WES237B – SU22

Lab3

Jetson TX2 - Processing Components

- Dual-core NVIDIA Denver2 + quad-core ARM Cortex-A57
- 256-core Pascal GPU
- 8GB LPDDR4, 128-bit interface
- 32GB eMMC
- 4kp60 H.264/H.265 encoder and decoder
- Dual ISPs (Image Signal Processors)
- 1.4 Gpps MIPI CSI camera ingest

Jetson TX2 - Denver2 and ARM

- IMPORTANT: we do not use Denver2 cores in our assignments. This is just for explaining the architecture
- Let’s check the CPUs:

quad-core ARM Cortex-A57

dual-core Denver2

```
nvidia@tegra-ubuntu:~/Desktop/jupyter_for_jetson$ lscpu
Architecture:          aarch64
Byte Order:            Little Endian
CPU(s):                6
On-line CPU(s) list:   0,3-5
Off-line CPU(s) list:  1,2
Thread(s) per core:    1
Core(s) per socket:    4
Socket(s):             1
Model name:            ARMv8 Processor rev 3 (v8l)
CPU max MHz:           2035.2000
CPU min MHz:           345.6000
L1d cache:             32K
L1i cache:             48K
L2 cache:              2048K
```

Jetson TX2 - Denver2 and ARM

- IMPORTANT: we do not use Denver2 cores in our assignments. This is just for explaining the architecture
- Let’s check the CPUs
- Query current CPU configuration:
  `sudo nvpmodel -q`
- Check available CPU configurations:
  `cat /etc/nvpmodel.conf`
- Set current CPU configuration:
  `sudo nvpmodel -m <id>`

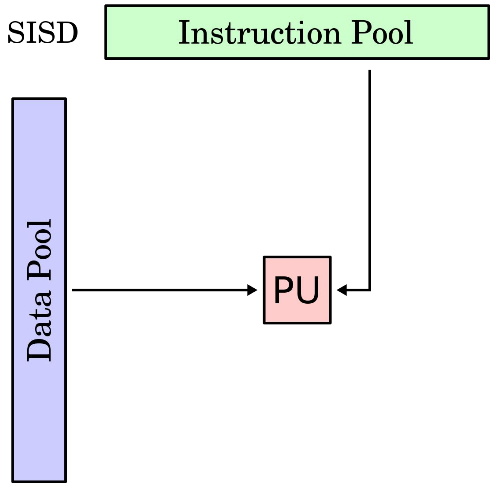
quad-core ARM Cortex-A57

dual-core Denver2

```
nvidia@tegra-ubuntu:~/Desktop/jupyter_for_jetson$ lscpu
Architecture:          aarch64
Byte Order:            Little Endian
CPU(s):                6
On-line CPU(s) list:   0,3-5
Off-line CPU(s) list:  1,2
Thread(s) per core:    1
Core(s) per socket:    4
Socket(s):             1
Model name:            ARMv8 Processor rev 3 (v8l)
CPU max MHz:           2035.2000
CPU min MHz:           345.6000
L1d cache:             32K
L1i cache:             48K
L2 cache:              2048K
```

SISD, SIMD, MIMD, & MISD

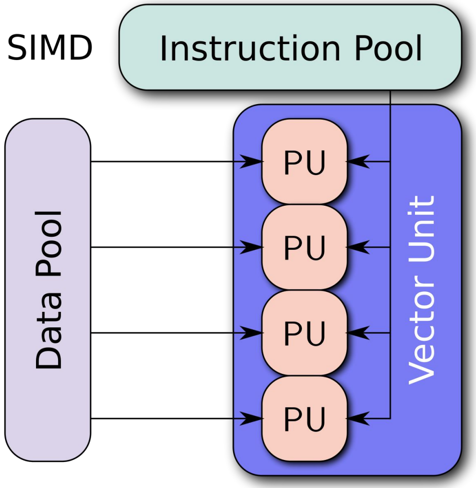
- SISD: Single Instruction, Single Data
  one processor that handles one algorithm using one source of data at a time



SISD, SIMD, MIMD, & MISD

- SISD: Single Instruction, Single Data
  one processor that handles one algorithm using one source of data at a time

- SIMD: Single Instruction, Multiple Data
  several processors that follow the same set of instructions, but each processor inputs different data into those instructions

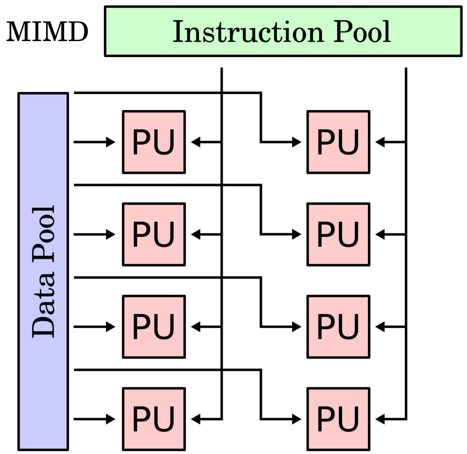


SISD, SIMD, MIMD, & MISD

- SISD: Single Instruction, Single Data
  one processor that handles one algorithm using one source of data at a time

