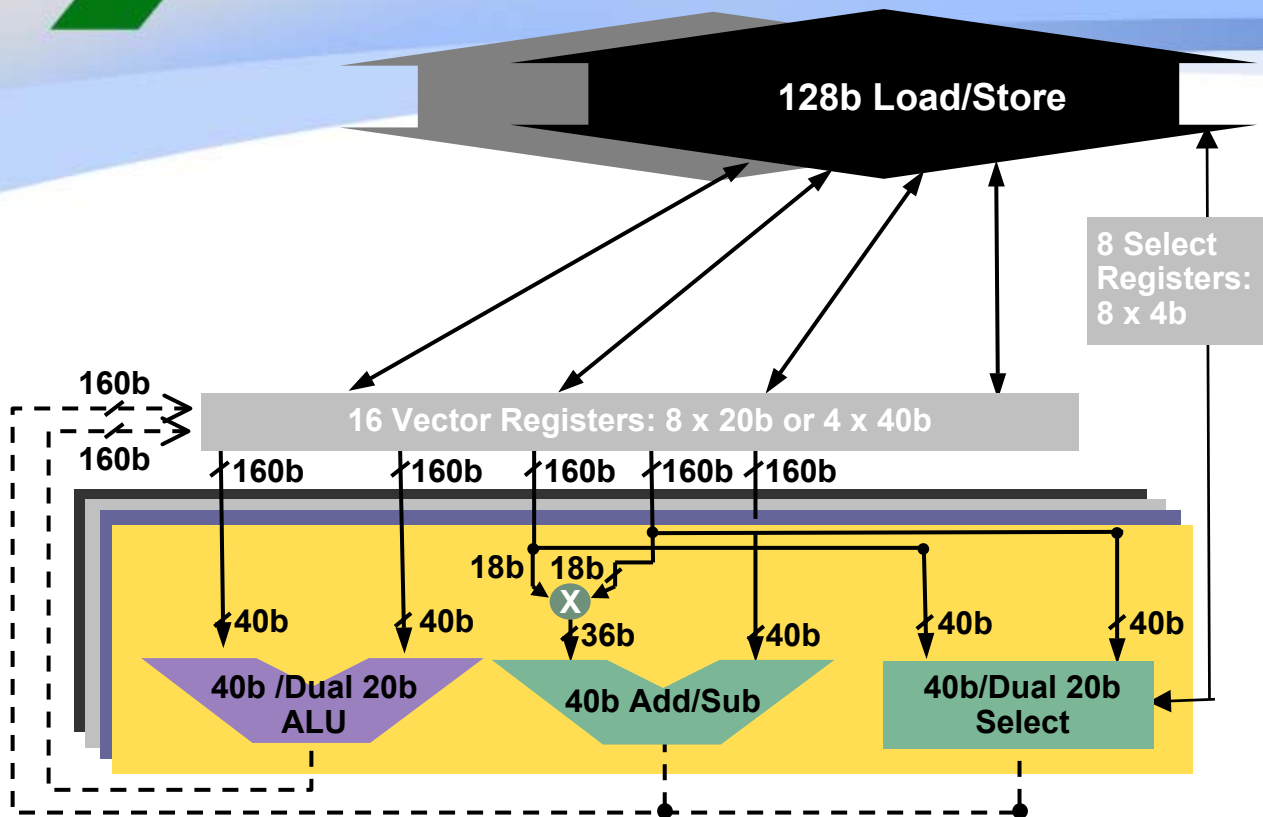
Parallel Operations



Issue 3 vector operations per cycle.

