



COSC 6365
Lecture 12
2008-02-21

CS@UH

Introduction to HPC

Lecture 12

Lennart Johnsson
Dept of Computer Science
Director TLC²



COSC 6365
Lecture 12
2008-02-21

CS@UH

Pipelining – Vectorization - SIMD

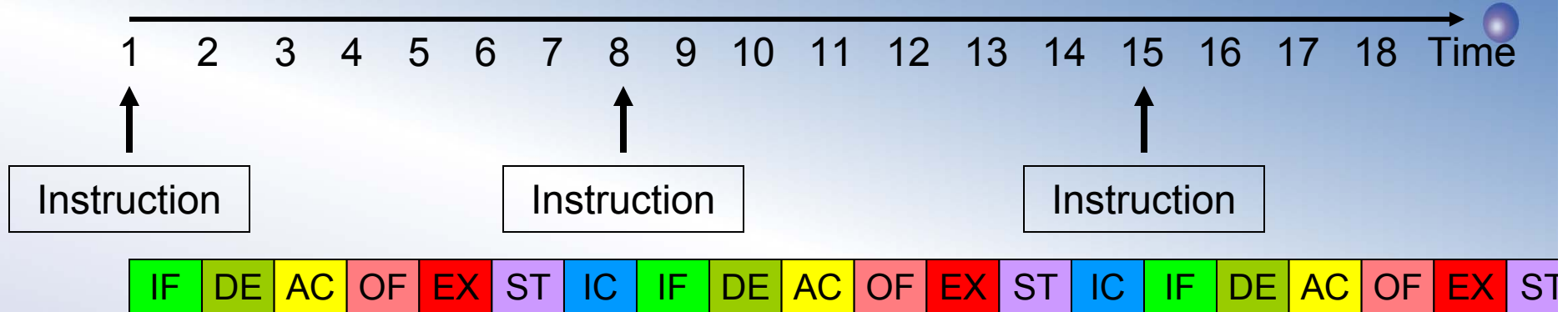
- Recall typical instruction functions
 1. Fetch instruction
 2. Decode instructions
 3. Compute operand addresses
 4. Fetch operands
 5. Execute operation
 6. Store result
 7. Increment program counter

If each step takes unit time, then an instruction is initiated every 7th time step.

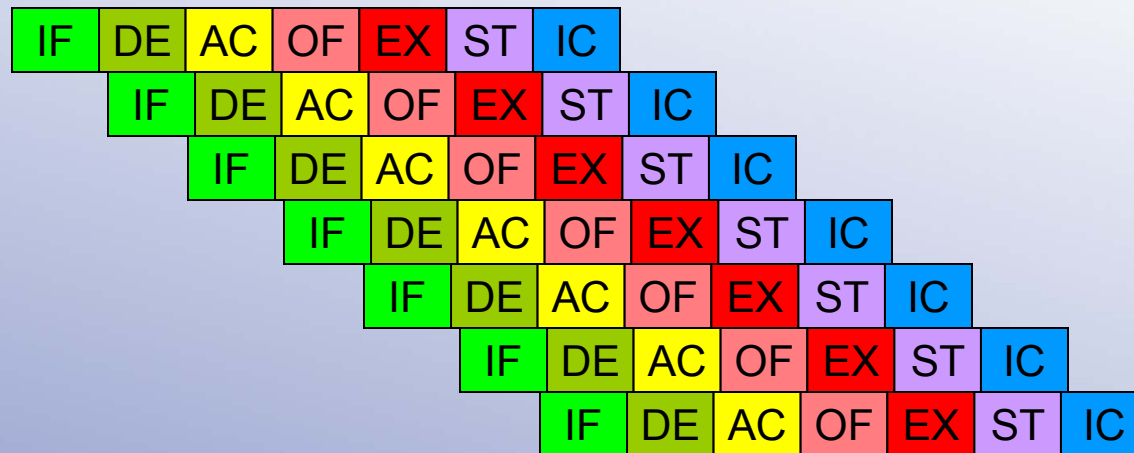
In reality some steps are more complex than others and takes longer time to execute.



Instruction execution



Pipeline



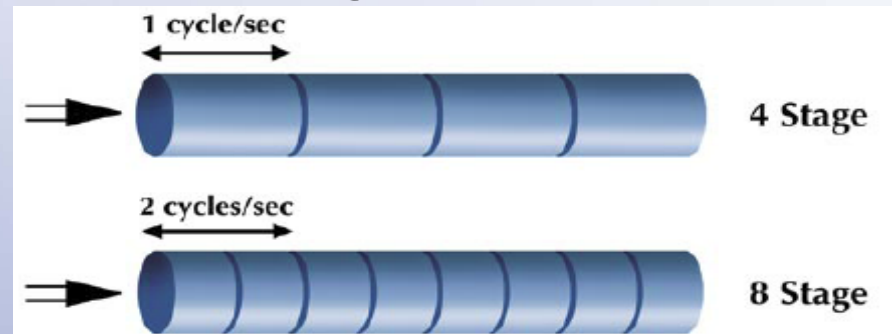


COSC 6365
Lecture 12
2008-02-21



Pipelining

- The Intel Xeon architecture optimizes frequency at the expense of pipeline execution efficiency. The Netburst architecture has 31 pipeline stages with clock rates up to 3.8 GHz. (The Itanium2 has 8 stages and clock rates up to 1.6 GHz))
- The AMD Opteron architecture optimizes pipeline execution efficiency at the expense of clock frequency. The pipeline has 12 stages. Clock rates up to 2.8 GHz





COSC 6365
Lecture 12
2008-02-21

CS@UH

Pipeline Concepts

- Latency
 - Time elapsed between initiation and completion of an operation
- Depth
 - Number of stages in the pipeline
- Initiation rate
 - The rate at which the first pipeline stage can service requests
- Completion rate
 - The rate at which the slowest stage can service requests
- Throughput
 - Same as Completion rate
- Startup time
 - The time required for the pipeline to reach a steady-state. It is the same as latency.



