

ESE532: System-on-a-Chip Architecture

Day 6: September 22, 2021
Data-Level Parallelism

Pickup board before lecture start 10:20am
or at end of lecture.



Penn ESE532 Fall 2021 -- DeHon

Today

Data-level Parallelism

- For Parallel Decomposition (Part 1)
- Architectures (Part 2)
- Concepts (Part 3)
- NEON

Penn ESE532 Fall 2021 -- DeHon

2

Message

- Data Parallelism easy basis for decomposition
 - When things are data parallel, exploiting for parallelism relatively easy
- Data Parallel architectures can be compact
 - pack more computations onto a fixed-size Integrated Circuit chip
 - OR perform computation in less area

Penn ESE532 Fall 2021 -- DeHon

3

Preclass 1

- 500 news articles
 - 3000 words each
- Count total occurrences of a string
- How can we exploit data-level parallelism on task?
- How much parallelism can we exploit?
 - If don't exploit parallelism within file?
 - If do exploit parallelism within file?

Penn ESE532 Fall 2021 -- DeHon

4

Parallel Decomposition

Penn ESE532 Fall 2021 -- DeHon

5

Data Parallel

- Data-level parallelism can serve as an organizing principle for parallel task decomposition
- Run computation on independent data in parallel

Penn ESE532 Fall 2021 -- DeHon

6

Exploit

- Can exploit with
 - Threads
 - Pipeline Parallelism
 - Instruction-level Parallelism
 - Fine-grained Data-Level Parallelism

Penn ESE532 Fall 2021 -- DeHon

7

Performance Benefit

- Ideally linear in number of processors (resources)
- Resource Bound:
 - $T_{dp} = (T_{single} \times N_{data}) / P$
- T_{single} = Latency on single data item
- $T_{dp} = T_{single} \times \lceil N_{data} / P \rceil$

Penn ESE532 Fall 2021 -- DeHon

8

Preclass 2 Common Examples

- What are common examples of DLP?
 - Simulation
 - Numerical Linear Algebra
 - Signal or Image Processing
 - Optimization

Penn ESE532 Fall 2021 -- DeHon

9

Hardware Architectures

Part 2

Penn ESE532 Fall 2021 -- DeHon

10

Idea

- If we're going to perform the same operations on different data, exploit that to reduce area, energy
- Reduced area means can have more computation on a fixed-size IC chip.

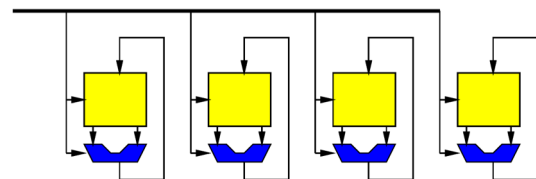
Penn ESE532 Fall 2021 -- DeHon

11

SIMD

- Single Instruction Multiple Data

Shared Instruction

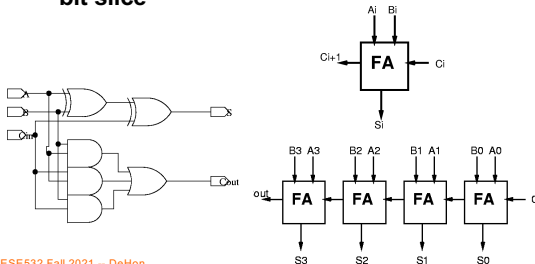


Penn ESE532 Fall 2021 -- DeHon

12

Ripple Carry Addition

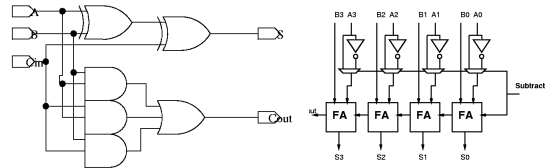
- Can define logic for each bit, then assemble:
 - bit slice



Penn ESE532 Fall 2021 -- DeHon

Arithmetic Logic Unit (ALU)

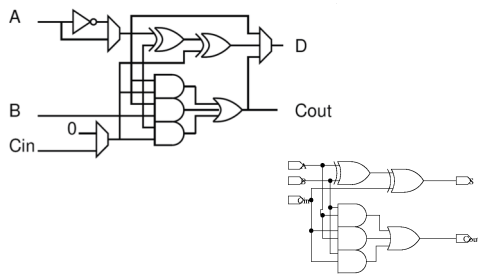
- Observe:
 - with small tweaks can get many functions with basic adder components



