CS 152 Computer Architecture and Engineering
CS252 Graduate Computer Architecture

Lecture 16 – RISC-V Vectors

Krste Asanovic
Electrical Engineering and Computer Sciences
University of California at Berkeley

http://www.eecs.berkeley.edu/~krste
http://inst.eecs.berkeley.edu/~cs152

Last Time in Lecture 16

GPU architecture

- Evolved from graphics-only, to more general-purpose computing
- GPUs programmed as attached accelerators, with software required to separate GPU from CPU code, move memory
- Many cores, each with many lanes
  - thousands of lanes on current high-end GPUs
- SIMT model has hardware management of conditional execution
  - code written as scalar code with branches, executed as vector code with predication

New RISC-V “V” Vector Extension

- Being added as a standard extension to the RISC-V ISA
  - An updated form of Cray-style vectors for modern microprocessors
- Today, a short tutorial on current draft standard, v0.7
  - v0.7 is intended to be close to final version of RISC-V vector extension
  - Still a work in progress, so details might change before standardization
  - **`https://github.com/riscv/riscv-v-spec`**
- **WARNING**: Lab 4 uses older version of vector ISA, since new tools not available yet
  - Most concepts carry over, if not programming details

RISC-V Scalar State

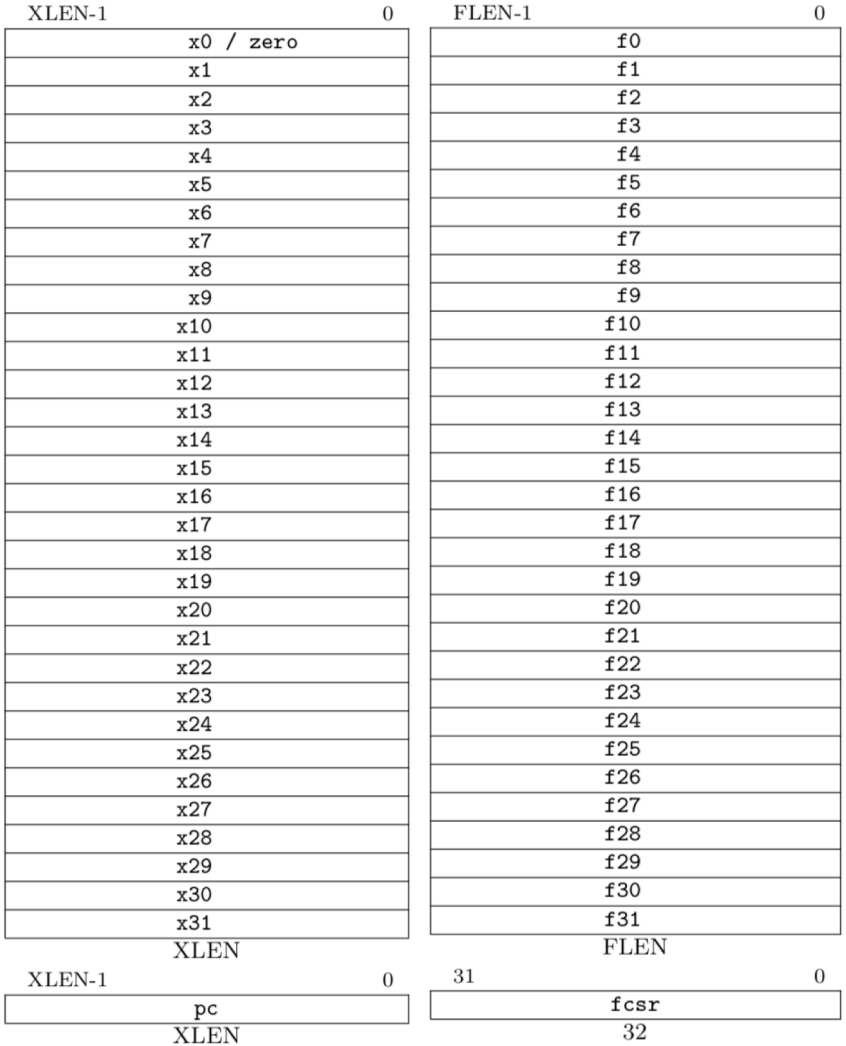
Program counter (**pc**)

