Vector Processors

Computer Science Department
CS 566 – Fall 2012

Eman Aldakheel
Ganesh Chandrasekaran
Prof. Ajay Kshemkalyani

OUTLINE

- What is Vector Processors
- Vector Processing & Parallel Processing
- Basic Vector Architecture
- Vector Instruction
- Vector Performance
- Advantages
- Disadvantages
- Applications
- Conclusion

Vector Processors

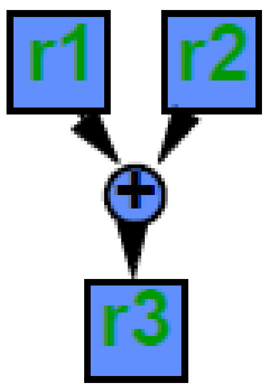
- A processor can operate on an entire vector in one instruction
- Work done automatically in parallel (simultaneously)
- The operand to the instructions are complete vectors instead of one element
- Reduce the fetch and decode bandwidth
- Data parallelism
- Tasks usually consist of:
  - Large active data sets
  - Poor locality
  - Long run times

Vector Processors (Cont'd)

- Each result independent of previous result
  - Long pipeline
  - Compiler ensures no dependencies
  - High clock rate
- Vector instructions access memory with known pattern
- Reduces branches and branch problems in pipelines
- Single vector instruction implies lots of work
  - Example:

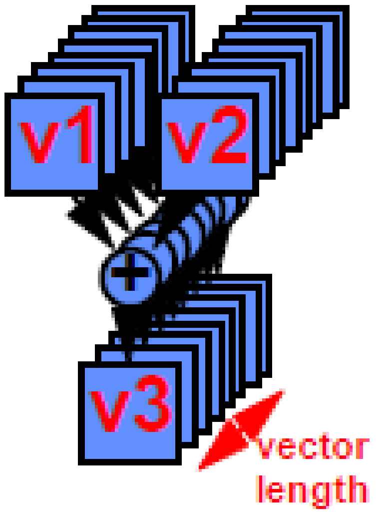
```
for(i=0; i<n; i++)
        c(i) = a(i) + b(i);
```

SCALAR
(1 operation)




```
add r3, r1, r2
```

VECTOR
(N operations)




```
add.vv v3, v1, v2
```

VECTOR PROCESSORS (CONT'D)

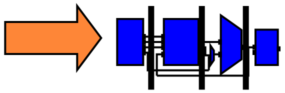
```
// C code
for(i=0;i<16; i++)
  b[i]+=a[i]
```

```
// Vectorized code
set     vl,16
vload   vr0,b
vload   vr1,a
vadd    vr0,vr0,vr1
vstore  vr0,b
```

Each vector instruction holds many units of independent operations

vadd

b[15]+=a[15]
b[14]+=a[14]
b[13]+=a[13]
b[12]+=a[12]
b[11]+=a[11]
b[10]+=a[10]
b[9]+=a[9]
b[8]+=a[8]
b[7]+=a[7]
b[6]+=a[6]
b[5]+=a[5]
b[4]+=a[4]
b[3]+=a[3]
b[2]+=a[2]
b[1]+=a[1]
b[0]+=a[0]



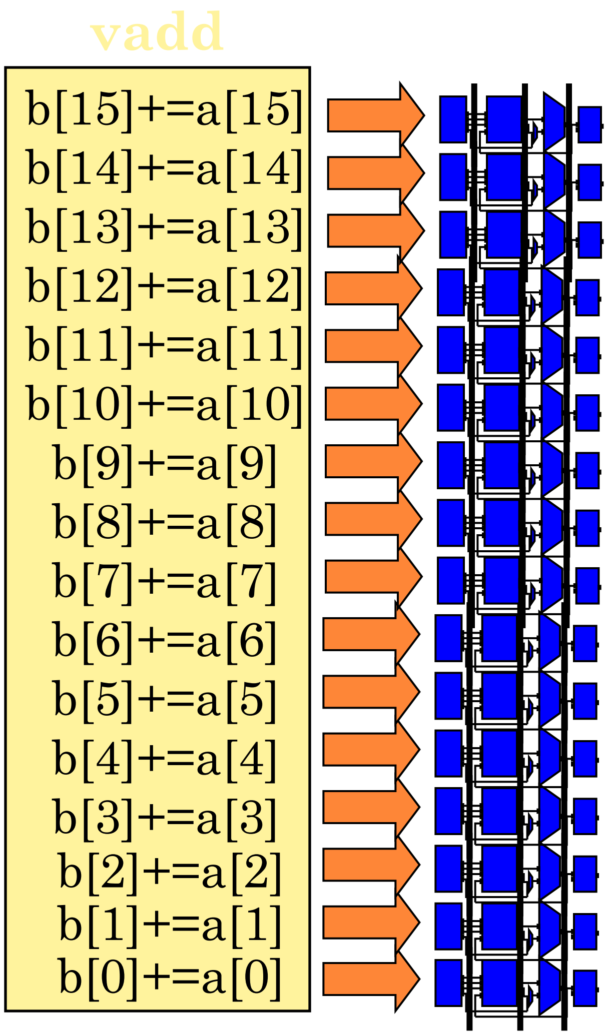
1 Vector Lane

Vector Processors (Cont'd)

```
// C code
for(i=0;i<16; i++)
  b[i]+=a[i]
```

```
// Vectorized code
set    vl,16
vload  vr0,b
vload  vr1,a
vadd   vr0,vr0,vr1
vstore vr0,b
```

