Porting & Optimising Code 32-bit to 64-bit

Matthew Gretton-Dann
Technical Lead - Toolchain Working Group

Linaro Connect, Dublin July 2013

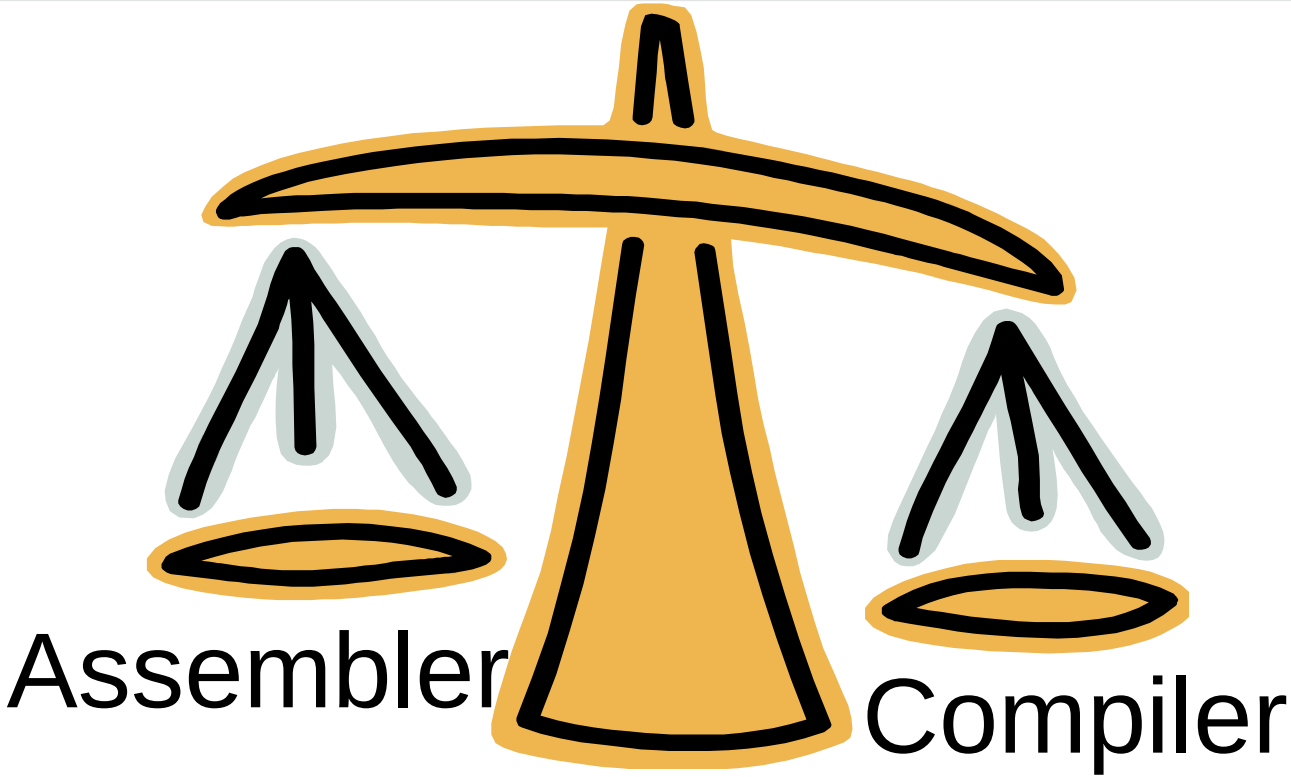
A Presentation of Four Parts

- Register Files
- Structure Layout & Data Models
- Atomics
- Vectorization & Neon Intrinsics

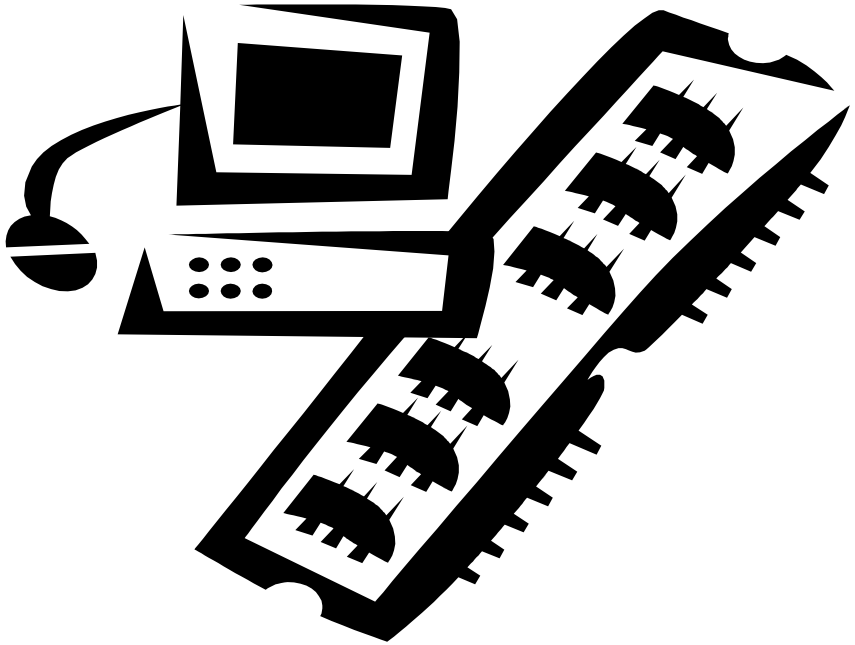
Simplification

- View for those writing apps
- No complicated kernel stuff
- Little Endian

Bias Warning



Why 64-bit?