- 128b Load / Store or Core Instruction
- MAC (SIMD 4x40b) or Select

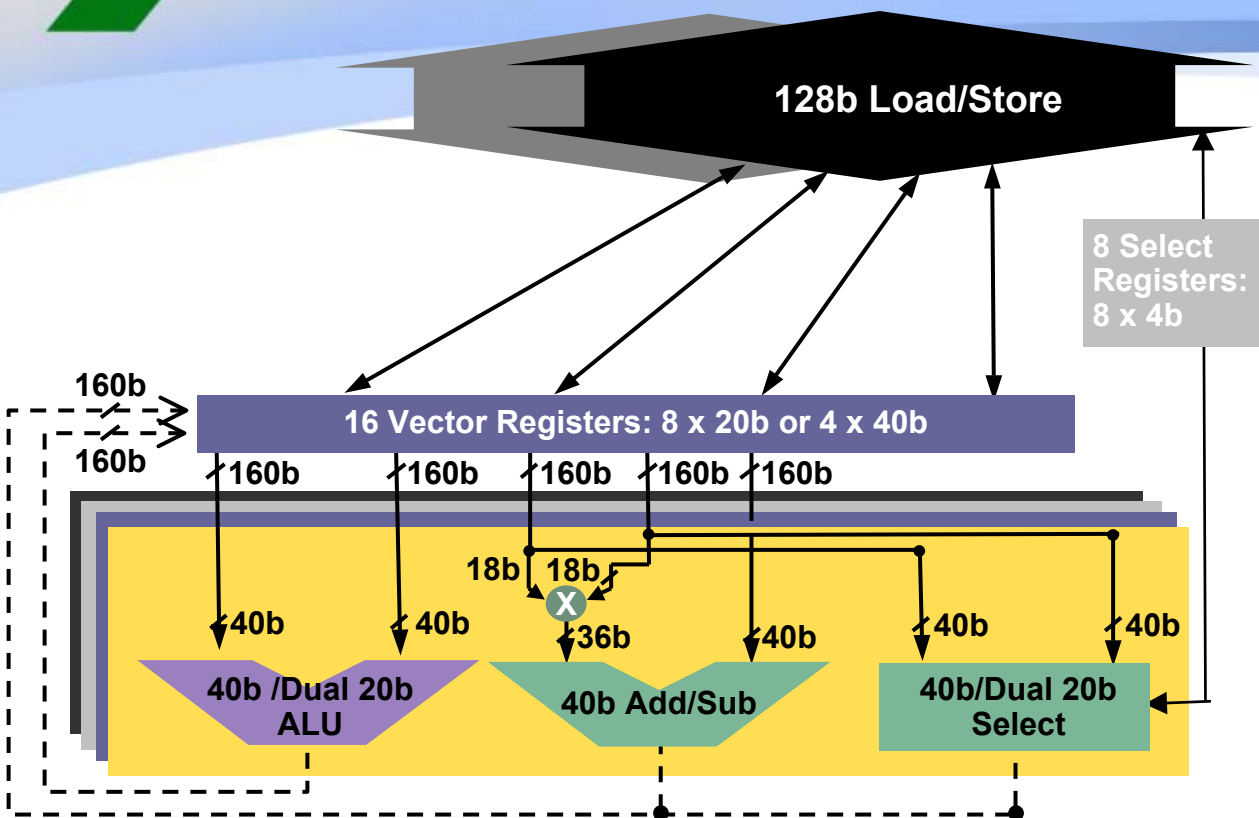
Parallel Operations



Issue 3 vector operations per cycle.

- 128b Load/Store or Core Instruction
- MAC (SIMD 4x40b) or Select
- ALU (SIMD 4x40b or 8x20b) or Second 128b Load/Store