- SIMD: Single Instruction, Multiple Data
  several processors that follow the same set of instructions, but each processor inputs different data into those instructions

- MIMD: Multiple Instructions, Multiple Data
  multiple processors, each capable of accepting its own instruction stream independently from the others. Each processor also pulls data from a separate data stream



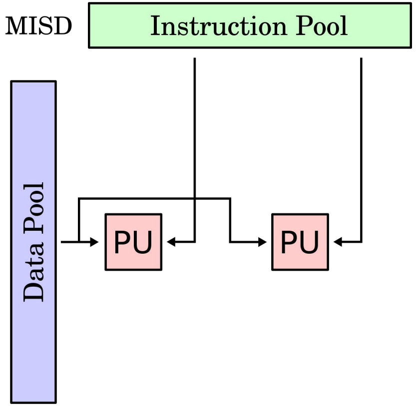
SISD, SIMD, MIMD, & MISD

- SISD: Single Instruction, Single Data
  one processor that handles one algorithm using one source of data at a time

- SIMD: Single Instruction, Multiple Data
  several processors that follow the same set of instructions, but each processor inputs different data into those instructions

- MIMD: Multiple Instructions, Multiple Data
  multiple processors, each capable of accepting its own instruction stream independently from the others. Each processor also pulls data from a separate data stream

- MISD: Multiple Instructions, Single Data
  multiple processors. Each processor uses a different algorithm but uses the same shared input data



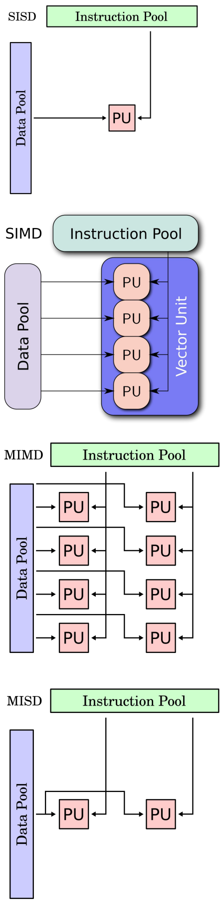
SISD, SIMD, MIMD, & MISD

- SISD: Single Instruction, Single Data
  one processor that handles one algorithm using one source of data at a time

- SIMD: Single Instruction, Multiple Data
  several processors that follow the same set of instructions, but each processor inputs different data into those instructions

- MIMD: Multiple Instructions, Multiple Data
  multiple processors, each capable of accepting its own instruction stream independently from the others. Each processor also pulls data from a separate data stream

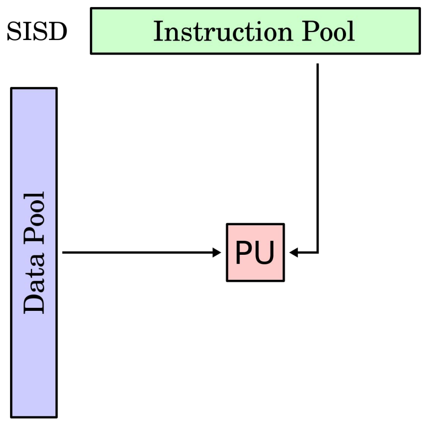
- MISD: Multiple Instructions, Single Data
  multiple processors. Each processor uses a different algorithm but uses the same shared input data



SISD on ARM

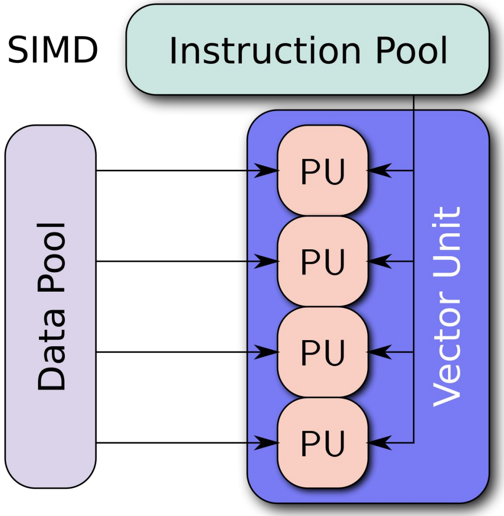
- SISD: Single Instruction, Single Data
  one processor that handles one algorithm using one source of data at a time

