Parallel and Distributed Programming
Instruction Level Parallelism (How to Approach Peak FLOPS on CPU)

Kenjiro Taura

2026/11/16

Contents

Introduction 2
How to approach peak FLOPS 4
Overcoming latency 12

Introduction

What you need to know to get a nearly peak FLOPS

- SIMD allows parallel operations *within a thread*
- however, SIMD alone cannot approach peak performance of a core
- the other key dimension is ***Instruction Level Parallelism (ILP)***

How to approach peak FLOPS

A simple FMA loop

- measure how fast we can iterate the following loop

```
float a, x, c;
for (i = 0; i < n; i++) {
  x = a * x + c;
}
```

- the code performs `n` FMAs and almost nothing else

```
$ ./fma_scalar 1 10 1000000
```

```
i=0 x=99.999718 core0=0 core1=0 cycles_per_iter=4.000088 ...
```

- it takes almost exactly **4 cycles** per `fmadd`
- throughput: **0.25 FMAs/cycle**

Looking into assembly

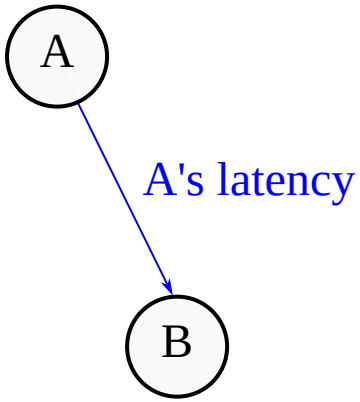
- the assembly shows it indeed does almost nothing else other than `fmadd`
- why 4 cycles?



```
.LBB0_6:
  fmadd s0,s9,s10,s8 // s0 = s9 * s10 + s8
  subs  x8,x8,#8
  fmadd s0,s9,s0,s8
  fmadd s0,s9,s0,s8
  fmadd s0,s9,s0,s8
  fmadd s0,s9,s0,s8
  fmadd s0,s9,s0,s8
  fmadd s0,s9,s0,s8
  fmadd s10,s9,s0,s8
  b.ne  .LBB0_6
```

It is the latency of `fmadd`

- if an output of instruction A becomes an input of B, then
  - ***B depends on A***
  - and B can execute only after the latency of A after A is executed




```
.LBB0_6:                                dependency
 fmadd s0,s9,s10,s8
 subs x8,x8,#8
 fmadd s0,s9,s0,s8
 fmadd s0,s9,s0,s8
 fmadd s0,s9,s0,s8
 fmadd s0,s9,s0,s8
 fmadd s0,s9,s0,s8
 fmadd s0,s9,s0,s8
 fmadd s10,s9,s0,s8
 b.ne .LBB0_6
```

SIMD FMA has the same latency

- do it on four numbers (note: `floatv`), using vector extension

```
floatv x = { ... };
for (i = 0; i < n; i++) {
  x = a * x + c;
}
```

```
./fma_neon 1 10 1000000
  ... cycles_per_iter=4.000267 ...
```

- the latency remains **4 cycles** per `fmla`
- good news: throughput becomes 4x : **0.25 SIMD FMAs/cycle** = **1 FMAs/cycle**
- bad news: still 16x slower than the processor's peak FLOPS (**4 SIMD FMAs/cycle**)

```
.LBB0_7:
  mov  v0.16b,v3.16b        // v0 = b
  subs x8,x8,#8
  fmla v0.4s,v1.4s,v2.s[0] // v0 += x * a
  mov  v1.16b,v3.16b        // v1 = b
  fmla v1.4s,v0.4s,v2.s[0] // v1 += v0 * a
  mov  v0.16b,v3.16b
  fmla v0.4s,v1.4s,v2.s[0]
  mov  v1.16b,v3.16b
  fmla v1.4s,v0.4s,v2.s[0]
  mov  v0.16b,v3.16b
  fmla v0.4s,v1.4s,v2.s[0]
  mov  v1.16b,v3.16b
  fmla v1.4s,v0.4s,v2.s[0]
  mov  v0.16b,v3.16b
  fmla v0.4s,v1.4s,v2.s[0]
  mov  v1.16b,v3.16b
  fmla v1.4s,v0.4s,v2.s[0]
  b.ne .LBB0_7
```

Note: why fmla + mov ?

- generated assembly is more complicated than scalar version
- it is because the compiler uses NEON instruction set, which has only `fmla` (multiply and *accumulate*) instruction for FMA

`fmla p,q,r` $\equiv$ `p += q * r`

- to repeat $x \leftarrow ax + b$ with `fmla`, we have to use extra `mov`

```
x = a x + b   ⇒   x' = a x + b   ⇒   x' = b
x = a x + b       x = a x' + b       x' += a x
                                     x = b
                                     x += a x'
```

```
.LBB0_7:
  mov  v0.16b,v3.16b
  subs x8,x8,#8
  fmla v0.4s,v1.4s,v2.s[0]
  mov  v1.16b,v3.16b
  fmla v1.4s,v0.4s,v2.s[0]
  mov  v0.16b,v3.16b
  fmla v0.4s,v1.4s,v2.s[0]
  mov  v1.16b,v3.16b
  fmla v1.4s,v0.4s,v2.s[0]
  mov  v0.16b,v3.16b
  fmla v0.4s,v1.4s,v2.s[0]
  mov  v1.16b,v3.16b
  fmla v1.4s,v0.4s,v2.s[0]
  mov  v0.16b,v3.16b
  fmla v0.4s,v1.4s,v2.s[0]
  mov  v1.16b,v3.16b
  fmla v1.4s,v0.4s,v2.s[0]
  b.ne .LBB0_7
```

SVE version

- tricky cast `(svfloat32_t)x` lets compiler use SVE, which has `fmadd`

```
floatv x = { ... };
for (i = 0; i < n; i++) {
  x = a * (svfloat32_t)x + c;
}
```

- add `-msve-vector-bits=128` to compile (otherwise the cast is not allowed)

```
$ clang++ -Wall -O3 -mcpu=neoverse-v2 -msve-
vector-bits=128 -o fma_sve ...
```

```
$ ./fma_sve 1 10 1000000
  ... cycles_per_iter=4.000112 ...
```

- the difference becomes important later

```
.LBB0_7:
  fmad  z0.s,p0/m,z2.s,z1.s
  subs  x8,x8,#8
  fmad  z0.s,p0/m,z2.s,z1.s
  fmad  z0.s,p0/m,z2.s,z1.s
  fmad  z0.s,p0/m,z2.s,z1.s
  fmad  z0.s,p0/m,z2.s,z1.s
  fmad  z0.s,p0/m,z2.s,z1.s
  fmad  z0.s,p0/m,z2.s,z1.s
  fmad  z0.s,p0/m,z2.s,z1.s
  b.ne  .LBB0_7
```

A model of superscalar CPU cores

- a virtual core can ***decode*** multiple instructions each cycle

```
ldr    s2, [x0], #4
add    x0, x0, #4
subs   x8, x8, #1
fmadd  s1, s1, s0, s2
b.ne   .LBB0_2
ldr    s2, [x0], #4
add    x0, x0, #4
subs   x8, x8, #1
fmadd  s1, s1, s0, s2
b.ne   .LBB0_2
ldr    s2, [x0], #4
add    x0, x0, #4
```

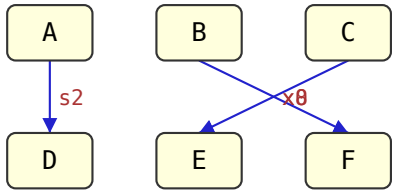


program **instruction window** **execution unit**

A model of superscalar CPU cores

- decoded instructions pile up in ***instruction queue*** and wait for inputs become available

```
A ldr    s2, [x0], #4
B add    x0, x0, #4
C subs   x8, x8, #1
D fmadd  s1, s1, s0, s2
E b.ne   .LBB0_2
F ldr    s2, [x0], #4
G add    x0, x0, #4
H subs   x8, x8, #1
I fmadd  s1, s1, s0, s2
J b.ne   .LBB0_2
K ldr    s2, [x0], #4
L add    x0, x0, #4
```



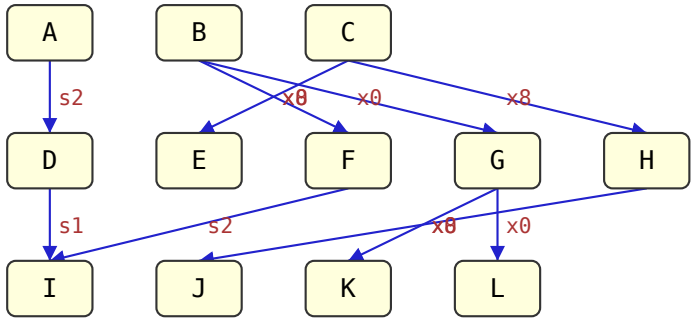


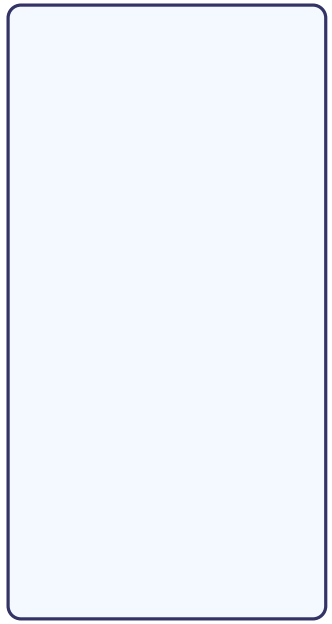
program **instruction window** **execution unit**

A model of superscalar CPU cores

- decoded instructions pile up in ***instruction queue*** and wait for inputs become available

```
A ldr    s2, [x0], #4
B add    x0, x0, #4
C subs   x8, x8, #1
D fmadd  s1, s1, s0, s2
E b.ne   .LBB0_2
F ldr    s2, [x0], #4
G add    x0, x0, #4
H subs   x8, x8, #1
I fmadd  s1, s1, s0, s2
J b.ne   .LBB0_2
K ldr    s2, [x0], #4
L add    x0, x0, #4
```



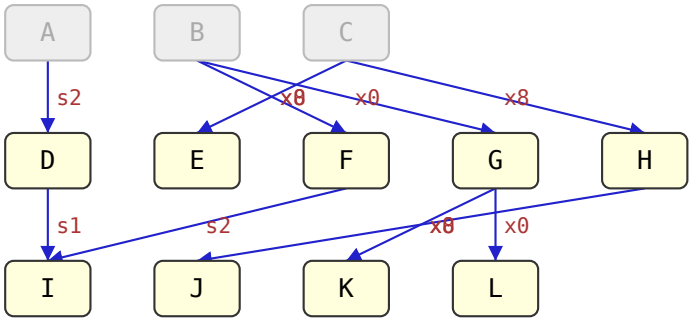


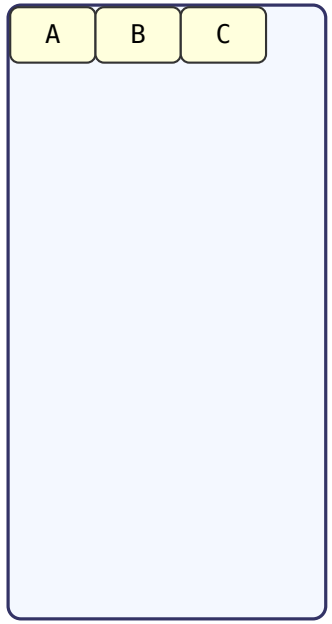
program **instruction window** **execution unit**

A model of superscalar CPU cores

- decoded instructions pile up in ***instruction queue*** and wait for inputs become available

```
ldr    s2, [x0], #4
add    x0, x0, #4
subs   x8, x8, #1
fmadd  s1, s1, s0, s2
b.ne   .LBB0_2
ldr    s2, [x0], #4
add    x0, x0, #4
subs   x8, x8, #1
fmadd  s1, s1, s0, s2
b.ne   .LBB0_2
ldr    s2, [x0], #4
add    x0, x0, #4
```





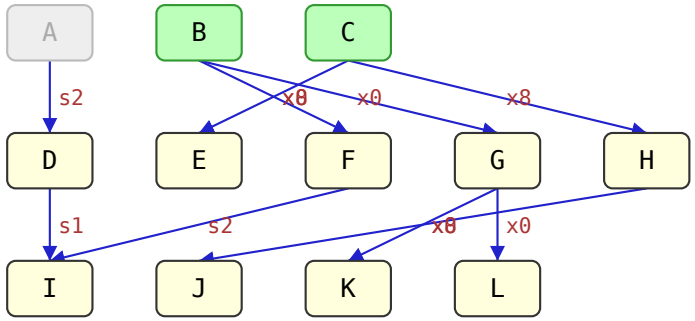
program | **instruction window** | **execution unit**

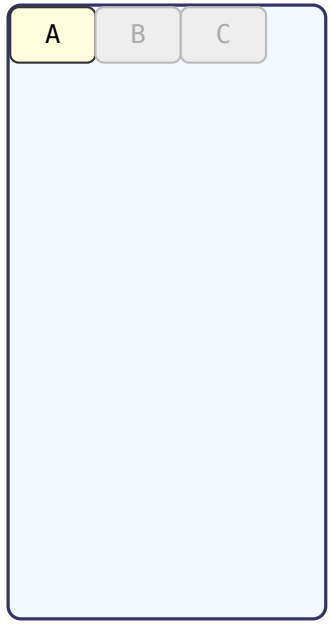
cycle 0

A model of superscalar CPU cores

- decoded instructions pile up in ***instruction queue*** and wait for inputs become available

```
ldr    s2, [x0], #4
add    x0, x0, #4
subs   x8, x8, #1
fmadd  s1, s1, s0, s2
b.ne   .LBB0_2
ldr    s2, [x0], #4
add    x0, x0, #4
subs   x8, x8, #1
fmadd  s1, s1, s0, s2
b.ne   .LBB0_2
ldr    s2, [x0], #4
add    x0, x0, #4
```