Table 2-1. Itanium® 2/ Itanium Processors Operation Latencies

		Consumer									
		Qual. Pred.	Branch Pred.	ALU	Load Store Addr	Multi-media	Store Data	Fmac	Fmisc	getf	setf
Producer	Adder: add, cmp, tbit, addp4, shladd, shladdp4, sum, logical ops, 64-bit immed. moves, movl, post-inc ops (includes post-inc stores, loads, lfetches)	n/a	n/a	1	1/(1-2) ¹	3	1	n/a	n/a	n/a	1
	Multimedia	n/a	n/a	3	3	2	3	n/a	n/a	n/a	3
	getf	n/a	n/a	5/9	6/9	6/9	5/9	n/a	n/a	n/a	6/9
	setf	n/a	n/a	n/a	n/a	n/a	6/2	6/2	6/2	6/2	n/a
	Fmac: fma, fms, fnma, fpma, fpms, fpnma, fadd, fnmpy, fsub, fpmpy, fpnmpy, fmpy, fnorm, xma, frcpa, frcpa, frsqta, fpsqta, fcvt, fpcvt	n/a	n/a	n/a	n/a	n/a	4/5	4/5	4/5	4/5	n/a
	Fmisc: fselect, fcmp, fclass, fmin, fmax, famin, famax, fmin, fmax, fpamin, fpcmp, fmerge, fmix, fsxt, fpack, fswap, fand, fandcm, for, fxor, fpmerge, fneg, fnegabs, fpabs, fpneg, fpnegabs	n/a	n/a	n/a	n/a	n/a	4/5	4/5	4/5	4/5	n/a
	INT side predicate write: cmp, tbit, tnat	1	0	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	FP side predicate write: fcmp	2	1/1	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	FP side predicate write: frcpa, frcpa, frsqta, fpsqta	2	2	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	Int Load ²	n/a	n/a	N	N+1	N+1	N	N	N	N	N
	FP Load ³	n/a	n/a	M+1	M+2	M+2	M+1	M+1	M+1	M+1	M+1
	IEU2: move_from_br, alloc	n/a	n/a	2	2	3	2	n/a	n/a	n/a	2
	Move to/from cr, ar ⁴	n/a	n/a	C	C	C	C	n/a	n/a	n/a	C
	Move to pr	1	0	2	2	3	2	n/a	n/a	n/a	n/a
	Move indirect ⁵	n/a	n/a	D	D	D	D	n/a	n/a	n/a	D

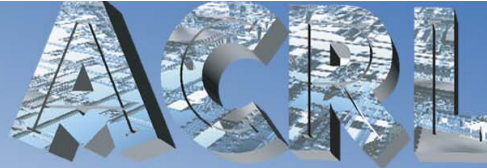
1. On the Itanium® processor, the address computation instruction must be in an M-slot type to avoid an extra cycle of latency.
2. N depends upon which level of cache is hit. For the Itanium processor, N=2 for L1D, N=6 for L2, N=21 for L3. For the Itanium 2 processor, N=1 for L1D, N=5 for L2, N=(12-15) for L3. These are minimum latencies.
3. M depends upon which level of cache is hit. For the Itanium processor, M=8 for L2 and M=24 for L3. For the Itanium 2 processor, M=5 for L2 and M=(12-15) for L3. These are minimum latencies. The "+" entries indicate one cycle is needed for format conversion.
4. Best-case values of C range from 2 to 35 cycles depending upon registers accessed. EC and LC accesses are 2 cycles. FPCR and CR accesses are 10-12 cycles.
5. Best-case values of D range from 6 to 35 cycles depending upon indirect registers accessed; lregs pkr and rr accesses are faster at 6 cycles.



COSC 6365
Lecture 12
2008-02-21

CS@UH

- <http://www.intel.com/design/processor/manuals/248966.pdf>
- <http://download.intel.com/design/Itanium2/manuals/25111003.pdf>