Memory

General Purpose Registers – 32-bit ARM

r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13 (SP)
r14 (LR)
r15 (PC)

General Purpose Registers

r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13 (SP)
r14 (LR)
r15 (PC)

General Purpose Registers

r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13 (SP)
r14 (LR)
r15 (PC)

r16
r17
r18
r19
r20
r21
r22
r23
r24
r25
r26
r27
r28
r29
r30

General Purpose Registers – 64-bit ARM

r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13
r14
r15

r16
r17
r18
r19
r20
r21
r22
r23
r24
r25
r26
r27
r28
r29
r30 (LR)

SP

PC

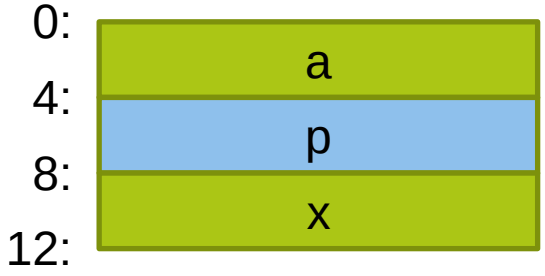
General Purpose Registers

General Purpose Registers – Consequences

- Easier to do 64-bit arithmetic!
- Less need to spill to the stack
- Spare registers to keep more temporaries

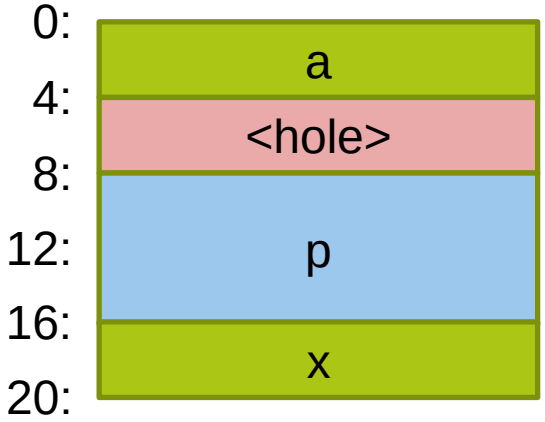
Structure Layout – 32-bit

```
struct foo {
  int32_t a;
  void* p;
  int32_t x;
};
```



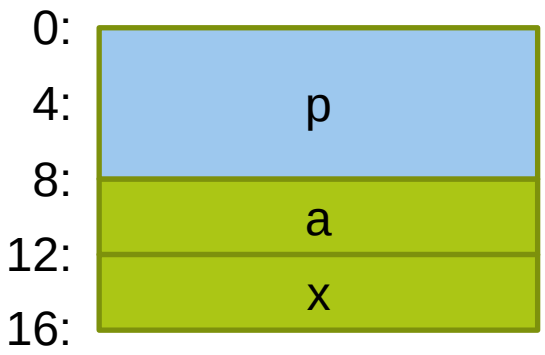
Structure Layout – 64-bit

```
struct foo {
  int32_t a;
  void* p;
  int32_t x;
};
```



Structure Layout – 64-bit

```
struct foo {
  void* p;
  int32_t a;
  int32_t x;
};
```



Brief Aside

- API: Application Programming Interface
  - Defines the interfaces a programmer may use
  - High level
- ABI: Application Binary Interface
  - Defines how to call functions, layout memory &c.
  - Low level