Penn ESE532 Fall 2021 -- DeHon

14

Arithmetic and Logic Unit



Penn ESE532 Fall 2021 -- DeHon

15

ALU Functions

- A+B w/ Carry
- B-A
- A xor B (squash carry)
- A*B (squash carry)
- /A

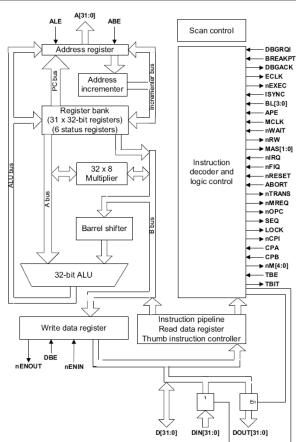
Key observation: every ALU bit does the same thing on different bits of the data word(s).

Penn ESE532 Fall 2021 -- DeHon

16

ARM v7 Core

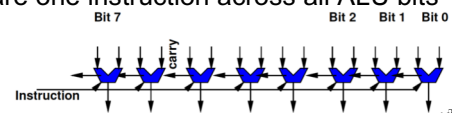
- ALU is Key compute operator in a processor



Penn ESE532 Fall 2021 -- DeHon

W-bit ALU as SIMD

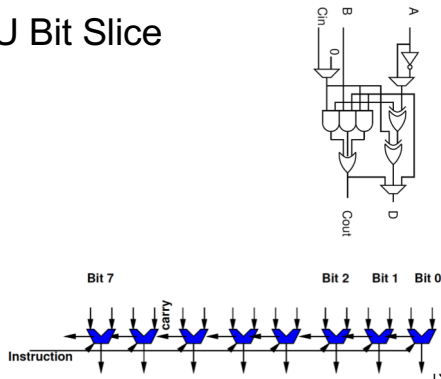
- Familiar idea
- A W-bit ALU (W=8, 16, 32, 64, ...) is SIMD
- Each bit of ALU works on separate bits
 - Performing the same operation on it
 - Trivial to see bitwise AND, OR, XOR
 - Also true for ADD (each bit performing Full Adder)
- Share one instruction across all ALU bits



Penn ESE532 Fall 2021 -- DeHon

18

ALU Bit Slice



Penn ESE532 Fall 2021 -- DeHon

Preclass 4

- What do we get when add 65280 to 257
 - 32b unsigned add?
 - 16b unsigned add?

Penn ESE532 Fall 2021 -- DeHon

20

ALU vs. SIMD ?

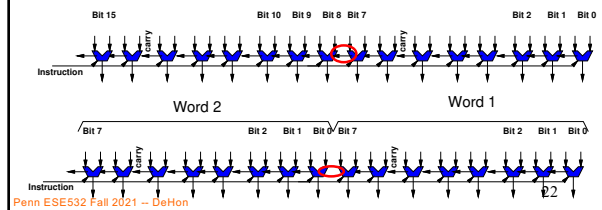
- What's different between
 - 128b wide ALU
 - SIMD datapath supporting eight 16b ALU operations

Penn ESE532 Fall 2021 -- DeHon

21

ALU vs. SIMD Example

- Concretely show:
 - 16b wide ALU
 - SIMD datapath with two 8b wide ALUs



Penn ESE532 Fall 2021 -- DeHon

22

ALU vs. SIMD ?

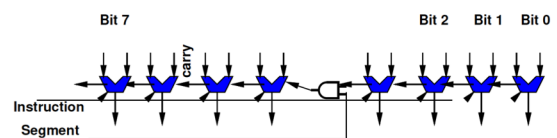
- How could we get both operations from the **same** hardware?
 - 128b wide ALU
 - SIMD datapath supporting eight 16b ALU operations

Penn ESE532 Fall 2021 -- DeHon

23

Segmented Datapath

- Add a few gates to convert a wide datapath into one supporting a set of smaller operations
 - Just need to squash the carry at points