Parallel Operations

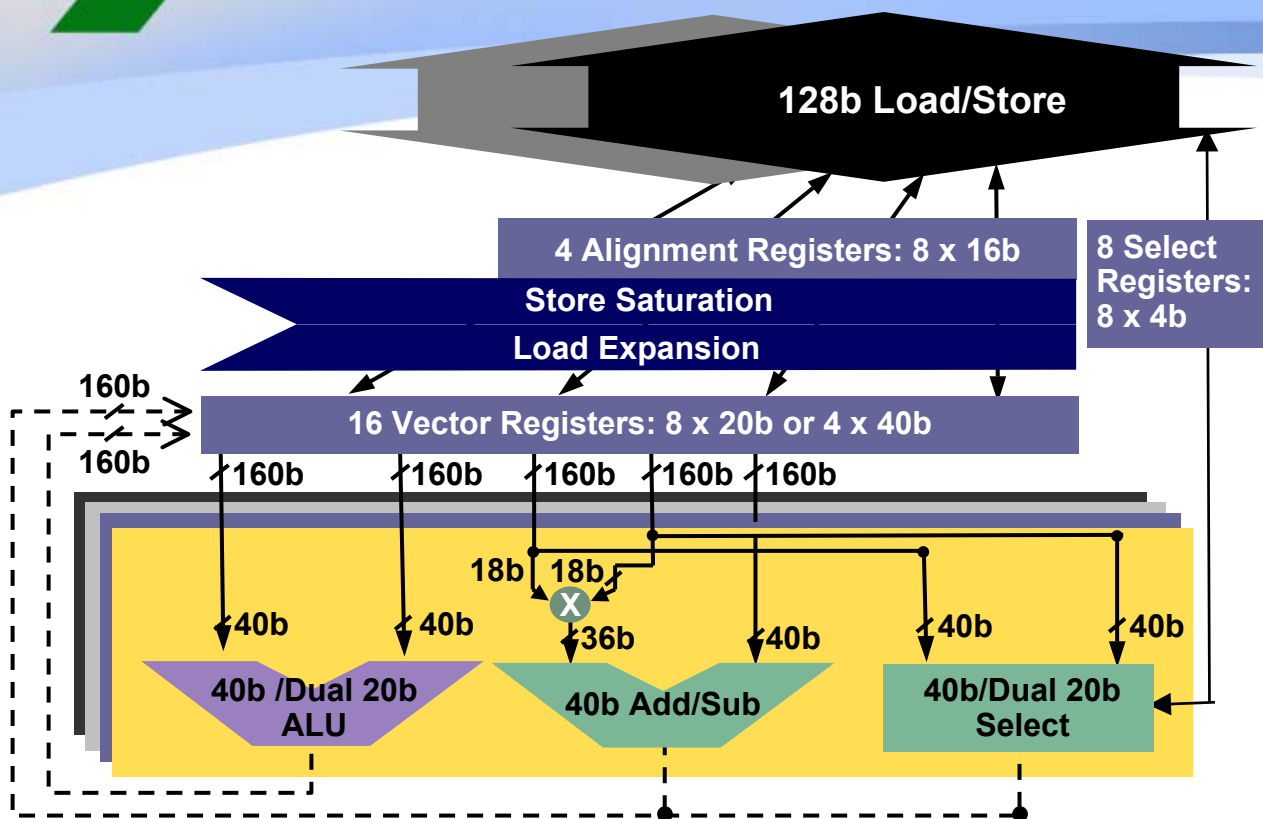


Issue 3 vector operations per cycle.

- 128b Load/Store or Core Instruction
- MAC (SIMD 4x40b) or Select
- ALU (SIMD 4x40b or 8x20b) or Second 128b Load/Store

Large vector register file:

16x160bits (320Bytes).
6 read, 3 write ports. 66 GB/sec.



- 2x128b Data Transfer. 12GB/sec @370MHz. Two independent address calculations
- Four alignment registers supporting unaligned vectors
- Loads sign extend to 20b/40b. Stores saturate

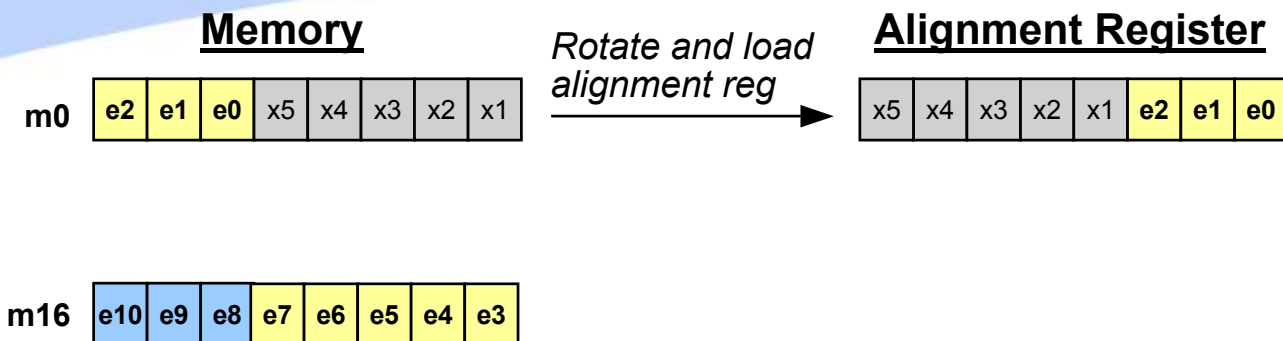
Issue 3 vector operations per cycle.