Each vector instruction holds many units of independent operations

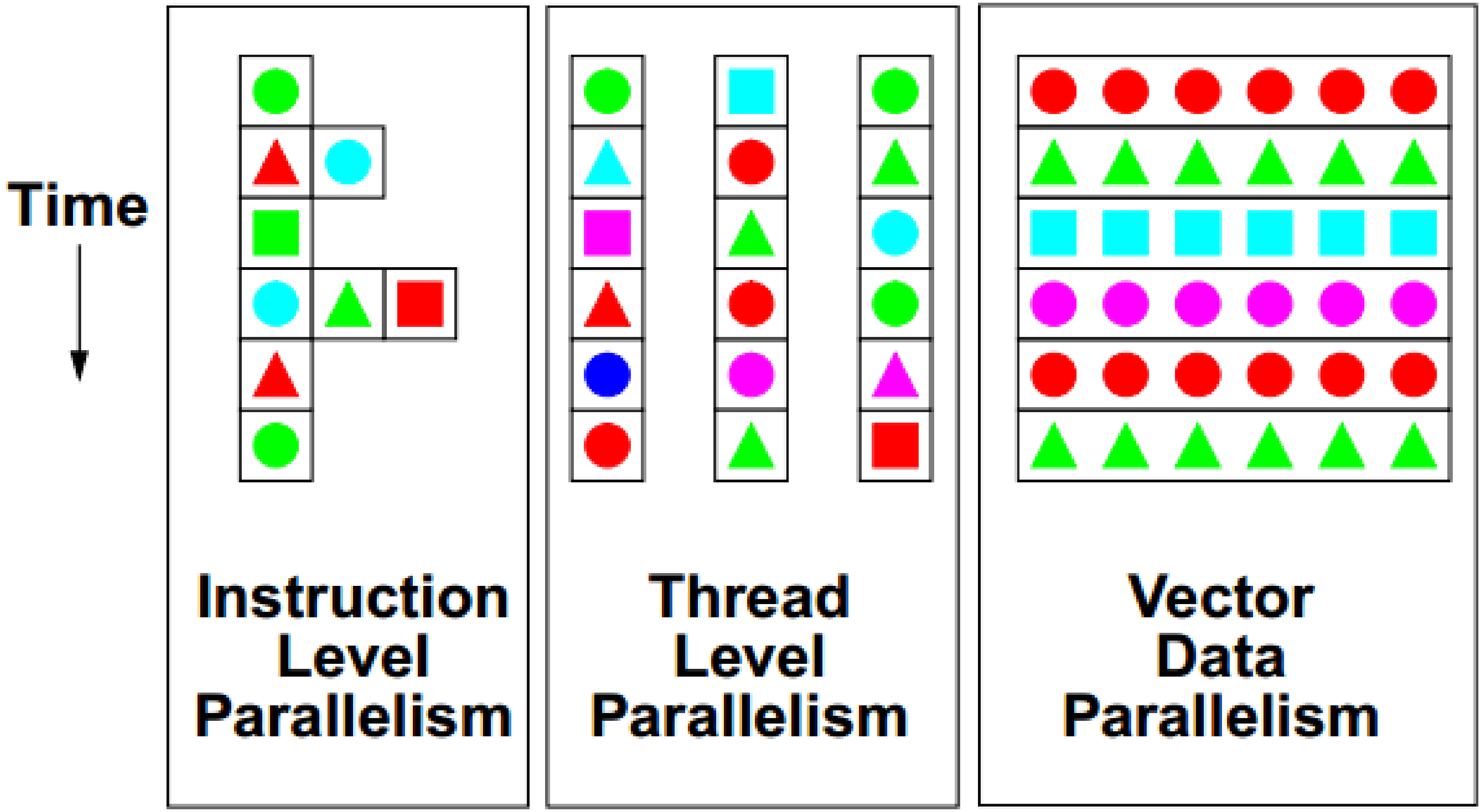


16 Vector Lanes

16x speedup

Vector Processors (Cont'd)

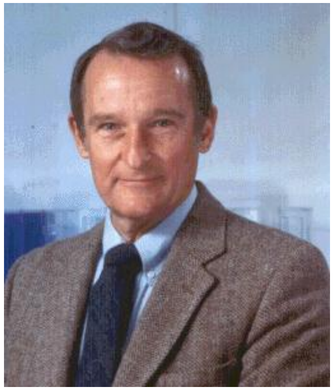
- The three major categories to exploit parallelism:
  - Instruction-level parallelism (ILP)
    - Multiple instructions from one instruction stream are executed simultaneously
  - Thread-level parallelism (TLP)
    - Multiple instruction streams are executed simultaneously
  - Vector data parallelism (DP)
    - The same operation is performed simultaneously on arrays of elements


Time
Instruction Level Parallelism
Thread Level Parallelism
Vector Data Parallelism

Vector Processing & Parallel Processing

- A vector processor is a CPU design wherein the instruction set includes operations that can perform mathematical operations on multiple data elements simultaneously
- This is in contrast to a scalar processor which handles one element at a time using multiple instructions
- Parallel computing is a form of computation in which many calculations are carried out simultaneously
- Large problems can often be divided into smaller ones which are then solved concurrently in parallel

Basic Vector Architecture



- Seymour Cray
  - The Father of Vector Processing and Supercomputing
- In 1951 he started working in computers when he joined Electronic Research Associates for producing early digital computers.
- His first work was in very first general-purpose scientific systems built
- After year of work he became an expert on digital computer technology
- During his six years with ERA he designed several other systems

Basic Vector Architecture (Cont'd)

- In 1957 left ERA with four other individuals to form Control Data Corporation
- When Cray was 34 he considered as a genius in designing high performance computers
- By 1960 he had completed his work on the design of the first computer to be fully transistorized
- He also had already started his design on the CDC 6600 the first supercomputer
  - The system would use three-dimensional packaging and an instruction set known as RISC

Basic Vector Architecture (Cont'd)