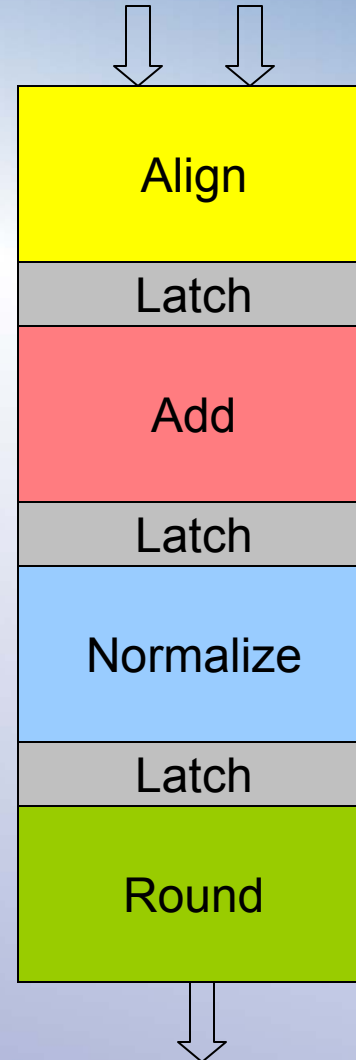
ADVANCED COMPUTING RESEARCH LABORATORY

COSC 6365
Lecture 12
2008-02-21

CS@UH

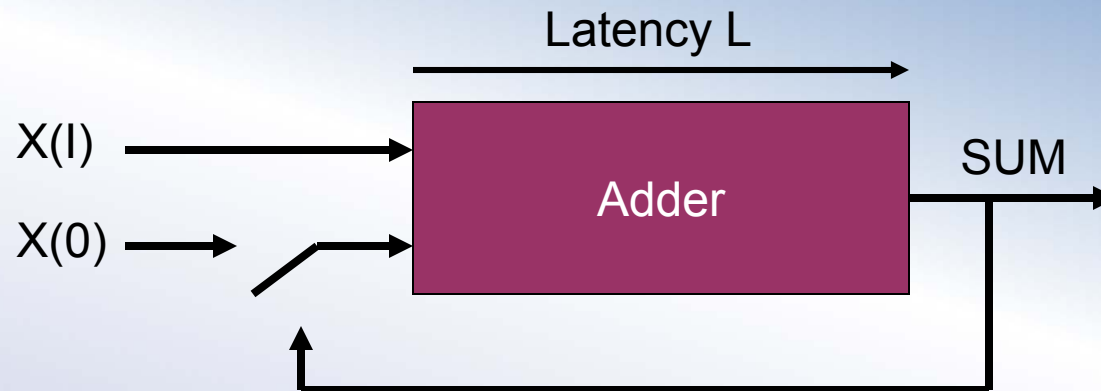
Arithmetic Units

Conceptual Adder
Pipeline





Reduction

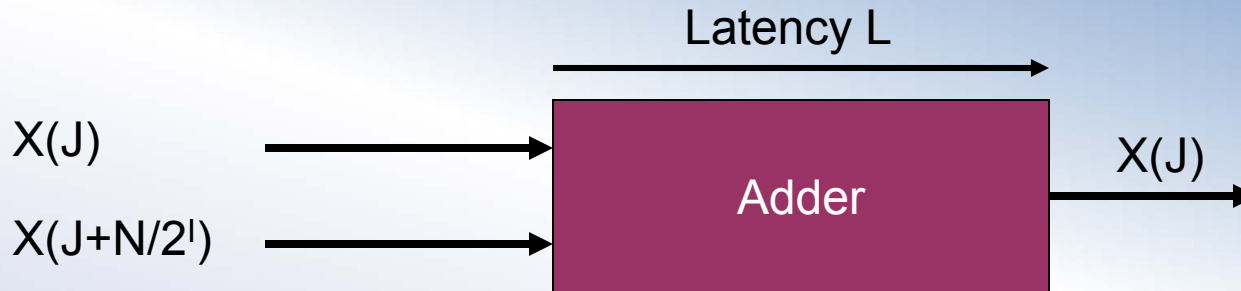


```
SUM = X(0)  
FOR I = 1 TO N-1 DO  
    SUM = SUM + X(I)  
ENDFOR
```

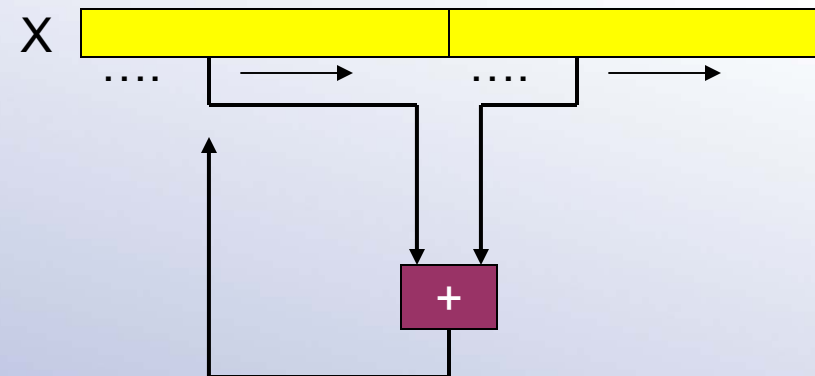
Time: $L \cdot N$



Restructured for pipelining

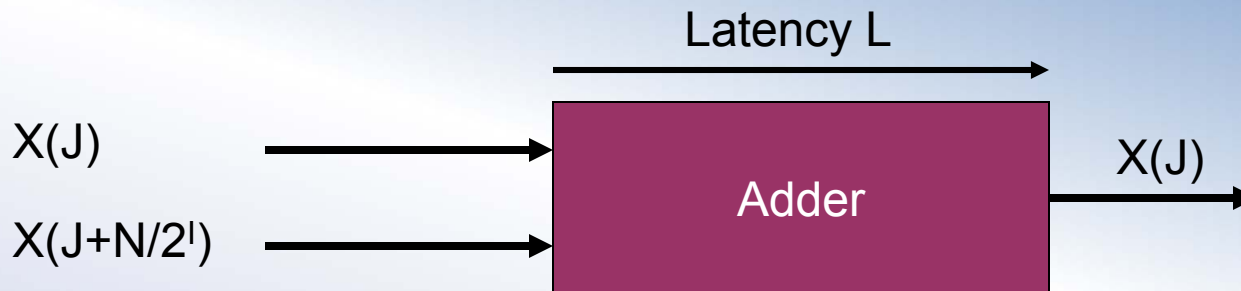


```
FOR I = 1 TO  $\log_2 N - 1$  DO
  FOR J = 0 TO  $N/2^I - 1$  DO
     $X(J) = X(J) + X(J + N/2^I)$ 
  ENDFOR
ENDFOR
```





Restructured for pipelining



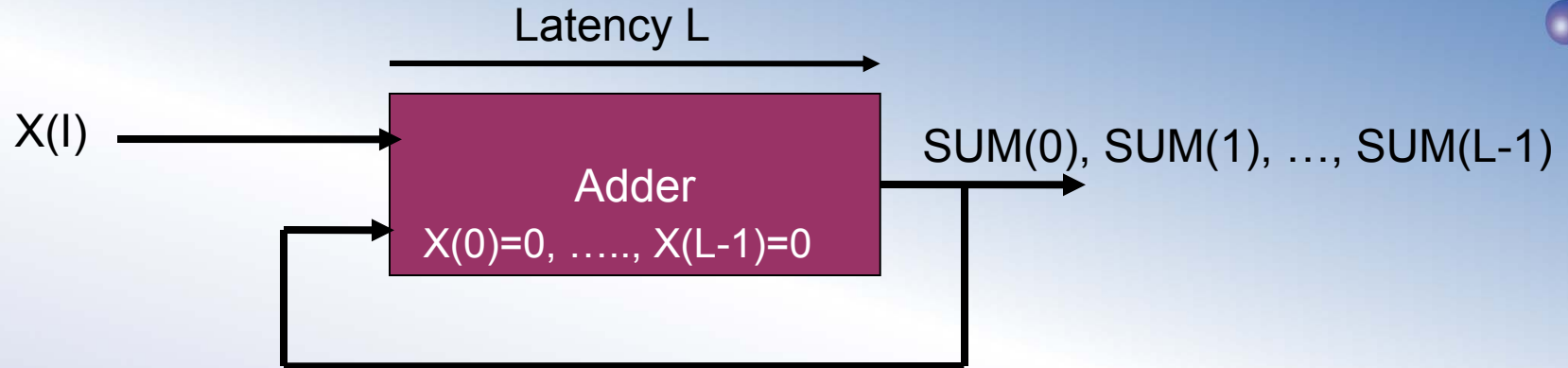
```

FOR I = 1 TO  $\log_2 N - 1$  DO
  FOR J = 0 TO  $N/2^I - 1$  DO
     $X(J) = X(J) + X(J + N/2^I)$ 
  ENDFOR
ENDFOR
  
