

ESE532: System-on-a-Chip Architecture

Day 6: September 19, 2018
Data-Level Parallelism



Penn ESE532 Fall 2018 -- DeHon

Today

- Data-level Parallelism
 - For Parallel Decomposition
 - Architectures
 - Concepts
 - NEON

Penn ESE532 Fall 2018 -- DeHon

2

Message

- Data Parallelism easy basis for decomposition
- Data Parallel architectures can be compact
 - pack more computations onto a fixed-size IC die
 - OR perform computation in less area

Penn ESE532 Fall 2018 -- DeHon

3

Preclass 1

- 400 news articles
- Count total occurrences of a string
- How can we exploit data-level parallelism on task?
- How much parallelism can we exploit?

Penn ESE532 Fall 2018 -- DeHon

4

Parallel Decomposition

Penn ESE532 Fall 2018 -- 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 2018 -- DeHon

6

Exploit

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

Penn ESE532 Fall 2018 -- DeHon

7

Thread Exploit DP

- How exploit threads for data-parallel text search?

Penn ESE532 Fall 2018 -- DeHon

8

Performance Benefit

- Ideally linear in number of processors (resources)
- Simple: $T_{dp} = N_{data} / P$
- $T_{single} =$ Latency on single data item
- $T_{dp} = \max(T_{single}, N_{data} / P)$

Penn ESE532 Fall 2018 -- DeHon

9

SPMD

- Single Program Multiple Data
- Only need to write code once
 - Get to use many times

Penn ESE532 Fall 2018 -- DeHon

10

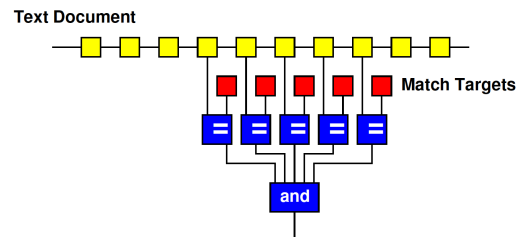
Pipeline Exploit

- How exploit hardware pipeline for text search?

Penn ESE532 Fall 2018 -- DeHon

11

Pipeline Text Search



Penn ESE532 Fall 2018 -- DeHon

12

Common Examples

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

Hardware Architectures

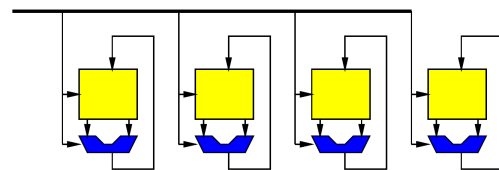
Idea

- 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 die.

SIMD

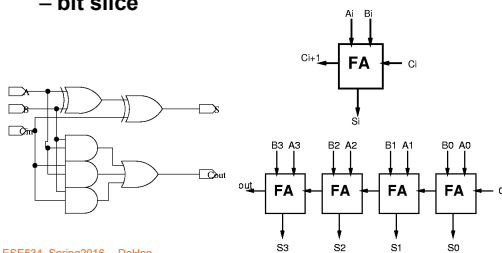
- Single Instruction Multiple Data

Shared Instruction



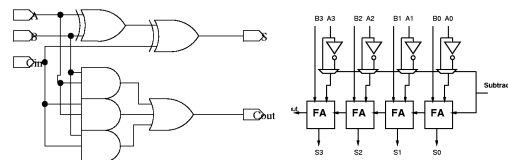
Ripple Carry Addition

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




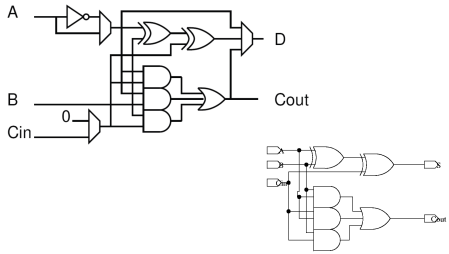
Arithmetic Logic Unit (ALU)