Penn ESE532 Fall 2021 -- DeHon

24

Segmented Datapath

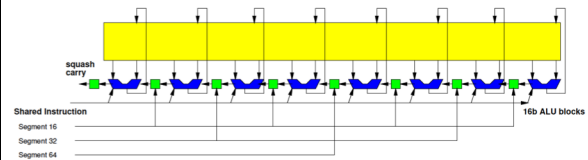
- Add a few gates to convert a wide datapath into one supporting a set of smaller operations
 - Just need to squash the carry at points
- But need to keep instructions (description) small
 - So typically have limited, homogeneous widths supported

Penn ESE532 Fall 2021 -- DeHon

25

Segmented 128b Datapath

- 1x128b, 2x64b, 4x32b, 8x16b



Penn ESE532 Fall 2021 -- DeHon

26

Vector

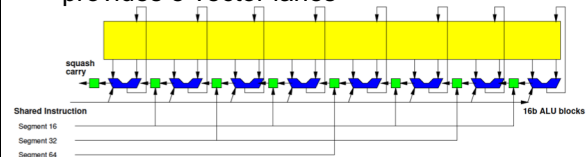
- In Math – 1D array of numbers
- Here – 1D array of numbers that we operate upon with SIMD operations
 - Perform the same operations on
- Vector A = [2, 3, 5, 7]
- Vector B = [4, 6, 8, 9]
- A+B = [6, 9, 13, 16]

Penn ESE532 Fall 2021 -- DeHon

27

Terminology: Vector Lane

- Each of the separate segments called a **Vector Lane**
 - Length of vector hardware natively supports
- For 16b data on 128b datapath, this provides 8 vector lanes

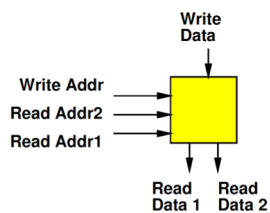


Penn ESE532 Fall 2021 -- DeHon

28

Register File

- Small Memory
- Usually with multiple ports
 - Ability to perform multiple reads and writes simultaneously
- Small
 - To make it fast (small memories fast)
 - Multiple ports are expensive

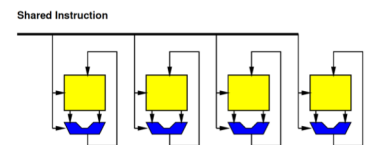


Penn ESE532 Fall 2021 -- DeHon

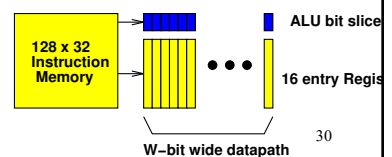
29

SIMD Datapath

- Logical View:



- Layout View:



Penn ESE532 Fall 2021 -- DeHon

30

Preclass 5

W	A(W)	% datapath	Instances	Peak 16b ops	Ratio
16		1.1	9	91	1.0
64					
128	1.8	43	56	448	4.9
256					

Complete: 64, 256

128 x 32 Instruction Memory

ALU bit slice

16 entry Register File

W-bit wide datapath

Penn ESE532 Fall 2021 – DeHon

32

To Scale Comparison

The diagram illustrates two processor architectures for comparison. The top architecture shows a 128 x 32 Instruction Memory connected to a single ALU bit slice and a 16 entry Register File. The bottom architecture shows a 128 x 32 Instruction Memory connected to a 16-entry Register File, which is then connected to a single ALU bit slice. The bottom architecture is labeled "To Scale Comparison".

128 x 32 Instruction Memory

ALU bit slice

16 entry Register File

128 x 32 Instruction Memory

ALU bit slice

16 entry Register File

To Scale Comparison

To Scale W=1024

The diagram illustrates a memory access pattern for a large matrix multiplication. On the left, a yellow vertical rectangle is labeled "128 x 32 iteration Memory". To its right is a stack of 1024 horizontal bars, each representing a row of the matrix. The bars are colored yellow and blue, indicating a striped access pattern. The stack is divided into four groups of 256 bars each, with a blue bar separating the groups. This visualizes the scaling of the memory access pattern for a large matrix size W=1024.