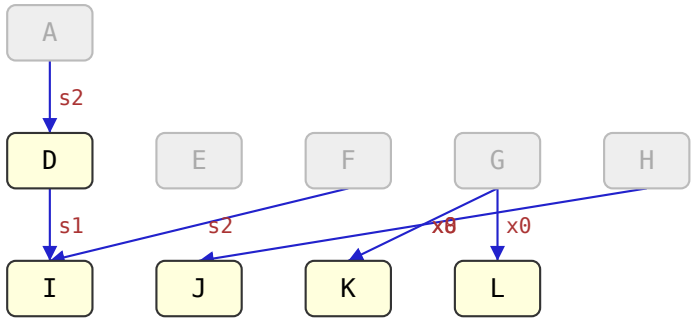
program | **instruction window** | **execution unit**

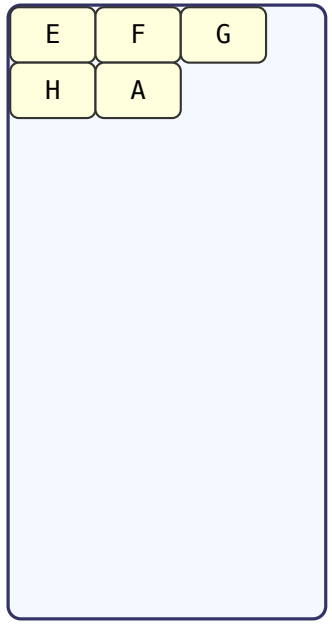
cycle 1

A model of superscalar CPU cores

- decoded instructions pile up in ***instruction queue*** and wait for inputs become available

```
ldr    s2, [x0], #4
add    x0, x0, #4
subs   x8, x8, #1
fmadd  s1, s1, s0, s2
b.ne   .LBB0_2
ldr    s2, [x0], #4
add    x0, x0, #4
subs   x8, x8, #1
fmadd  s1, s1, s0, s2
b.ne   .LBB0_2
ldr    s2, [x0], #4
add    x0, x0, #4
```





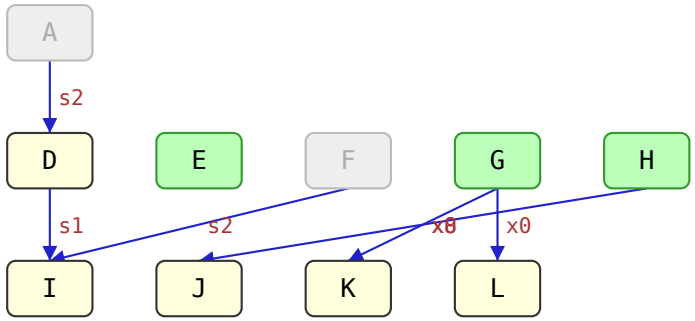
program | **instruction window** | **execution unit**

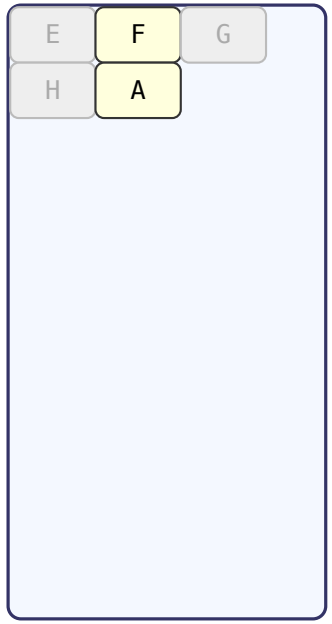
cycle 1

A model of superscalar CPU cores

- instructions are executed once inputs become available (***out of order*** execution)

```
ldr    s2, [x0], #4
add    x0, x0, #4
subs   x8, x8, #1
fmadd  s1, s1, s0, s2
b.ne   .LBB0_2
ldr    s2, [x0], #4
add    x0, x0, #4
subs   x8, x8, #1
fmadd  s1, s1, s0, s2
b.ne   .LBB0_2
ldr    s2, [x0], #4
add    x0, x0, #4
```





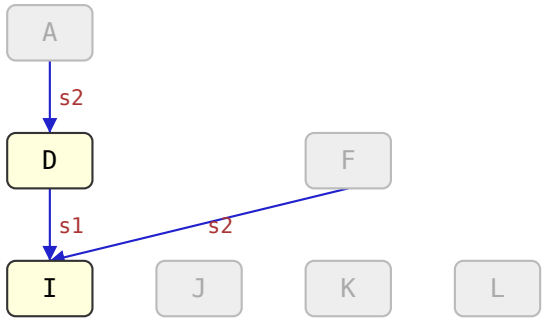
program | **instruction window** | **execution unit**

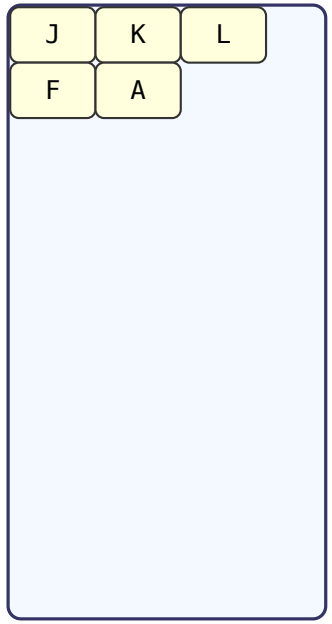
cycle 2

A model of superscalar CPU cores

- instructions are executed once inputs become available (***out of order*** execution)

```
ldr    s2, [x0], #4
add    x0, x0, #4
subs   x8, x8, #1
fmadd  s1, s1, s0, s2
b.ne   .LBB0_2
ldr    s2, [x0], #4
add    x0, x0, #4
subs   x8, x8, #1
fmadd  s1, s1, s0, s2
b.ne   .LBB0_2
ldr    s2, [x0], #4
add    x0, x0, #4
```





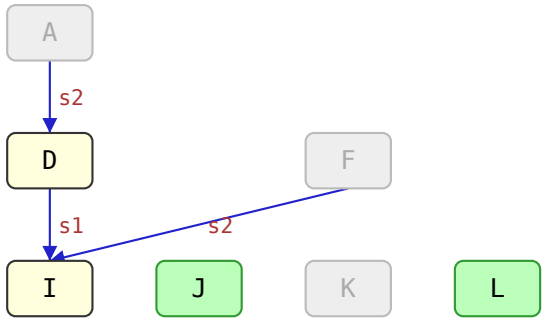
program | **instruction window** | **execution unit**

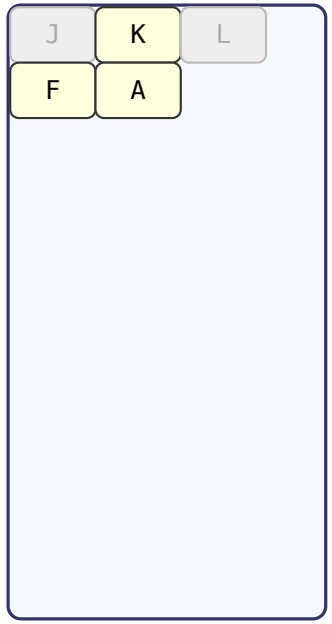
cycle 2

A model of superscalar CPU cores

- instructions are executed once inputs become available (***out of order*** execution)

```
ldr    s2, [x0], #4
add    x0, x0, #4
subs   x8, x8, #1
fmadd  s1, s1, s0, s2
b.ne   .LBB0_2
ldr    s2, [x0], #4
add    x0, x0, #4
subs   x8, x8, #1
fmadd  s1, s1, s0, s2
b.ne   .LBB0_2
ldr    s2, [x0], #4
add    x0, x0, #4
```





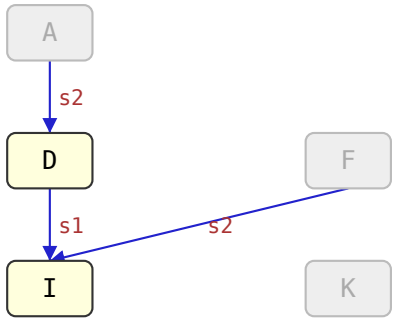
program | **instruction window** | **execution unit**

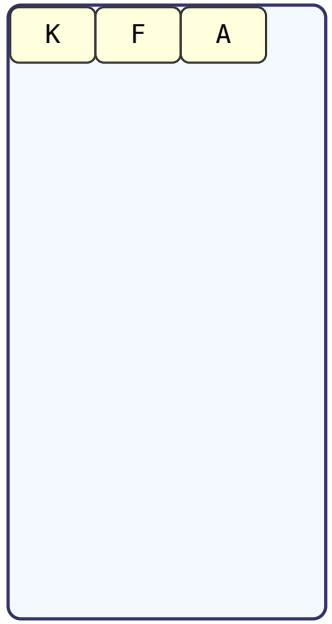
cycle 3

A model of superscalar CPU cores

- instructions are executed once inputs become available (***out of order*** execution)

```
ldr    s2, [x0], #4
add    x0, x0, #4
subs   x8, x8, #1
fmadd  s1, s1, s0, s2
b.ne   .LBB0_2
ldr    s2, [x0], #4
add    x0, x0, #4
subs   x8, x8, #1
fmadd  s1, s1, s0, s2
b.ne   .LBB0_2
ldr    s2, [x0], #4
add    x0, x0, #4
```





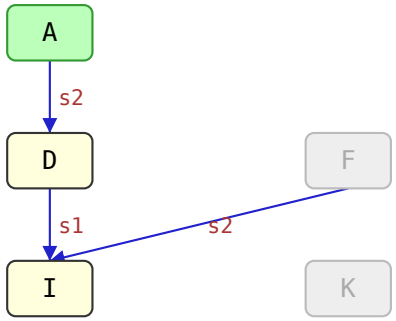
program | **instruction window** | **execution unit**

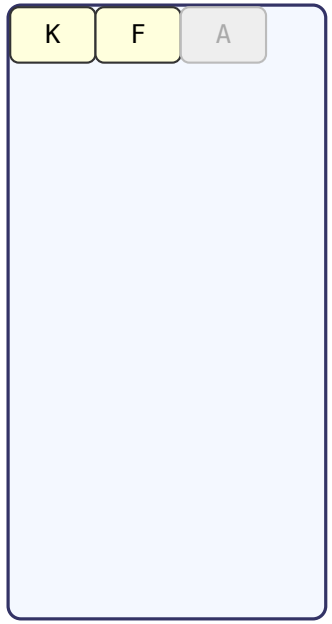
cycle 3

A model of superscalar CPU cores

- instructions are executed once inputs become available (***out of order*** execution)

```
ldr    s2, [x0], #4
add    x0, x0, #4
subs   x8, x8, #1
fmadd  s1, s1, s0, s2
b.ne   .LBB0_2
ldr    s2, [x0], #4
add    x0, x0, #4
subs   x8, x8, #1
fmadd  s1, s1, s0, s2
b.ne   .LBB0_2
ldr    s2, [x0], #4
add    x0, x0, #4
```





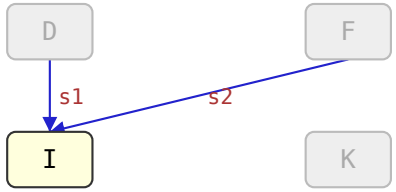
program | **instruction window** | **execution unit**

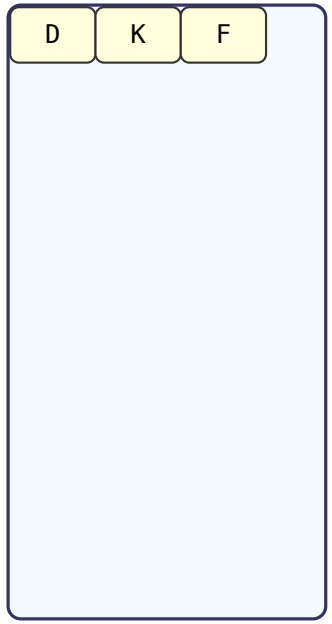
cycle 4

A model of superscalar CPU cores

- instructions are executed once inputs become available (***out of order*** execution)

```
ldr    s2, [x0], #4
add    x0, x0, #4
subs   x8, x8, #1
fmadd  s1, s1, s0, s2
b.ne   .LBB0_2
ldr    s2, [x0], #4
add    x0, x0, #4
subs   x8, x8, #1
fmadd  s1, s1, s0, s2
b.ne   .LBB0_2
ldr    s2, [x0], #4
add    x0, x0, #4
```





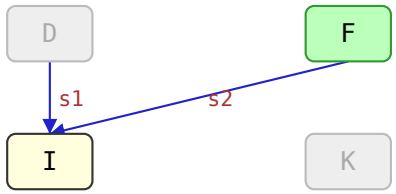
program **instruction window** **execution unit**