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



Arithmetic and Logic Unit

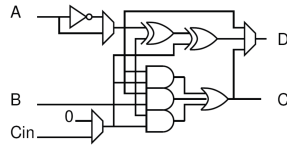




19

Penn ESE534 Fall2016 -- DeHon

ALU Functions



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

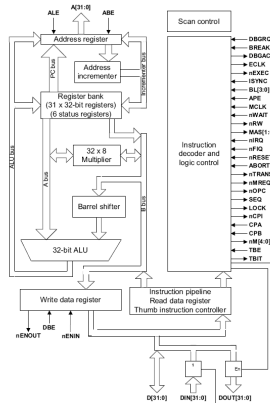
Key observation: every ALU bit does the same thing on different bits of the data word(s).

20

Penn ESE534 Fall2016 -- DeHon

ARM v7 Core

- ALU is Key compute operator in a processor
- Compute
 - ALU
 - Multiplier

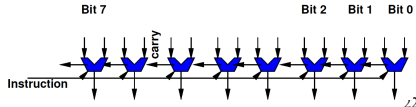


22

Penn ESE532 Fall 2018 -- DeHon

W-bit ALU as SIMD

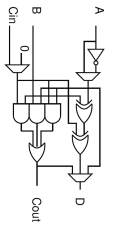
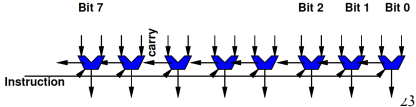
- Familiar idea
- A W-bit ALU (W=8, 16, 32, 64, ...) is SIMD
- Each bit of ALU works on separate bits
 - Performing the same operation on it
 - Trivial to see bitwise AND, OR, XOR
 - Also true for ADD (each bit performing Full Adder)
- Share one instruction across all ALU bits



22

Penn ESE532 Fall 2018 -- DeHon

ALU Bit Slice

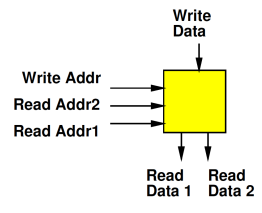



23

Penn ESE532 Fall 2018 -- DeHon

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

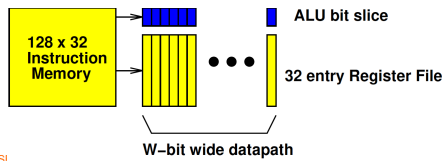


24

Penn ESE532 Fall 2018 -- DeHon

Preclass 2

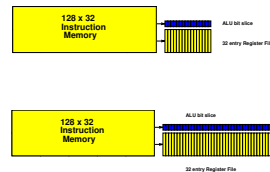
- Area $W=16$?
- Area $W=128$?
- Number in 10^8
 - $W=16$
 - $W=128$
- Perfect Pack Ratio?
- Why?



Penn ESL

25

To Scale Comparison

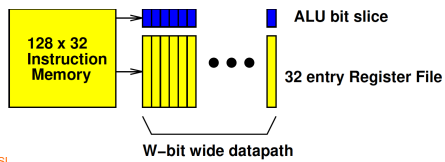


Penn ESE532 Fall 2018 -- DeHon

26

Preclass 2

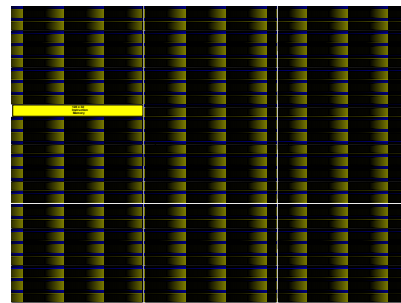
- W for single datapath in 10^8 ?
- Perfect 16b pack ratio?
- Compare $W=128$ perfect pack ratio?