128 x 32
iteration
Memory

44

Penn ESE532 Fall 2021 -- DeHon

Preclass 6: Vector Length

- May not match physical hardware length
 - Logical (application need): Vector Length
 - Physical (hardware provide): Vector Lanes

```
vadd(int *a, int *b, int *res,  
     int vector_length) {  
    for (int i=0;i<vector_length;i++)  
        res[i]=a[i]+b[i];  
}
```

Penn ESE532 Fall 2021 -- DeHon

Preclass 6: Vector Length

- May not match physical hardware length
 - Logical (application need): Vector Length
 - Physical (hardware provide): Vector Lanes
- How support
 - Vector length < Vector Lanes? (5, 8)
 - Vector length > Vector Lanes? (24, 8)

36

Preclass 6: Vector Length

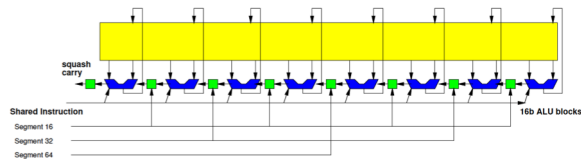
- May not match physical hardware length
 - Logical (application need): Vector Length
 - Physical (hardware provide): Vector Lanes
- Efficiency (utilization) when
 - Vector length < Vector Lanes? (5, 8)
 - Vector length > Vector Lanes? (24, 8)
 - Vector length % (Vector Lanes) != 0 (13, 8)
 - E.g. vector length 13, for 8 vector lanes

37

Performance

- Resource Bound

$$T_{\text{vector}} = \lceil N_{\text{op}} / VL \rceil \times T_{\text{cycle}}$$



Penn ESE532 Fall 2021 -- DeHon

38

Preclass 3: Opportunity

- Don't need 64b variables for lots of things
- Natural data sizes?
 - Audio samples?
 - Input from A/D?
 - Video Pixels?
 - X, Y coordinates for 4K x 4K image?

Penn ESE532 Fall 2021 -- DeHon

39

Vector Computation

- Easy to map to SIMD flow if can express computation as operation on vectors
 - Vector Add
 - Vector Multiply
 - Dot Product

Penn ESE532 Fall 2021 -- DeHon

40

Concepts

Part 3

Penn ESE532 Fall 2021 -- DeHon

41

Terminology: Scalar

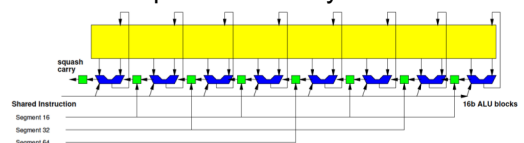
- Simple: non-vector
- When we have a vector unit controlled by a normal (non-vector) processor core often need to distinguish:
 - Vector operations that are performed on the vector unit
 - Normal=non-vector=scalar operations performed on the base processor core

Penn ESE532 Fall 2021 -- DeHon

42

Vector Register File

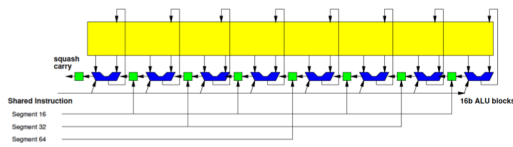
- Need to be able to feed the SIMD compute units
 - Not be bottlenecked on data movement to the SIMD ALU
- Wide RF (register file) to supply
- With wide path to memory



Penn ESE532 Fall 2021 -- DeHon

Point-wise Vector Operations

- Easy – just like wide-word operations (now with segmentation)

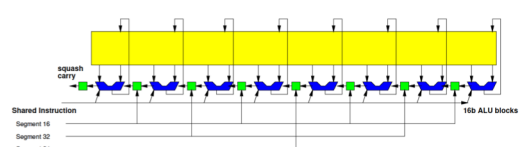


Penn ESE532 Fall 2021 -- DeHon

44

Point-wise Vector Operations

- ...but alignment matters.
- If not aligned, need to perform data movement operations to get aligned



Penn ESE532 Fall 2021 -- DeHon

45

Ideal

- for ($i=0; i<64; i++$)
– $c[i]=a[i]+b[i]$
- No data dependencies
- Access every element
- Number of operations is a multiple of number of vector lanes

Penn ESE532 Fall 2021 -- DeHon

46

Skipping Elements?