32x32/64-bit integer registers (**x0-x31**)
- **x0** always contains a 0

Floating-point (FP), adds 32 registers (**f0-f31**)
- each can contain a single- or double-precision FP value (32-bit or 64-bit IEEE FP)

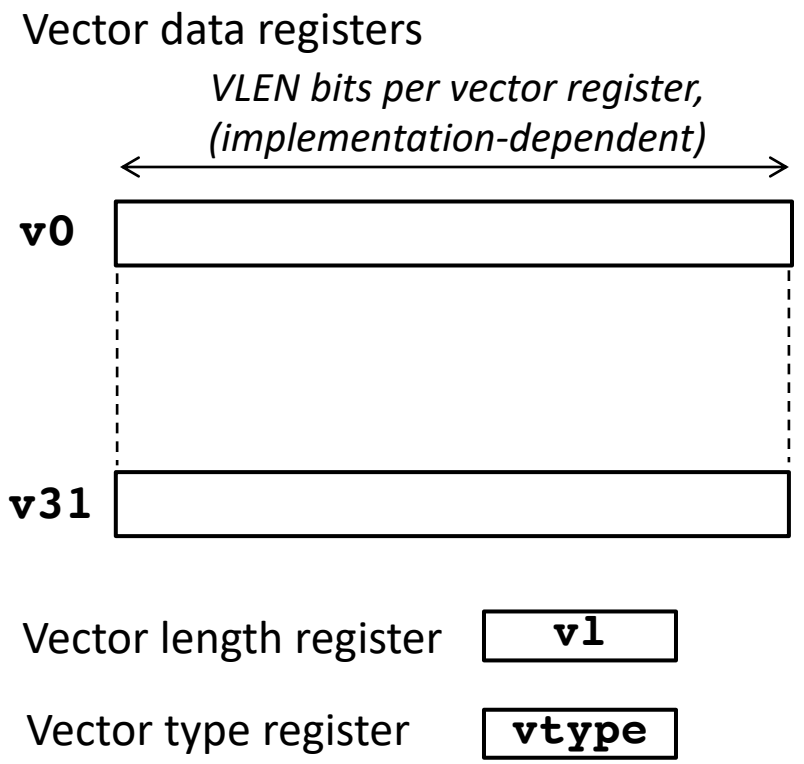
FP status register (**fcsr**), used for FP rounding mode & exception reporting

ISA string options:
- RV32I (XLEN=32, no FP)
- RV32IF (XLEN=32, FLEN=32)
- RV32ID (XLEN=32, FLEN=64)
- RV64I (XLEN=64, no FP)
- RV64IF (XLEN=64, FLEN=32)
- RV64ID (XLEN=64, FLEN=64)



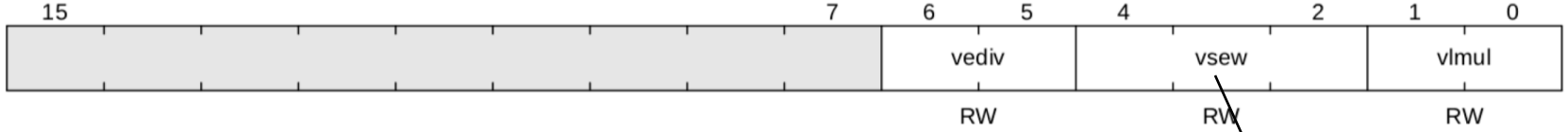
Vector Extension Additional State

- 32 vector data registers, **`v0–v31`**, each VLEN bits long
- Vector length register **`vl`**
- Vector type register **`vtype`**
- Other control registers:
  - **`vstart`**
    - For trap handling
  - **`vrm/vxsat`**
    - Fixed-point rounding mode/saturation
    - Also appear in **`fcsr`**