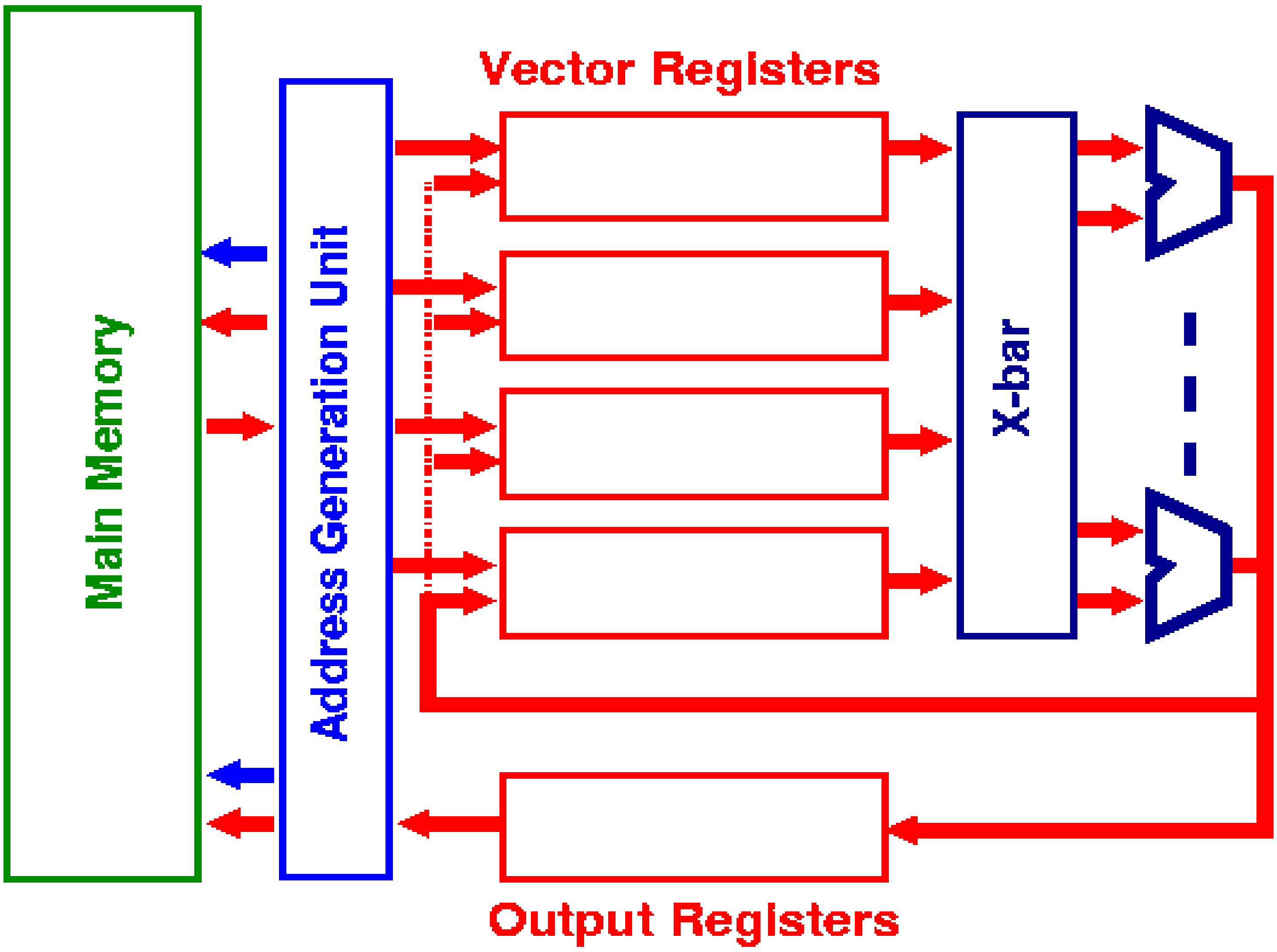
- The 8600 was the last system that Cray worked on while at CDC
- In 1968 he realized that he would need more than just higher clock speed if he wanted to reach his goals for performance
- The concept of parallelism took root
- Cray designed the system with 4 processors running in parallel but all sharing the same memory
- In 1972 he packed away the design of the 8600 in favor of something completely new

Basic Vector Architecture (Cont'd)

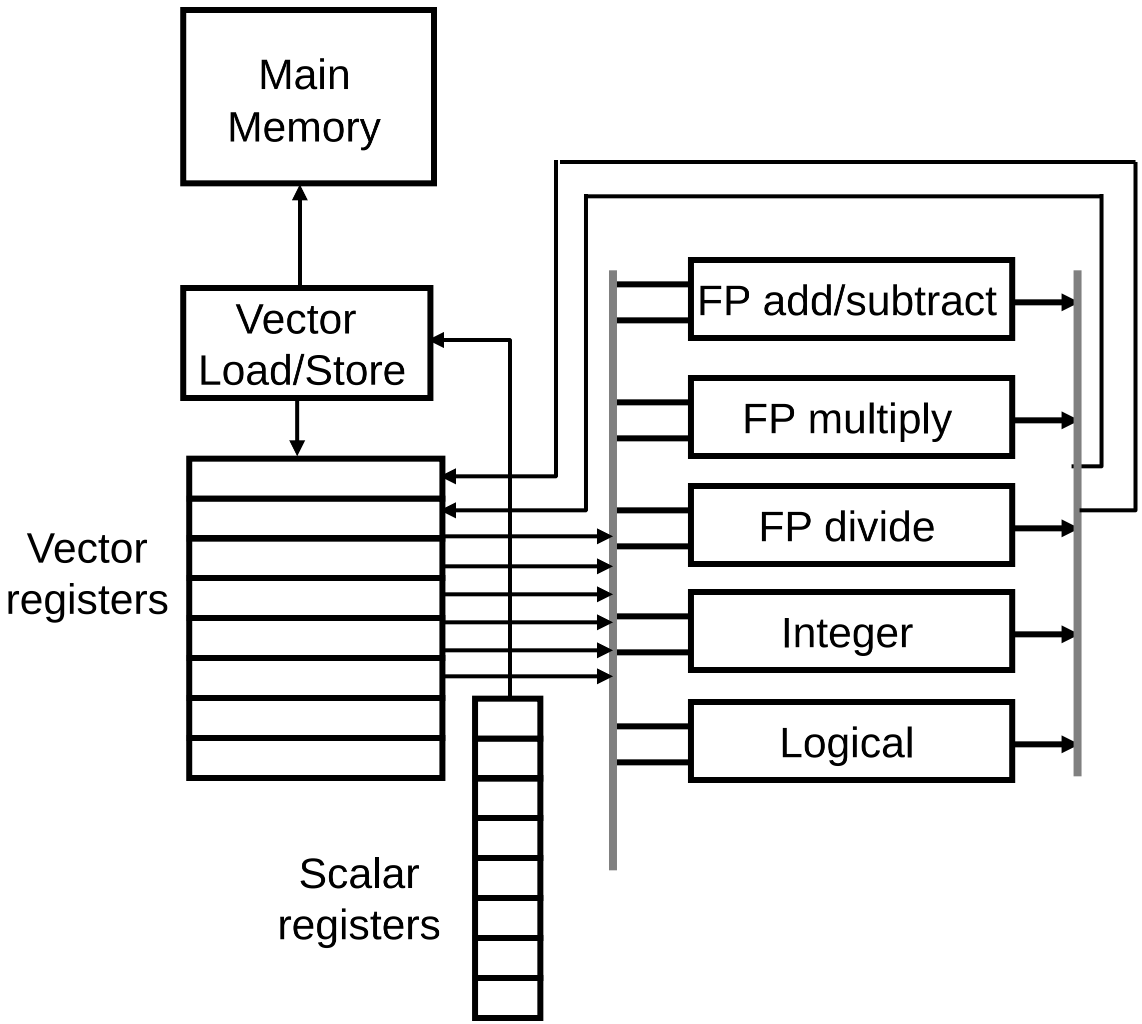
- His solution was that a greater performance could come from a uniprocessor with a different design
  - This design included Vector capabilities
- CRAY-1 the first computer produced by Cray Research which implemented with a single processor utilizing vector processing to achieve maximum performance (8 registers with 64 64-bit words in each)
- Cray-1 had separate pipelines for different instruction types allowing vector chaining. 80-240 MFlops
- Cray believed that physical designs should always be elegant, having as much importance as meeting performance goals