- Example: your previous assignment ran sequentially on CPU cores



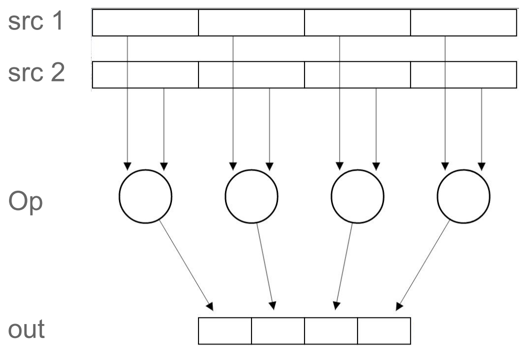
SIMD on ARM

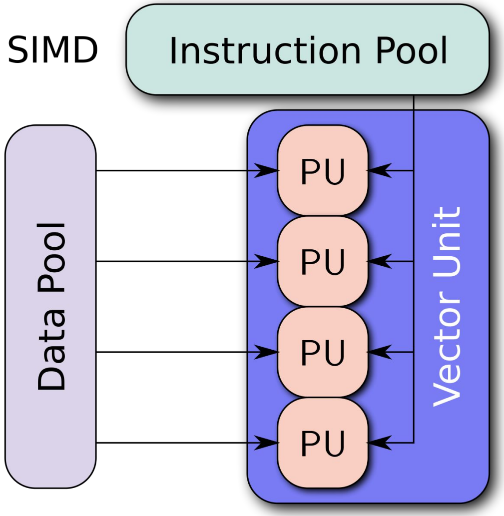
- SIMD: Single Instruction, Multiple Data
  several processors that follow the same set of instructions, but each processor inputs different data into those instructions



SIMD on ARM

- SIMD: Single Instruction, Multiple Data
  several processors that follow the same set of instructions, but each processor inputs different data into those instructions