Data Models

ILP32

Data Models

ILP32

LP64

Data Models

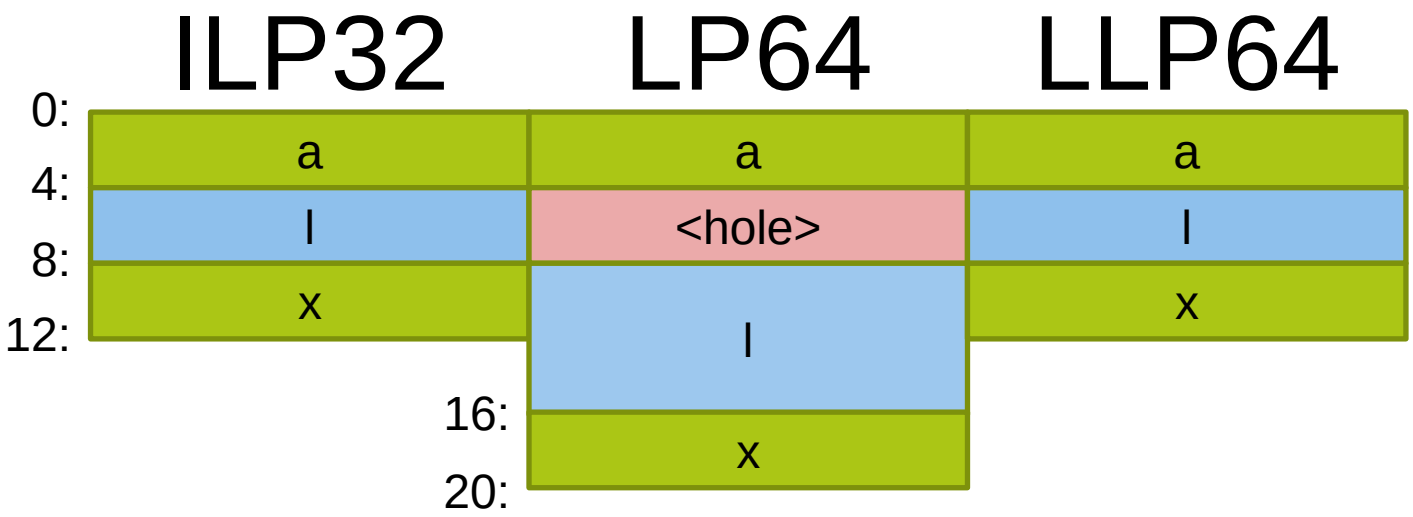
ILP32

LP64

LLP64

Data Models

```
struct foo {
  int a;
  long l;
  int x;
};
```



That’s It...

One more thing...

One more thing...

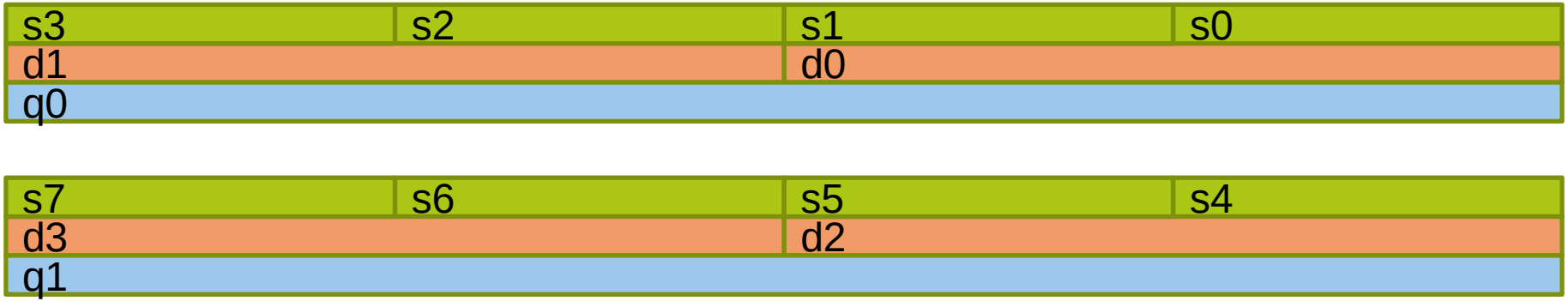
- Remove conditionalisation

Two more things...

- Remove conditionalisation
- Add some new load/store semantics

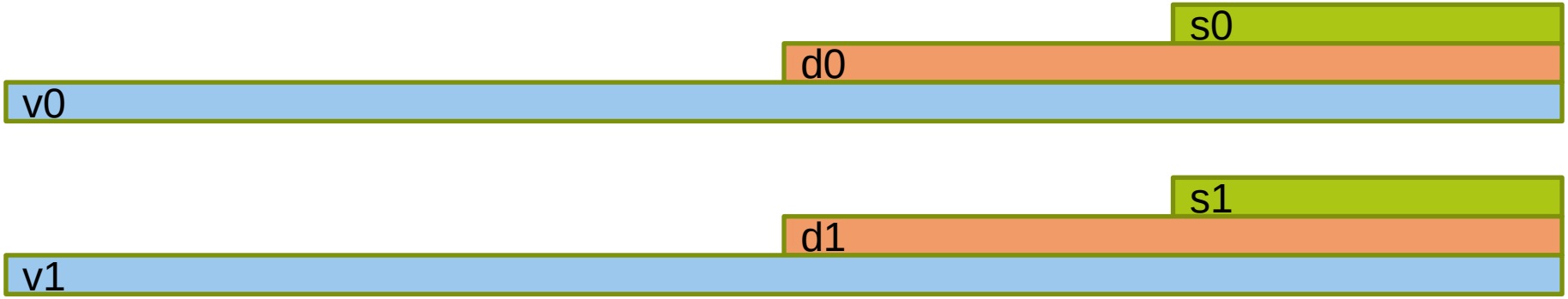
Three more things...

- Remove conditionalisation
- Add some new load/store semantics
- Change the register layout for the floating-point/SIMD registers



Three more things...

- Remove conditionalisation
- Add some new load/store semantics
- Change the register layout for the floating-point/SIMD registers



Four more things...

- Remove conditionalization
- Add some new load/store semantics
- Change the register layout for the float-point/SIMD registers
- Add some more SIMD instructions

Many more things...

- Remove conditionalization
- Add some new load/store semantics
- Change the register layout for the float-point/SIMD registers
- Add some more SIMD instructions
- ...

Atomics

```
#if defined(__GNUC__) &&
    (defined(__i386__) ||
    defined(__x86_64__))
int
AtomicAdd(volatile int* ptr, int increment)
{
  int temp = increment;
  __asm__ __volatile__(
  "lock; xaddl %0,%1”
  : "+r" (temp), "+m" (*ptr) : : "memory");
  return temp + increment;
}
```

```
#else
int AtomicAdd (volatile int* ptr, int
    increment)
{
  *ptr += increment;
  return *ptr;
}
#endif
```

Atomics

type __atomic_add_fetch (type *ptr, type val, int memmodel)

These built-in functions perform the operation suggested by the name, and return the result of the operation. That is,

{ *ptr op= val; return *ptr; }

All memory models are valid.

