Fast Big-Integer Arithmetic on SVP64 at up to 256-bits/cycle and beyond

Jacob R. Lifshay

FOSDEM 2024

libreSOC

What is SVP64?

- Vectorization Extension for PowerISA developed by Libre-SOC

What is SVP64?

- Vectorization Extension for PowerISA developed by Libre-SOC
- Basically, a way to modify nearly any PowerISA instruction to run it in a HW loop.

What is SVP64?

- Vectorization Extension for PowerISA developed by Libre-SOC
- Basically, a way to modify nearly any PowerISA instruction to run it in a HW loop.

  Simple Example:

```
setvl 0, 0, 3, 0, 1, 1 # makes stuff run 3 times
sv.add *r3, *r15, r12  # adds 3 times
```

What is SVP64?

- Vectorization Extension for PowerISA developed by Libre-SOC
- Basically, a way to modify nearly any PowerISA instruction to run it in a HW loop.

  Simple Example:

```
setvl 0, 0, 3, 0, 1, 1 # makes stuff run 3 times
sv.add *r3, *r15, r12  # adds 3 times

# expands to:
add r3, r15, r12  # no * means r12 doesn't increment
add r4, r16, r12  # * means r3 and r15 increment
add r5, r17, r12
```

Big-Integer Addition on SVP64

How can we use SVP64 to add 256-bit integers?

Big-Integer Addition on SVP64

How can we use SVP64 to add 256-bit integers?

```
setvl 0, 0, 4, 0, 1, 1 # makes stuff run 4 times
addic r0, r0, 0  # clear CA (carry flag)
sv.adde *r4, *r4, *r8  # carry-propagating add
```