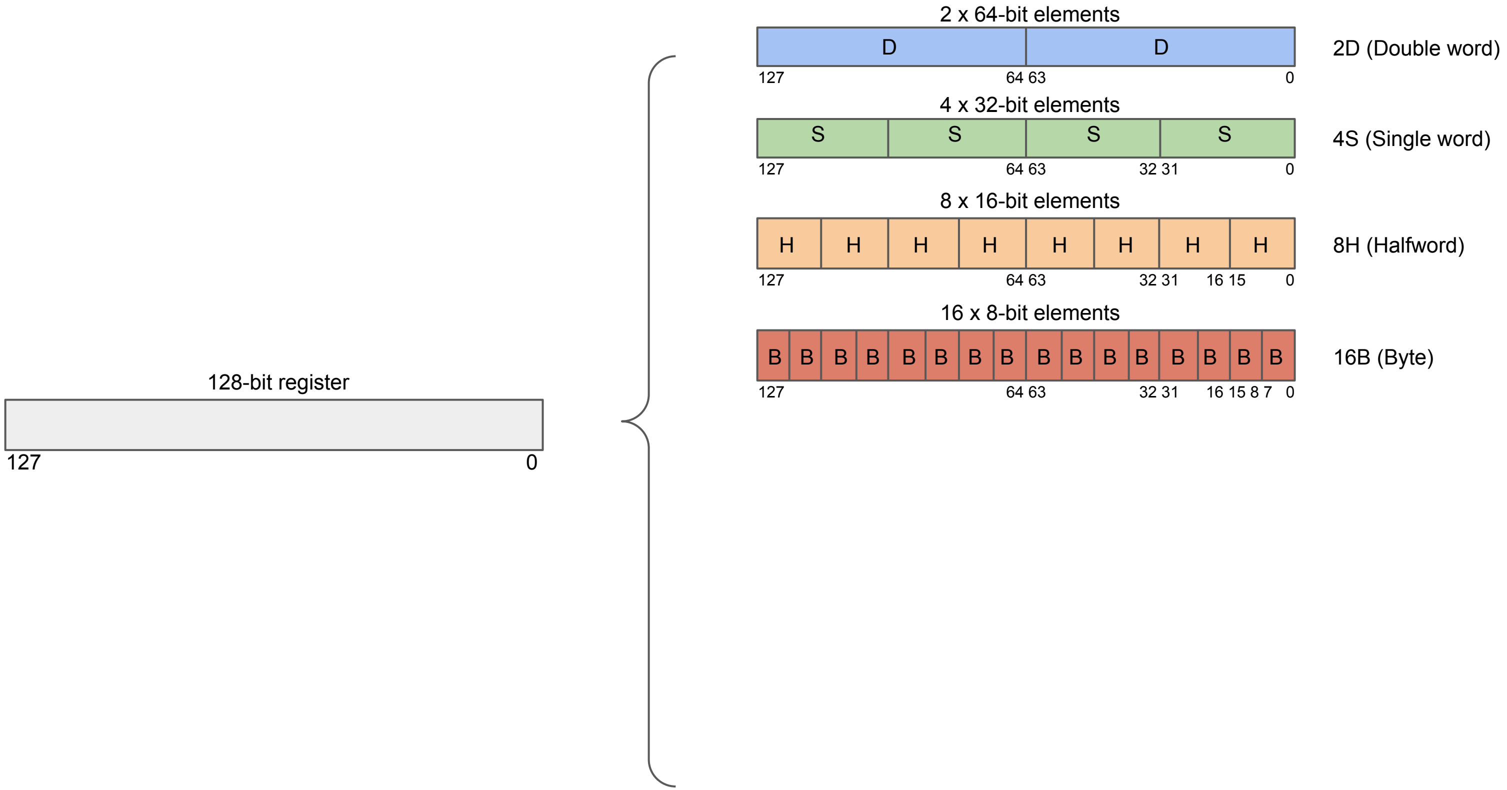




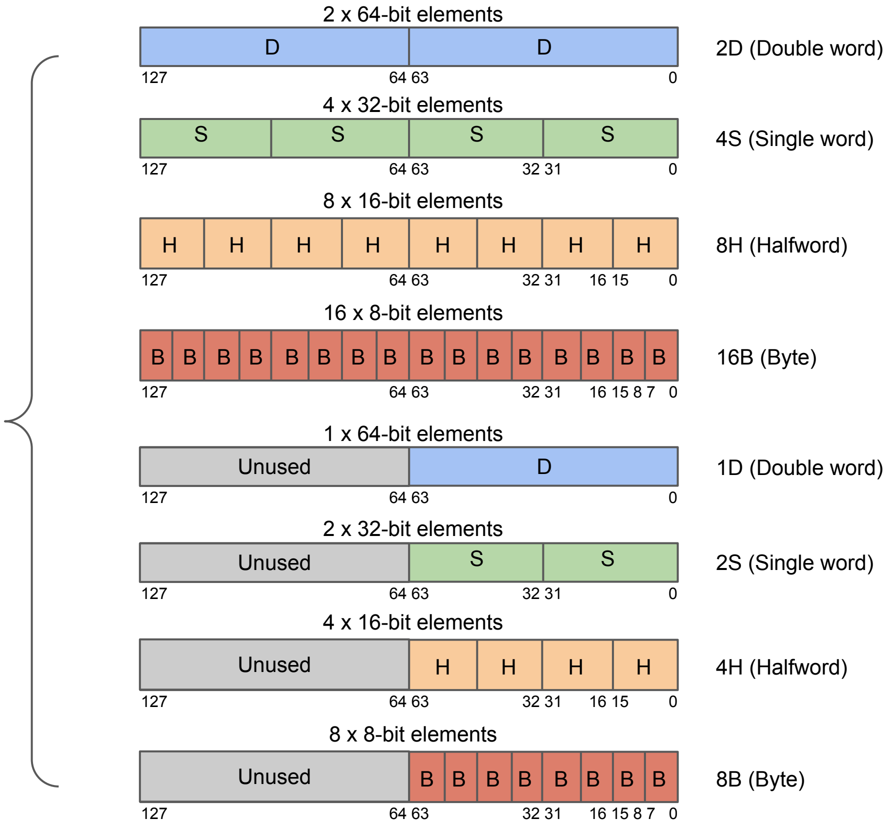
ARM Neon Programming

- ARMv8 Neon Unit:
  - Fully integrated into the processor and uses processor’s resources for loop control, caching, and integer operations
  - Uses 128-bit registers for SIMD processing
  - It’s register file is a collection of registers that can be accessed as (8, 16, 32, 64, 128)-bit registers
  - Registers contain *vector* of elements. The same element position in the input and output registers is referred to as a *lane*
  - Each Neon instruction results in “n” parallel operation, where “n” is the number of lanes

Neon Register and Element Size



Neon Register and Element Size



ARMv8 (Jetson)
32 x 128-bit vector registers
31 x 64-bit general purpose registers

ARMv7 (PYNQ)
16 x 128-bit vector registers

https://developer.arm.com/-/media/Files/pdf/graphics-and-multimedia/ARM_CPU_Architecture.pdf
https://developer.arm.com/architectures/instruction-sets/simd-isas/neon/neon-programmers-guide-for-armv8-a/introducing-neon-for-armv8-a/single-page