```

Time: I=1	$L + (N/2) - 1$
I=2	$L + (N/4) - 1$
.....	
I= $\log_2 N$	L
	$N + (L - 1) \log_2 N$

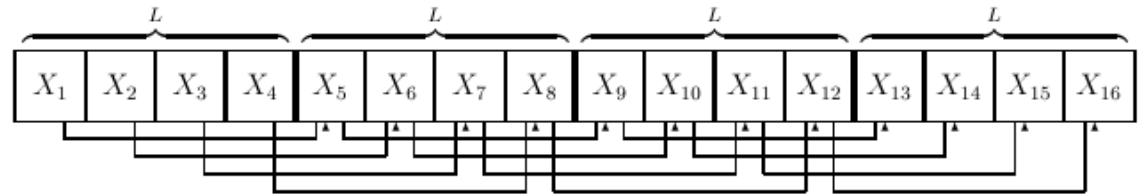


Restructured for pipelining v. 2



```

FORALL I = 1 TO L-1 DO
    SUM(I) = X(I)
ENDFORALL
FORALL J=0 TO L-1 DO
    FOR I=L TO N-1 STEP L DO
        SUM(J) = SUM(J)+X(I+J)
    ENDFOR
ENDFORALL
    
```

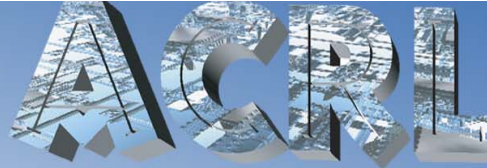


Time: Phase 1 (FORALL)
N+L
Phase 2 (not shown)
(Alg 1 or Alg 2)
L*L or L+(L-1)log₂L



Summary of vectorized addition

Alg	Arithm. ops	Cycles	Load	Store	Scalar ops	Vector ops	Vector length
1	N-1	$(N-1)*L$	N	1	N	0	0
2	N-1	$N-1+(L-1)\log_2 N$	$2(N-1)$	N-1	0	$\log_2 N$	$N/2, N/4, \dots, 1$
3	N-1	$N+2L-1+(L-1)\log_2 L$	N	1	0	$\log_2 N+1$	$N-L, L, L/2, \dots$