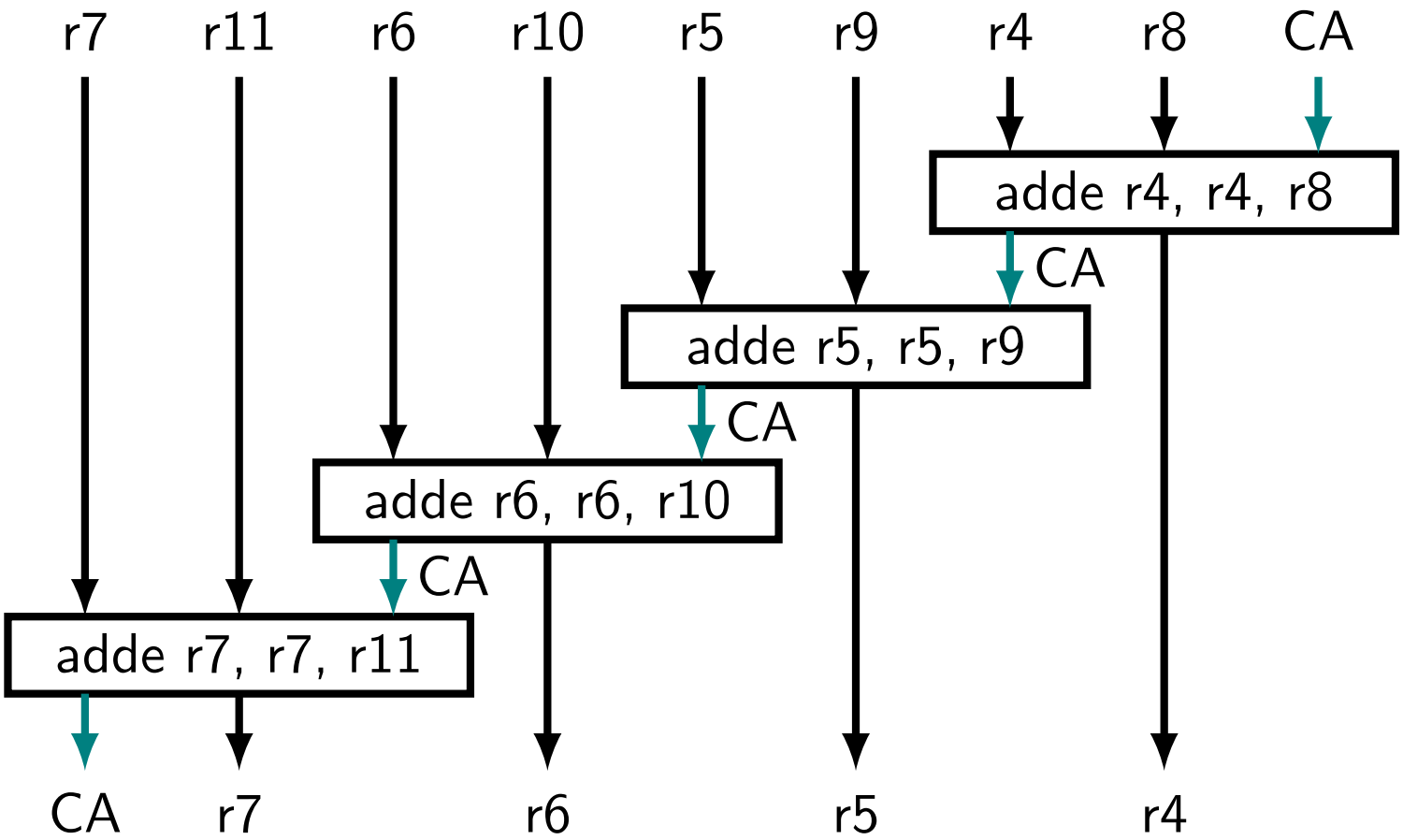
Big-Integer Addition on SVP64

How can we use SVP64 to add 256-bit integers?

```
setvl 0, 0, 4, 0, 1, 1 # makes stuff run 4 times
addic r0, r0, 0  # clear CA (carry flag)
sv.adde *r4, *r4, *r8  # carry-propagating add