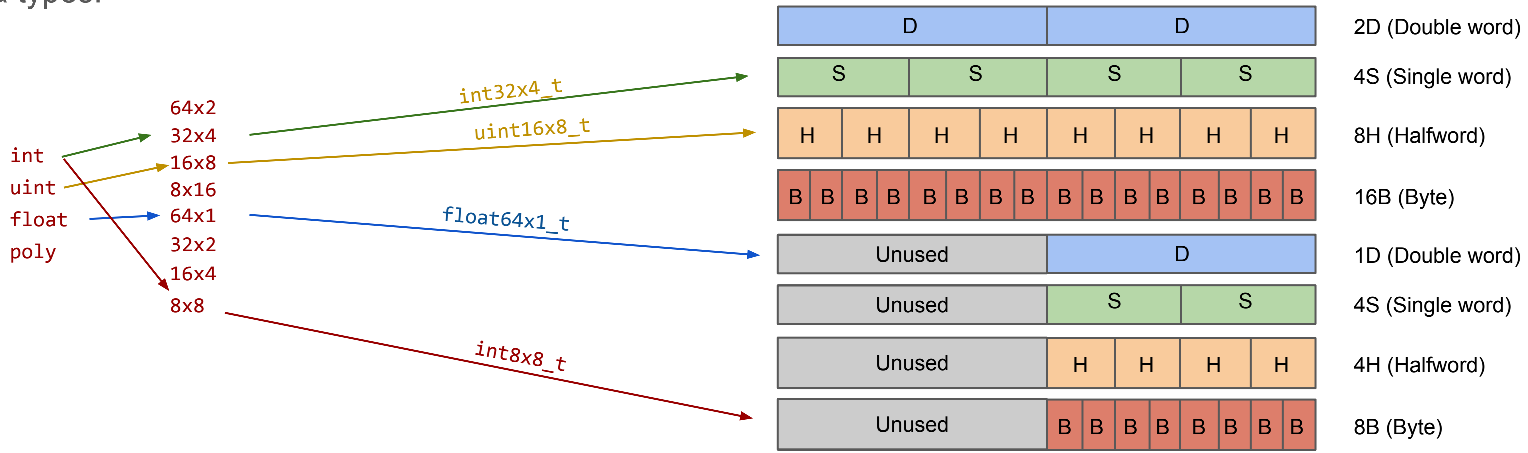
Neon Intrinsics

- Are functions calls that compiler replaces with an (or a sequence of) appropriate Neon instruction(s)
- Functions: https://developer.arm.com/architectures/instruction-sets/simd-isas/neon/intrinsics?page=1

Neon Intrinsics

- Are functions calls that compiler replaces with an (or a sequence of) appropriate Neon instruction(s)
- Functions: https://developer.arm.com/architectures/instruction-sets/simd-isas/neon/intrinsics?page=1

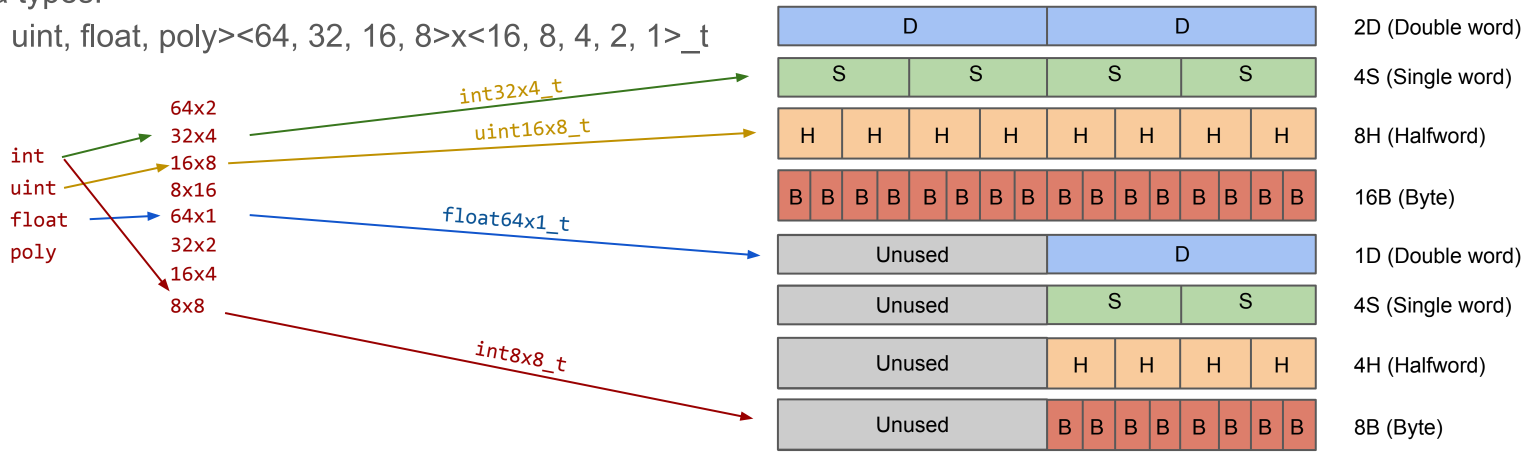
- Data types:



Neon Intrinsics

- Are functions calls that compiler replaces with an (or a sequence of) appropriate Neon instruction(s)
- Functions: https://developer.arm.com/architectures/instruction-sets/simd-isas/neon/intrinsics?page=1

- Data types:
  <int, uint, float, poly><64, 32, 16, 8>x<16, 8, 4, 2, 1>_t



Compile Neon

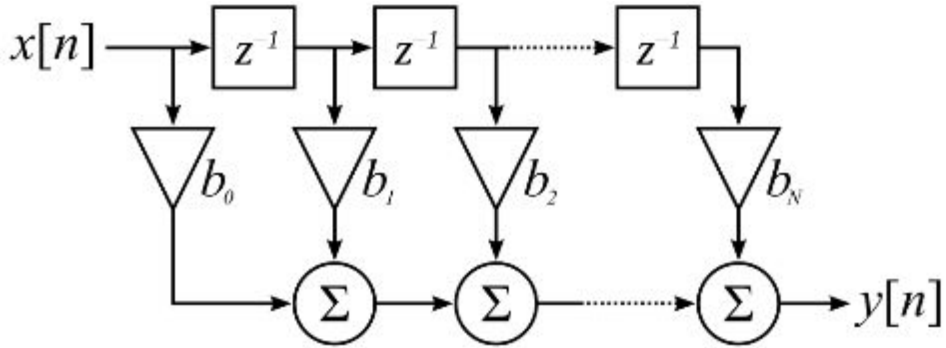
- ARMv7 (PYNQ) requires *-mfpu=neon* and *-O1 -ftree-vectorize*
- ARMv8 (Jetson) requires *-O1 -ftree-vectorize*
- **Lab Work**:
    - Complete *neon.c* (provided)
    - Compile it with: `gcc -mfpu=neon neon.c -o neon`
    - Run: `./neon`