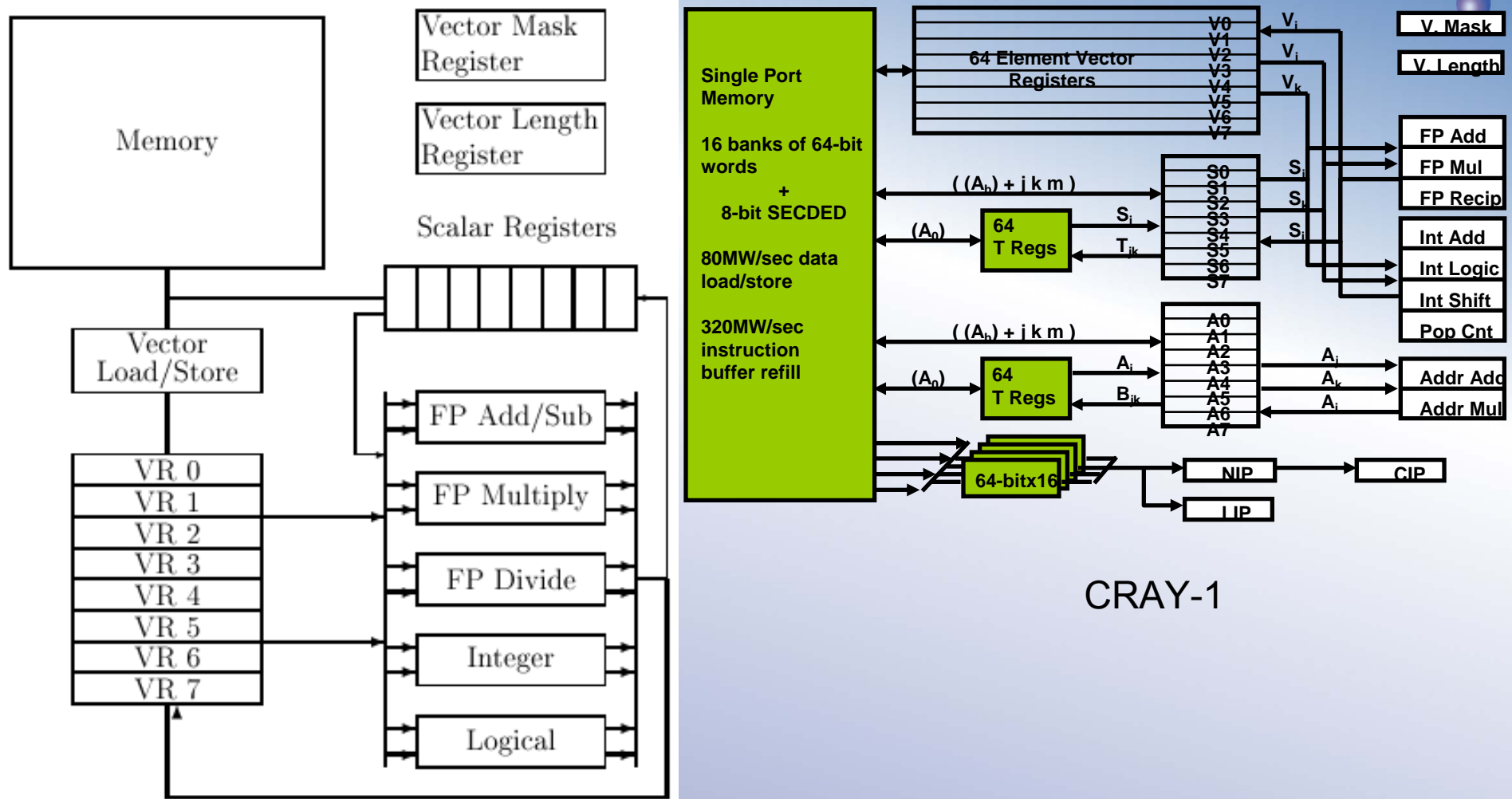


ADVANCED COMPUTING RESEARCH LABORATORY

COSC 6365
Lecture 12
2008-02-21



Generic Vector Architecture



CRAY-1



COSC 6365
Lecture 12
2008-02-21

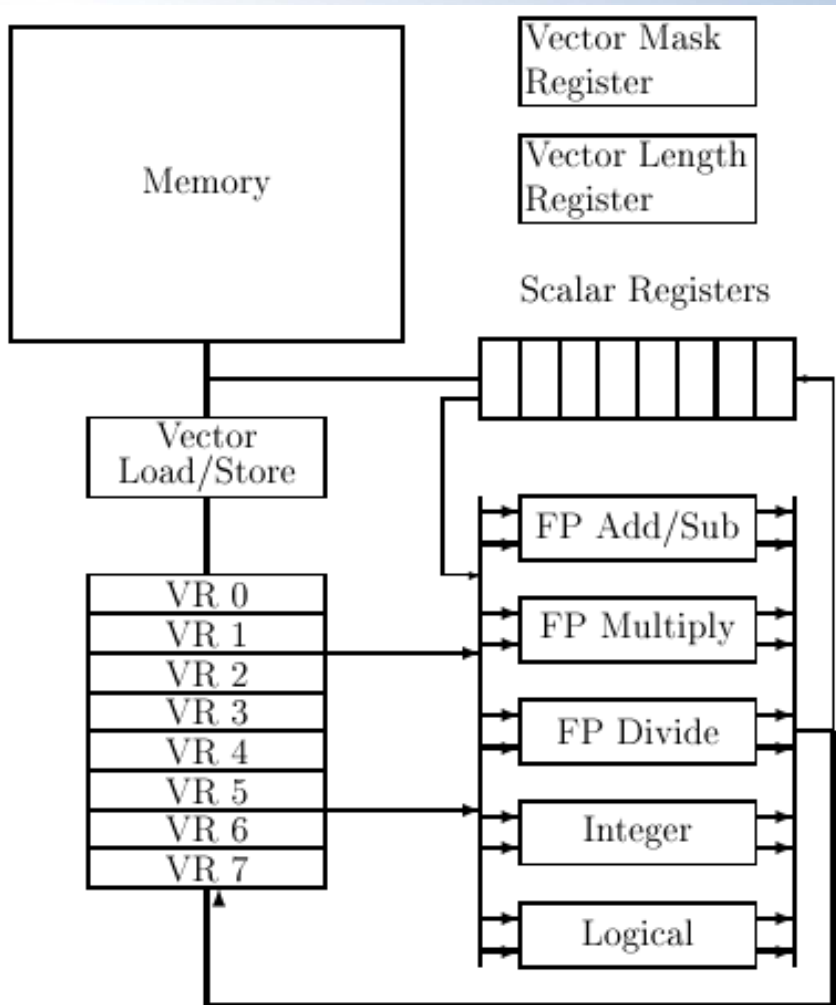
CS@UH

Vector architecture concepts

- Vector-register architecture
- Vector-memory architecture
- Vector Registers
 - Often reconfigurable (depth vs numbers)
- Vector Length
- Vector Mask
- Vector units
 - Load and Store
 - Floating-point
 - Integer
 -



Generic Vector Architecture



General Characteristics	Vector Operation Costs	
	Operation	Depth
8 Vector Registers (MVL=64) 8 Scalar Registers 1 Load/Store Path (VP-1) 2 Load Paths/1 Store Path (VP-3)	Vector Add	6
	Vector Multiply	7
	Vector Divide	20
	Vector Load	12
	Vector Store	12
	Vector Copy ³	1
	Vector Stall	4
	Vector Loop	15
	Vector Base	10



COSC 6365
Lecture 12
2008-02-21



Performance model

- Vector base
 - Accounts for the time to set up control of vector operations
- Vector loop
 - Accounts for the time for control of each loop iteration
- Strip mining
 - Restructuring of a loop into vector operations with vectors that fit in vector registers

$$T_N = T_{base} + \left\lceil \frac{N}{MVL} \right\rceil \times (T_{loop} + T_{start}) + N \times T_{elem.}$$



Vector Summation

Example 1: Computing $y = \sum_{i=0}^{N-1} x(i)$ with Algorithm 2 on VP-1

```

FOR I = 1 TO log2N-1 DO
  FOR J = 0 TO N/2I-1 DO
    X(J) = X(J)+X(J+N/2I)
  ENDFOR
ENDFOR

```

- Load a segment of X
- Load a second segment of X
- Perform a vector add on the two segments
- Store the result

General Characteristics	Vector Operation Costs	
	Operation	Depth
8 Vector Registers (MVL=64)	Vector Add	6
8 Scalar Registers	Vector Multiply	7
1 Load/Store Path (VP-1)	Vector Divide	20
2 Load Paths/1 Store Path (VP-3)	Vector Load	12
	Vector Store	12
	Vector Copy ³	1
	Vector Stall	4
	Vector Loop	15
	Vector Base	10

- Load 1st segment of X 12+VL
- Load 2nd segment of X 12+VL
- Add 6+VL
- Stall (add depends on 2nd load) 4
- Store 12+VL
- Stall (store depends on add) 4
- Total loop 50+4VL