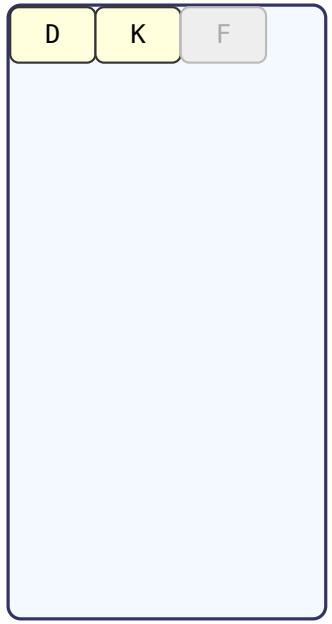
cycle 4

A model of superscalar CPU cores

- instructions are executed once inputs become available (***out of order*** execution)

```
ldr    s2, [x0], #4
add    x0, x0, #4
subs   x8, x8, #1
fmadd  s1, s1, s0, s2
b.ne   .LBB0_2
ldr    s2, [x0], #4
add    x0, x0, #4
subs   x8, x8, #1
fmadd  s1, s1, s0, s2
b.ne   .LBB0_2
ldr    s2, [x0], #4
add    x0, x0, #4
```





program | **instruction window** | **execution unit**

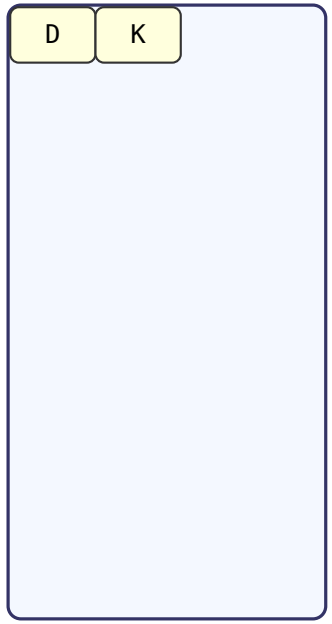
cycle 5

A model of superscalar CPU cores

- instructions are executed once inputs become available (***out of order*** execution)

```
ldr    s2, [x0], #4
add    x0, x0, #4
subs   x8, x8, #1
fmadd  s1, s1, s0, s2
b.ne   .LBB0_2
ldr    s2, [x0], #4
add    x0, x0, #4
subs   x8, x8, #1
fmadd  s1, s1, s0, s2
b.ne   .LBB0_2
ldr    s2, [x0], #4
add    x0, x0, #4
```





program | **instruction window** | **execution unit**

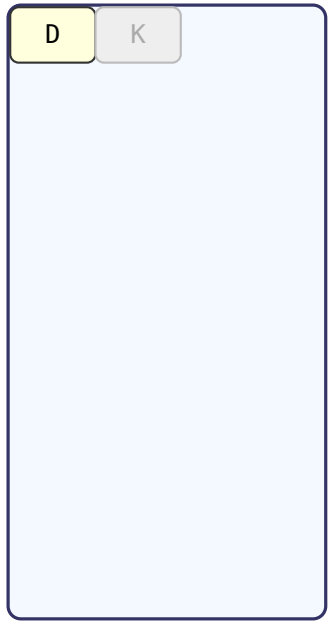
cycle 5

A model of superscalar CPU cores

- instructions are executed once inputs become available (***out of order*** execution)

```
ldr    s2, [x0], #4
add    x0, x0, #4
subs   x8, x8, #1
fmadd  s1, s1, s0, s2
b.ne   .LBB0_2
ldr    s2, [x0], #4
add    x0, x0, #4
subs   x8, x8, #1
fmadd  s1, s1, s0, s2
b.ne   .LBB0_2
ldr    s2, [x0], #4
add    x0, x0, #4
```





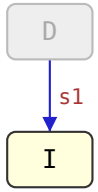
program **instruction window** **execution unit**

cycle 6

A model of superscalar CPU cores

- instructions are executed once inputs become available (***out of order*** execution)

```
ldr    s2, [x0], #4
add    x0, x0, #4
subs   x8, x8, #1
fmadd  s1, s1, s0, s2
b.ne   .LBB0_2
ldr    s2, [x0], #4
add    x0, x0, #4
subs   x8, x8, #1
fmadd  s1, s1, s0, s2
b.ne   .LBB0_2
ldr    s2, [x0], #4
add    x0, x0, #4
```





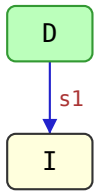
program | **instruction window** | **execution unit**

cycle 6

A model of superscalar CPU cores

- instructions are executed once inputs become available (***out of order*** execution)

```
ldr    s2, [x0], #4
add    x0, x0, #4
subs   x8, x8, #1
fmadd  s1, s1, s0, s2
b.ne   .LBB0_2
ldr    s2, [x0], #4
add    x0, x0, #4
subs   x8, x8, #1
fmadd  s1, s1, s0, s2
b.ne   .LBB0_2
ldr    s2, [x0], #4
add    x0, x0, #4
```





program | **instruction window** | **execution unit**

cycle 8

A model of superscalar CPU cores

- instructions are executed once inputs become available (***out of order*** execution)

```
ldr    s2, [x0], #4
add    x0, x0, #4
subs   x8, x8, #1
fmadd  s1, s1, s0, s2
b.ne   .LBB0_2
ldr    s2, [x0], #4
add    x0, x0, #4
subs   x8, x8, #1
fmadd  s1, s1, s0, s2
b.ne   .LBB0_2
ldr    s2, [x0], #4
add    x0, x0, #4
```





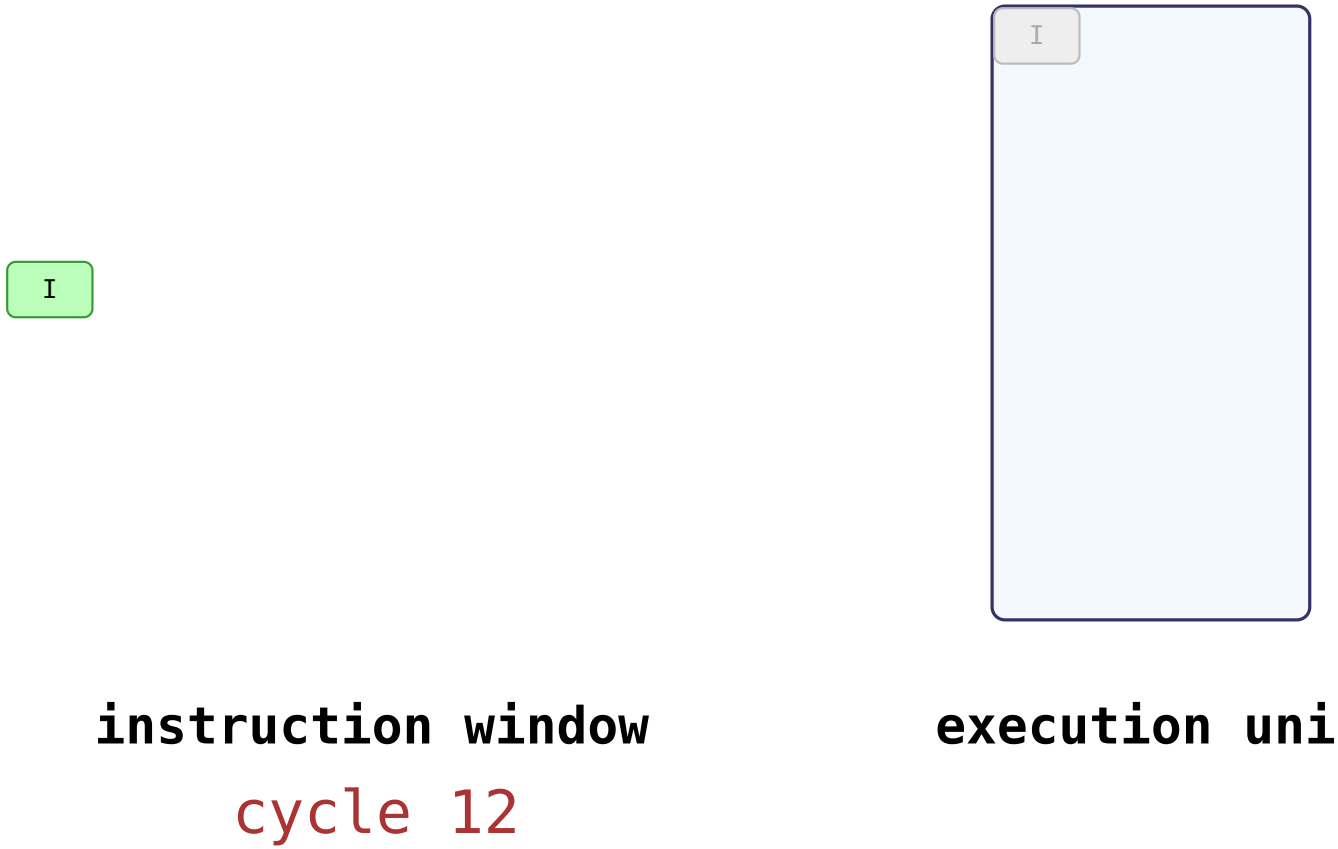
program **instruction window** **execution unit**

cycle 8

A model of superscalar CPU cores

- instructions are executed once inputs become available (***out of order*** execution)

```
ldr    s2, [x0], #4
add    x0, x0, #4
subs   x8, x8, #1
fmadd  s1, s1, s0, s2
b.ne   .LBB0_2
ldr    s2, [x0], #4
add    x0, x0, #4
subs   x8, x8, #1
fmadd  s1, s1, s0, s2
b.ne   .LBB0_2
ldr    s2, [x0], #4
add    x0, x0, #4
```



program **instruction window** **execution unit**

cycle 12

A model of superscalar CPU cores

- instructions are executed once inputs become available (***out of order*** execution)

```
ldr    s2, [x0], #4
add    x0, x0, #4
subs   x8, x8, #1
fmadd  s1, s1, s0, s2
b.ne   .LBB0_2
ldr    s2, [x0], #4
add    x0, x0, #4
subs   x8, x8, #1
fmadd  s1, s1, s0, s2
b.ne   .LBB0_2
ldr    s2, [x0], #4
add    x0, x0, #4
```



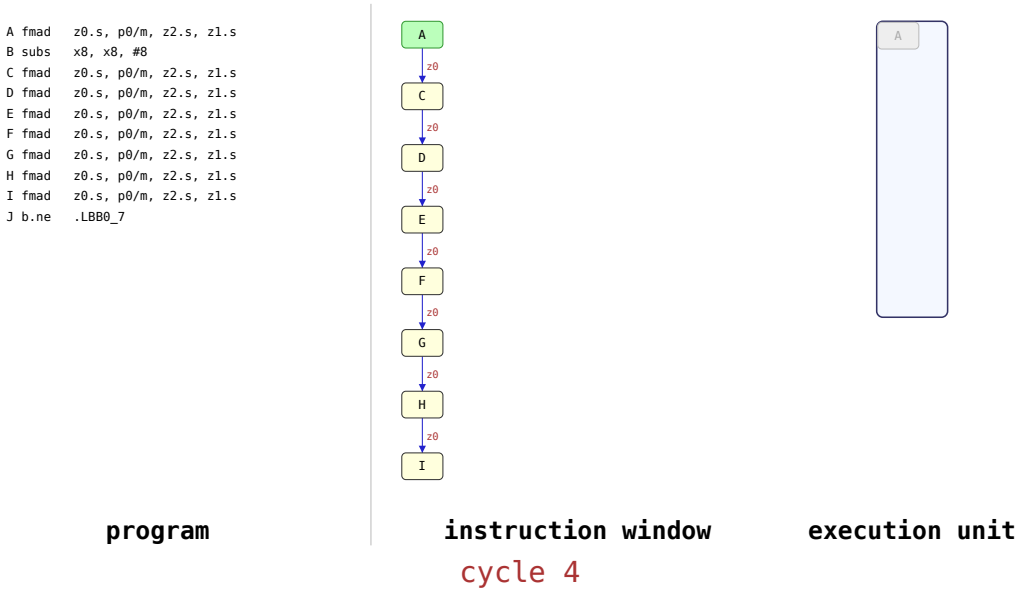
program | **instruction window** | **execution unit**

cycle 12

Overcoming latency

So how to improve *performance*?

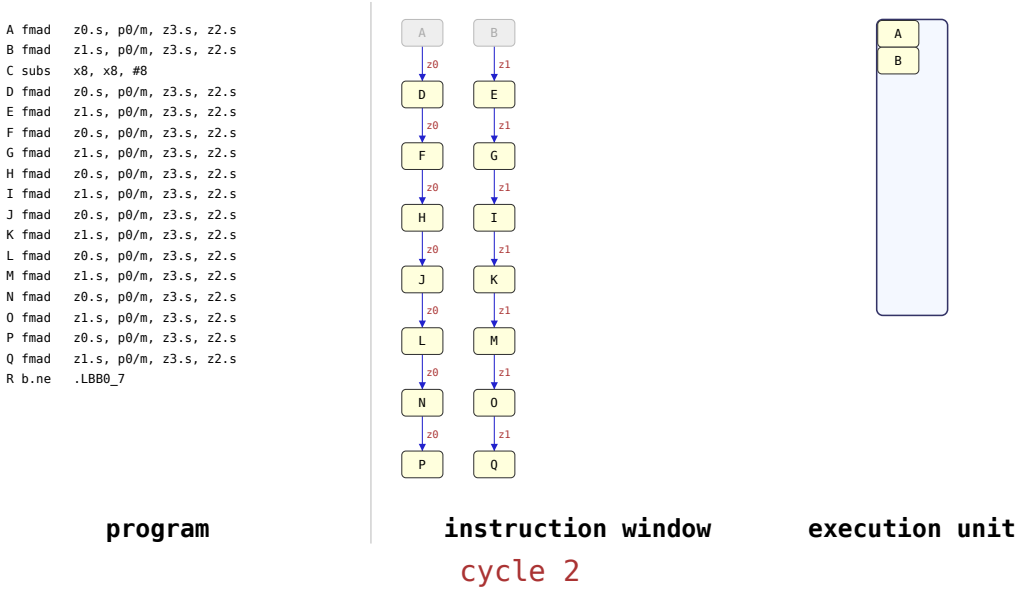
- increase parallelism (no other ways)!
- we ***can't*** make a serial chain of dependent computation run faster than determined by latencies



So how to improve *performance*?