Atomics

```
int AtomicAdd(volatile int* ptr, int increment)
{
  return __atomic_add_fetch (ptr, increment, memmodel);
}
```

Atomics

- There are basically three types of memory model defined by C++11 which GCC's support is based upon:
  - Sequentially Consistent
  - Acquire/Release
  - Relaxed

Atomics – Sequentially Consistent

```
a = 1;
x.store(20);
```

```
if (x.load() == 20)
  assert (a == 1);
```

Atomics – Relaxed

```
a = 1;
x.store(20, memory_order_relaxed);
```

```
if ( x.load(memory_order_relaxed)) ==
     20)
  assert (a == 1);
```

Atomics – Acquire/Release

```
x.store (10, memory_order_release);
```

```
y.store (20, memory_order_release);
```

```
assert (y.load (memory_order_acquire)
    == 20 &&
    x.load (memory_order_acquire) ==
    0)
```

```
assert (y.load (memory_order_acquire)
    == 0 &&
    x.load (memory_order_acquire) ==
    10)
```

Atomics – Sequentially Consistent

```
x.store (10);
```

```
y.store (20);
```

```
assert (y.load () == 20 &&
    x.load () == 0)
```

```
assert (y.load () == 0 &&
    x.load () == 10)
```

Atomics – Sequentially Consistent

```
x.store (10);
```

```
y.store (20);
```

```
assert (y.load () == 20 &&
    x.load () == 0)
```

```
assert (y.load () == 0 &&
    x.load () == 10)
```

Atomics – Acquire/Release

```
a = 1;
x.store(20, memory_order_release);
```

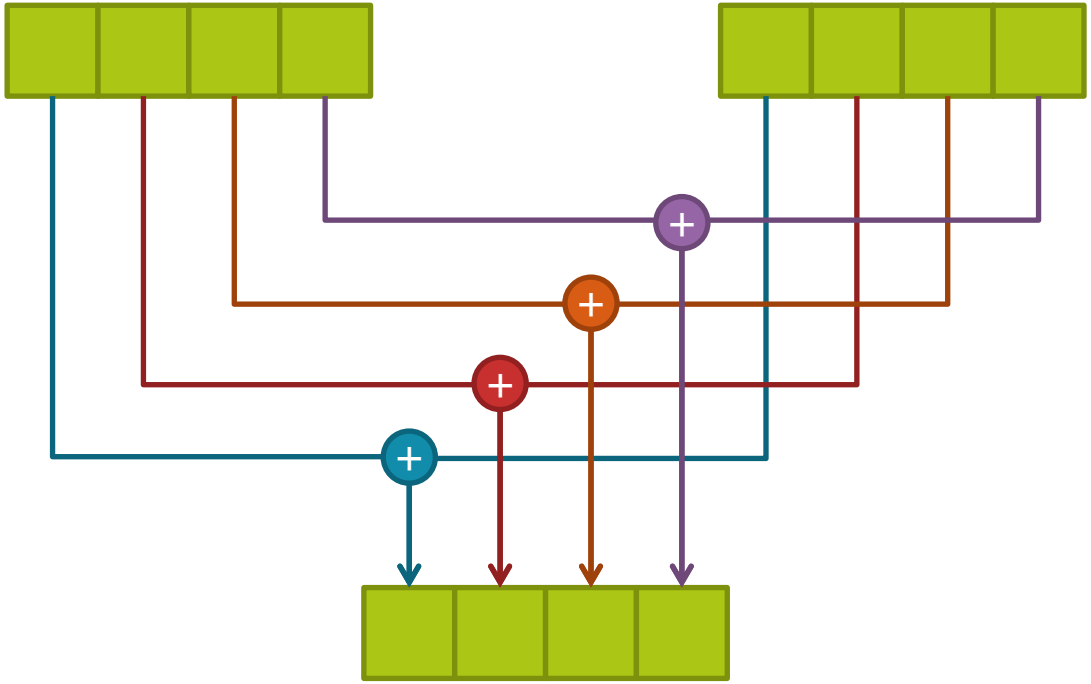
```
if (x.load(memory_order_acquire) ==
    20)
  assert (a == 1);
```

Atomics

```
int AtomicAdd(volatile int* ptr, int increment)
{
  return __atomic_add_fetch (ptr, increment, __ATOMIC_SEQ_CST);
}
```

And Now For Something Completely Different...

```
add:
    vld1.32   {q9}, [r1]!
    vld1.32   {q8}, [r2]!
    vadd.i32  q8, q9, q8
    subs      r3, r3, #4
    vst1.32   {q8}, [r0]!
    bne       add
    bx        lr
```