# expands to:
addic r0, r0, 0  # clear CA (carry flag)
adde r4, r4, r8
adde r5, r5, r9
adde r6, r6, r10
adde r7, r7, r11
```

Big-Integer Addition on SVP64

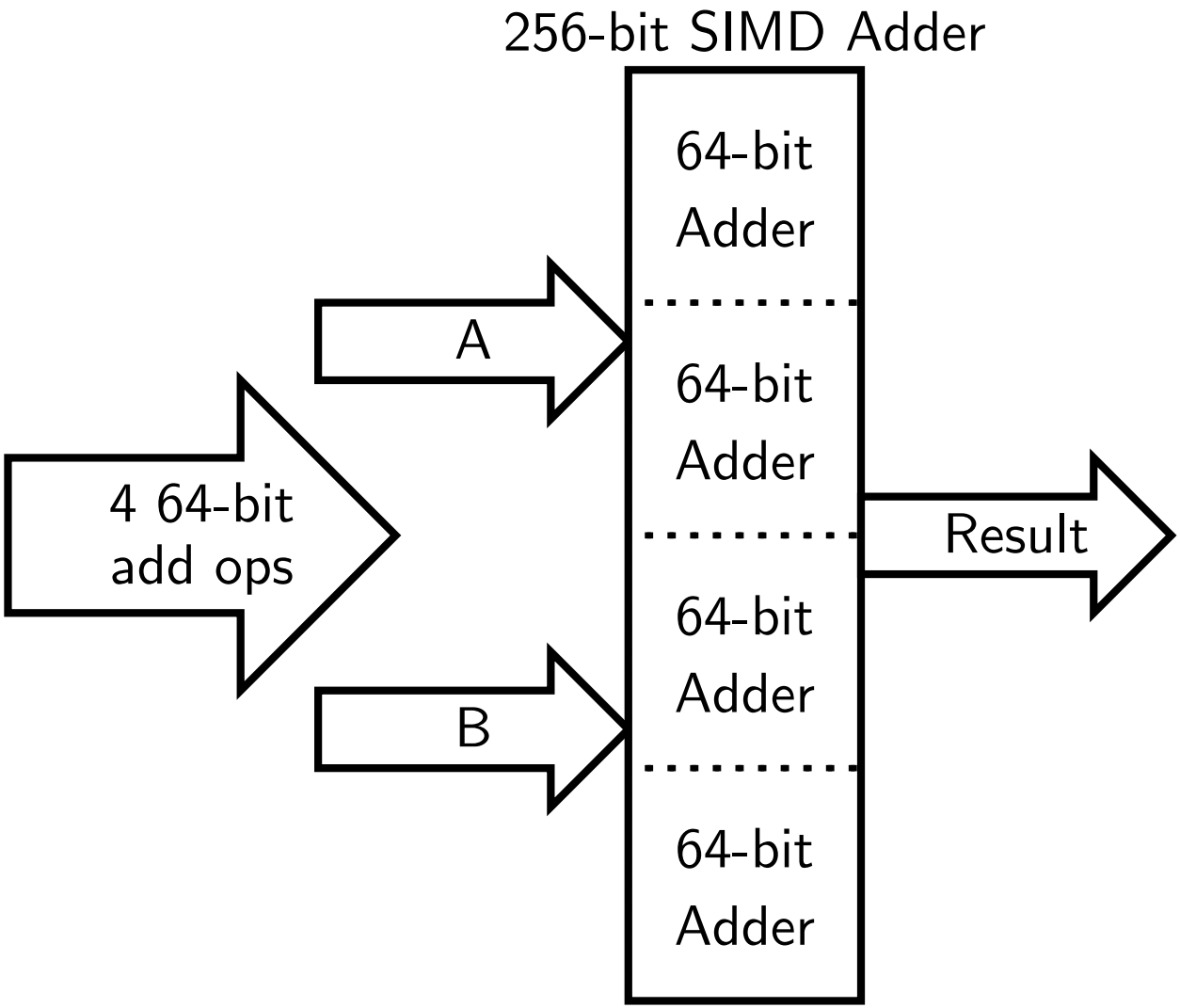
How can we use SVP64 to add 256-bit integers?