- we can only increase ***throughput***, by running multiple ***independent*** chains
- we expect that the following will finish in the same number of cycles as the original one despite it performs twice as many flops

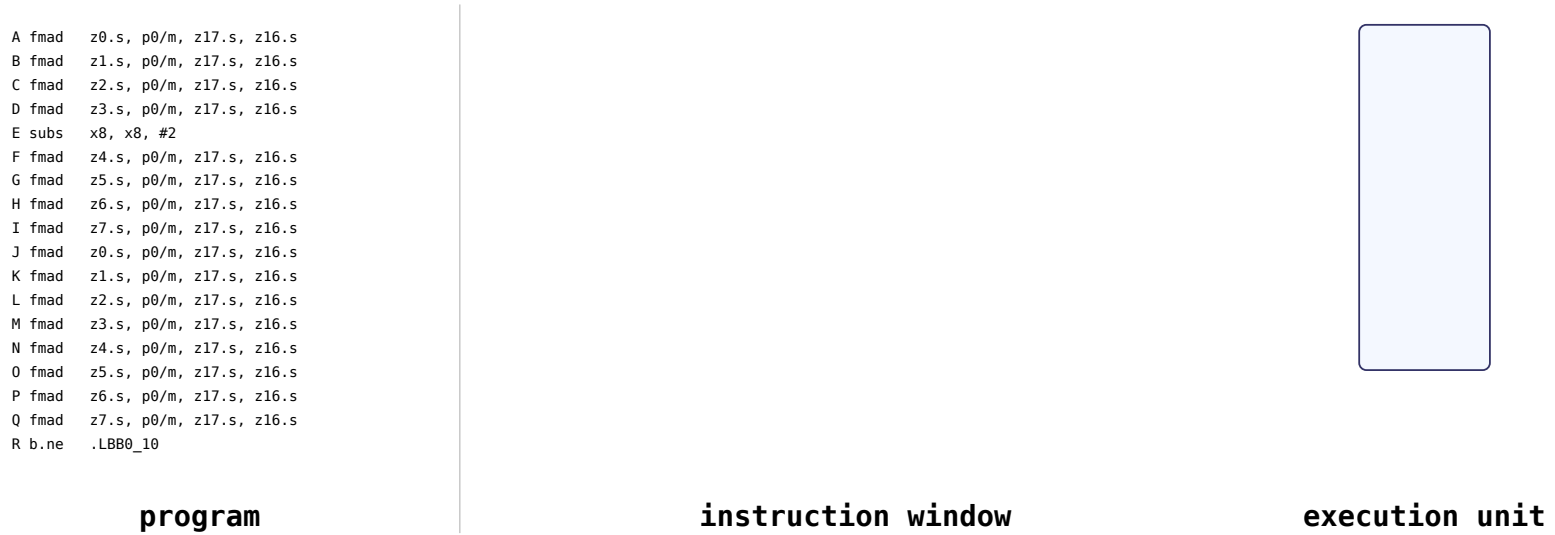
```
floatv x0, x1;
for (i = 0; i < n; i++) {
  x0 = a * (svfloat32_t)x0 + c;
  x1 = a * (svfloat32_t)x1 + c;
}
```



Increase the number of chains further …

- we expect the following to reach peak FLOPS with sufficiently large C

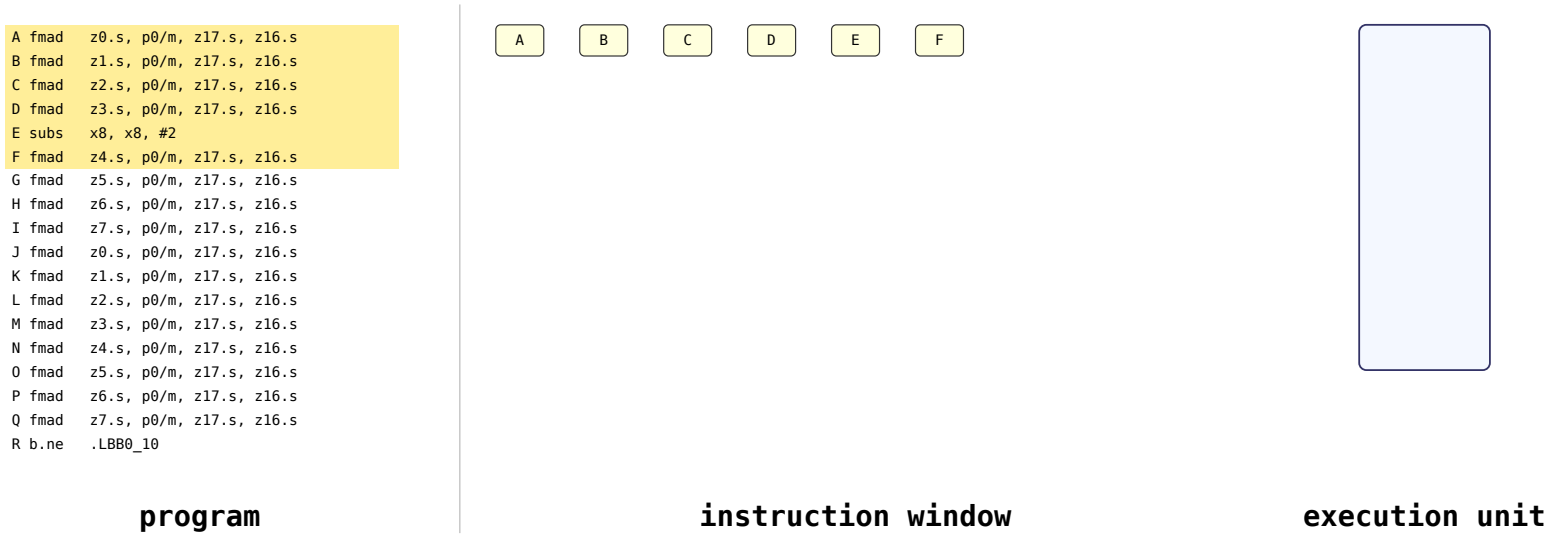
```
floatv x[C/L];
for (long k = 0; k < n; k++)
  for (int c = 0; c < C/L; c++)
    x[c] = a * (svfloat32_t)x[c] + b;
```



Increase the number of chains further …

- we expect the following to reach peak FLOPS with sufficiently large C

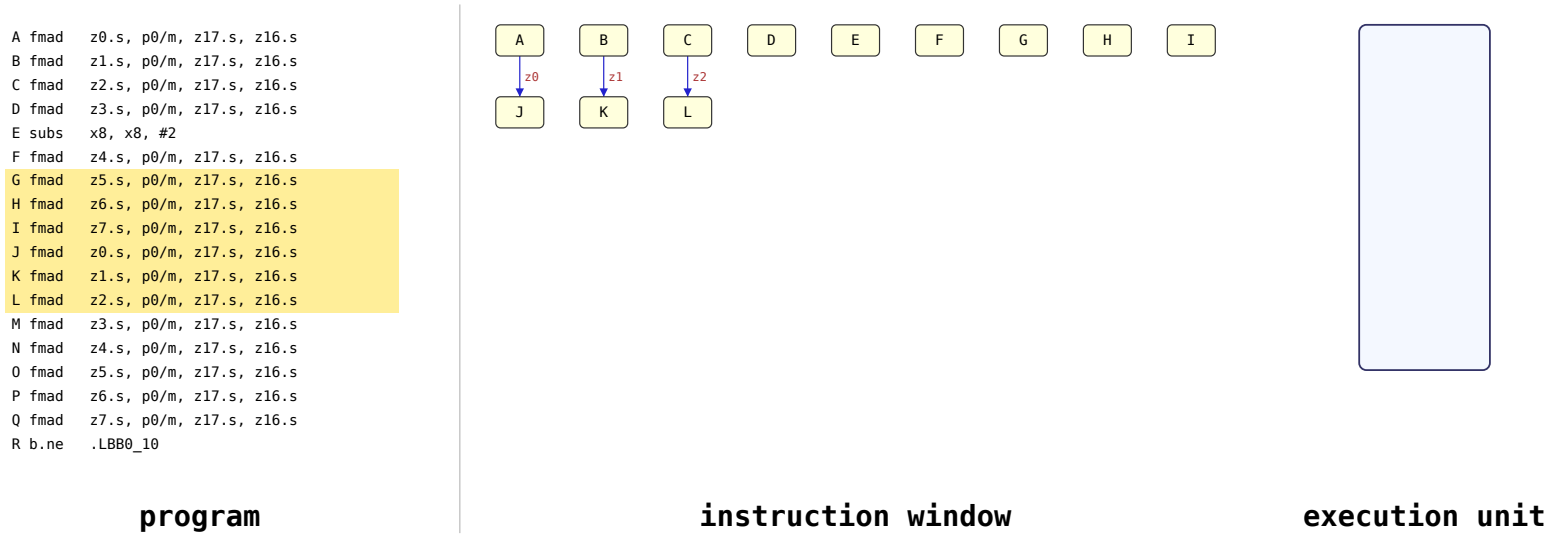
```
floatv x[C/L];
for (long k = 0; k < n; k++)
  for (int c = 0; c < C/L; c++)
    x[c] = a * (svfloat32_t)x[c] + b;
```



Increase the number of chains further …

- we expect the following to reach peak FLOPS with sufficiently large C

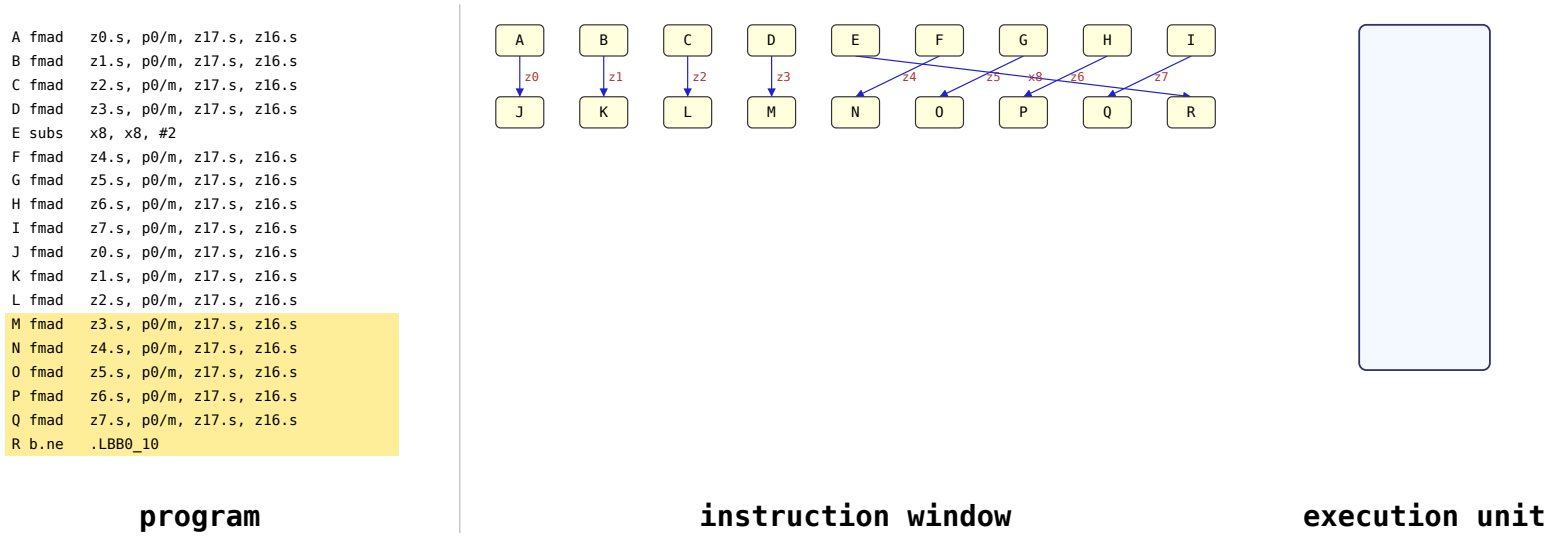
```
floatv x[C/L];
for (long k = 0; k < n; k++)
  for (int c = 0; c < C/L; c++)
    x[c] = a * (svfloat32_t)x[c] + b;
```



Increase the number of chains further …

- we expect the following to reach peak FLOPS with sufficiently large C

```
floatv x[C/L];
for (long k = 0; k < n; k++)
  for (int c = 0; c < C/L; c++)
    x[c] = a * (svfloat32_t)x[c] + b;
```