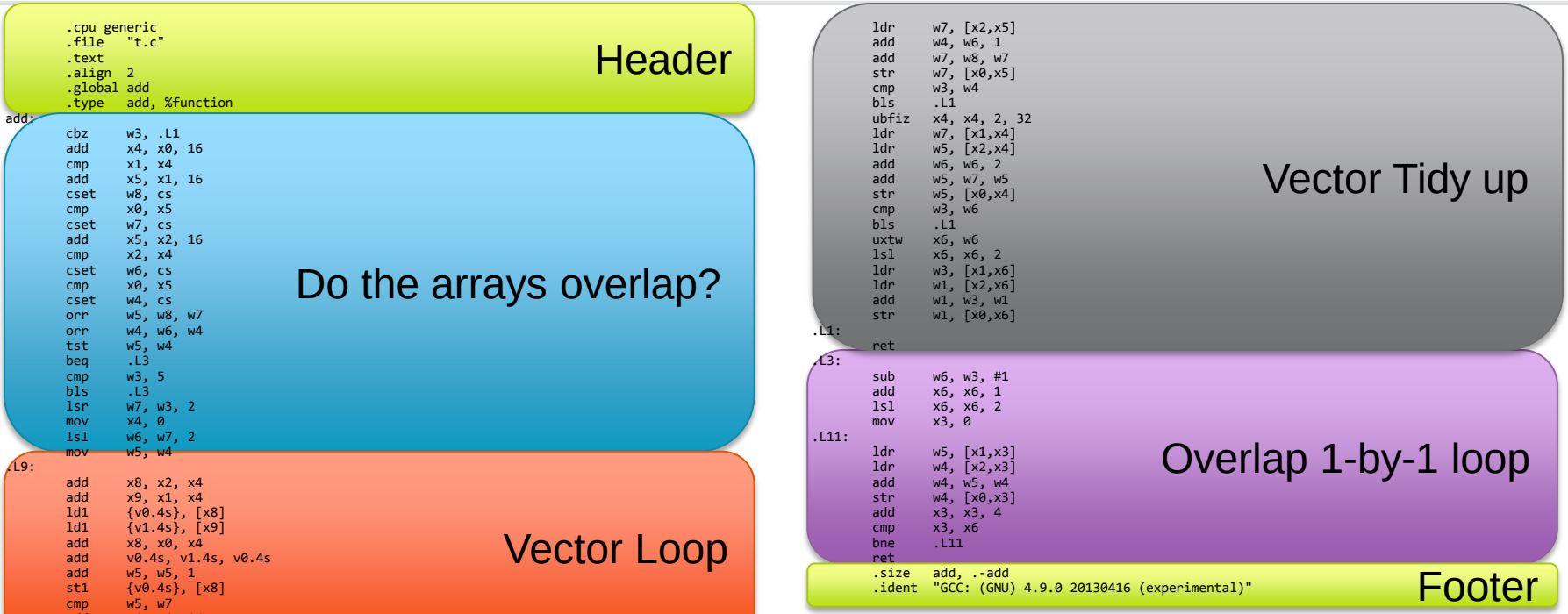
Autovectorisation

```
void add(int *a, const int *b, const int *c, unsigned n)
{
  unsigned i;
  for (i = 0; i < n; ++i)
    a[i] = b[i] + c[i];
}
```

Autovectorisation

```
        .cpu generic
        .file   "t.c"
        .text
        .align  2
        .global add
        .type   add, %function
add:
        cbz     w3, .L1
        add     x4, x0, 16
        cmp     x1, x4
        add     x5, x1, 16
        cset    w8, cs
        cmp     x0, x5
        cset    w7, cs
        add     x5, x2, 16
        cmp     x2, x4
        cset    w6, cs
        cmp     x0, x5
        cset    w4, cs
        orr     w5, w8, w7
        orr     w4, w6, w4
        tst     w5, w4
        beq     .L3
        cmp     w3, 5
        bls     .L3
        lsr     w7, w3, 2
        mov     x4, 0
        lsl     w6, w7, 2
        mov     w5, w4
.L9:
        add     x8, x2, x4
        add     x9, x1, x4
        ld1     {v0.4s}, [x8]
        ld1     {v1.4s}, [x9]
        add     x8, x0, x4
        add     v0.4s, v1.4s, v0.4s
        add     w5, w5, 1
        st1     {v0.4s}, [x8]
        cmp     w5, w7
        add     x4, x4, 16
        bcc     .L9
        cmp     w3, w6
        beq     .L1
        uxtw    x5, w6
        lsl     x5, x5, 2
        ldr     w8, [x1,x5]
        ldr     w7, [x2,x5]
        add     w4, w6, 1
        add     w7, w8, w7
        str     w7, [x0,x5]
        cmp     w3, w4
        bls     .L1
        ubfiz   x4, x4, 2, 32
        ldr     w7, [x1,x4]
        ldr     w5, [x2,x4]
        add     w6, w6, 2
        add     w5, w7, w5
        str     w5, [x0,x4]
        cmp     w3, w6
        bls     .L1
        uxtw    x6, w6
        lsl     x6, x6, 2
        ldr     w3, [x1,x6]
        ldr     w1, [x2,x6]
        add     w1, w3, w1
        str     w1, [x0,x6]
.L1:
        ret
.L3:
        sub     w6, w3, #1
        add     x6, x6, 1
        lsl     x6, x6, 2
        mov     x3, 0
.L11:
        ldr     w5, [x1,x3]
        ldr     w4, [x2,x3]
        add     w4, w5, w4
        str     w4, [x0,x3]
        add     x3, x3, 4
        cmp     x3, x6
        bne     .L11
        ret
        .size   add, .-add
        .ident  "GCC: (GNU) 4.9.0 20130416 (experimental)"
```