Basic Vector Architecture (Cont'd)

- Pipeline architecture may have a number of steps
- There is no standard when it comes to pipelining technique
- In the Cray-1 there is fourteen stages to perform vector operations

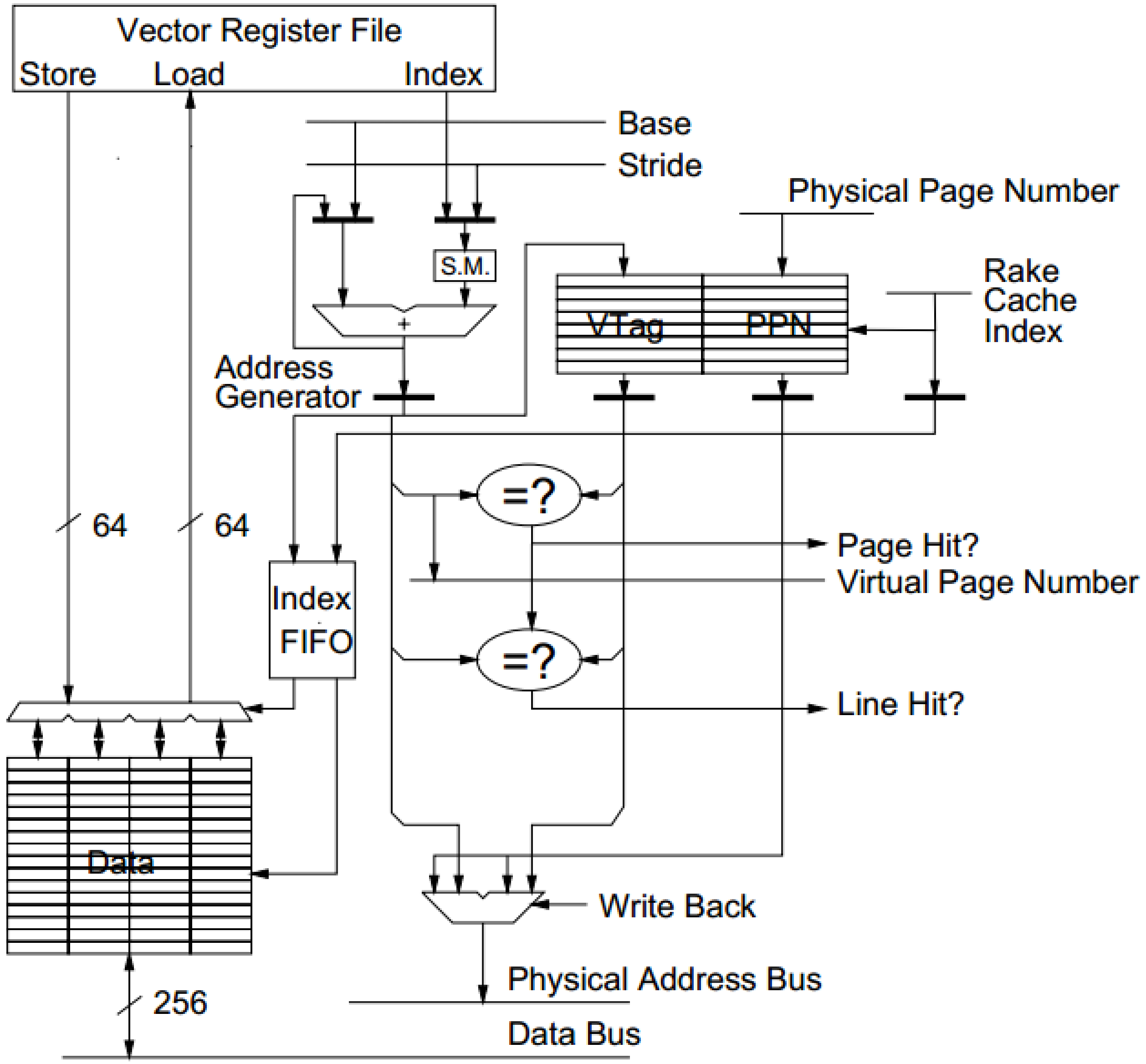


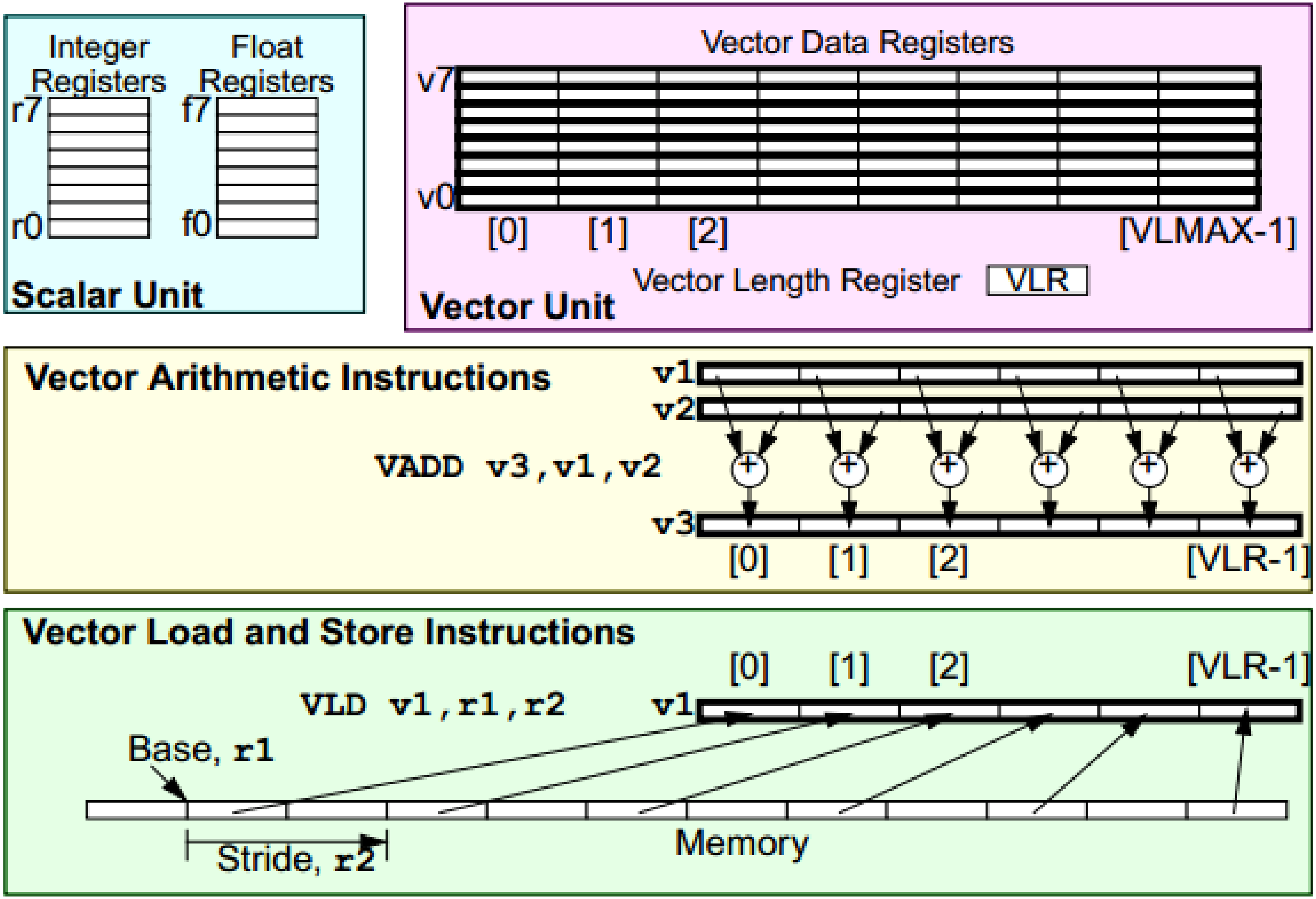