- How does this work with datapath?
– Assume loaded $a[0], a[1], \dots, a[63]$ and $b[0], b[1], \dots, b[63]$ into vector register file
- for ($i=0; i<64; i=i+2$)
– $c[i/2]=a[i]+b[i]$

Penn ESE532 Fall 2021 -- DeHon

47

Stride

- Stride: the distance between vector elements used
- for ($i=0; i<64; i=i+2$)
– $c[i/2]=a[i]+b[i]$
- Accessing data with stride=2

Penn ESE532 Fall 2021 -- DeHon

48

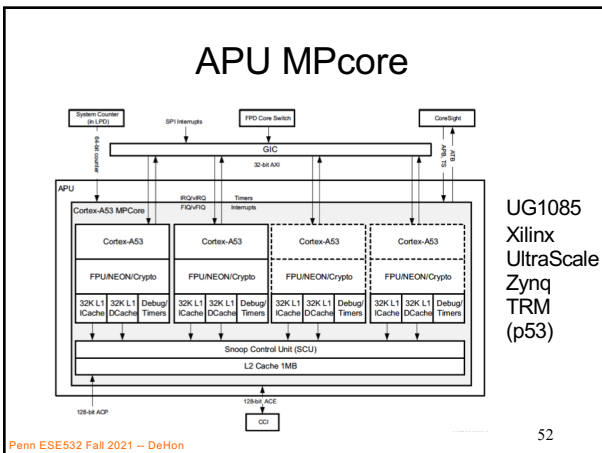
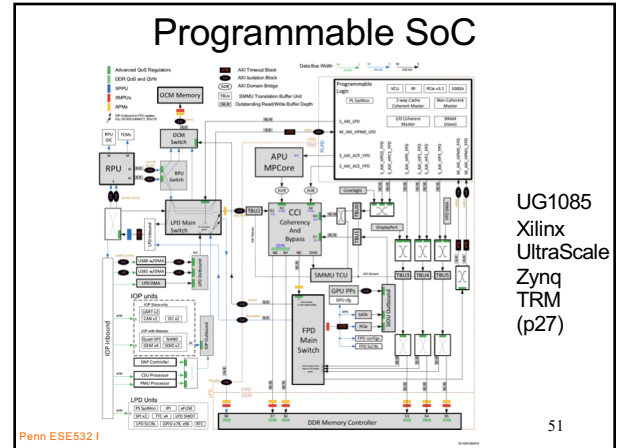
Load/Store

- Strided load/stores
– Some architectures will provide strided memory access that compact when read into register file
- Scatter/gather
– Some architectures will provide memory operations to grab data from different places to construct a dense vector

Penn ESE532 Fall 2021 -- DeHon

49

ARM Vector Accelerator



- 128b wide register file, 16 registers
- Support
 - 2x64b
 - 4x32b (also Single-Precision Float)
 - 8x16b
 - 16x8b

- VADD – basic vector
- VCEQ – compare equal
 - Sets to all 0s or 1s, useful for masking
- VMIN – avoid using if's
- VMLA – accumulating multiply
- VPADAL – maybe useful for reduce
 - Vector pair-wise add
- VEXT – for “shifting” vector alignment
- VLDn – deinterleaving load

2-issue
In-order
1 NEON pipe
64b wide
8-stage pipe

The diagram illustrates the Cortex-A9 pipeline architecture. On the left, the front end consists of a 2-issue, in-order, 1 NEON pipe, 64b wide, and 8-stage. This front end feeds into a pipeline with stages: Lowest, Decode, Issue, and then a multi-ported execution stage. The execution stage is divided into Integer, Multiply, Floating-Point / NEON, Dual Issue, and Load/Store. The output of the execution stage is a Writeback stage. The diagram shows the flow of data from the front end through the pipeline stages to the writeback stage.

Figure 1 Cortex-A7 Pipeline

<https://www.anandtech.com/show/8718/the-samsung-galaxy-note-4-exynos-review/3>

<https://arstechnica.com/gadgets/2011/10/arms-new-cortex-a7-is-tailor-made-for-android-superphones/>

Big Ideas

- Data Parallelism easy basis for decomposition
- Data Parallel architectures can be compact – pack more computations onto a chip
 - SIMD, Pipelined
 - Benefit by sharing (instructions)
 - Performance can be brittle
 - Drop from peak as mismatch

Penn ESE532 Fall 2021 -- DeHon

56

Admin

- Remember feedback
- Reading for Day 7 online
- HW3 due Friday
- HW4 out
- Ultra96 distribution

Penn ESE532 Fall 2021 -- DeHon

57