Compile Neon

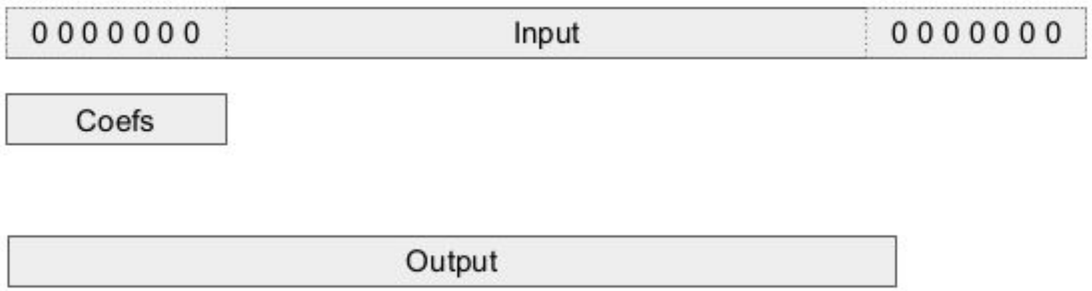
- ARMv7 (PYNQ) requires *-mfpu=neon* and *-O1 -ftree-vectorize*
- ARMv8 (Jetson) requires *-O1 -ftree-vectorize*
- **Lab Work**:
  - Complete *neon.c* (provided)
  - Compile it with: `gcc -mfpu=neon neon.c -o neon`
  - Run: `./neon`
  - Modify the code to add *250* to *data* instead of *3*