Vector Registers
Main Memory
Address Generation Unit
X-bar
Output Registers


Main Memory
Vector Load/Store
Vector registers
Scalar registers
FP add/subtract
FP multiply
FP divide
Integer
Logical

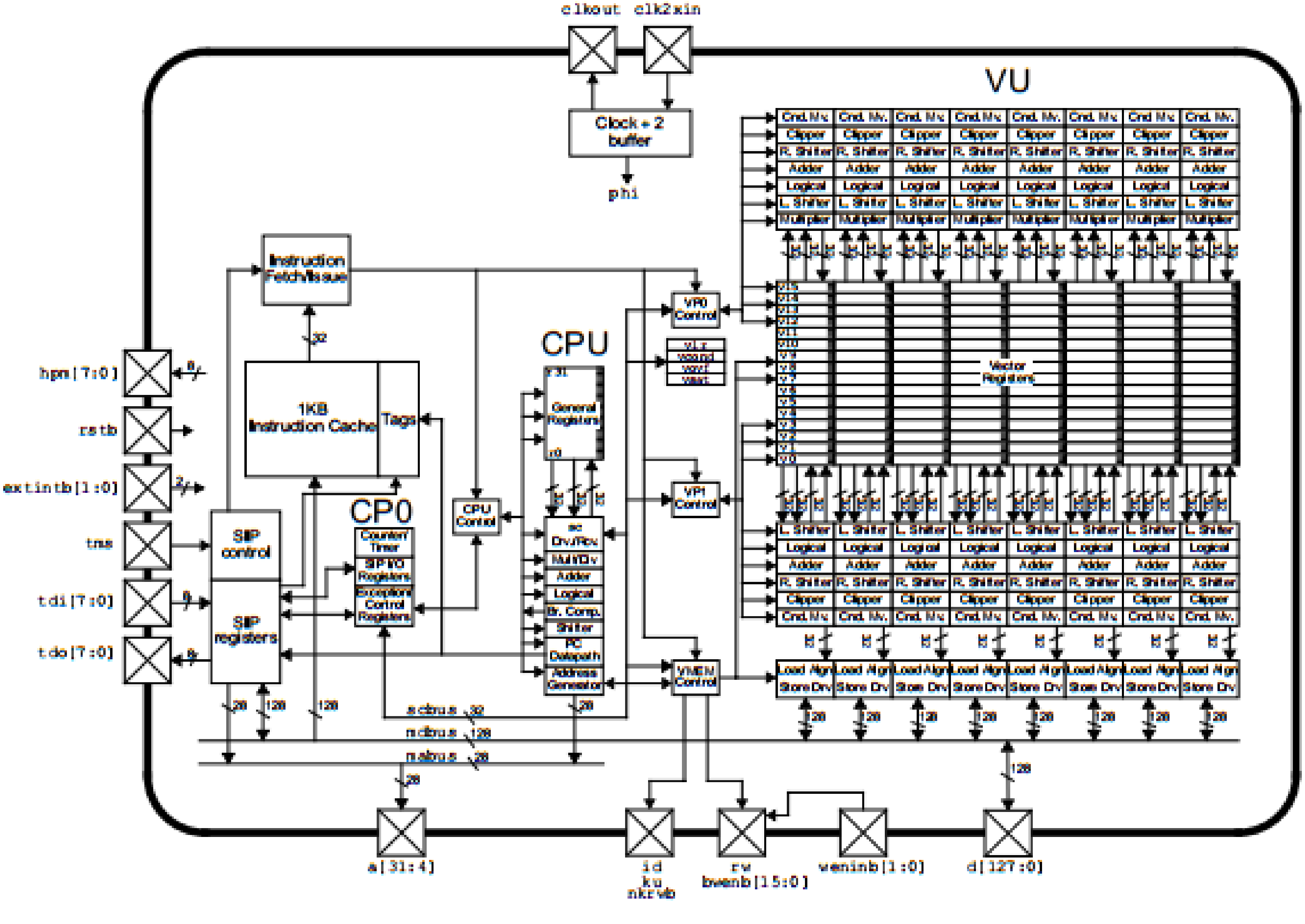
Basic Vector Architecture (Cont'd)