Autovectorisation



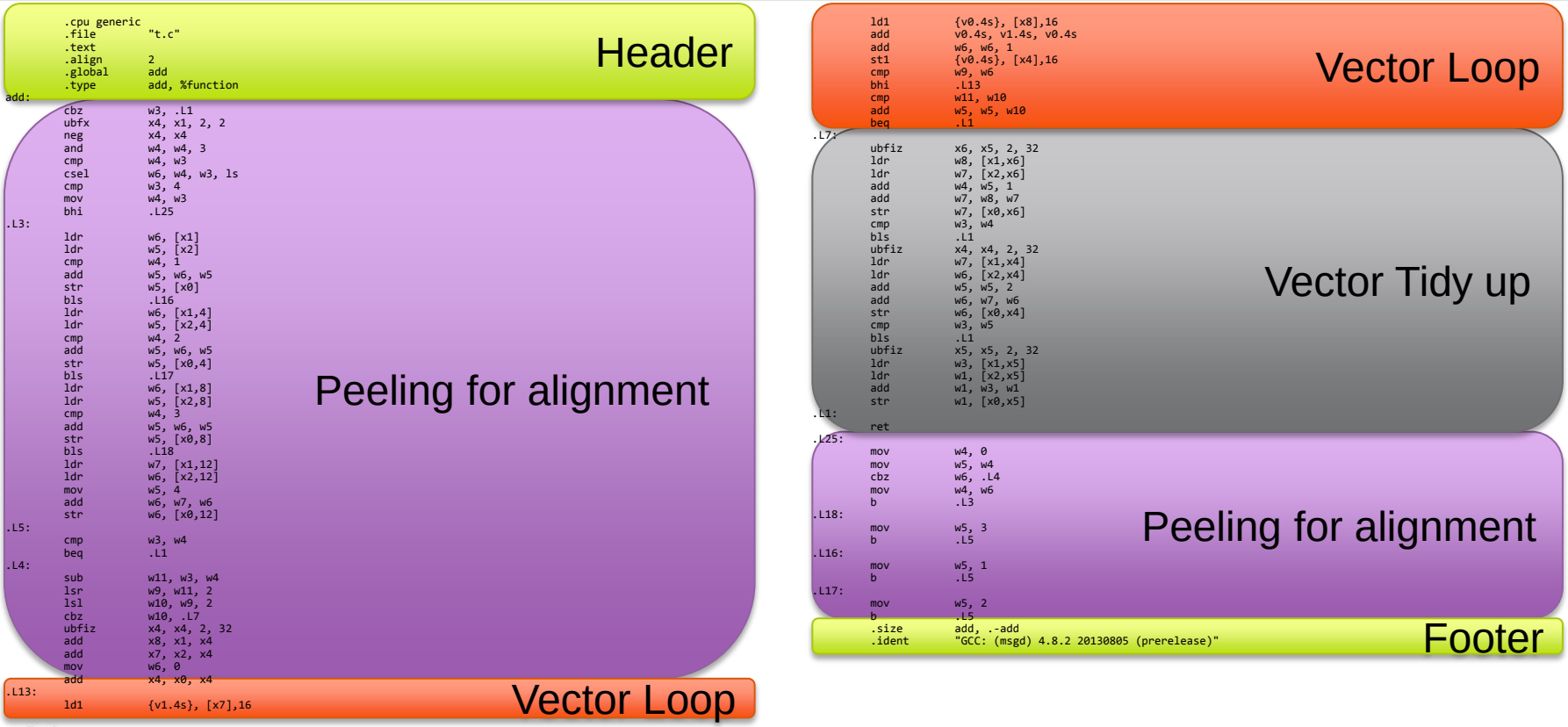
Autovectorisation

```
void add(int * restrict a, const int * restrict b, const int * restrict c, unsigned n)
{
  unsigned i;
  for (i = 0; i < n; ++i)
    a[i] = b[i] + c[i];
}
```