FIR Filtering

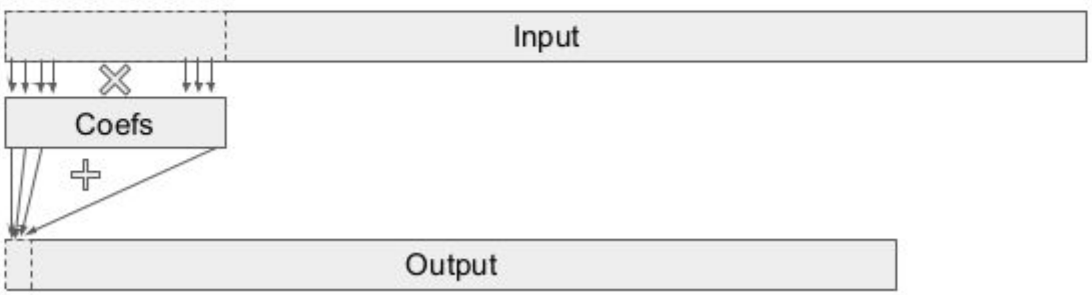
1D Convolution



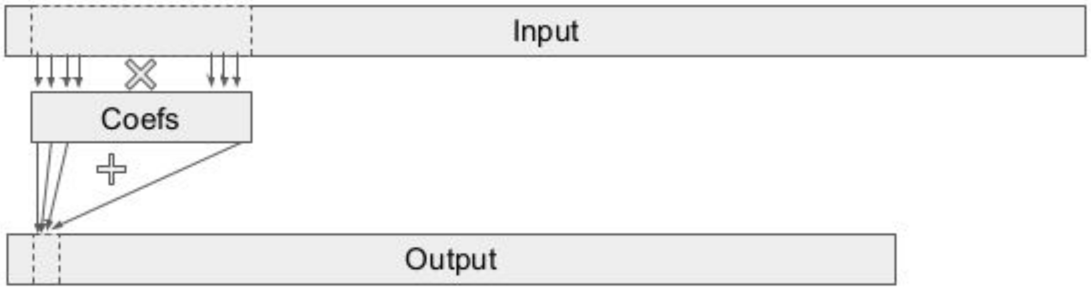
1D Convolution



1D Convolution

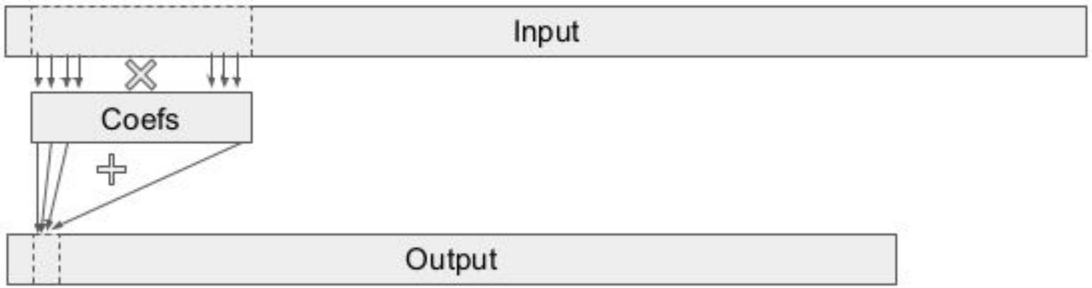


1D Convolution



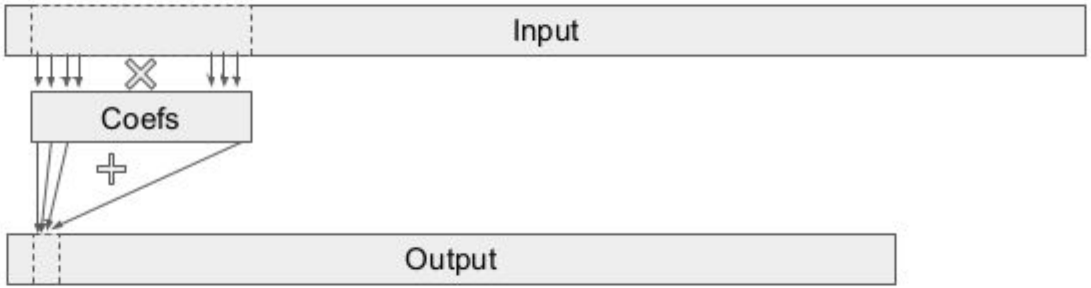
1D Convolution

- 2 nested loops
- Loop through (size of input - size of filter)
  - For each filter coefficient
    - Multiply by the input and accumulate
  - Store result in the output



1D Convolution

- Complete the naive implementation in src/fir.cpp



Loop Unrolling

Unroll coefficient loop (inner loop) by 4:

- Manually duplicate the single line of code
- Increment loop variable by 4

Gprof Profiling

- Profiling tool (like perf), but will provide information on independent function calls within the executable
    - Perf will only provide a cycle count and execution time for the entire executable.
- Compile flag `-pg`
- Running the executable
- There should now be a report `gmon.out` in the directory
- **Make sure you remove the gmon.out if you run the program again.**

Gprof Profiling

- View the report with `gprof -b <EXEC-NAME> gmon.out`
  - For this lab, the command is `gprof -b lab3_fir gmon.out`
- `fir()` takes up 50% of execution time
- `fir_opt()` takes up 42.9% of execution time

```
                        Call graph

granularity: each sample hit covers 4 byte(s) for 1.79% of 0.56 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.04    0.52                 main [1]
                0.28    0.00       1/1           fir(float*, float*, float*, int, int) [2]
                0.24    0.00       1/1           fir_opt(float*, float*, float*, int, int) [3]
                0.00    0.00       3/3           std::sqrt(float) [11]
                0.00    0.00       1/1           designLPF(float*, int, float, float) [16]
-----------------------------------------------
                0.28    0.00       1/1           main [1]
[2]     50.0    0.28    0.00       1         fir(float*, float*, float*, int, int) [2]
-----------------------------------------------
                0.24    0.00       1/1           main [1]
[3]     42.9    0.24    0.00       1         fir_opt(float*, float*, float*, int, int) [3]
-----------------------------------------------
                0.00    0.00       3/3           main [1]
[11]     0.0    0.00    0.00       3         std::sqrt(float) [11]
-----------------------------------------------
                0.00    0.00       1/1           __libc_csu_init [24]
[12]     0.0    0.00    0.00       1         _GLOBAL__sub_I__Z3firPfS_S_ii [12]
                0.00    0.00       1/1           __static_initialization_and_destruction_0(int, int) [14]
-----------------------------------------------
                0.00    0.00       1/1           __libc_csu_init [24]
[13]     0.0    0.00    0.00       1         _GLOBAL__sub_I_main [13]
                0.00    0.00       1/1           __static_initialization_and_destruction_0(int, int) [15]
-----------------------------------------------
                0.00    0.00       1/1           _GLOBAL__sub_I__Z3firPfS_S_ii [12]
[14]     0.0    0.00    0.00       1         __static_initialization_and_destruction_0(int, int) [14]
-----------------------------------------------
                0.00    0.00       1/1           _GLOBAL__sub_I_main [13]
[15]     0.0    0.00    0.00       1         __static_initialization_and_destruction_0(int, int) [15]
-----------------------------------------------
                0.00    0.00       1/1           main [1]
[16]     0.0    0.00    0.00       1         designLPF(float*, int, float, float) [16]
-----------------------------------------------
```