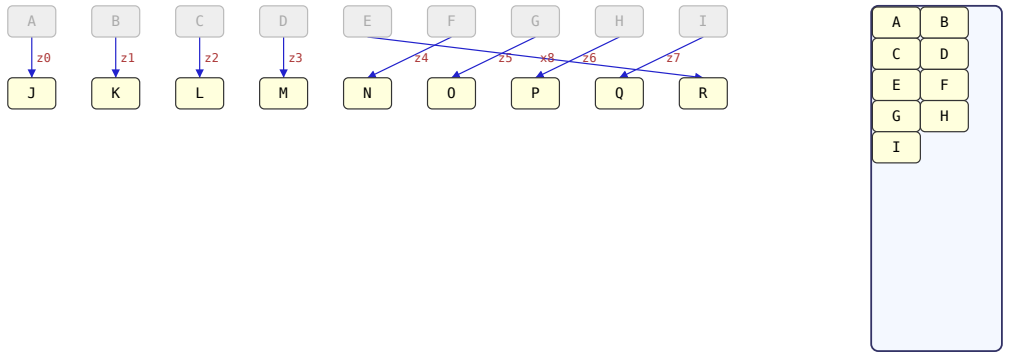


Increase the number of chains further …

- we expect the following to reach peak FLOPS with sufficiently large C

```
floatv x[C/L];
for (long k = 0; k < n; k++)
  for (int c = 0; c < C/L; c++)
    x[c] = a * (svfloat32_t)x[c] + b;
```

```
A fmad   z0.s, p0/m, z17.s, z16.s
B fmad   z1.s, p0/m, z17.s, z16.s
C fmad   z2.s, p0/m, z17.s, z16.s
D fmad   z3.s, p0/m, z17.s, z16.s
E subs   x8, x8, #2
F fmad   z4.s, p0/m, z17.s, z16.s
G fmad   z5.s, p0/m, z17.s, z16.s
H fmad   z6.s, p0/m, z17.s, z16.s
I fmad   z7.s, p0/m, z17.s, z16.s
J fmad   z0.s, p0/m, z17.s, z16.s
K fmad   z1.s, p0/m, z17.s, z16.s
L fmad   z2.s, p0/m, z17.s, z16.s
M fmad   z3.s, p0/m, z17.s, z16.s
N fmad   z4.s, p0/m, z17.s, z16.s
O fmad   z5.s, p0/m, z17.s, z16.s
P fmad   z6.s, p0/m, z17.s, z16.s
Q fmad   z7.s, p0/m, z17.s, z16.s
R b.ne   .LBB0_10
```



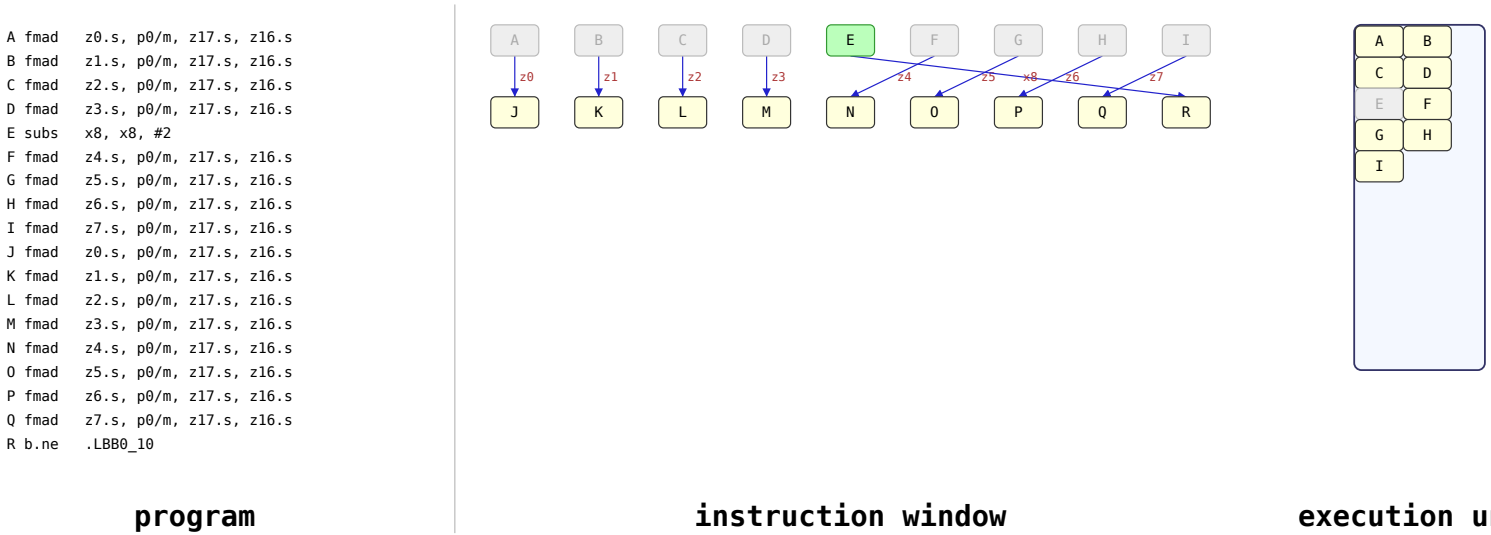
program | **instruction window** | **execution unit**

cycle 0

Increase the number of chains further …

- we expect the following to reach peak FLOPS with sufficiently large C

```
floatv x[C/L];
for (long k = 0; k < n; k++)
  for (int c = 0; c < C/L; c++)
    x[c] = a * (svfloat32_t)x[c] + b;
```

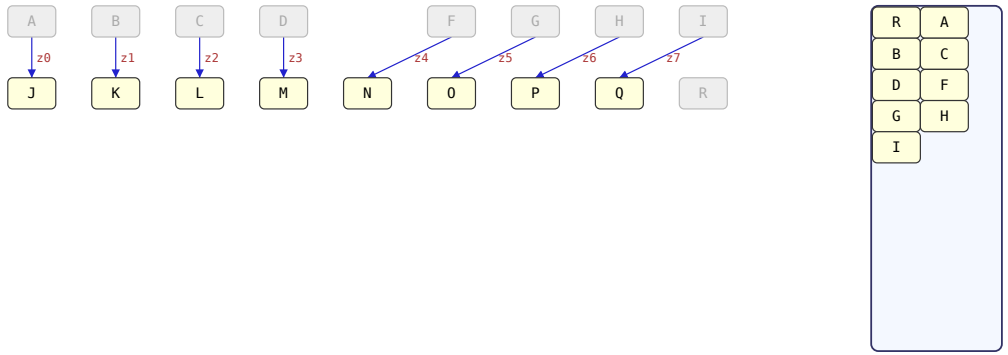


Increase the number of chains further …

- we expect the following to reach peak FLOPS with sufficiently large C

```
floatv x[C/L];
for (long k = 0; k < n; k++)
  for (int c = 0; c < C/L; c++)
    x[c] = a * (svfloat32_t)x[c] + b;
```

```
A fmad   z0.s, p0/m, z17.s, z16.s
B fmad   z1.s, p0/m, z17.s, z16.s
C fmad   z2.s, p0/m, z17.s, z16.s
D fmad   z3.s, p0/m, z17.s, z16.s
E subs   x8, x8, #2
F fmad   z4.s, p0/m, z17.s, z16.s
G fmad   z5.s, p0/m, z17.s, z16.s
H fmad   z6.s, p0/m, z17.s, z16.s
I fmad   z7.s, p0/m, z17.s, z16.s
J fmad   z0.s, p0/m, z17.s, z16.s
K fmad   z1.s, p0/m, z17.s, z16.s
L fmad   z2.s, p0/m, z17.s, z16.s
M fmad   z3.s, p0/m, z17.s, z16.s
N fmad   z4.s, p0/m, z17.s, z16.s
O fmad   z5.s, p0/m, z17.s, z16.s
P fmad   z6.s, p0/m, z17.s, z16.s
Q fmad   z7.s, p0/m, z17.s, z16.s
R b.ne   .LBB0_10
```



program **instruction window** **execution unit**

cycle 1

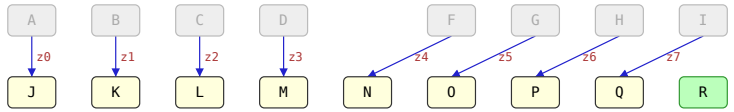
Increase the number of chains further …

- we expect the following to reach peak FLOPS with sufficiently large C

```
floatv x[C/L];
for (long k = 0; k < n; k++)
  for (int c = 0; c < C/L; c++)
    x[c] = a * (svfloat32_t)x[c] + b;
```

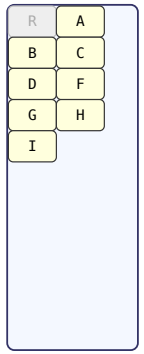
```
A fmad   z0.s, p0/m, z17.s, z16.s
B fmad   z1.s, p0/m, z17.s, z16.s
C fmad   z2.s, p0/m, z17.s, z16.s
D fmad   z3.s, p0/m, z17.s, z16.s
E subs   x8, x8, #2
F fmad   z4.s, p0/m, z17.s, z16.s
G fmad   z5.s, p0/m, z17.s, z16.s
H fmad   z6.s, p0/m, z17.s, z16.s
I fmad   z7.s, p0/m, z17.s, z16.s
J fmad   z0.s, p0/m, z17.s, z16.s
K fmad   z1.s, p0/m, z17.s, z16.s
L fmad   z2.s, p0/m, z17.s, z16.s
M fmad   z3.s, p0/m, z17.s, z16.s
N fmad   z4.s, p0/m, z17.s, z16.s
O fmad   z5.s, p0/m, z17.s, z16.s
P fmad   z6.s, p0/m, z17.s, z16.s
Q fmad   z7.s, p0/m, z17.s, z16.s
R b.ne   .LBB0_10
```

program



instruction window

cycle 2



execution unit

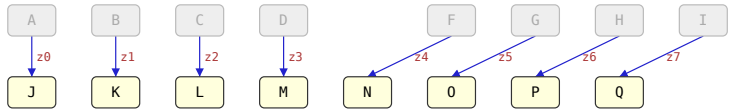
Increase the number of chains further …

- we expect the following to reach peak FLOPS with sufficiently large C

```
floatv x[C/L];
for (long k = 0; k < n; k++)
  for (int c = 0; c < C/L; c++)
    x[c] = a * (svfloat32_t)x[c] + b;
```

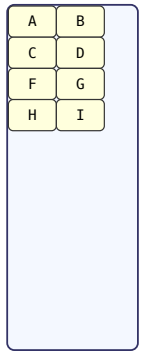
```
A fmad   z0.s, p0/m, z17.s, z16.s
B fmad   z1.s, p0/m, z17.s, z16.s
C fmad   z2.s, p0/m, z17.s, z16.s
D fmad   z3.s, p0/m, z17.s, z16.s
E subs   x8, x8, #2
F fmad   z4.s, p0/m, z17.s, z16.s
G fmad   z5.s, p0/m, z17.s, z16.s
H fmad   z6.s, p0/m, z17.s, z16.s
I fmad   z7.s, p0/m, z17.s, z16.s
J fmad   z0.s, p0/m, z17.s, z16.s
K fmad   z1.s, p0/m, z17.s, z16.s
L fmad   z2.s, p0/m, z17.s, z16.s
M fmad   z3.s, p0/m, z17.s, z16.s
N fmad   z4.s, p0/m, z17.s, z16.s
O fmad   z5.s, p0/m, z17.s, z16.s
P fmad   z6.s, p0/m, z17.s, z16.s
Q fmad   z7.s, p0/m, z17.s, z16.s
R b.ne   .LBB0_10
```

program



instruction window

cycle 2



execution unit

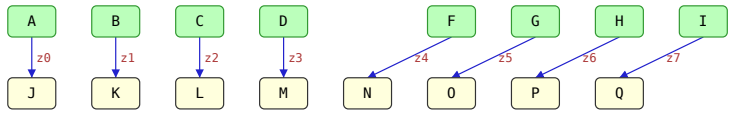
Increase the number of chains further …

- we expect the following to reach peak FLOPS with sufficiently large C

```
floatv x[C/L];
for (long k = 0; k < n; k++)
  for (int c = 0; c < C/L; c++)
    x[c] = a * (svfloat32_t)x[c] + b;
```

```
A fmad   z0.s, p0/m, z17.s, z16.s
B fmad   z1.s, p0/m, z17.s, z16.s
C fmad   z2.s, p0/m, z17.s, z16.s
D fmad   z3.s, p0/m, z17.s, z16.s
E subs   x8, x8, #2
F fmad   z4.s, p0/m, z17.s, z16.s
G fmad   z5.s, p0/m, z17.s, z16.s
H fmad   z6.s, p0/m, z17.s, z16.s
I fmad   z7.s, p0/m, z17.s, z16.s
J fmad   z0.s, p0/m, z17.s, z16.s
K fmad   z1.s, p0/m, z17.s, z16.s
L fmad   z2.s, p0/m, z17.s, z16.s
M fmad   z3.s, p0/m, z17.s, z16.s
N fmad   z4.s, p0/m, z17.s, z16.s
O fmad   z5.s, p0/m, z17.s, z16.s
P fmad   z6.s, p0/m, z17.s, z16.s
Q fmad   z7.s, p0/m, z17.s, z16.s
R b.ne   .LBB0_10
```