Penn ESL

27

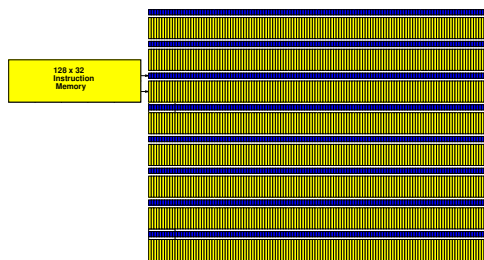
To Scale $W=9088$



Penn ESE532 Fall 2018 -- DeHon

28

To Scale $W=1024$



Penn ESE532 Fall 2018 -- DeHon

29

ALU vs. SIMD ?

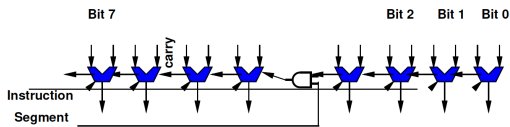
- What's different between
 - 128b wide ALU
 - SIMD datapath supporting eight 16b ALU operations

Penn ESE532 Fall 2018 -- DeHon

30

Segmented Datapath

- Relatively easy (few additional 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 2018 -- DeHon

31

Segmented Datapath

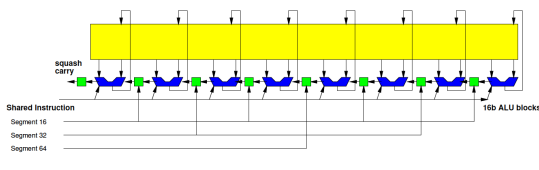
- Relatively easy (few additional gates) to convert a wide datapath into one supporting a set of smaller operations
 - Just need to squash the carry at points
- But need to keep instructions (description) small
 - So typically have limited, homogeneous widths supported

Penn ESE532 Fall 2018 -- DeHon

32

Segmented 128b Datapath

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

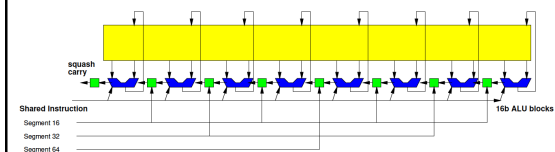


Penn ESE532 Fall 2018 -- DeHon

33

Terminology: Vector Lane

- Each of the separate segments called a **Vector Lane**
- For 16b data, this provides 8 vector lanes

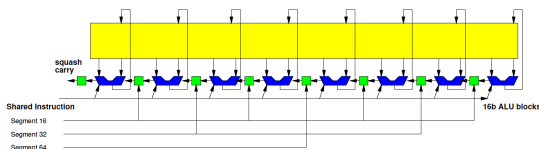


Penn ESE532 Fall 2018 -- DeHon

34

Performance

- Ideally, pack into vector lanes
- $T_{\text{vector}} = N_{\text{op}} / VL$



Penn ESE532 Fall 2018 -- DeHon

35

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 2018 -- DeHon

36

Vector Computation

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

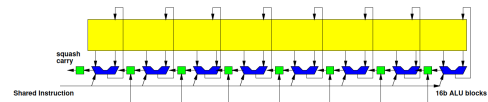
Concepts

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

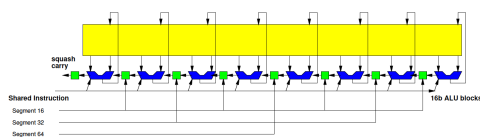
Vector Register File

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



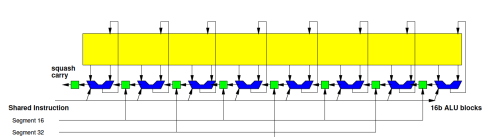
Point-wise Vector Operations

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



Point-wise Vector Operations

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



Ideal

- for (i=0;i<64;i=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 2018 -- DeHon

43

Vector Length

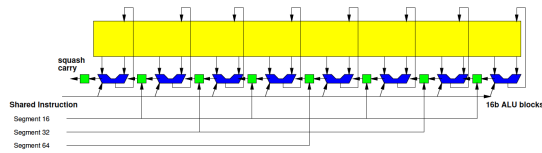
- May not match physical hardware length
- What happens when
 - Vector length > hardware SIMD operators?
 - Vector length < hardware SIMD operators?
 - Vector length % hwd operators !=0
 - E.g. vector length 20, for 8 hwd operators

Penn ESE532 Fall 2018 -- DeHon

44

Performance

- $T_{\text{vector}} = \text{ceil}(N_{\text{op}}/VL) * T_{\text{cycle}}$



Penn ESE532 Fall 2018 -- DeHon

45

Skipping Elements?