Autovectorisation

```
        .cpu generic
        .file   "t.c"
        .text
        .align  2
        .global add
        .type   add, %function
add:
        cbz     w3, .L1
        ubfx    x4, x1, 2, 2
        neg     x4, x4
        and     w4, w4, 3
        cmp     w4, w3
        csel    w6, w4, w3, ls
        cmp     w3, 4
        mov     w4, w3
        bhi     .L25
.L3:
        ldr     w6, [x1]
        ldr     w5, [x2]
        cmp     w4, 1
        add     w5, w6, w5
        str     w5, [x0]
        bls     .L16
        ldr     w6, [x1,4]
        ldr     w5, [x2,4]
        cmp     w4, 2
        add     w5, w6, w5
        str     w5, [x0,4]
        bls     .L17
        ldr     w6, [x1,8]
        ldr     w5, [x2,8]
        cmp     w4, 3
        add     w5, w6, w5
        str     w5, [x0,8]
        bls     .L18
        ldr     w7, [x1,12]
        ldr     w6, [x2,12]
        mov     w5, 4
        add     w6, w7, w6
        str     w6, [x0,12]
.L5:
        cmp     w3, w4
        beq     .L1
.L4:
        sub     w11, w3, w4
        lsr     w9, w11, 2
        lsl     w10, w9, 2
        cbz     w10, .L7
        ubfiz   x4, x4, 2, 32
        add     x8, x1, x4
        add     x7, x2, x4
        mov     w6, 0
        add     x4, x0, x4
.L13:
        ld1     {v1.4s}, [x7],16
        ld1     {v0.4s}, [x8],16
        add     v0.4s, v1.4s, v0.4s
        add     w6, w6, 1
        st1     {v0.4s}, [x4],16
        cmp     w9, w6
        bhi     .L13
        cmp     w11, w10
        add     w5, w5, w10
        beq     .L1
.L7:
        ubfiz   x6, x5, 2, 32
        ldr     w8, [x1,x6]
        ldr     w7, [x2,x6]
        add     w4, w5, 1
        add     w7, w8, w7
        str     w7, [x0,x6]
        cmp     w3, w4
        bls     .L1
        ubfiz   x4, x4, 2, 32
        ldr     w7, [x1,x4]
        ldr     w6, [x2,x4]
        add     w5, w5, 2
        add     w6, w7, w6
        str     w6, [x0,x4]
        cmp     w3, w5
        bls     .L1
        ubfiz   x5, x5, 2, 32
        ldr     w3, [x1,x5]
        ldr     w1, [x2,x5]
        add     w1, w3, w1
        str     w1, [x0,x5]
.L1:
        ret
.L25:
        mov     w4, 0
        mov     w5, w4
        cbz     w6, .L4
        mov     w4, w6
        b       .L3
.L18:
        mov     w5, 3
        b       .L5
.L16:
        mov     w5, 1
        b       .L5
.L17:
        mov     w5, 2
        b       .L5
        .size   add, .-add
        .ident  "GCC: (msgd) 4.8.2 20130805 (prerelease)"
```

Autovectorisation




```
        .cpu generic
        .file   "t.c"
        .text
        .align  2
        .global add
        .type   add, %function
add:
        cbz     w3, .L1
        ubfx    x4, x1, 2, 2
        neg     x4, x4
        and     w4, w4, 3
        cmp     w4, w3
        csel    w6, w4, w3, ls
        cmp     w3, 4
        mov     w4, w3
        bhi     .L25
.L3:
        ldr     w6, [x1]
        ldr     w5, [x2]
        cmp     w4, 1
        add     w5, w6, w5
        str     w5, [x0]
        bls     .L16
        ldr     w6, [x1,4]
        ldr     w5, [x2,4]
        cmp     w4, 2
        add     w5, w6, w5
        str     w5, [x0,4]
        bls     .L17
        ldr     w6, [x1,8]
        ldr     w5, [x2,8]
        cmp     w4, 3
        add     w5, w6, w5
        str     w5, [x0,8]
        bls     .L18
        ldr     w7, [x1,12]
        ldr     w6, [x2,12]
        mov     w5, 4
        add     w6, w7, w6
        str     w6, [x0,12]
.L5:
        cmp     w3, w4
        beq     .L1
.L4:
        sub     w11, w3, w4
        lsr     w9, w11, 2
        lsl     w10, w9, 2
        cbz     w10, .L7
        ubfiz   x4, x4, 2, 32
        add     x8, x1, x4
        add     x7, x2, x4
        mov     w6, 0
        add     x4, x0, x4
.L13:
        ld1     {v1.4s}, [x7],16
        ld1     {v0.4s}, [x8],16
        add     v0.4s, v1.4s, v0.4s
        add     w6, w6, 1
        st1     {v0.4s}, [x4],16
        cmp     w9, w6
        bhi     .L13
        cmp     w11, w10
        add     w5, w5, w10
        beq     .L1
.L7:
        ubfiz   x6, x5, 2, 32
        ldr     w8, [x1,x6]
        ldr     w7, [x2,x6]
        add     w4, w5, 1
        add     w7, w8, w7
        str     w7, [x0,x6]
        cmp     w3, w4
        bls     .L1
        ubfiz   x4, x4, 2, 32
        ldr     w7, [x1,x4]
        ldr     w6, [x2,x4]
        add     w5, w5, 2
        add     w6, w7, w6
        str     w6, [x0,x4]
        cmp     w3, w5
        bls     .L1
        ubfiz   x5, x5, 2, 32
        ldr     w3, [x1,x5]
        ldr     w1, [x2,x5]
        add     w1, w3, w1
        str     w1, [x0,x5]
.L1:
        ret
.L25:
        mov     w4, 0
        mov     w5, w4
        cbz     w6, .L4
        mov     w4, w6
        b       .L3
.L18:
        mov     w5, 3
        b       .L5
.L16:
        mov     w5, 1
        b       .L5
.L17:
        mov     w5, 2
        b       .L5
        .size   add, .-add
        .ident  "GCC: (msgd) 4.8.2 20130805 (prerelease)"
```

Autovectorisation

```
void add(int * restrict a, const int * restrict b, const int * restrict c, unsigned n)
{
  unsigned i;
  a = __builtin_assume_aligned (a, 32);
  b = __builtin_assume_aligned (b, 32);
  b = __builtin_assume_aligned (c, 32);
  for (i = 0; i < n; ++i)
    a[i] = b[i] + c[i];
}
```