program



instruction window

cycle 4

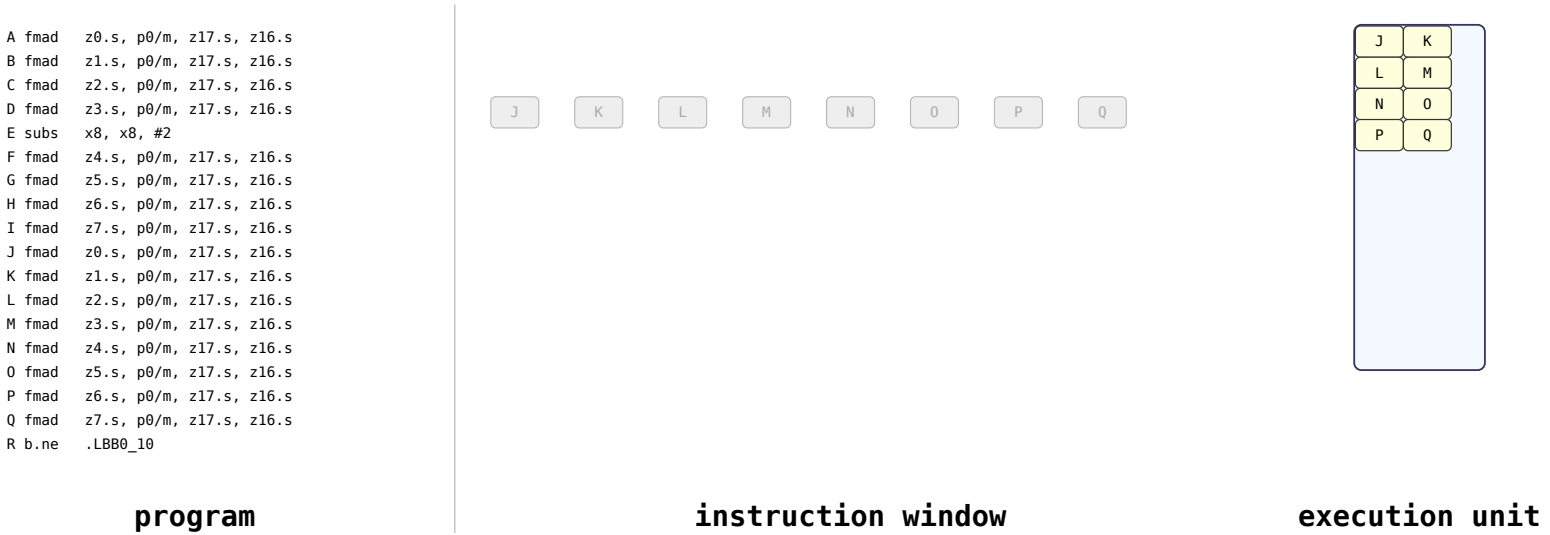


execution unit

Increase the number of chains further …

- we expect the following to reach peak FLOPS with sufficiently large C

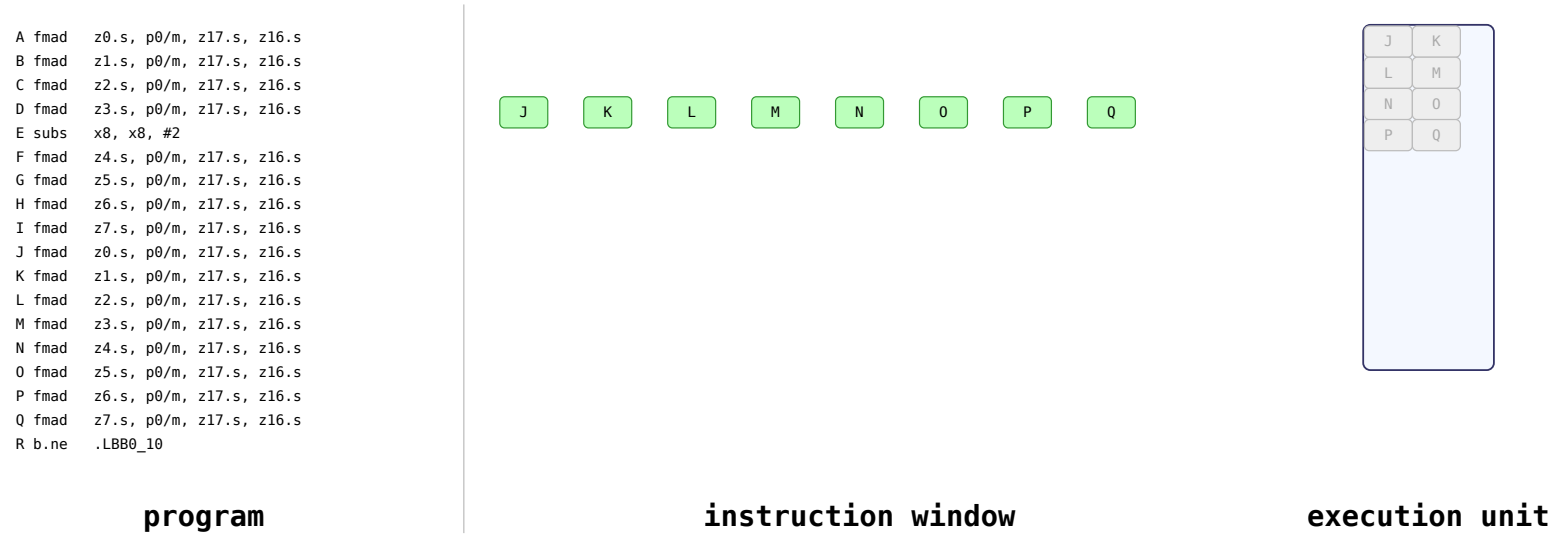
```
floatv x[C/L];
for (long k = 0; k < n; k++)
  for (int c = 0; c < C/L; c++)
    x[c] = a * (svfloat32_t)x[c] + b;
```



Increase the number of chains further …

- we expect the following to reach peak FLOPS with sufficiently large C

```
floatv x[C/L];
for (long k = 0; k < n; k++)
  for (int c = 0; c < C/L; c++)
    x[c] = a * (svfloat32_t)x[c] + b;
```



Increase the number of chains further …

- we expect the following to reach peak FLOPS with sufficiently large C

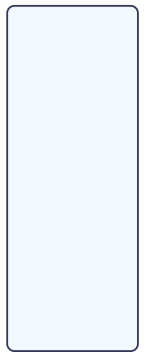
```
floatv x[C/L];
for (long k = 0; k < n; k++)
  for (int c = 0; c < C/L; c++)
    x[c] = a * (svfloat32_t)x[c] + b;
```

```
A fmad   z0.s, p0/m, z17.s, z16.s
B fmad   z1.s, p0/m, z17.s, z16.s
C fmad   z2.s, p0/m, z17.s, z16.s
D fmad   z3.s, p0/m, z17.s, z16.s
E subs   x8, x8, #2
F fmad   z4.s, p0/m, z17.s, z16.s
G fmad   z5.s, p0/m, z17.s, z16.s
H fmad   z6.s, p0/m, z17.s, z16.s
I fmad   z7.s, p0/m, z17.s, z16.s
J fmad   z0.s, p0/m, z17.s, z16.s
K fmad   z1.s, p0/m, z17.s, z16.s
L fmad   z2.s, p0/m, z17.s, z16.s
M fmad   z3.s, p0/m, z17.s, z16.s
N fmad   z4.s, p0/m, z17.s, z16.s
O fmad   z5.s, p0/m, z17.s, z16.s
P fmad   z6.s, p0/m, z17.s, z16.s
Q fmad   z7.s, p0/m, z17.s, z16.s
R b.ne   .LBB0_10
```

program

instruction window

cycle 8



execution unit

When performance hits plateau?

- up to 14 chains: as expected (latency 4, throughput up to 4)
- ≥ 15 chains: each interval runs at either 16 or 8 FMAs/cycle (16 is the peak)

C	chains	cycles/iter	FMAs/cycle
16	4	4.00	4.0
32	8	4.00	8.0
48	12	4.00	12.0
56	14	4.00	14.0
60	15	5.73	10.5
64	16	6.02	10.6
72	18	6.73	10.7
96	24	8.85	10.8

Execution units and dispatch ports

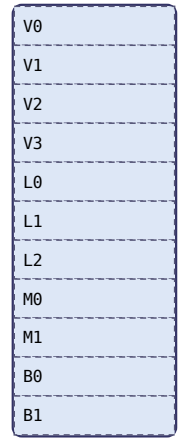
- each instruction goes to a specific ***execution unit*** handling that instruction type
- one execution unit can accept only one instruction in a cycle
- $\Rightarrow$ the number of units for specific instruction types determines the peak throughput of that specific instruction types

Execution units and dispatch ports

```
A fmad   z0.s, p0/m, z17.s, z16.s
B fmad   z1.s, p0/m, z17.s, z16.s
C fmad   z2.s, p0/m, z17.s, z16.s
D fmad   z3.s, p0/m, z17.s, z16.s
E subs   x8, x8, #2
F fmad   z4.s, p0/m, z17.s, z16.s
G fmad   z5.s, p0/m, z17.s, z16.s
H fmad   z6.s, p0/m, z17.s, z16.s
I fmad   z7.s, p0/m, z17.s, z16.s
J fmad   z0.s, p0/m, z17.s, z16.s
K fmad   z1.s, p0/m, z17.s, z16.s
L fmad   z2.s, p0/m, z17.s, z16.s
M fmad   z3.s, p0/m, z17.s, z16.s
N fmad   z4.s, p0/m, z17.s, z16.s
O fmad   z5.s, p0/m, z17.s, z16.s
P fmad   z6.s, p0/m, z17.s, z16.s
Q fmad   z7.s, p0/m, z17.s, z16.s
R b.ne   .LBB0_10
```

program

instruction window



execution unit

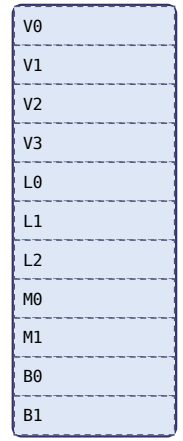
Execution units and dispatch ports

```
A fmad   z0.s, p0/m, z17.s, z16.s
B fmad   z1.s, p0/m, z17.s, z16.s
C fmad   z2.s, p0/m, z17.s, z16.s
D fmad   z3.s, p0/m, z17.s, z16.s
E subs   x8, x8, #2
F fmad   z4.s, p0/m, z17.s, z16.s
G fmad   z5.s, p0/m, z17.s, z16.s
H fmad   z6.s, p0/m, z17.s, z16.s
I fmad   z7.s, p0/m, z17.s, z16.s
J fmad   z0.s, p0/m, z17.s, z16.s
K fmad   z1.s, p0/m, z17.s, z16.s
L fmad   z2.s, p0/m, z17.s, z16.s
M fmad   z3.s, p0/m, z17.s, z16.s
N fmad   z4.s, p0/m, z17.s, z16.s
O fmad   z5.s, p0/m, z17.s, z16.s
P fmad   z6.s, p0/m, z17.s, z16.s
Q fmad   z7.s, p0/m, z17.s, z16.s
R b.ne   .LBB0_10
```

program



instruction window

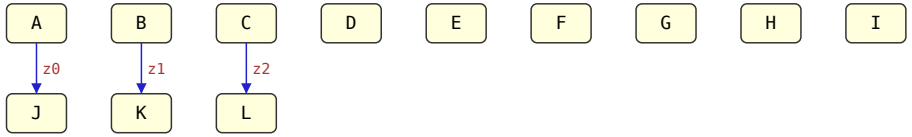


execution unit

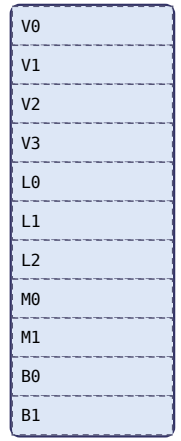
Execution units and dispatch ports

```
A fmad   z0.s, p0/m, z17.s, z16.s
B fmad   z1.s, p0/m, z17.s, z16.s
C fmad   z2.s, p0/m, z17.s, z16.s
D fmad   z3.s, p0/m, z17.s, z16.s
E subs   x8, x8, #2
F fmad   z4.s, p0/m, z17.s, z16.s
G fmad   z5.s, p0/m, z17.s, z16.s
H fmad   z6.s, p0/m, z17.s, z16.s
I fmad   z7.s, p0/m, z17.s, z16.s
J fmad   z0.s, p0/m, z17.s, z16.s
K fmad   z1.s, p0/m, z17.s, z16.s
L fmad   z2.s, p0/m, z17.s, z16.s
M fmad   z3.s, p0/m, z17.s, z16.s
N fmad   z4.s, p0/m, z17.s, z16.s
O fmad   z5.s, p0/m, z17.s, z16.s
P fmad   z6.s, p0/m, z17.s, z16.s
Q fmad   z7.s, p0/m, z17.s, z16.s
R b.ne   .LBB0_10
```

program



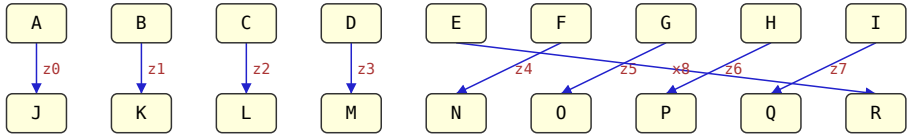
instruction window

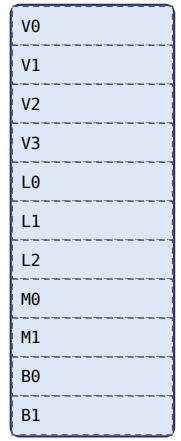


execution unit

Execution units and dispatch ports

```
A fmad   z0.s, p0/m, z17.s, z16.s
B fmad   z1.s, p0/m, z17.s, z16.s
C fmad   z2.s, p0/m, z17.s, z16.s
D fmad   z3.s, p0/m, z17.s, z16.s
E subs   x8, x8, #2
F fmad   z4.s, p0/m, z17.s, z16.s
G fmad   z5.s, p0/m, z17.s, z16.s
H fmad   z6.s, p0/m, z17.s, z16.s
I fmad   z7.s, p0/m, z17.s, z16.s
J fmad   z0.s, p0/m, z17.s, z16.s
K fmad   z1.s, p0/m, z17.s, z16.s
L fmad   z2.s, p0/m, z17.s, z16.s
M fmad   z3.s, p0/m, z17.s, z16.s
N fmad   z4.s, p0/m, z17.s, z16.s
O fmad   z5.s, p0/m, z17.s, z16.s
P fmad   z6.s, p0/m, z17.s, z16.s
Q fmad   z7.s, p0/m, z17.s, z16.s
R b.ne   .LBB0_10
```