Vector Type Register

vtype register layout



Bits	Contents
1:0	vlmul[1:0]
4:2	vsew[2:0]
6:5	vediv[1:0]
XLEN-1:7	Reserved (write 0)

vsew[2:0]			SEW
0	0	0	8
0	0	1	16
0	1	0	32
0	1	1	64
1	0	0	128
1	0	1	256
1	1	0	512
1	1	1	1024

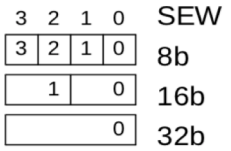
`vsew[2:0]` field encodes standard element width (SEW) in bits of elements in vector register (SEW = $8*2^{\texttt{vsew}}$)

`vlmul[1:0]` encodes vector register length multiplier (LMUL = $2^{\texttt{vlmul}}$ = 1-8)

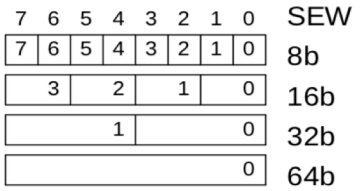
`vediv[1:0]` encodes how vector elements are divided into equal sub-elements (EDIV = $2^{\texttt{vediv}}$ = 1-8)

Example Vector Register Data Layouts (LMUL=1)

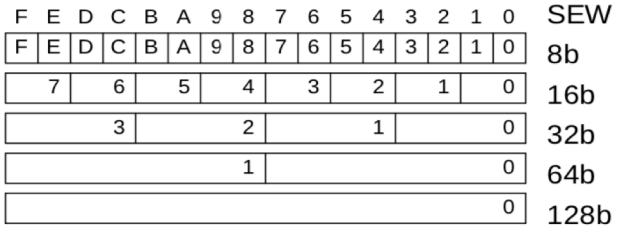
VLEN=32b



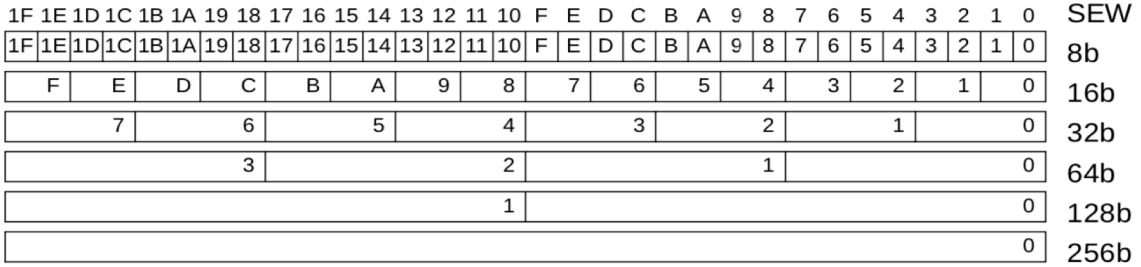
VLEN=64b



VLEN=128b



VLEN = 256b



Setting vector configuration, `vsetvli/vsetvl`

The **`vsetvl{i}`** configuration instructions set the **`vtype`** register, and also set the **`vl`** register, returning the **`vl`** value in a scalar register

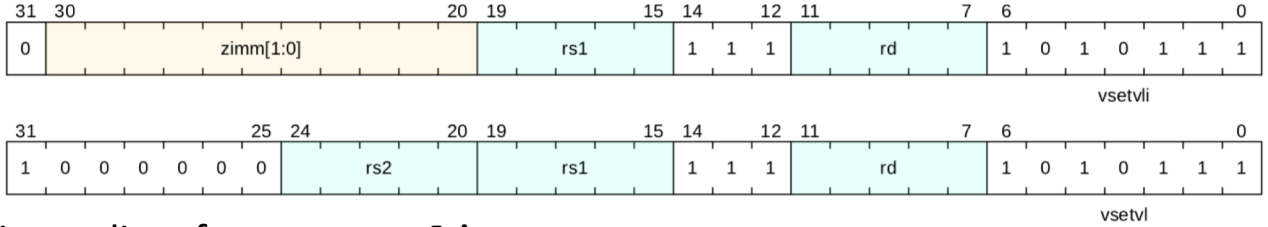
```
vsetvli rd, rs1, e8 # Set SEW=8, vl=min(VLEN/SEW,rs1), rd=vl
```

Resulting machine vector length setting

Requested application vector length

`vtype` parameters (SEW,LMUL,EDIV) encoded as immediate in instruction

Instruction encoding



Usually use immediate form, **`vsetvli`**, to set **`vtype`** parameters.