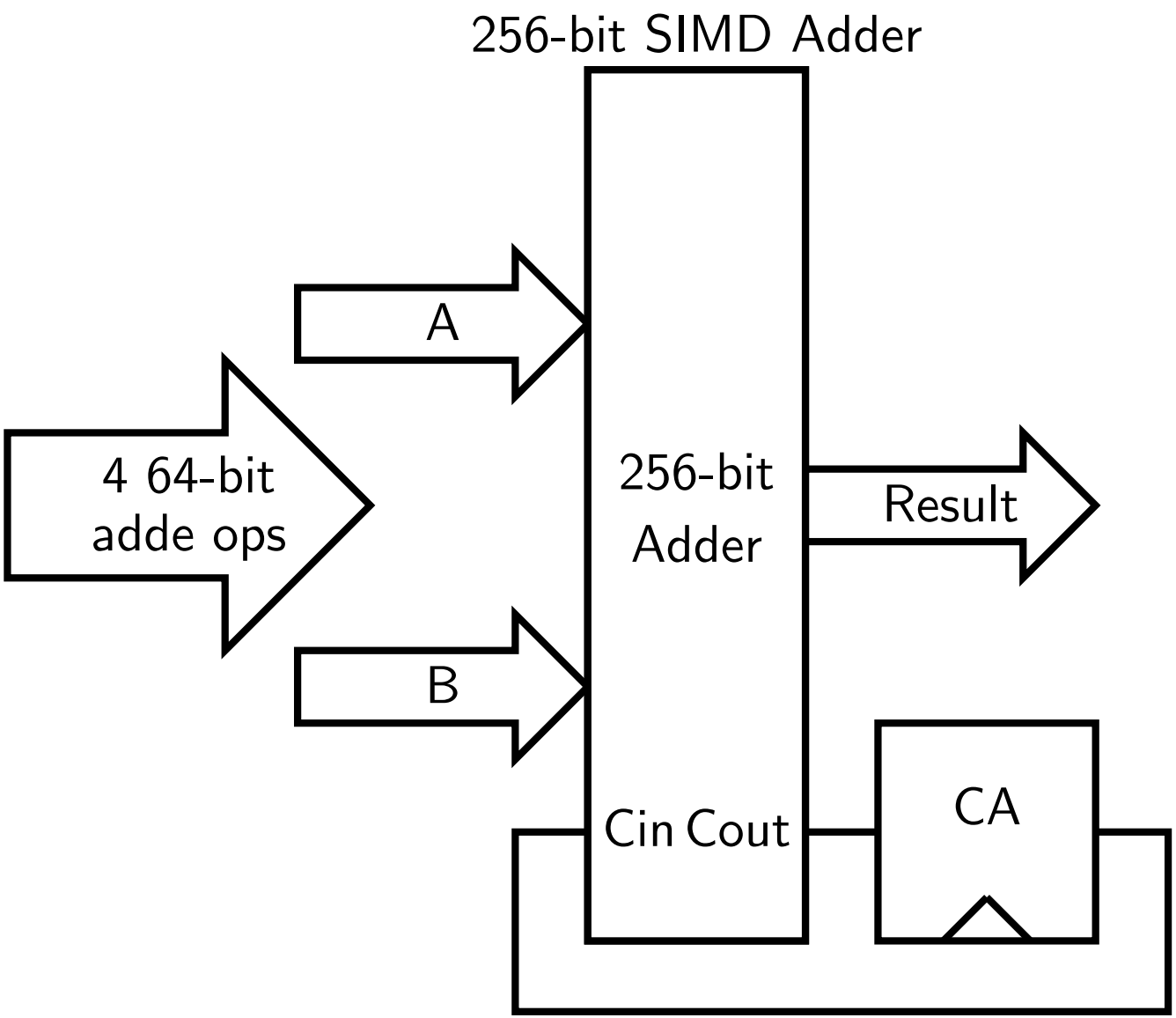
Big-Integer Addition on an example CPU

Disclaimer: SVP64 is designed for everything from tiny to big and fast CPUs, this example only shows a hypothetical big and fast CPU design

Big-Integer Addition on an example CPU



Big-Integer Addition on an example CPU



Big-Integer Multiply on SVP64

How can we use SVP64 to Multiply a 64-bit by a 256-bit integer?

Big-Integer Multiply on SVP64

How can we use SVP64 to Multiply a 64-bit by a 256-bit integer?

- new instruction: `maddedu RT, RA, RB, RC`

Big-Integer Multiply on SVP64

How can we use SVP64 to Multiply a 64-bit by a 256-bit integer?

- new instruction: `maddedu RT, RA, RB, RC`
- $64 \times 64 + 64 \rightarrow$ 128-bit Multiply-Add

Big-Integer Multiply on SVP64

How can we use SVP64 to Multiply a 64-bit by a 256-bit integer?

- new instruction: `maddedu RT, RA, RB, RC`
- $64 \times 64 + 64 \rightarrow$ 128-bit Multiply-Add
- Semantics as used in this presentation (somewhat simplified):

```
result = (RA * RB) + RC
RT = LSB_HALF(result)
RC = MSB_HALF(result)
```

Big-Integer Multiply on SVP64

How can we use SVP64 to Multiply a 64-bit by a 256-bit integer?

Big-Integer Multiply on SVP64

How can we use SVP64 to Multiply a 64-bit by a 256-bit integer?