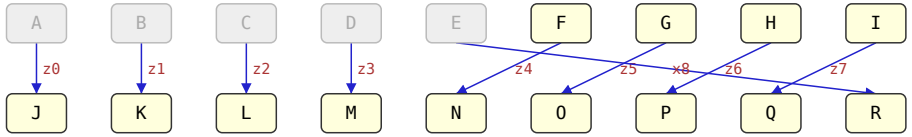


program **instruction window** **execution unit**

Execution units and dispatch ports

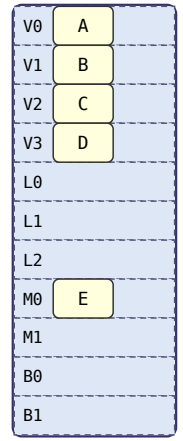
```
A fmad   z0.s, p0/m, z17.s, z16.s
B fmad   z1.s, p0/m, z17.s, z16.s
C fmad   z2.s, p0/m, z17.s, z16.s
D fmad   z3.s, p0/m, z17.s, z16.s
E subs   x8, x8, #2
F fmad   z4.s, p0/m, z17.s, z16.s
G fmad   z5.s, p0/m, z17.s, z16.s
H fmad   z6.s, p0/m, z17.s, z16.s
I fmad   z7.s, p0/m, z17.s, z16.s
J fmad   z0.s, p0/m, z17.s, z16.s
K fmad   z1.s, p0/m, z17.s, z16.s
L fmad   z2.s, p0/m, z17.s, z16.s
M fmad   z3.s, p0/m, z17.s, z16.s
N fmad   z4.s, p0/m, z17.s, z16.s
O fmad   z5.s, p0/m, z17.s, z16.s
P fmad   z6.s, p0/m, z17.s, z16.s
Q fmad   z7.s, p0/m, z17.s, z16.s
R b.ne   .LBB0_10
```

program



instruction window

cycle 0

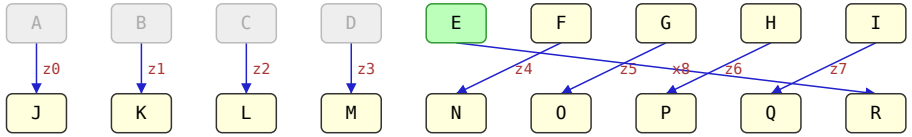


execution unit

Execution units and dispatch ports

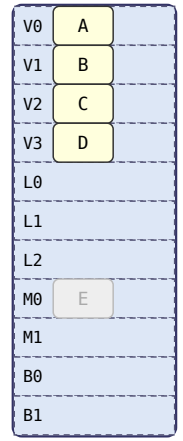
```
A fmad   z0.s, p0/m, z17.s, z16.s
B fmad   z1.s, p0/m, z17.s, z16.s
C fmad   z2.s, p0/m, z17.s, z16.s
D fmad   z3.s, p0/m, z17.s, z16.s
E subs   x8, x8, #2
F fmad   z4.s, p0/m, z17.s, z16.s
G fmad   z5.s, p0/m, z17.s, z16.s
H fmad   z6.s, p0/m, z17.s, z16.s
I fmad   z7.s, p0/m, z17.s, z16.s
J fmad   z0.s, p0/m, z17.s, z16.s
K fmad   z1.s, p0/m, z17.s, z16.s
L fmad   z2.s, p0/m, z17.s, z16.s
M fmad   z3.s, p0/m, z17.s, z16.s
N fmad   z4.s, p0/m, z17.s, z16.s
O fmad   z5.s, p0/m, z17.s, z16.s
P fmad   z6.s, p0/m, z17.s, z16.s
Q fmad   z7.s, p0/m, z17.s, z16.s
R b.ne   .LBB0_10
```

program



instruction window

cycle 1

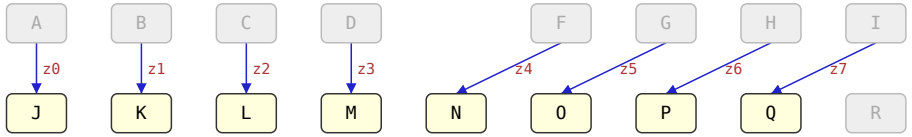


execution unit

Execution units and dispatch ports

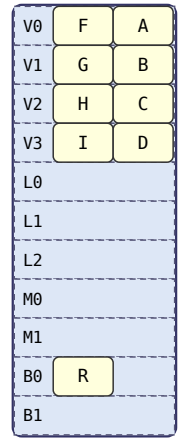
```
A fmad   z0.s, p0/m, z17.s, z16.s
B fmad   z1.s, p0/m, z17.s, z16.s
C fmad   z2.s, p0/m, z17.s, z16.s
D fmad   z3.s, p0/m, z17.s, z16.s
E subs   x8, x8, #2
F fmad   z4.s, p0/m, z17.s, z16.s
G fmad   z5.s, p0/m, z17.s, z16.s
H fmad   z6.s, p0/m, z17.s, z16.s
I fmad   z7.s, p0/m, z17.s, z16.s
J fmad   z0.s, p0/m, z17.s, z16.s
K fmad   z1.s, p0/m, z17.s, z16.s
L fmad   z2.s, p0/m, z17.s, z16.s
M fmad   z3.s, p0/m, z17.s, z16.s
N fmad   z4.s, p0/m, z17.s, z16.s
O fmad   z5.s, p0/m, z17.s, z16.s
P fmad   z6.s, p0/m, z17.s, z16.s
Q fmad   z7.s, p0/m, z17.s, z16.s
R b.ne   .LBB0_10
```

program



instruction window

cycle 1

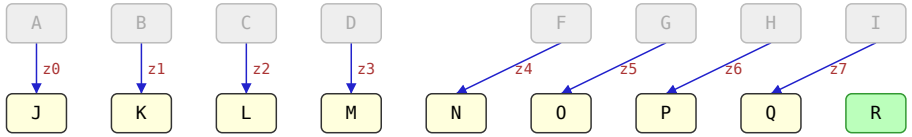


execution unit

Execution units and dispatch ports

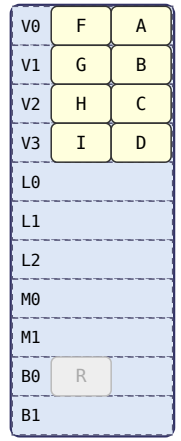
```
A fmad   z0.s, p0/m, z17.s, z16.s
B fmad   z1.s, p0/m, z17.s, z16.s
C fmad   z2.s, p0/m, z17.s, z16.s
D fmad   z3.s, p0/m, z17.s, z16.s
E subs   x8, x8, #2
F fmad   z4.s, p0/m, z17.s, z16.s
G fmad   z5.s, p0/m, z17.s, z16.s
H fmad   z6.s, p0/m, z17.s, z16.s
I fmad   z7.s, p0/m, z17.s, z16.s
J fmad   z0.s, p0/m, z17.s, z16.s
K fmad   z1.s, p0/m, z17.s, z16.s
L fmad   z2.s, p0/m, z17.s, z16.s
M fmad   z3.s, p0/m, z17.s, z16.s
N fmad   z4.s, p0/m, z17.s, z16.s
O fmad   z5.s, p0/m, z17.s, z16.s
P fmad   z6.s, p0/m, z17.s, z16.s
Q fmad   z7.s, p0/m, z17.s, z16.s
R b.ne   .LBB0_10
```

program



instruction window

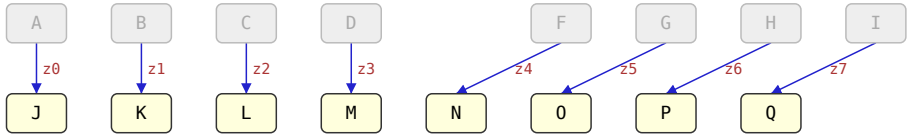
cycle 2

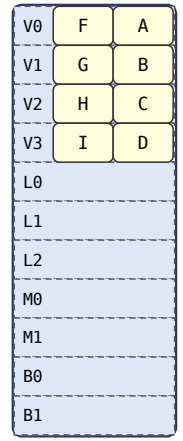


execution unit

Execution units and dispatch ports

```
A fmad   z0.s, p0/m, z17.s, z16.s
B fmad   z1.s, p0/m, z17.s, z16.s
C fmad   z2.s, p0/m, z17.s, z16.s
D fmad   z3.s, p0/m, z17.s, z16.s
E subs   x8, x8, #2
F fmad   z4.s, p0/m, z17.s, z16.s
G fmad   z5.s, p0/m, z17.s, z16.s
H fmad   z6.s, p0/m, z17.s, z16.s
I fmad   z7.s, p0/m, z17.s, z16.s
J fmad   z0.s, p0/m, z17.s, z16.s
K fmad   z1.s, p0/m, z17.s, z16.s
L fmad   z2.s, p0/m, z17.s, z16.s
M fmad   z3.s, p0/m, z17.s, z16.s
N fmad   z4.s, p0/m, z17.s, z16.s
O fmad   z5.s, p0/m, z17.s, z16.s
P fmad   z6.s, p0/m, z17.s, z16.s
Q fmad   z7.s, p0/m, z17.s, z16.s
R b.ne   .LBB0_10
```





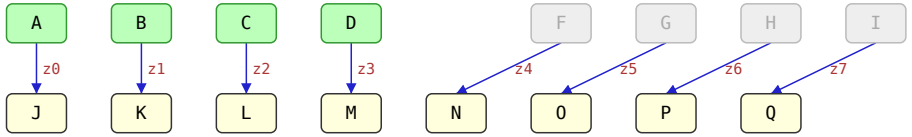
program | **instruction window** | **execution unit**

cycle 2

Execution units and dispatch ports

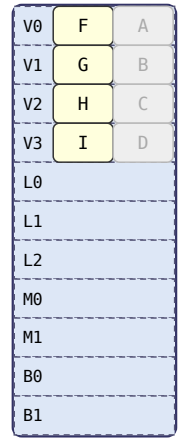
```
A fmad   z0.s, p0/m, z17.s, z16.s
B fmad   z1.s, p0/m, z17.s, z16.s
C fmad   z2.s, p0/m, z17.s, z16.s
D fmad   z3.s, p0/m, z17.s, z16.s
E subs   x8, x8, #2
F fmad   z4.s, p0/m, z17.s, z16.s
G fmad   z5.s, p0/m, z17.s, z16.s
H fmad   z6.s, p0/m, z17.s, z16.s
I fmad   z7.s, p0/m, z17.s, z16.s
J fmad   z0.s, p0/m, z17.s, z16.s
K fmad   z1.s, p0/m, z17.s, z16.s
L fmad   z2.s, p0/m, z17.s, z16.s
M fmad   z3.s, p0/m, z17.s, z16.s
N fmad   z4.s, p0/m, z17.s, z16.s
O fmad   z5.s, p0/m, z17.s, z16.s
P fmad   z6.s, p0/m, z17.s, z16.s
Q fmad   z7.s, p0/m, z17.s, z16.s
R b.ne   .LBB0_10
```

program



instruction window

cycle 4

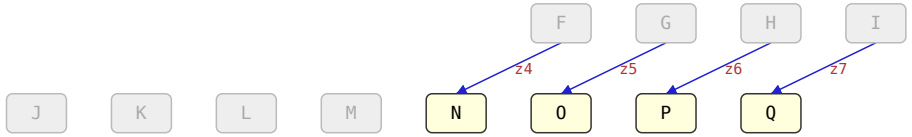


execution unit

Execution units and dispatch ports

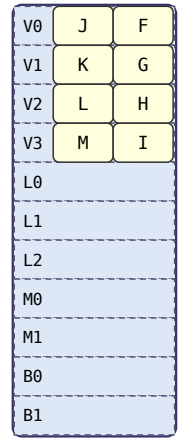
```
A fmad   z0.s, p0/m, z17.s, z16.s
B fmad   z1.s, p0/m, z17.s, z16.s
C fmad   z2.s, p0/m, z17.s, z16.s
D fmad   z3.s, p0/m, z17.s, z16.s
E subs   x8, x8, #2
F fmad   z4.s, p0/m, z17.s, z16.s
G fmad   z5.s, p0/m, z17.s, z16.s
H fmad   z6.s, p0/m, z17.s, z16.s
I fmad   z7.s, p0/m, z17.s, z16.s
J fmad   z0.s, p0/m, z17.s, z16.s
K fmad   z1.s, p0/m, z17.s, z16.s
L fmad   z2.s, p0/m, z17.s, z16.s
M fmad   z3.s, p0/m, z17.s, z16.s
N fmad   z4.s, p0/m, z17.s, z16.s
O fmad   z5.s, p0/m, z17.s, z16.s
P fmad   z6.s, p0/m, z17.s, z16.s
Q fmad   z7.s, p0/m, z17.s, z16.s
R b.ne   .LBB0_10
```