The register version **`vsetvl`** is usually used for context save/restore

`vsetvl{i}` operation

- The first scalar register argument, rs1, is the requested application vector length (AVL)
- The type argument (either immediate or second register) indicates how the vector registers should be configured
  - Configuration includes size of each element
- The vector length is set to the minimum of requested AVL and the maximum supported vector length (VLMAX) in the new configuration
  - VLMAX = LMUL*VLEN/SEW
  - **`vl`** = min(AVL, VLMAX)
- The value placed in **`vl`** is also written to the scalar destination register rd

Simple stripmined vector memcpy example

Set configuration, calculate vector strip length

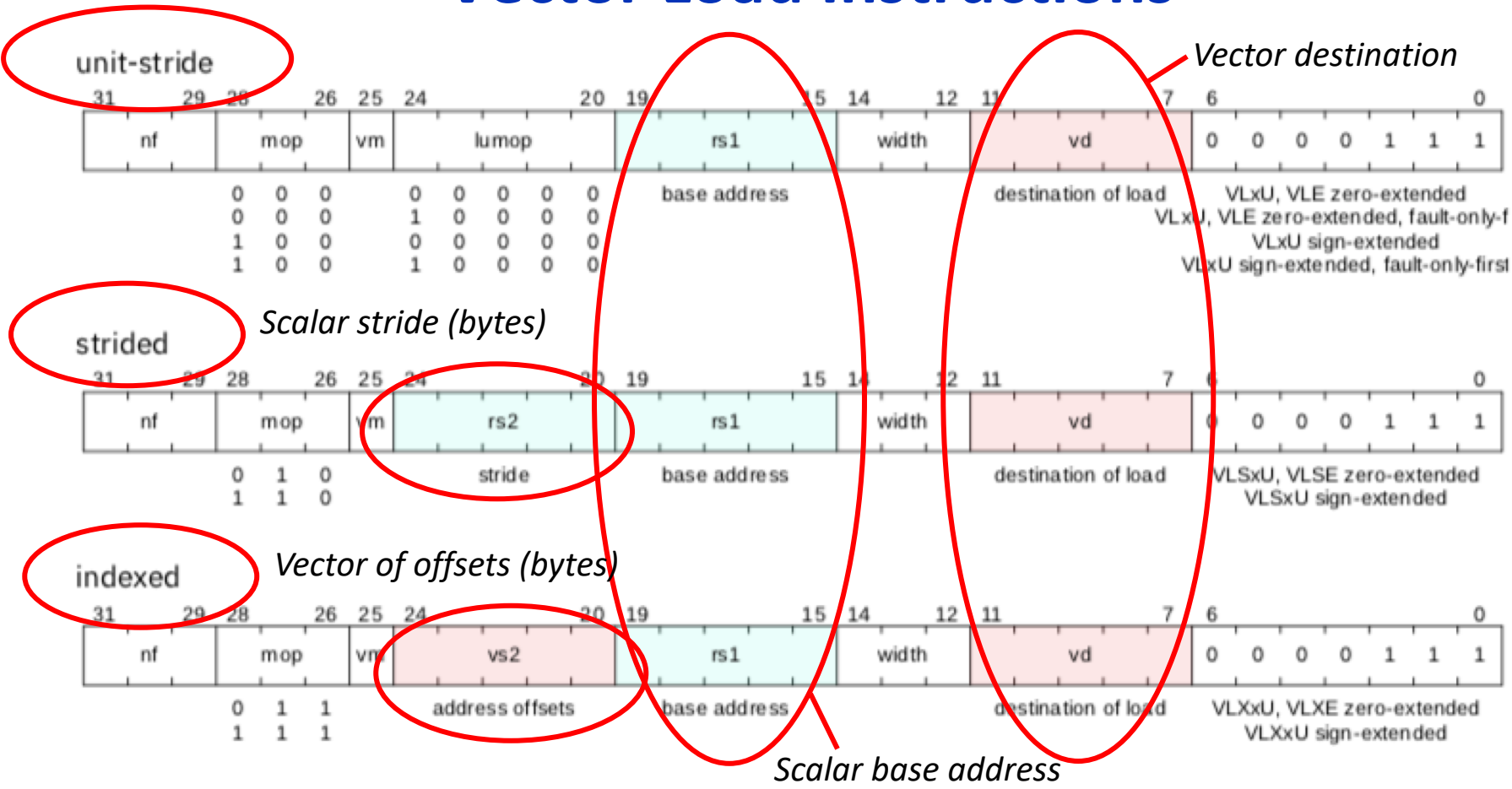
Unit-stride vector load bytes

Unit-stride vector store bytes

```
# void *memcpy(void* dest, const void* src, size_t n)
# a0=dest, a1=src, a2=n
#
memcpy:
    mv a3, a0 # Copy destination
loop:
    vsetvli t0, a2, e8      # Vectors of 8b
    vlb.v v0, (a1)                  # Load bytes
    add a1, a1, t0                  # Bump pointer
    sub a2, a2, t0                  # Decrement count
    vsb.v v0, (a3)                  # Store bytes
    add a3, a3, t0                  # Bump pointer
    bnez a2, loop                   # Any more?
    ret                             # Return
```

Binary machine code can run on machines with any VLEN!

Vector Load Instructions



Vector Store Instructions

Vector Unit-Stride Loads/Stores

```
# vd destination, rs1 base address, vm is mask encoding (v0.t or <missing>)
vlb.v         vd, (rs1), vm     # 8b signed
vlh.v         vd, (rs1), vm     # 16b signed
vlw.v         vd, (rs1), vm     # 32b signed

vlbu.v        vd, (rs1), vm     # 8b unsigned
vlhu.v        vd, (rs1), vm     # 16b unsigned