- Data is read into vector registers which are FIFO queues
  - Can hold 50-100 floating point values
- The instruction set:
  - Loads a vector register from a location in memory
  - Performs operations on elements in vector registers
  - Stores data back into memory from the vector registers
- A vector processor is easy to program parallel SIMD computer
- Memory references and computations are overlapped to bring about a tenfold speed increase


Vector Register File
Store
Load
Index
Base
Stride
Physical Page Number
S.M.
VTag
PPN
Rake
Cache
Index
Address
Generator
64
64
=?
Page Hit?
Virtual Page Number
Index
FIFO
=?
Line Hit?
Data
Write Back
Physical Address Bus
256
Data Bus



Integer Registers
Float Registers
r7
r0
f7
f0
Scalar Unit
Vector Data Registers
v7
v0
[0] [1] [2] [VLMAX-1]
Vector Length Register VLR
Vector Unit
Vector Arithmetic Instructions
VADD v3,v1,v2
v1
v2
v3
[0] [1] [2] [VLR-1]
Vector Load and Store Instructions
VLD v1,r1,r2
[0] [1] [2] [VLR-1]
v1
Base, r1
Stride, r2
Memory


clkout
clk2xin
VU
Clock ÷ 2 buffer
phi
Instruction Fetch/Issue
VP0 Control
VP1 Control
CPU
General Registers
hpm[7:0]
rstb
extintb[1:0]
tms
tdi[7:0]
tdo[7:0]
1KB Instruction Cache
Tags
CP0
Counter/ Timer
SIP I/O Registers
Exception Control Registers
CPU Control
SIP control
SIP registers
Vector Registers
Logical
Adder
Clipper
Shifter
Multiplier
L. Shifter
R. Shifter
VMEM Control
Load Align Store Drv
scbus
ncbus
nabus
a[31:4]
rw
bwenb[15:0]
d[127:0]

Processor (year)	Clock rate (MHz)	Vector registers	Elements per register (64-bit elements)	Vector arithmetic units	Vector load-store units	Lanes
Cray-1 (1976)	80	8	64	6: FP add, FP multiply, FP reciprocal, integer add, logical, shift	1	1
Cray X-MP (1983) Cray Y-MP (1988)	118 166	8	64	8: FP add, FP multiply, FP reciprocal, integer add, 2 logical, shift, population count/parity	2 loads 1 store	1
Cray-2 (1985)	244	8	64	5: FP add, FP multiply, FP reciprocal/sqrt, integer add/shift/population count, logical	1	1
Fujitsu VP100/VP200 (1982)	133	8–256	32–1024	3: FP or integer add/logical, multiply, divide	2	1 (VP100) 2 (VP200)
Hitachi S810/S820 (1983)	71	32	256	4: FP multiply-add, FP multiply/divide-add unit, 2 integer add/logical	3 loads 1 store	1 (S810) 2 (S820)
Convex C-1 (1985)	10	8	128	2: FP or integer multiply/divide, add/logical	1	1 (64 bit) 2 (32 bit)
NEC SX/2 (1985)	167	8 + 32	256	4: FP multiply/divide, FP add, integer add/logical, shift	1	4
Cray C90 (1991) Cray T90 (1995)	240 460	8	128	8: FP add, FP multiply, FP reciprocal, integer add, 2 logical, shift, population count/parity	2 loads 1 store	2
NEC SX/5 (1998)	312	8 + 64	512	4: FP or integer add/shift, multiply, divide, logical	1	16
Fujitsu VPP5000 (1999)	300	8–256	128–4096	3: FP or integer multiply, add/logical, divide	1 load 1 store	16
Cray SV1 (1998) SV1ex (2001)	300 500	8	64	8: FP add, FP multiply, FP reciprocal, integer add, 2 logical, shift, population count/parity	1 load-store 1 load	2 8 (MSP)
VMIPS (2001)	500	8	64	5: FP multiply, FP divide, FP add, integer add/shift, logical	1 load-store	1

Vector Instruction

- Instructions available depends on what components the processor contains.
- For a case, we take the VMIPS processor developed in 2001, that has the following components:
  - Floating Point Multiply
  - Floating Point Divide
  - Floating Point Add
  - Integer Add/Shift
  - Logical
- Integer Add/Shift exploits the additive nature of multiplication and the built-in Shift-Add procedure implemented in processors.