program



instruction window

cycle 4

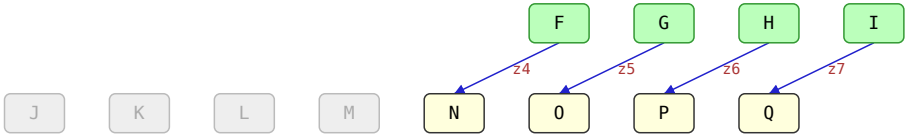


execution unit

Execution units and dispatch ports

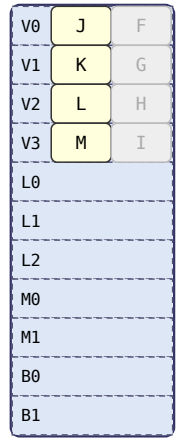
```
A fmad   z0.s, p0/m, z17.s, z16.s
B fmad   z1.s, p0/m, z17.s, z16.s
C fmad   z2.s, p0/m, z17.s, z16.s
D fmad   z3.s, p0/m, z17.s, z16.s
E subs   x8, x8, #2
F fmad   z4.s, p0/m, z17.s, z16.s
G fmad   z5.s, p0/m, z17.s, z16.s
H fmad   z6.s, p0/m, z17.s, z16.s
I fmad   z7.s, p0/m, z17.s, z16.s
J fmad   z0.s, p0/m, z17.s, z16.s
K fmad   z1.s, p0/m, z17.s, z16.s
L fmad   z2.s, p0/m, z17.s, z16.s
M fmad   z3.s, p0/m, z17.s, z16.s
N fmad   z4.s, p0/m, z17.s, z16.s
O fmad   z5.s, p0/m, z17.s, z16.s
P fmad   z6.s, p0/m, z17.s, z16.s
Q fmad   z7.s, p0/m, z17.s, z16.s
R b.ne   .LBB0_10
```

program



instruction window

cycle 5



execution unit

Execution units and dispatch ports

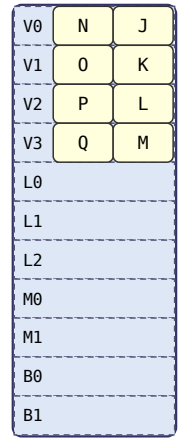
```
A fmad   z0.s, p0/m, z17.s, z16.s
B fmad   z1.s, p0/m, z17.s, z16.s
C fmad   z2.s, p0/m, z17.s, z16.s
D fmad   z3.s, p0/m, z17.s, z16.s
E subs   x8, x8, #2
F fmad   z4.s, p0/m, z17.s, z16.s
G fmad   z5.s, p0/m, z17.s, z16.s
H fmad   z6.s, p0/m, z17.s, z16.s
I fmad   z7.s, p0/m, z17.s, z16.s
J fmad   z0.s, p0/m, z17.s, z16.s
K fmad   z1.s, p0/m, z17.s, z16.s
L fmad   z2.s, p0/m, z17.s, z16.s
M fmad   z3.s, p0/m, z17.s, z16.s
N fmad   z4.s, p0/m, z17.s, z16.s
O fmad   z5.s, p0/m, z17.s, z16.s
P fmad   z6.s, p0/m, z17.s, z16.s
Q fmad   z7.s, p0/m, z17.s, z16.s
R b.ne   .LBB0_10
```

program



instruction window

cycle 5



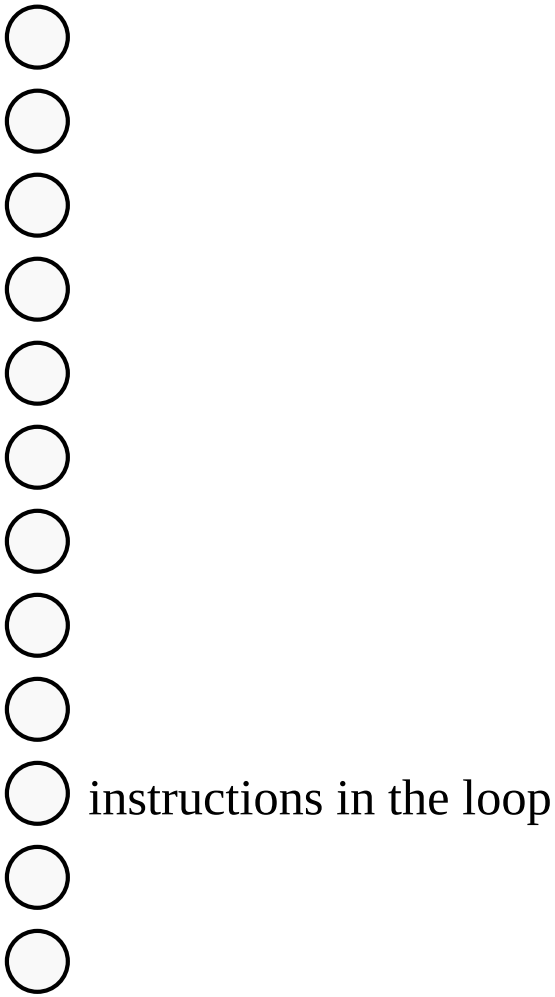
execution unit

Execution units (pipelines) of Neoverse V2

units	count	instructions
B	2	branches
S0 – S3	4	simple integer ops
M0, M1	2	integer ops incl. multiply
L01, L2	3	loads (L01 also store addresses)
D	2	store data
V0 – V3	4	FP and SIMD (`fadd`, `fmla`, `mov v`, …)

An approximate performance model for loops

cycles per iteration of a loop

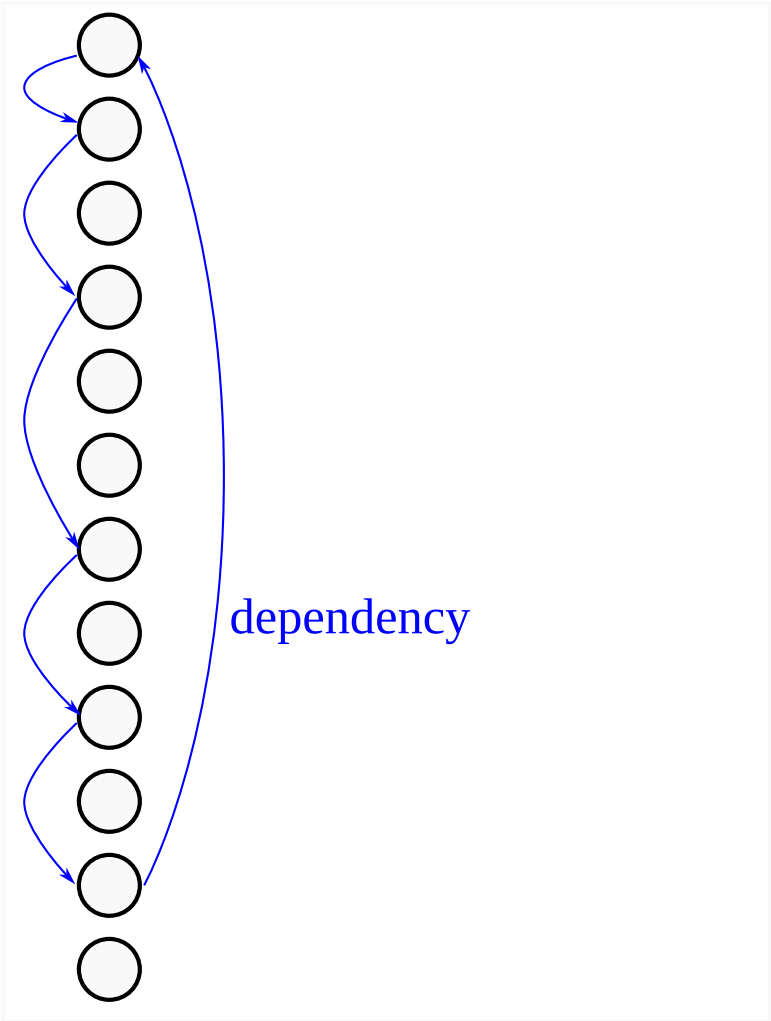


An approximate performance model for loops

cycles per iteration of a loop

1. latency-imposed limit:

$$\max_{C \in \text{ loop-carried chains}} \sum \text{latency of insns in } C$$



An approximate performance model for loops

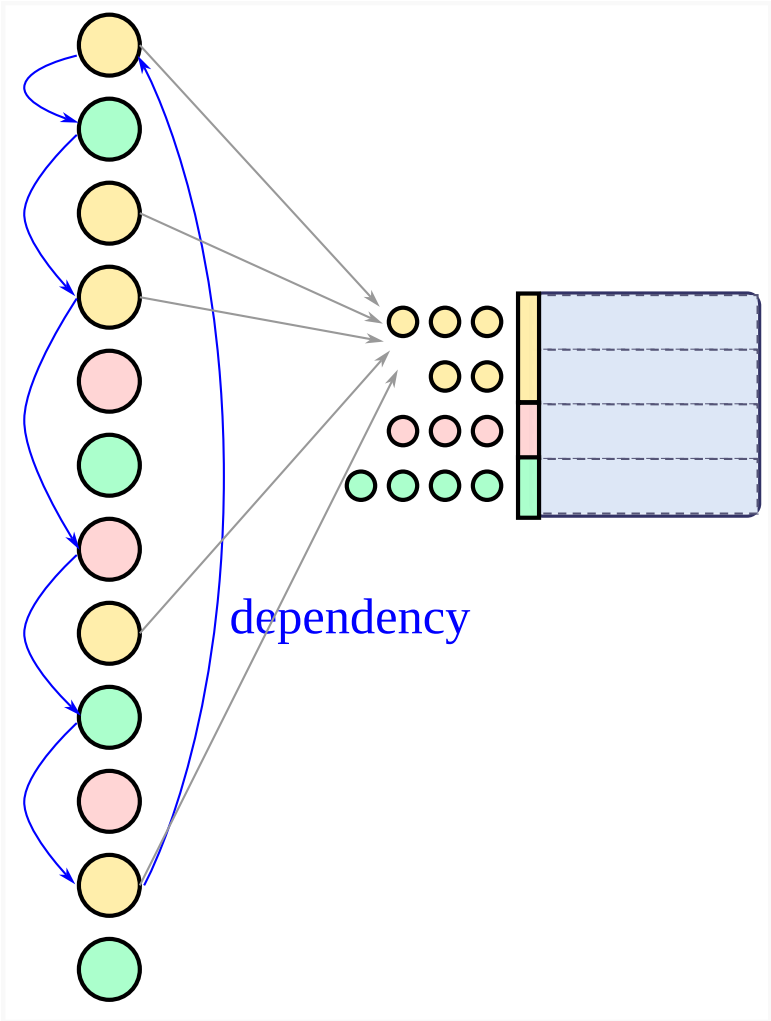
cycles per iteration of a loop

1. latency-imposed limit:

$$\max_{C \in \text{ loop-carried chains}} \sum \text{latency of insns in } C$$

2. throughput-imposed limit:

$$\max_{t \in \text{ instruction types}} \frac{\#\{\text{instructions of type } t \text{ in the loop}\}}{\#\{\text{execution units for type } t\}}$$



Resources for details

- Arm Neoverse V2 Software Optimization Guide
  - latency, throughput and execution units (pipelines) of every instruction
- Arm Intrinsics
- LLVM Machine Code Analyzer (`llvm-mca`) (later)
- for Intel/AMD CPUs: Software optimization resources by Agner, uops.info

LLVM Machine Code Analyzer (`llvm-mca`)

- a great tool to do just that
- given a code sequence, it shows
  - latency and
  - execution units

  of each instruction and, based on them, ***calculates the number of cycles per iteration***,
- under some simplifying assumptions
  - the given sequence repeats many times
  - no cache misses (!)
  - no dependencies through memory (load does not depend on earlier stores)
  - no branch misprediction

How to use `llvm-mca`

1. generate assembly (get `program.s`) by, e.g.,

   ```
   $ clang -O3 -mcpu=neoverse-v2 ... program.c -S
   ```

2. find the loop you want to analyze in the assembly
3. sandwich it by **“// LLVM-MCA-BEGIN”** and **“// LLVM-MCA-END”**

   ```
   // LLVM-MCA-BEGIN
   .LBB0_4:
      ...
      b.ne .LBB0_4
   // LLVM-MCA-END
   ```

4. run `llvm-mca` tool on the assembly code

   ```
   $ llvm-mca -mcpu=neoverse-v2 program.s
   ```

How to use `llvm-mca`

- it shows
  - latency of each instruction
  - execution units used by each instruction

  and how many instructions use each of the units (therefore the throughput limit of the loop)
- with `--timeline` option,

  ```
  $ llvm-mca -mcpu=neoverse-v2 --timeline program.s
  ```

  it also shows when each instruction gets dispatched, executed, and retired (particularly instructive)

Summary

- to approach the peak FLOPS of a CPU core, you need
  - SIMD (4 floats per instruction on Grace)
  - ILP: enough independent instructions to hide the latency (latency $\times$ throughput $= 4 \times 4 =$ 16 independent FMAs on Grace)
  - attention to every instruction in the loop (e.g., `mov`, loads), as all of them consume execution resources
- tools: compiler's assembly output (`-S`), `llvm-mca`, the optimization guide
- on the GPU, the same latency is hidden by many threads instead