- 128b Load/Store or Core Instruction
- MAC (SIMD 4x40b) or Select
- ALU (SIMD 4x40b or 8x20b) or Second 128b Load/Store

Large vector register file:

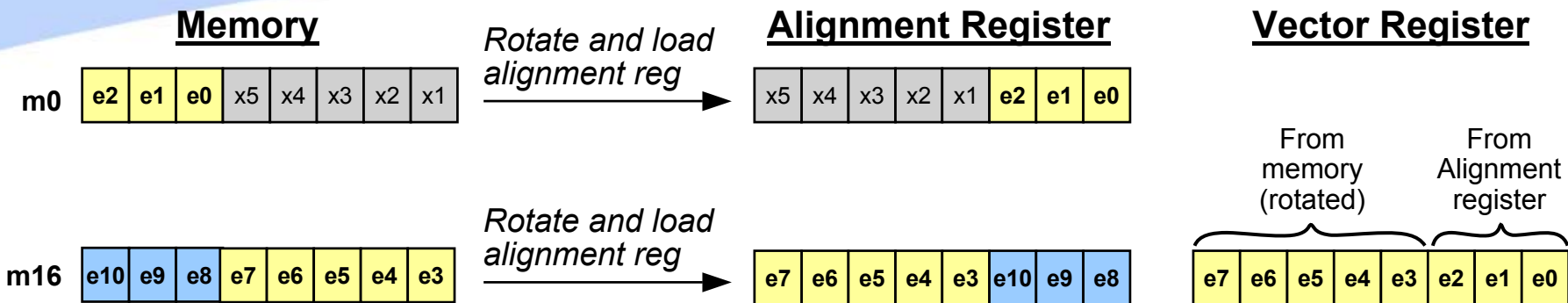
16x160bits (320Bytes).
6 read, 3 write ports. 66 GB/sec.

Unaligned Vector Load/Store



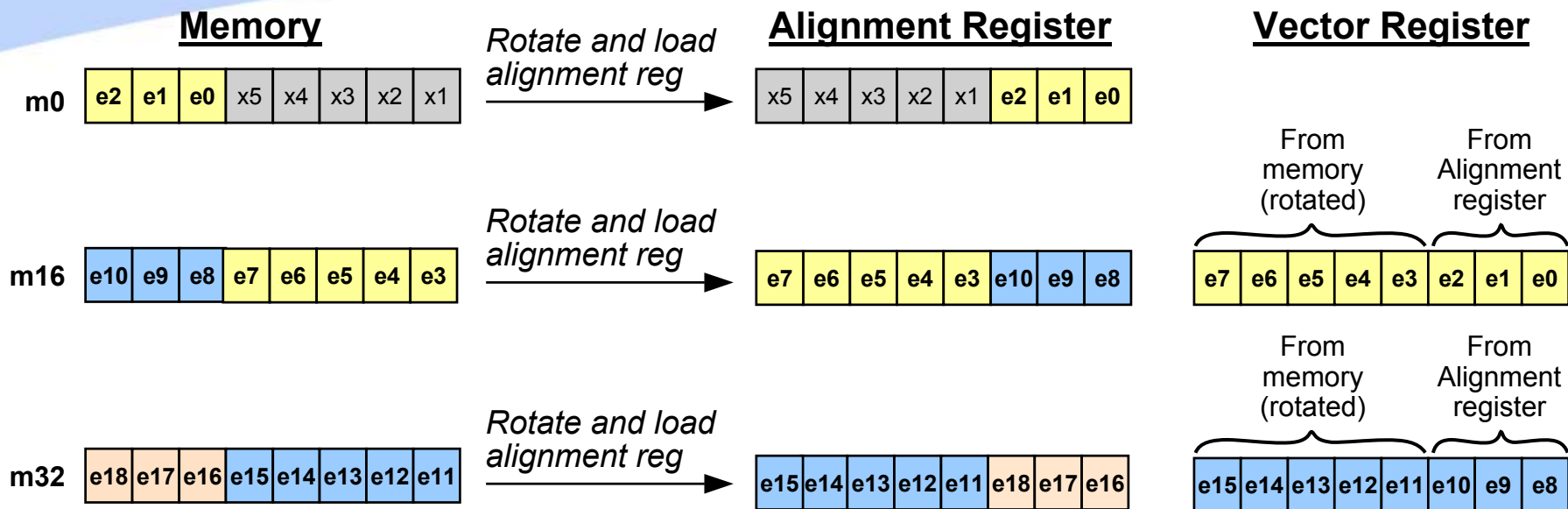
- Data rotated based on least significant address bits.
- 4 x128b alignment registers. Initialized to hold partial vector.