SIMD Instructions

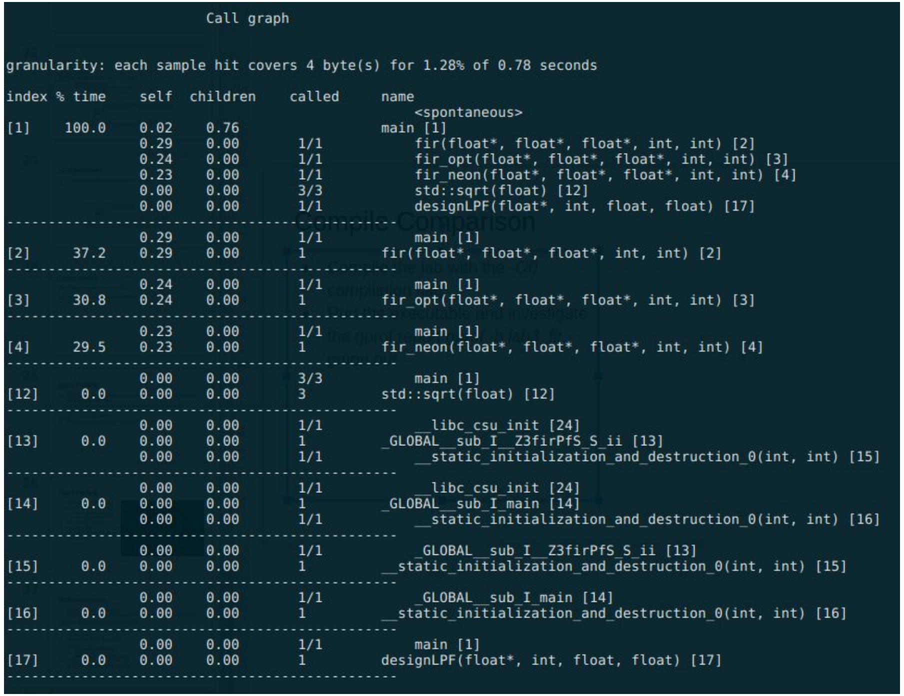
- Include <arm_neon.h> (already done for the lab)
- Add compiler flag: -mfpu=neon (only on PYNQ, not on Jetson. Already done for lab)

Replace the unrolled loop body by NEON Instructions

1. Declare SIMD registers: Use 128-bits SIMD vectors
    a. Float 32-bit x 4
2. Initialize output SIMD vector with 0
3. Inside the loop:
    a. Load input data into SIMD vector
    b. Load coefficients into SIMD vector
    c. Multiply-accumulate into output SIMD vector
4. Add 4 values together then store in output array

Compile Comparison

- Compile the lab with the *-O0* compilation flag
- Run the executable and investigate the gprof report *gprof -b lab3_fir gmon.out*



```
                        Call graph


granularity: each sample hit covers 4 byte(s) for 1.28% of 0.78 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.02    0.76                 main [1]
                0.29    0.00       1/1           fir(float*, float*, float*, int, int) [2]
                0.24    0.00       1/1           fir_opt(float*, float*, float*, int, int) [3]
                0.23    0.00       1/1           fir_neon(float*, float*, float*, int, int) [4]
                0.00    0.00       3/3           std::sqrt(float) [12]
                0.00    0.00       1/1           designLPF(float*, int, float, float) [17]
-----------------------------------------------
                0.29    0.00       1/1           main [1]
[2]     37.2    0.29    0.00       1         fir(float*, float*, float*, int, int) [2]
-----------------------------------------------
                0.24    0.00       1/1           main [1]
[3]     30.8    0.24    0.00       1         fir_opt(float*, float*, float*, int, int) [3]
-----------------------------------------------
                0.23    0.00       1/1           main [1]
[4]     29.5    0.23    0.00       1         fir_neon(float*, float*, float*, int, int) [4]
-----------------------------------------------
                0.00    0.00       3/3           main [1]
[12]     0.0    0.00    0.00       3         std::sqrt(float) [12]
-----------------------------------------------
                0.00    0.00       1/1           __libc_csu_init [24]
[13]     0.0    0.00    0.00       1         _GLOBAL__sub_I__Z3firPfS_S_ii [13]
                0.00    0.00       1/1           __static_initialization_and_destruction_0(int, int) [15]
-----------------------------------------------
                0.00    0.00       1/1           __libc_csu_init [24]
[14]     0.0    0.00    0.00       1         _GLOBAL__sub_I_main [14]
                0.00    0.00       1/1           __static_initialization_and_destruction_0(int, int) [16]
-----------------------------------------------
                0.00    0.00       1/1           _GLOBAL__sub_I__Z3firPfS_S_ii [13]
[15]     0.0    0.00    0.00       1         __static_initialization_and_destruction_0(int, int) [15]
-----------------------------------------------
                0.00    0.00       1/1           _GLOBAL__sub_I_main [14]
[16]     0.0    0.00    0.00       1         __static_initialization_and_destruction_0(int, int) [16]
-----------------------------------------------
                0.00    0.00       1/1           main [1]
[17]     0.0    0.00    0.00       1         designLPF(float*, int, float, float) [17]
-----------------------------------------------
```

Compile Comparison

- Change the compile flag from *-O0* to *-O1*
- Run the executable and investigate the gprof report *gprof -b lab3_fir gmon.out*

https://linux.die.net/man/1/gcc