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

Penn ESE532 Fall 2018 -- DeHon

46

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 2018 -- DeHon

47

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 2018 -- DeHon

48

Dot Product

- What happens when need a dot product?
- `res=0;`
- `for (i=0;i<N;i++)`
 - `res+=a[i]*b[i]`

Penn ESE532 Fall 2018 -- DeHon

49

Reduction

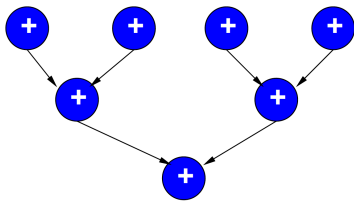
- Common operations where want to perform a combining operation to reduce a vector to a scalar
 - Sum values in vector
 - AND, OR
- Reduce Operation

Penn ESE532 Fall 2018 -- DeHon

50

Reduce Tree

- Efficiently handled with reduce tree

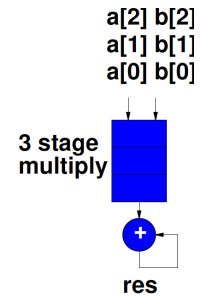


Penn ESE532 Fall 2018 -- DeHon

51

Reduce in Pipeline

- Comes almost for free in pipeline



Penn ESE532 Fall 2018 -- DeHon

52

Vector Reduce Instruction

- Usually include support for vector reduce operation
 - Doesn't need to add much to delay
 - Maybe even faster than performing larger operation
 - 8 16x16 multiplies with sum reduce less complex than one 128x128 multiply
 - ...can exploit datapath of larger operation

Penn ESE532 Fall 2018 -- DeHon

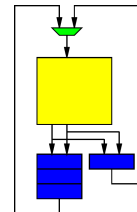
53

Dot Product Revisited

- `for (i=0;i<N;i++)`
 - `res+=a[i]*b[i]`
- a in R0—R4
- b in R4—R7
- With 3 cycle pipelined multiply

- What happens if try to implement dot product as:

- `MPY R0, R4, R14`
- `ADD R14, R15, R15`
- `MPY R1, R5, R14`
- `ADD R14, R15, R15`
- ...

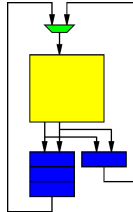


Penn ESE532 Fall 2018 -- DeHon

54

Dot Product Revisited

- for (i=0; i<N; i++)
 - res+=a[i]*b[i]
 - a in R0—R4
 - b in R4—R7
- How should order (reformulate) instructions exploiting data-level parallelism?
- MPY R0, R4, R14
 - ADD R14, R15, R15
 - MPY R1, R5, R14
 - ADD R14, R15, R15

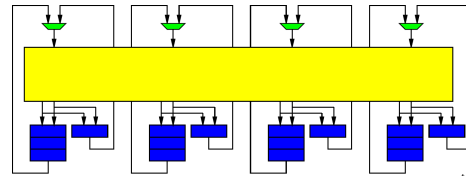


55

Penn ESE532 Fall 2018 -- DeHon

Pipelined Vector Units

- Will get both pipelining and parallel vector lanes
- Exploit data-level parallelism for both

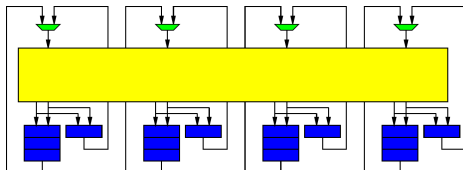


56

Penn ESE532 Fall 2018 -- DeHon

Pipelined Vector Units

$$T_{\text{vector}} \sim \text{ceil}(N_{\text{op}}/VL) * T_{\text{cycle}} + CP * T_{\text{cycle}}$$



57

Penn ESE532 Fall 2018 -- DeHon

Conditionals?