Unaligned Vector Load/Store

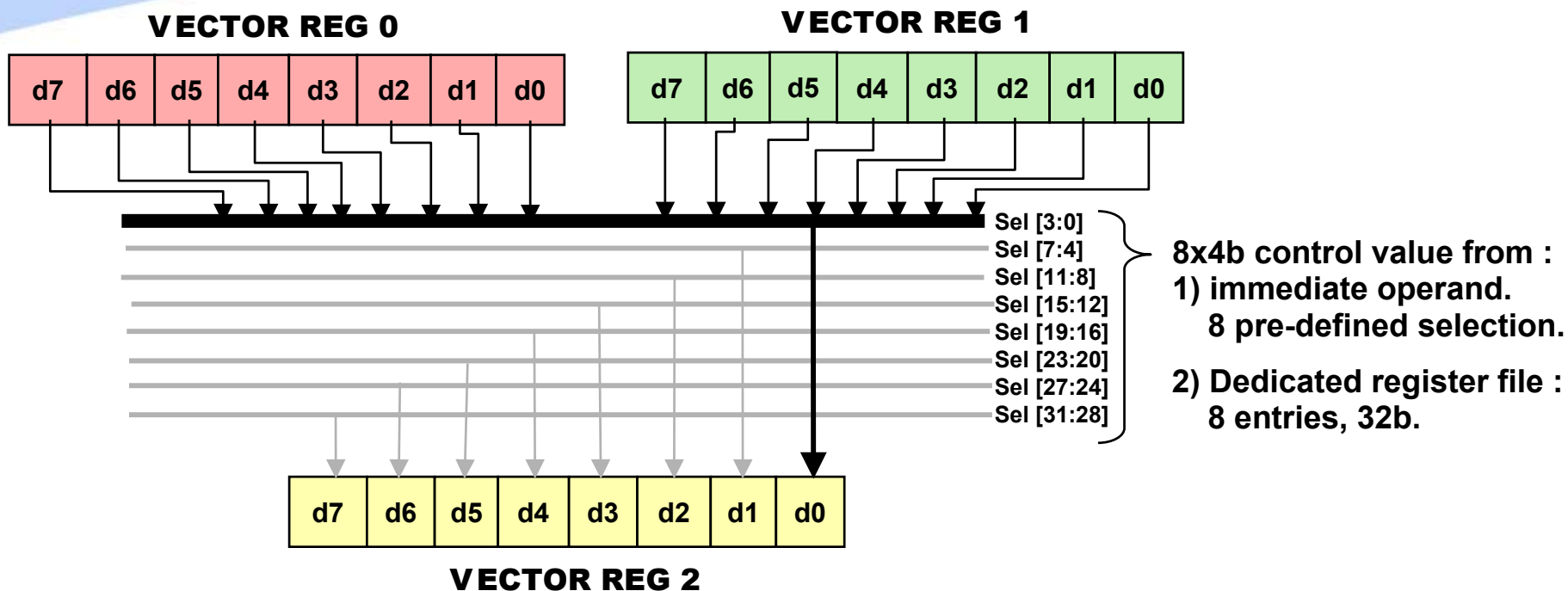


- ▀ Data rotated based on least significant address bits.
- ▀ 4 x128b alignment registers. Initialized to hold partial vector.
- ▀ Subsequent loads merge load data with contents of alignment register, and prime alignment register for next load.