Vector Summation

General Characteristics	Vector Operation Costs	
	Operation	Depth
8 Vector Registers (MVL=64)	Vector Add	6
8 Scalar Registers	Vector Multiply	7
1 Load/Store Path (VP-1)	Vector Divide	20
2 Load Paths/1 Store Path (VP-3)	Vector Load	12
	Vector Store	12
	Vector Copy ³	1
	Vector Stall	4
	Vector Loop	15
	Vector Base	10

Load 1 st segment of X	12+VL
Load 2 nd segment of X	12+VL
Add	6+VL
Stall (add depends on 2nd load)	4
Store	12+VL
Stall (store depends on add)	4
Total loop	50+4VL

Loop time:

$$T_{N/2^I} = 10 + \left\lceil \frac{N/2^I}{MVL} \right\rceil (15 + 50) + 4N/2^I$$

Total time:

$$\begin{aligned} T_N &= \sum_{I=1}^{\log_2 N} \left(10 + \left\lceil \frac{N/2^I}{MVL} \right\rceil (15 + 50) + 4N/2^I \right) \\ &= 10 \log_2 N + 65 \sum_{I=1}^{\log_2 N} \left\lceil \frac{N/2^I}{MVL} \right\rceil + 4(N - 1) \end{aligned}$$



Vector Summation VP-3

Example 2: Computing $y = \sum_{i=0}^{N-1} x(i)$ with Algorithm 2 on VP-3

Load first two segments of X
Perform a vector addition on the two segments

Load next two segments of X, store previous result
Perform a vector addition on the new segments

Load next two segments of X, store previous result
Perform a vector addition on the new segments

⋮

Store final result

First segment

Loop	Load	Stall	Add
15	12+NmodMVL	4	6+NmodMVL

Middle segments

Loop	Load	Stall	Add
15	12+MVL	4	6+MVL

Last segment

Loop	Store
15	12+MVL



Vector Summation VP-3

Vector load, store	12+VL
Vector add	6+VL
Stall for vector add	4 (vector add must wait for loads)
Total	$\frac{22+2VL}{}$

$$T_{N/2^I} = 10 + \underbrace{\left[\frac{N/2^I}{MVL} \right] (15 + 22)}_{\text{First and Middle Segments}} + \underbrace{(15 + 12 + \min(MVL, N/2^I))}_{\text{Final Store}} + 2N/2^I$$

$$\begin{aligned} T_N &= \sum_{I=1}^{\log_2 N} \left(10 + \left[\frac{N/2^I}{MVL} \right] (15 + 22) + (15 + 12 + \min(MVL, N/2^I)) + 2N/2^I \right) \\ &= 37 \log_2 N + 37 \sum_{I=1}^{\log_2 N} \left[\frac{N/2^I}{MVL} \right] + \sum_{I=1}^{\log_2 N} \left(\min(MVL, N/2^I) \right) + 2(N - 1). \end{aligned}$$



Vector Summation

r_∞ asymptotic rate

$N_{1/2}$ Size for which
 r_∞ is achieved

For VP-1 $r_\infty = 0.199$ ops/cycle

For VP-3 $r_\infty = 0.388$ ops/cycle

N	VP-1		VP-3	
	Cycles	Ops/cycle	Cycles	Ops/cycle
20	452	0.0442	428	0.0467
40	607	0.0658	562	0.0712
60	686	0.0874	621	0.0966
64	702	0.0912	633	0.1011
65	782	0.0830	711	0.0914
80	842	0.0950	756	0.1058
100	921	0.1085	815	0.1226
120	1001	0.1199	875	0.1371
128	1033	0.1239	899	0.1424
129	1178	0.1094	1013	0.1272
140	1222	0.1145	1041	0.1344
160	1302	0.1228	1091	0.1466
180	1382	0.1302	1140	0.1578
200	1461	0.1368	1190	0.1680
300	2067	0.1451	1578	0.1901
400	2531	0.1580	1839	0.2174
500	2931	0.1706	2064	0.2422
1000	5461	0.1831	3461	0.2889
1500	8057	0.1862	4864	0.3084
2000	10511	0.1903	6154	0.3250
5000	25782	0.1939	14160	0.3531
10000	50927	0.1964	27184	0.3679



COSC 6365
Lecture 12
2008-02-21

CS@UH

Strip-Mining

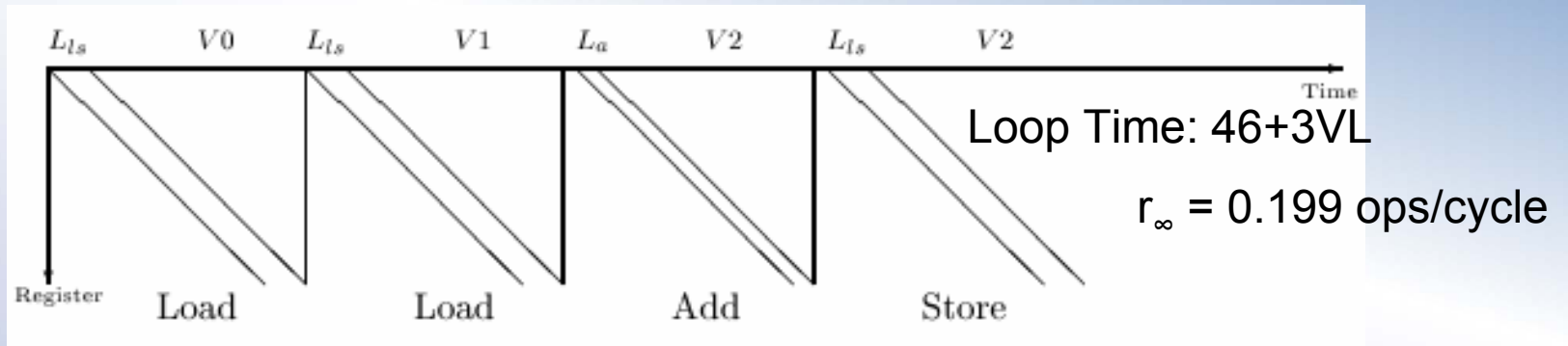
```
FOR I = 1 TO N DO  
    Y(I) = Y(I)+X(I)  
ENDFOR
```

```
VL=NmodMVL  
LOW=1  
FOR J = 0 TO N/MVL DO  
    FOR I = LOW TO LOW+VL-1 DO  
        Y(I) = Y(I)+X(I)  
    ENDFOR  
    LOW = LOW+VL  
    VL = MVL  
ENDFOR
```