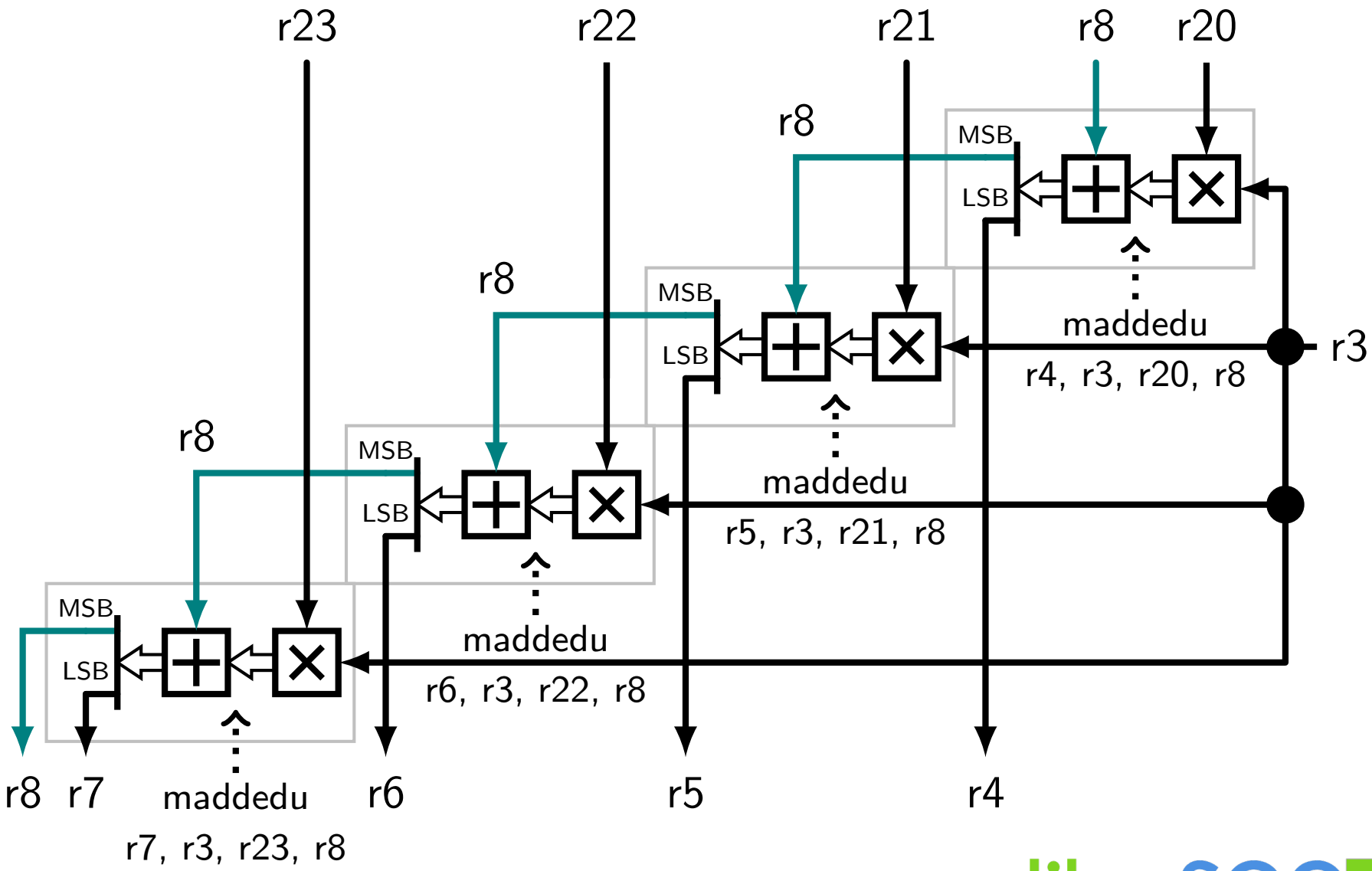
```
# 64-bit input in r3
# 256-bit input in r20-23
# 320-bit output in r4-8
setvl 0, 0, 4, 0, 1, 1 # makes stuff run 4 times
li r8, 0  # clear carry register
sv.maddedu *r4, r3, *r20, r8  # carrying multiply
```

Big-Integer Multiply on SVP64

How can we use SVP64 to Multiply a 64-bit by a 256-bit integer?