Unaligned Vector Load/Store

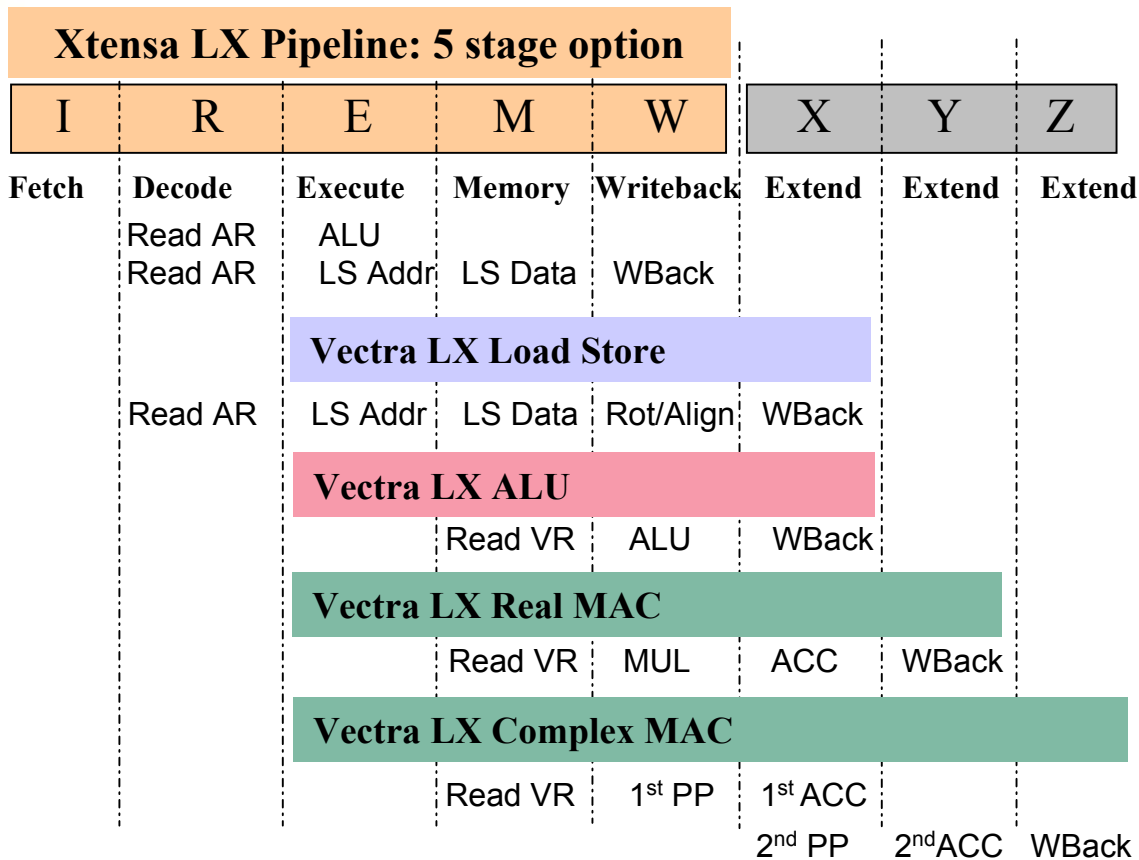


- Throughput: 1 unaligned load/store per cycle using standard compiled memory
- Aligned load/store can be issued from 2nd load/store slot in parallel to these operations



Powerful SELECT instructions to define general elements transfer

- Each target element can be individually selected from the source vectors
- Easy to implement replication, rotation, shift, interleaving...



- ▀ DSP computation placed in late pipe stages to eliminate load delay slot.
- ▀ Additional stages for extra computation: Multiply/Add and Complex Multiply/Add
- ▀ Can also be combined with 7-stage option of Xtensa LX for longer memory access

Area	Approximately 250,000 gates (Xtensa + Vectra)
Max Frequency	370MHz in 0.13 μ m technology, using high performance libraries and synthesis flow
Power	0.5mW/MHz for 370MHz target 0.25mW/MHz for 100MHz target
Performance	994 cycles for 256pt Radix-4 FFT



Complete Xtensa LX Software Tools: Vector Compiler for Vectra LX

```
short a[128], b[128], c[128];
void min_mul()
{ int i;
  for (i=0; i<128; i++) {
    c[i] = MIN(a[i]*b[i], 16384);
  }
}
```

C code

```
loopgtz a 1,.L
{pack40 v6,v3,v5; mul18.0 v4,v0,v1; lvs16.iu v0,a8,16}
{min40 v3,v2,v7; nop; lvs16.iu v1,a9,16}
{min40 v5,v4,v7; mul18.1 v2,v0,v1; svs16.iu v6,a10,16}
.L:
```

Compiled Loop

Compiler vectorize a, b, c -> A, B, C. 8 elements per vector

Software pipelined loop overlaps operations from different iterations: N-2, N-1, N

(N-2) PACK even/odd from MIN	(N-1) AxB even	(N) Load A
(N-1) MIN odd		(N) Load B
(N-1) MIN even	(N) AXB odd	(N-2) Store C



Complete Xtensa LX Software Tools: Vector Data Type for Vectra LX

```
short a[128], b[128], c[128];  
void min_mul()  
{int result=0;  
int i;  
for (i=0; i<128; i++) {  
    c[i] = MIN(a[i]*b[i], 16384);}}
```

C code

C/C++ compiler and debugger support vector types just like other C types.