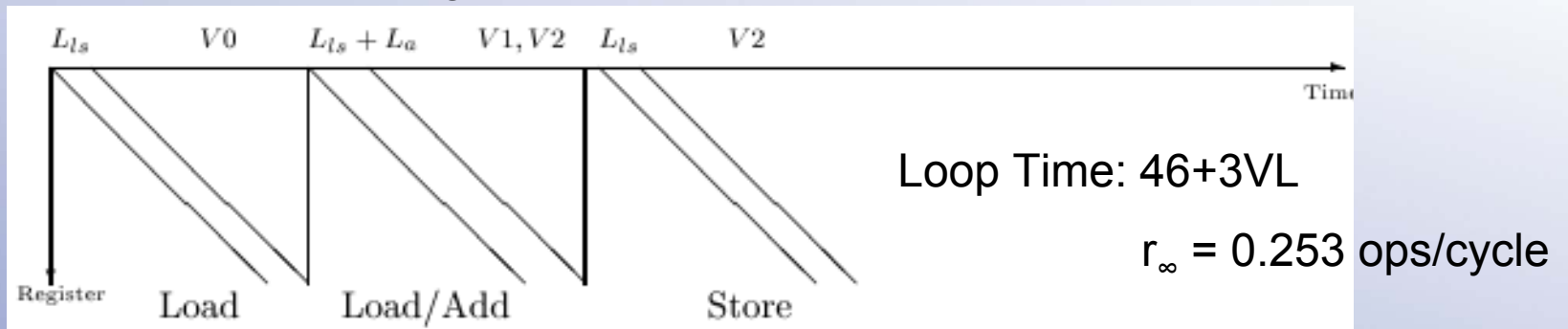


Chaining (building a connected pipeline)

Vector Summation VP-1

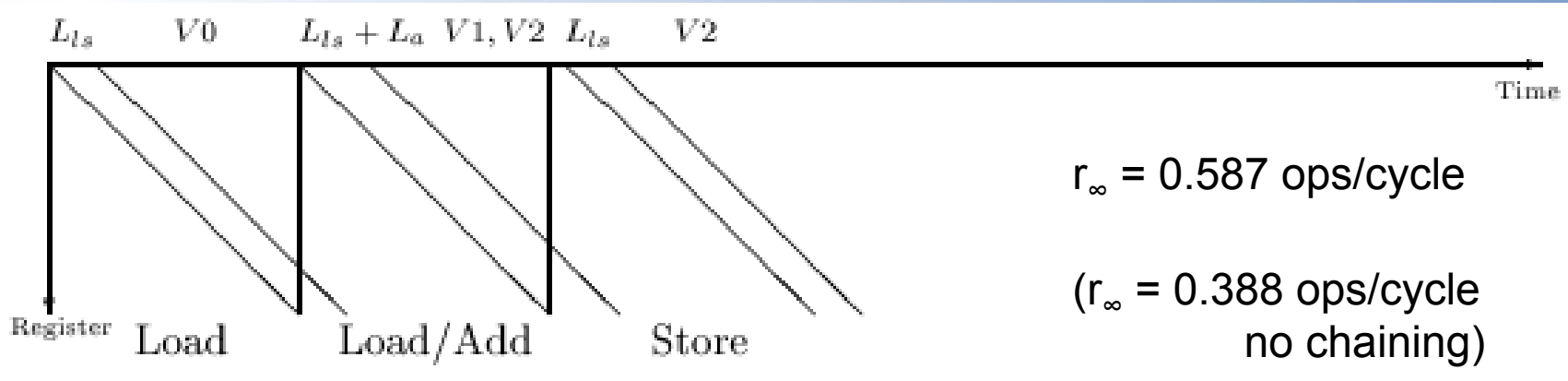


Vector Summation with chaining VP-1





Vector Summation



Load two segments of X
 Add segments of X
 Store the result

Load two segments	12+VL
Vector Add	6 (chained with load)
Vector Store	12 (chained with add)
Total	<u>30+VL</u>



COSC 6365
Lecture 12
2008-02-21

CS@UH

Program transformations for vectorization - dependencies

- *True data dependencies*: read-after-write (RAW)
- *Anti-dependence*: write-after-read (WAR)
- *Output dependence*: write-after-write (WAW)
- *Loop-carried dependence*: a dependence of one of the above types occurs from one iteration to another



COSC 6365
Lecture 12
2008-02-21

CS@UH

Common program transformations for vectorization

- Variable Renaming
- Statement Reordering
- Loop Distribution
- Loop Reordering
- Scalar Expansion
- Variable Copying
- Index Splitting
- Node Splitting
- Loop Unrolling
- Loop Peeling
- Loop Rerolling
- Loop Collapsing



Variable Renaming

```
FOR I = 1 TO 100 DO  
[1]    Y(I) = X(I)/S  
[2]    X(I) = X(I)+S  
[3]    Z(I) = Y(I)+S  
[4]    Y(I) = S - Y(I)  
ENDFOR
```

RAW: 1 to 3 and 1 to 4

WAR: 1 to 2

WAW: 1 to 4

No loop carried dependence

```
FOR I = 1 TO 100 DO  
[1]    T(I) = X(I)/S  
[2]    X1(I) = X(I)+S  
[3]    Z(I) = T(I)+S  
[4]    Y(I) = S-T(I)  
ENDFOR
```