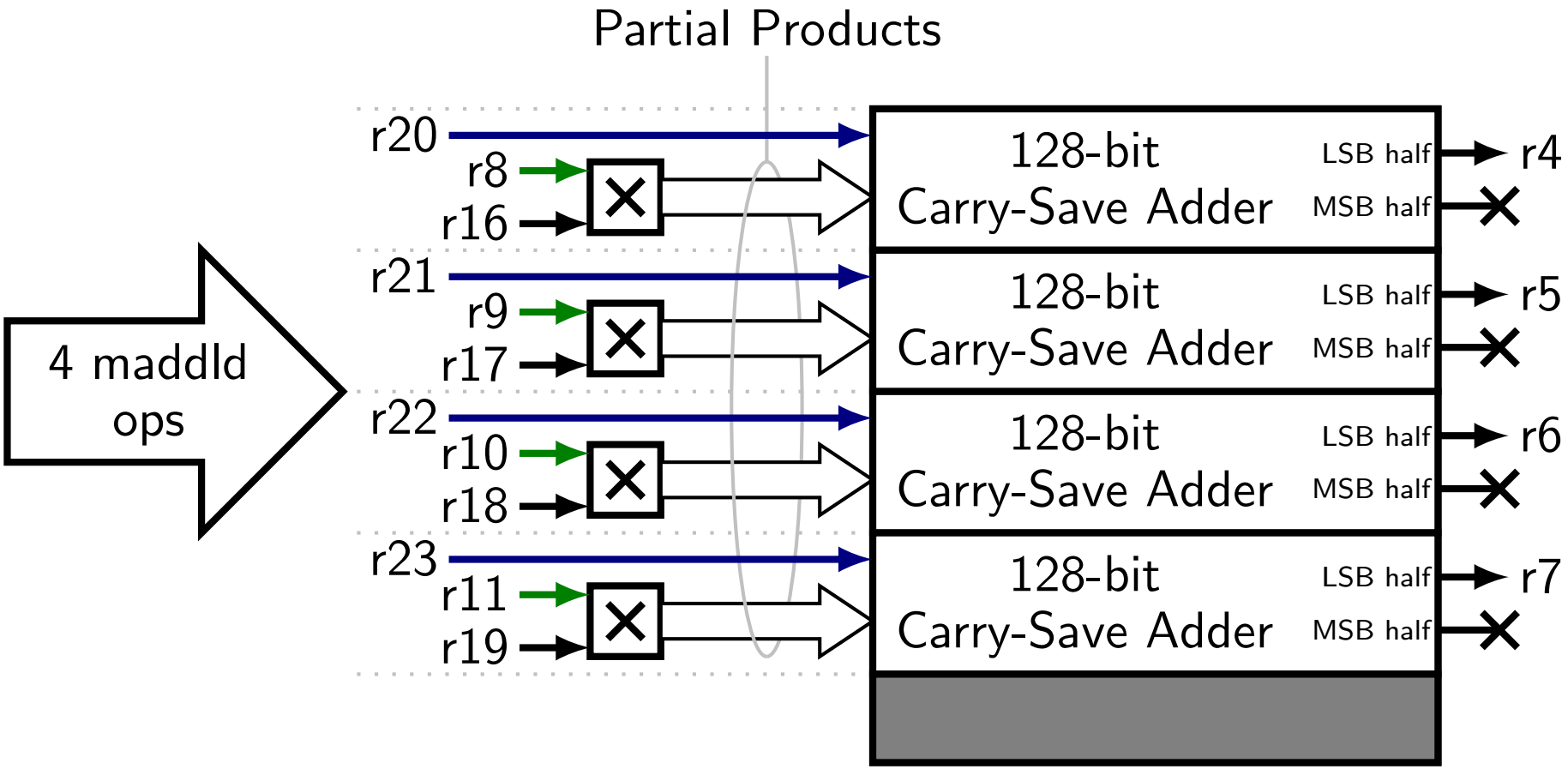
```
# 64-bit input in r3
# 256-bit input in r20-23
# 320-bit output in r4-8
setvl 0, 0, 4, 0, 1, 1 # makes stuff run 4 times
li r8, 0  # clear carry register
sv.maddedu *r4, r3, *r20, r8  # carrying multiply

# expands to:
li r8, 0
maddedu r4, r3, r20, r8
maddedu r5, r3, r21, r8
maddedu r6, r3, r22, r8
maddedu r7, r3, r23, r8
```