Algorithm expressed using vector data types. Type conversions implicit.

```
vec8x16 a[16], b[16], c[16];  
void min_mul()  
{ vec4x40 V0, V1;  
vec4x40 V_con = 16384;  
int i;  
WUR_VSAR(0);  
for(i = 0; i <16; i++)  
{V0 = MUL18_0(a[i], b[i]);  
V1 = MUL18_1(a[i], b[i]);  
V0 = MIN40(V0, V_con);  
V1 = MIN40(V1, V_con);  
c[i] = PACK40(V1, V0); }}
```

Vector Data Type

```
loopgtz a11,.L
  lvs16.iu v0,a8,16
  mul18.0 v4,v0,v1
  pack40 v6,v3,v5
  lvs16.iu v1,a9,16
  min40 v3,v2,v7
  svs16.iu v6,a10,16
  mul18.1 v2,v0,v1
  min40 v5,v4,v7
```

```
.L
```

Assembly Input

Assembled Output

```
loopgtz a11,.L
{pack40 v6,v3,v5; mul18.0 v4,v0,v1; lvs16.iu v0,a8,16}
{min40 v3,v2,v7; nop; lvs16.iu v1,a9,16}
{min40 v5,v4,v7; mul18.1 v2,v0,v1; svs16.iu v6,a10,16}
.L:
```

- ▀ Analyze data dependence from straight line assembly code stream
- ▀ Model Vectra LX resource
- ▀ Automatic schedule instructions into Vectra LX bundles.

Complete Xtensa LX Software Tools: Pipeline Accurate ISS for Vectra LX

```

.....
addi.n    a4, a4, -1
loopnez   a4, .Lwinddown
{lvs16.iu v4, a2, 16; mula18.0 v2, v0, v1; nop}
{lvs16.iu v1, a3, 16; mula18.1 v3, v0, v1; mov160 v0, v4}
.Lwinddown:
.....

```

Code Stream

Traditional Instruction Set Simulator:

- Model the state of an instruction
- Estimate cycle
- Instruction order approximates event order

ADDI	I	R	E	M	W					
LOOP		I	R	E	M	W				
B1:LVS			I	R	E	M	W	X		
B1:MULA					E	M	W	X	Y	
B2:LVS				I	R	E	M	W	X	
B2:MULA						E	M	W	X	Y
B2:MOV						E	M	W	X	

Xtensa LX Pipeline Accurate Instruction Set Simulator:

- Model the state of the pipeline
- Simulate cycle & event order



Creating Your Own DSP Extensions: Add to, or Modify Vectra LX, using TIE

Vectra LX was designed using TIE (*Tensilica Instruction Extensions*)