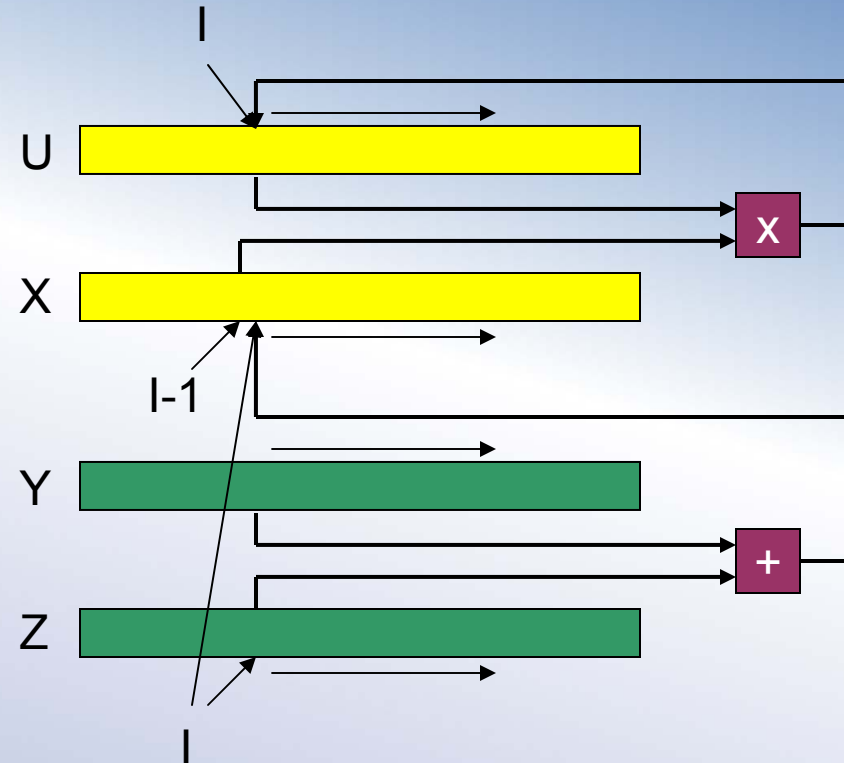
RAW: eliminated



Statement reordering

```
FOR I = 1 TO 100 DO  
[1] U(I) = X(I-1)*U(I)  
[2] X(I) = Y(I)+Z(I)  
ENDFOR
```

Loop carried dependence on X





Statement reordering

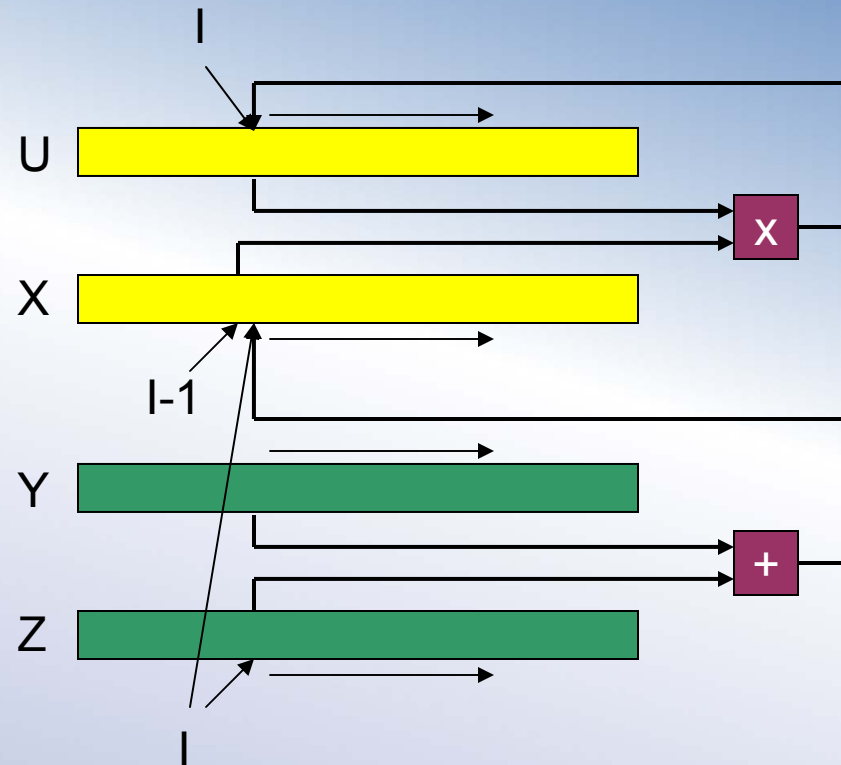
```
FOR I = 1 TO 100 DO
```

```
[2] X(I) = Y(I)+Z(I)
```

```
[1] U(I) = X(I-1)*U(I)
```

```
ENDFOR
```

Forward dependence on X
Can be vectorized





Loop distribution

```
FOR I = 1 TO 100 DO  
[2]   X(I) = Y(I)+Z(I)  
[1]   U(I) = X(I-1)*U(I)  
ENDFOR
```

```
FOR I = 1 TO 100 DO  
[2]   X(I) = Y(I)+Z(I)  
ENDFOR  
FOR I = 1 TO 100 DO  
[1]   U(I) = X(I-1)*U(I)  
ENDFOR
```

X(1:100) = Y(1:100)+Z(1:100)

U(1:100) = X(0:99)*U(1:100)



Loop distribution

```
FOR I = 1 TO 100 DO  
[1]      Z(I) = X(I-2)*Y(I)  
[2]      U(I) = Y(I)+Y(I-1)/2  
[3]      X(I) = Z(I)+2  
ENDFOR
```

Reorder

```
FOR I = 1 TO 100 DO  
[1]      Z(I) = X(I-2)*Y(I)  
[3]      X(I) = Z(I)+2  
[2]      U(I) = Y(I)+Y(I-1)/2  
ENDFOR  
      U(1:100) = Y(1:100)+Y(0:99)/2
```

Distribute

```
FOR I = 1 TO 100 DO  
[1]      Z(I) = X(I-2)*Y(I)  
[3]      X(I) = Z(I)+2  
ENDFOR  
FOR I = 1 TO 100 DO  
[2]      U(I) = Y(I)+Y(I-1)/2  
ENDFOR
```