Instructions in VMIPS

Instr.	Operands	Comment	Operation
ADDV.D	V1,V2,V3	vector + vector	V1=V2+V3
ADDSV.D	V1,F0,V2	scalar + vector	V1=F0+V2
MULV.D	V1,V2,V3	vector x vector	V1=V2xV3
MULSV.D	V1,F0,V2	scalar x vector	V1=F0xV2
SUBV.D	V1,V2,V3	vector - vector	V1=V2-V3
SUBSV.D	V1,F0,V2	scalar – vector	V1=F0-V2
SUBVS.D	V1,V2,F0	vector - scalar	V1=V2- F0
DIVV.D	V1,V2,V3	vector / vector	V1=V2/V3
DIVSV.D	V1,F0,V2	scalar / vector	V1=F0/V2
DIVVS.D	V1,V2,F0	vector / scalar	V1=V2/F0

INSTRUCTIONS IN VMIPS(CONT'D)

Instr.	Operands Comment	Operation
LV	V1,R1	Load vector register V1 from memory starting at address R1
SV	R1,V1	Store vector register V1 into memory starting at address R1
LVWS	V1,(R1,R2)	Load V1 from address at R1 and stride at R2 as R1+i*R2
SVWS	(R1, R2), V1	Store with Stride
LVI	V1,(R1+V2)	Load V1 with vector whose elements are at R1+ V2(i)
SVI	(R1+V2),V1	Store V1 to a vector whose elements are R1+V2(i)
CVI	V1,R1	Create an index vector by storing values i*R1 into V1.

Logical operation

- S- -V.D and S- -VS.D

Here - - is replaced by the corresponding Logical Operators as per need.

EQ – Equal to

NE – Not Equal

GT – Greater Than

LT – Less Than

GE – Greater than or Equal to

LE – Less than or Equal to

Compare each value from S and V and put 1 in corresponding bit vector if result is True and 0 if False. Put the resulting Bit Vector in the Vector Mask Register.

VECTOR MASK REGISTER

Instr.	Operands	Operation Comment
POP	R1,VM	Count the number of 1s in Vector mask register and store result in R1
CVM		Set VMR to all 1s
MVTM	VM,F0	Move contents of F0 to VMR
MVFM	F0,VM	Move contents of VMR to F0

Vector Performance

- Vector execution time depends on:
  - Length of operand vectors
  - Data Dependencies
  - Structural Hazards
- **Initiation rate:** rate at which a vector unit consumes new operands and produces new results.
- **Convoy:** set of vector instructions that can begin execution in same clock (Assuming no Data dependencies or structural hazards since all instructions in a convoy begin execution at the same clock period)
- **Chime:** approx. time to execute a convoy

EXAMPLE

```
LV V1,Rx              ;load vector X
MULVS.D V2,V1,F0      ;vector-scalar multiply
LV V3,Ry              ;load vector Y
ADDV.D V4,V2,V3       ;add
SV Ry,V4              ;store the result
```

1. First LV is in a separate convoy since MULVS depends on its execution
2. MULVS and second LV can be in same convoy since they are independent.
3. ADDV is in a separate convoy
4. SV is in the fourth convoy since it needs ADDV to complete.

No. of chimes for completion is 4 and 2 Floating point operations take place in that time. So rate is 2 FLOPS per cycle. Assuming 10 elements in vector, no. of clock cycles needed is 40.

Role of startup time

- Startup time - Time latency from pipelining of vector operation. Assuming vector length of n,

Unit	Startup Overhead(Cycles)
Load and store unit	12
Multiply Unit	7
Add Unit	6

Convoy	Starting time	First result time	Last result time
LV	0	12	11+n(12-1+n)
MULVS.D LV	12+n	12+n+12	23+2n
ADDV.D	24+2n	24+2n+6	29+3n
SV	30+3n	30+3n+12	41+4n