vlwu.v        vd, (rs1), vm     # 32b unsigned

vle.v         vd, (rs1), vm     # SEW

# vs3 store data, rs1 base address, vm is mask encoding (v0.t or <missing>)
vsb.v         vs3, (rs1), vm    # 8b store
vsh.v         vs3, (rs1), vm    # 16b store
vsw.v         vs3, (rs1), vm    # 32b store
vse.v         vs3, (rs1), vm    # SEW store
```

Vector Strided Load/Store Instructions

```
# vd destination, rs1 base address, rs2 byte stride
vlsb.v          vd, (rs1), rs2, vm  # 8b
vlsh.v          vd, (rs1), rs2, vm  # 16b
vlsw.v          vd, (rs1), rs2, vm  # 32b

vlsbu.v         vd, (rs1), rs2, vm  # unsigned 8b
vlshu.v         vd, (rs1), rs2, vm  # unsigned 16b
vlswu.v         vd, (rs1), rs2, vm  # unsigned 32b

vlse.v          vd, (rs1), rs2, vm  # SEW

# vs3 store data, rs1 base address, rs2 byte stride
vssb.v          vs3, (rs1), rs2, vm # 8b
vssh.v          vs3, (rs1), rs2, vm # 16b
vssw.v          vs3, (rs1), rs2, vm # 32b
vsse.v          vs3, (rs1), rs2, vm # SEW
```

Vector Indexed Loads/Stores

```
# vd destination, rs1 base address, vs2 indices
vlxb.v        vd, (rs1), vs2, vm  # 8b
vlxh.v        vd, (rs1), vs2, vm  # 16b
vlxw.v        vd, (rs1), vs2, vm  # 32b

vlxbu.v       vd, (rs1), vs2, vm  # 8b unsigned
vlxhu.v       vd, (rs1), vs2, vm  # 16b unsigned
vlxwu.v       vd, (rs1), vs2, vm  # 32b unsigned

vlxe.v        vd, (rs1), vs2, vm  # SEW

# Vector ordered-indexed store instructions
# vs3 store data, rs1 base address, vs2 indices
vsxb.v        vs3, (rs1), vs2, vm # 8b
vsxh.v        vs3, (rs1), vs2, vm # 16b
vsxw.v        vs3, (rs1), vs2, vm # 32b
vsxe.v        vs3, (rs1), vs2, vm # SEW

# Vector unordered-indexed store instructions
vsuxb.v       vs3, (rs1), vs2, vm # 8b
vsuxh.v       vs3, (rs1), vs2, vm # 16b
vsuxw.v       vs3, (rs1), vs2, vm # 32b
vsuxe.v       vs3, (rs1), vs2, vm # SEW
```