Define DSP register file
16 entries 160b wide

Define DSP instruction:
SIMD Multiply/Add of
even elements

Specify pipeline
schedule for Multiply/
Add instruction

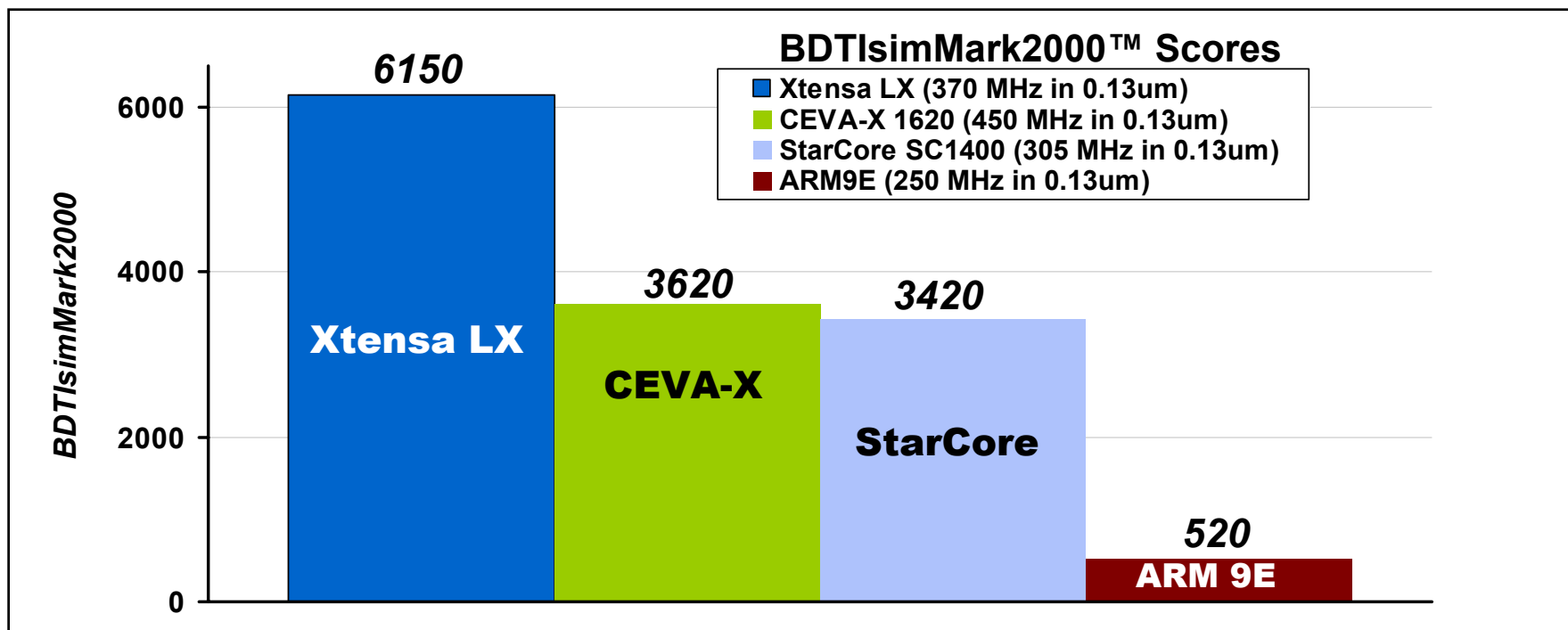
```
regfile vec 160 16 v
operation MULA18.0 {inout vec acc, in vec m0, in vec m1} {} {
  wire [39:0] sum0 = m0[ 17: 0] * m1[ 17: 0] + acc[ 40: 0];
  wire [39:0] sum1 = m0[ 57: 40] * m1[ 57: 40] + acc[ 79: 40];
  wire [39:0] sum2 = m0[ 97: 80] * m1[ 97: 80] + acc[119: 80];
  wire [39:0] sum3 = m0[137:120] * m1[137:120] + acc[159:120];
  assign accum = {sum3, sum2, sum1, sum0}; }
schedule mula {MULA18.0} {
  use m0 4; use m1 4; use acc 5; def acc 5; }
```

- Specification: Verilog description of semantics of custom instructions
 - Optionally add custom register files or states
 - Optionally use pre-defined register files from Xtensa LX, Vectra LX
- TIE Compiler automatically extends Xtensa LX processor
 - Updated Synthesizable RTL
 - Matching SW development tools

BDTI Benchmark: Xtensa LX Processor with Vectra LX engine

Xtensa LX Configuration includes Vectra LX DSP Engine + 11 custom extensions

- Viterbi trellis decode (3) ,Viterbi trellis traceback (2), Bit stream unpacking (2), Multiply intensive filter (4) - add a total of 30K additional gates



The BDTIsimMark2000™ is a summary measure of DSP speed. See www.BDTI.com for info. Scores © 2004 BDTI.

Xtensa LX configuration as tested by BDTI: 248,600 “gates” (equivalent NAND2X cell area) at post-synthesis; 4.4mm² actual layout area; 3D extracted final layout timing under worst case conditions: 369 MHz



Summary: Xtensa LX Processor Optimal DSP Solutions for Every Application

Vectra LX: New programmable DSP engine

- *Integrated with configurable Xtensa LX processor core*
- *Competitive performance compared to other general DSPs*

Application specific extensions

- *Target specific algorithms*
- *Augment performance. Augment efficiency.*
- *Delivers highest-ever BDTI benchmark performance*

Tensilica Instruction Extension (TIE) methodology

- *Easy for Tensilica licensees to create application-specific DSP extensions*
- *Easy for Tensilica licensees to extend Vectra LX engine*
- *Automatically produce updated processor RTL + matching software tools.*
- *Achieve the right performance + power efficiency for YOUR application*