VMIPS Execution Time

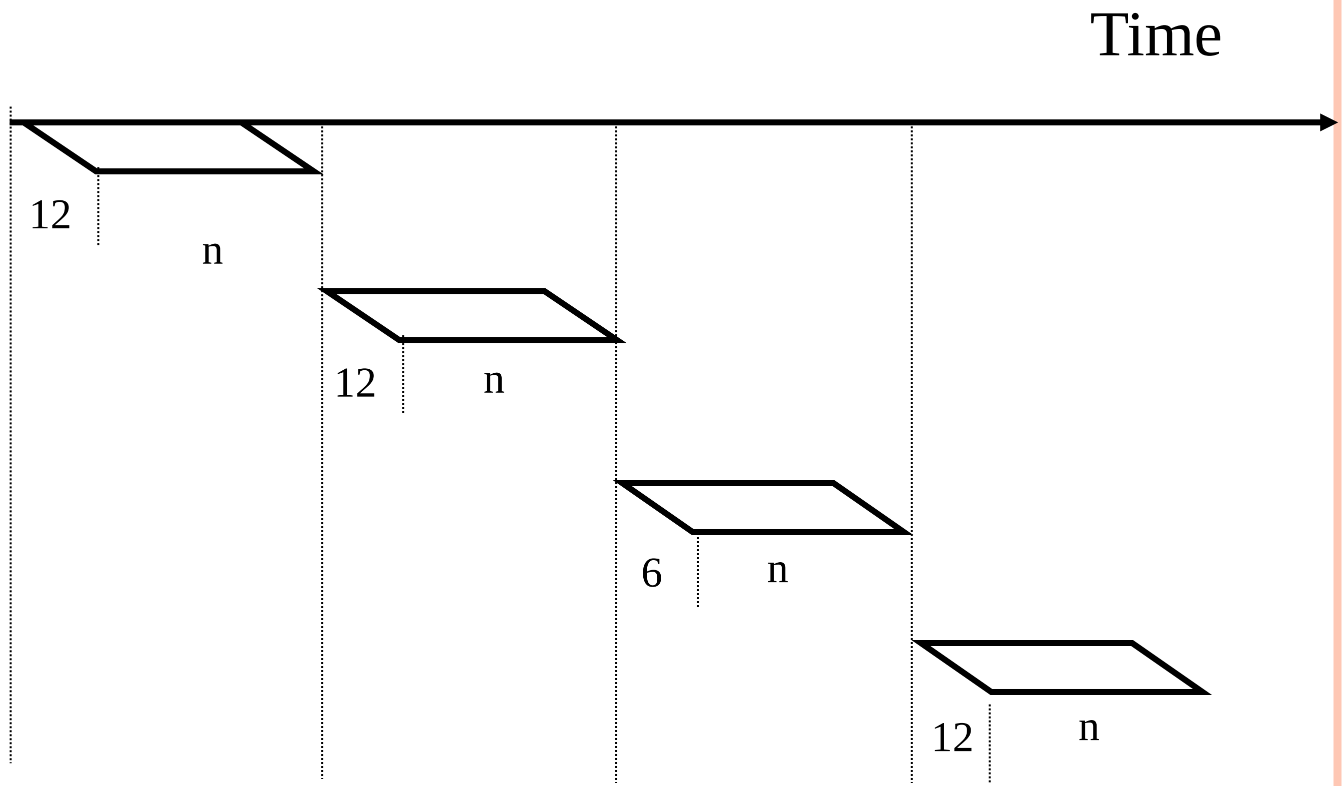
1: LV V1,Rx

2: MULV V2,F0,V1

LV V3,Ry

3: ADDV V4,V2,V3

4: SV Ry,V4



Memory Units

- Start up time for a load is time needed to get first word from memory to register. If rest of the vector can be supplied without any need to stall, then Initiation time = Rate at which new words are fetched and stored.
- Startup time is longer for large LSU.
- Memory banks are better than normal interleaving because:
  - Multiple loads or store per clock can be done. If Load and store operations are in a single convoy with different vectors, banks are better.
  - Ability to store or load data words that are not sequential.
  - Sharing of memory between multiple processors since each processor will generate its own stream of addresses.

Vector length

- VMIPS has a vector length of 64. But in real world applications vector lengths are not exactly 64. For example, adding just first n elements of a vector.
- Vector Length register is used for this purpose.
- VLR controls the length of any vector operation by defining their length.
- Its value cannot be greater than the length of the vector registers. (64 in this case)
- This works when the length of data is less than the Maximum Vector Length of a processor. But in real world applications, data in vectors in memory can be greater than the MVL of the processor.
- In this case, we use a technique called Strip Mining.

Strip mining

- Splitting data such that each vector operation is done for a size less than or equal to MVL.
- Done by a simple loop with MOD operator as control point.

```
i = 0;
VL = n mod MVL;
for (j=0; j<n/MVL; j++){
        for(i<VL; i++)
        {Y(i)=a*X(i)+Y(i)}
VL = MVL;}
```

Improving performance :Conditional statements

- Conditional statements affect vectorization.

```
for (i=0;i<100;i++)
        if (a[i] !=0)
                a[i] = a[i] – b[i];
end
```

- Cannot be vectorized normally due to presence of if statement.
- Can be overcome by using the VMR

```
LV          V1,Ra ;         load vector A into V1
LV          V2,Rb ;         load vector B
L.D         F0,#0 ;         load FP zero into F0
SNEVS.D     V1,F0 ;         sets VM(i) to 1 if V1(i)!=F0
SUBV.D      V1,V1,V2 ;      subtract under vector mask
SV          Ra,V1 ;         store the result in A
```

IMPROVING PERFORMANCE

- If we think of the registers used as not one big block but group of individual registers, we can pipeline data to improve performance.
- For example,

```
MULV.D  V1,V2,V3
ADDV.D  V4,V1,V5
```

needs to be in separate convoys if we approach register as a whole block.

- If we consider it as group of individual registers, each containing one value, then second ADDV can start as soon as first element becomes available.
- Increases convoy size and increases HW

ADVANTAGES

- Each result is independent of previous results - allowing high clock rates.
- A single vector instruction performs a great deal of work - meaning less fetches and fewer branches (and in turn fewer mispredictions).
- Vector instructions access memory a block at a time which results in very low memory latency.
- Less memory access = faster processing time.
- Lower cost due to low number of operations compared to scalar counterparts.

DISADVANTAGES

- Works well only with data that can be executed in highly or completely parallel manner.
- Needs large blocks of data to operate on to be efficient because of the recent advances increasing speed of accessing memory.
- Severely lacking in performance compared to normal processors on scalar data.
- High price of individual chips due to limitations of on-chip memory.
- Increased code complexity needed to vectorize the data.
- High cost in design and low returns compared to superscalar microprocessors.

APPLICATIONS

- Useful in applications that involve comparing or processing large blocks of data.
- Multimedia Processing (compress., graphics, audio synth, image proc.)
- Standard benchmark kernels (Matrix Multiply, FFT, Convolution, Sort)
- Lossy Compression (JPEG, MPEG video and audio)
- Lossless Compression (Zero removal, RLE, Differencing, LZW)
- Cryptography (RSA, DES/IDEA, SHA/MD5)
- Speech and handwriting recognition
- Operating systems/Networking (memcpy, memset, parity, checksum)
- Databases (hash/join, data mining, image/video serving)

Conclusion

- The Vector machine is faster at performing mathematical operations on larger vectors.
- The Vector processing computer's vector register architecture makes it better able to compute vast amounts of data quickly.
- While Vector Processing is not widely popular today, it still represents a milestone in supercomputing achievement.
- It is still in use today in home PC's as SIMD units which augment the scalar CPU when necessary (usually GPUs).
- Since scalar processors designed can also be used for general applications their cost per unit is reduced drastically. Such is not the case for vector processors/supercomputers.
- Vector processors will continue to have a future in Large Scale computing and certain applications but can never reach the popularity of Scalar microprocessors.

Thank You

References

- http://www.eecs.berkeley.edu/~krste/thesis.pdf
- http://www.geo.fmi.fi/~pjanhune/papers/
- www.eecg.toronto.edu/~yiannac/docs/cases08.**ppt**
- www.cct.lsu.edu/~scheinin/Parallel/**VectorProcessors**.ppt
- http://www.ece.uah.edu/~milenka
- www.docin.com/p-380535422.html
- What is difference between vector processing and parallel processing?i have xp so what type of processing i have? | Answerbag http://www.answerbag.com/q_view/1833623#ixzz2Bc4wyjwI
- Vector Processors by Mark Smotherman, Assoicate Professor,School of Computing, Clemson University
- Vector Processing by Aleksandar Milenkovic, Electrical and Computer Engineering, University of Alabama in Huntsville
- Vector Processing by David A. Patterson and Jan Rabaey
- Vector Processors by Ryan McPherson
- Vector Processors by Pratyusa Manadhata, Vyas Sekar
- Vector Processing by Ben Helmer, Matt Sagerstrand, Daniel Yingling
- Vector Processors by Brian Anderson, Mike Jutt, Ryan Scanlon