Vector Length Multiplier, LMUL

- Gives fewer but longer vector registers
- Set by **`vlmul[1:0]`** field in **`vtype`** during **`setvli`**

LMUL=2

F E D C	B A 9 8	7 6 5 4	3 2 1 0	Byte
3	2	1	0	v2 * n + 0
7	6	5	4	v2 * n + 1

LMUL=4

F E D C	B A 9 8	7 6 5 4	3 2 1 0	Byte
9	8	1	0	v4 * n + 0
B	A	3	2	v4 * n + 1
D	C	5	4	v4 * n + 2
F	E	7	6	v4 * n + 3

LMUL=8 stripmined vector memcpy example

```
# void *memcpy(void* dest, const void* src, size_t n)
# a0=dest, a1=src, a2=n
#
memcpy:
    mv a3, a0 # Copy destination
loop:
    vsetvli t0, a2, e8,m8  # Vectors of 8b
    vlb.v v0, (a1)                  # Load bytes
      add a1, a1, t0                # Bump pointer
      sub a2, a2, t0                # Decrement count
    vsb.v v0, (a3)                  # Store bytes
      add a3, a3, t0                # Bump pointer
      bnez a2, loop                 # Any more?
      ret                           # Return
```

Set configuration, calculate vector strip length

Unit-stride vector load bytes

Unit-stride vector store bytes

Combine eight vector registers into group

(v0,v1,…,v7)

Binary machine code can run on machines with any VLEN!

Mixed-Width Loops

- Have different element widths in one loop, even in one instruction
- Want same number of elements in each vector register, even if different bits/element
- Solution: Keep SEW/LMUL constant

VLEN=256b, SLEN=128b

SEW=8b, LMUL=1, VLMAX=32

1F	1E	1D	1C	1B	1A	19	18	17	16	15	14	13	12	11	10	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	Byte
1F	1E	1D	1C	1B	1A	19	18	17	16	15	14	13	12	11	10	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	v1 * n + 0

SEW=16b, LMUL=2, VLMAX=32

1F	1E	1D	1C	1B	1A	19	18	17	16	15	14	13	12	11	10	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	Byte
17		16		15		14		13		12		11		10		7		6		5		4		3		2		1		0		v2 * n + 0
1F		1E		1D		1C		1B		1A		19		18		F		E		D		C		B		A		9		8		v2 * n + 1

SEW=32b, LMUL=4, VLMAX=32

1F	1E	1D	1C	1B	1A	19	18	17	16	15	14	13	12	11	10	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	Byte
13				12				11				10				3				2				1				0				v4 * n + 0
17				16				15				14				7				6				5				4				v4 * n + 1
1B				1A				19				18				B				A				9				8				v4 * n + 2
1F				1E				1D				1C				F				E				D				C				v4 * n + 3

SEW=64b, LMUL=8, VLMAX=32

1F	1E	1D	1C	1B	1A	19	18	17	16	15	14	13	12	11	10	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	Byte
11								10								1								0								v8 * n + 0
13								12								3								2								v8 * n + 1
15								14								5								4								v8 * n + 2
17								16								7								6								v8 * n + 3
19								18								9								8								v8 * n + 4
1B								1A								B								A								v8 * n + 5
1D								1C								D								C								v8 * n + 6
1F								1E								F								E								v8 * n + 7

CS152 Administrivia

- PS 4 due Friday April 5 in Section
- Lab 4 out on Friday
- Lab 3 due Monday April 8

CS252 Administrivia

Next week readings: Cray-1, VLIW & Trace Scheduling