Big-Integer Multiply on SVP64



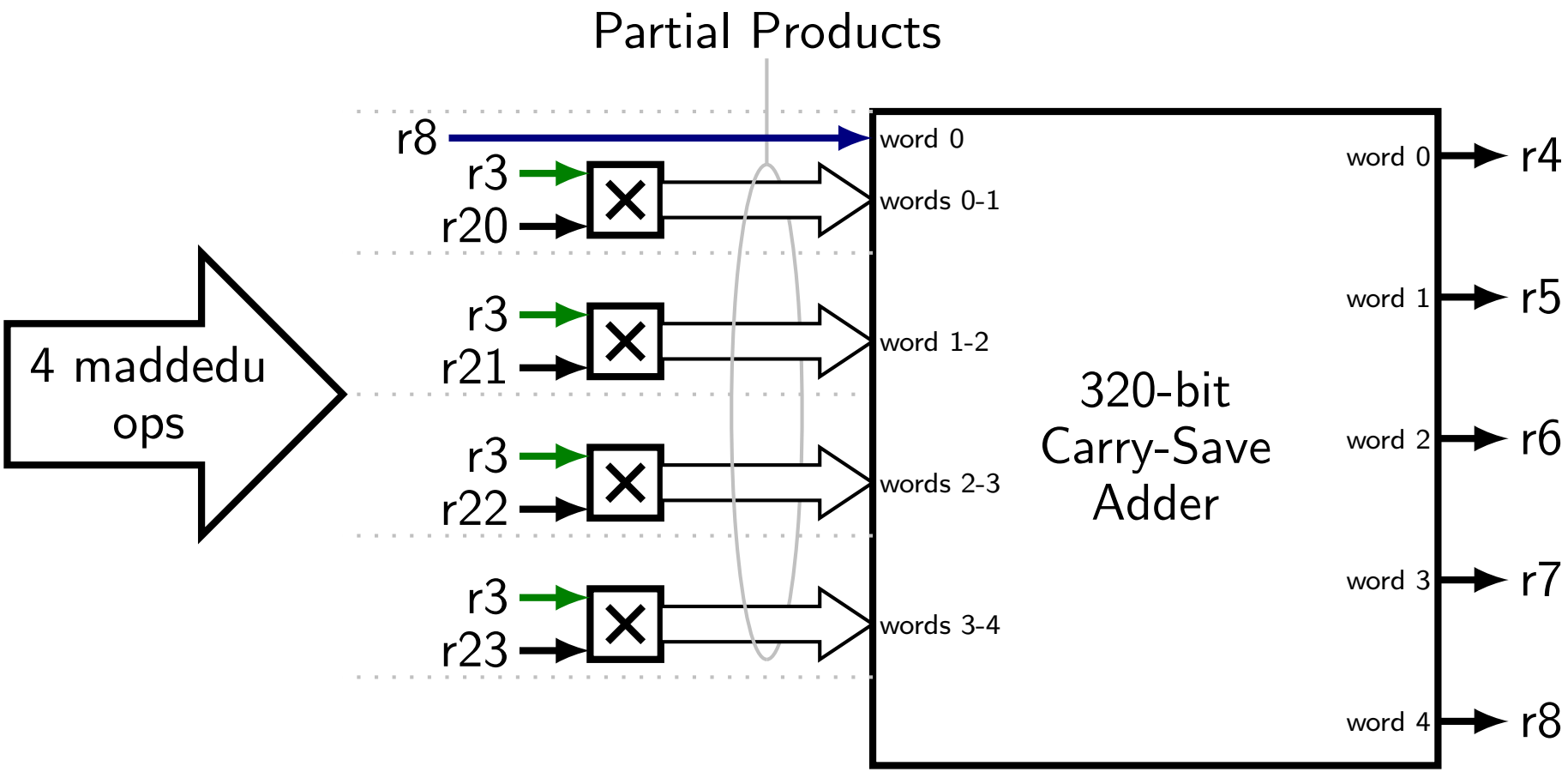
Big-Integer Multiply on an example CPU

```
sv.maddld *r4, *r8, *r16, *r20  # mul-add
```



Big-Integer Multiply on an example CPU

```
sv.maddedu *r4, r3, *r20, r8  # carrying multiply
```



- Discussion: `https://lists.libre-soc.org`
- IRC #libre-soc on OFTC or Libera
- Matrix #_oftc_#libre-soc:matrix.org
- `https://libre-soc.org/`
- Thanks to NLnet for funding this: `https://nlnet.nl/assure`
- `https://libre-soc.org/nlnet/#faq`