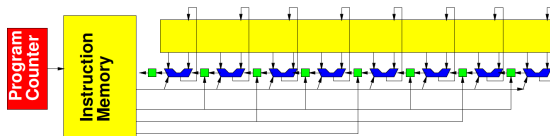
- What happens if want to do something different?
- For (i=0; i<8; i++)
 - if (a[i]<b[i])
 - d[i]=a[i]+c[i]
 - else
 - d[i]=b[i]+c[i]

58

Penn ESE532 Fall 2018 -- DeHon

Conditionals

- Only have one Program Counter
 - Cannot implement conditional via branching

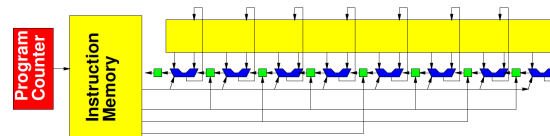


59

Penn ESE532 Fall 2018 -- DeHon

Conditionals

- Only have one instruction
 - Cannot perform separate operations on each ALU in datapath



60

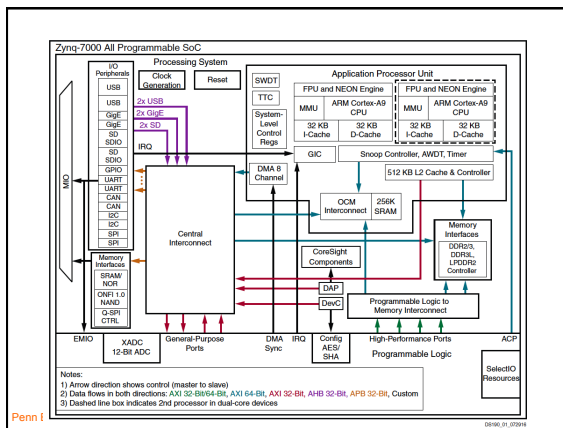
Penn ESE532 Fall 2018 -- DeHon

Conditionals

- Only have one Program Counter
 - Cannot implement conditional via branching
- Only have one instruction
 - Cannot perform separate operations on each ALU in datapath
- Must perform an invariant operation sequence
- Simple answer: prevent using SIMD unit
- Avoid when possible: max, min, ...
- Better: ...later in course

Neon

ARM Vector Accelerator on Zynq



Neon Vector

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

Sample Instructions

- 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

Neon Notes

- Didn't see
 - Vector-wide reduce operation
- Do need to think about operations being pipelined within lanes

ARM Cortex A-7 Pipeline

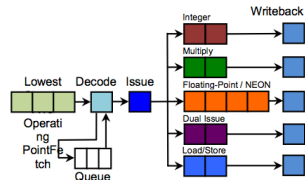


Figure 1 Cortex-A7 Pipeline

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

Penn ESE532 Fall 2018 -- DeHon

67

ARM Cortex A-15 Pipeline

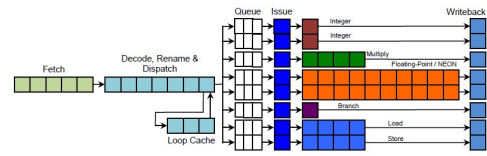


Figure 2 Cortex-A15 Pipeline

<https://community.arm.com/processors/t/discussions/4271/out-of-order-for-cortex-a15>

(expect A9 is somewhere between these two)

Penn ESE532 Fall 2018 -- DeHon

68

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 2018 -- DeHon

69

Admin

- Reading for Day 7 online
- HW3 due Friday
- HW4 out

Penn ESE532 Fall 2018 -- DeHon

70