Autovectorisation

```
        .cpu    generic
        .file   "t.c"
        .text
        .align  2
        .global add
        .type   add, %function
add:
        cbz     w3, .L1
        lsr     w6, w3, 2
        lsl     w4, w6, 2
        cbz     w4, .L10
        cmp     w3, 3
        bls     .L10
        mov     x5, 0
        mov     w7, w5
.L9:
        add     x8, x2, x5
        add     x9, x1, x5
        ld1     {v0.4s}, [x8]
        ld1     {v1.4s}, [x9]
        add     x8, x0, x5
        add     v0.4s, v1.4s, v0.4s
        add     w7, w7, 1
        st1     {v0.4s}, [x8]
        cmp     w6, w7
        add     x5, x5, 16
        bhi     .L9
        cmp     w3, w4
        beq     .L1
.L3:
        uxtw    x6, w4
        lsl     x6, x6, 2
        ldr     w8, [x1,x6]
        ldr     w7, [x2,x6]
        add     w5, w4, 1
        add     w7, w8, w7
        str     w7, [x0,x6]
        cmp     w3, w5
        bls     .L1
        ubfiz   x5, x5, 2, 32
        ldr     w7, [x1,x5]
        ldr     w6, [x2,x5]
        add     w4, w4, 2
        add     w6, w7, w6
        str     w6, [x0,x5]
        cmp     w3, w4
        bls     .L1
        ubfiz   x4, x4, 2, 32
        ldr     w3, [x1,x4]
        ldr     w1, [x2,x4]
        add     w1, w3, w1
        str     w1, [x0,x4]
.L1:
        ret
.L10:
        mov     w4, 0
        b       .L3
        .size   add, .-add
        .ident  "GCC: (msgd) 4.8.2 20130805 (prerelease)"
```

Autovectorisation

```
    .cpu    generic
    .file   "t.c"
    .text
    .align  2
    .global add
    .type   add, %function
```
Header

```
add:
    cbz     w3, .L1
    lsr     w6, w3, 2
    lsl     w4, w6, 2
    cbz     w4, .L10
    cmp     w3, 3
    bls     .L10
    mov     x5, 0
    mov     w7, w5
```
Function Setup

```
.L9:
    add     x8, x2, x5
    add     x9, x1, x5
    ld1     {v0.4s}, [x8]
    ld1     {v1.4s}, [x9]
    add     x8, x0, x5
    add     v0.4s, v1.4s, v0.4s
    add     w7, w7, 1
    st1     {v0.4s}, [x8]
    cmp     w6, w7
    add     x5, x5, 16
    bhi     .L9
    cmp     w3, w4
    beq     .L1
```
Vector Loop

```
.L3:
    uxtw    x6, w4
    lsl     x6, x6, 2
    ldr     w8, [x1,x6]
    ldr     w7, [x2,x6]
```
Vector Tidy up

```
    add     w5, w4, 1
    add     w7, w8, w7
    str     w7, [x0,x6]
    cmp     w3, w5
    bls     .L1
    ubfiz   x5, x5, 2, 32
    ldr     w7, [x1,x5]
    ldr     w6, [x2,x5]
    add     w4, w4, 2
    add     w6, w7, w6
    str     w6, [x0,x5]
    cmp     w3, w4
    bls     .L1
    ubfiz   x4, x4, 2, 32
    ldr     w3, [x1,x4]
    ldr     w1, [x2,x4]
    add     w1, w3, w1
    str     w1, [x0,x4]
.L1:
    ret
```
Vector Tidy up

```
.L10:
    mov     w4, 0
    b       .L3
```
Function Setup

```
    .size   add, .-add
    .ident  "GCC: (msgd) 4.8.2 20130805 (prerelease)"
```
Footer

Autovectorisation

```
void add(int * restrict a, const int * restrict b, const int * restrict c, unsigned n)
{
  unsigned i;
  assert (n % 4 == 0)
  a = __builtin_assume_aligned (a, 32);
  b = __builtin_assume_aligned (b, 32);
  b = __builtin_assume_aligned (c, 32);
  for (i = 0; i < n; ++i)
    a[i] = b[i] + c[i];
}
```

Neon Intrinsics

```
#include <arm_neon.h>

void add (int * __restrict a, const int * __restrict b, const int * __restrict c, unsigned n)
{
  unsigned i;
  int32x4_t* va = (int32x4_t*)a;
  const int32x4_t* vb = (const int32x4_t*)b;
  const int32x4_t* vc = (const int32x4_t*)c;

  for (i = 0; i < n; i += 4)
    *(va++) = vaddq_s32 (*(vb++), *(vc++));
}
```

Neon Intrinsics

```
    .cpu    generic
    .file   "t.c"
    .text
    .align  2
    .global add
    .type   add, %function
add:
    cbz     w3, .L1
    mov     w4, 0
.L3:
    ld1     {v1.4s}, [x1],16
    ld1     {v0.4s}, [x2],16
    add     v0.4s, v1.4s, v0.4s
    add     w4, w4, 4
    st1     {v0.4s}, [x0],16
    cmp     w3, w4
    bhi     .L3
.L1:
    ret
    .size   add, .-add
    .ident  "GCC: (GNU) 4.9.0
    20130902 (experimental)"
```

Neon Intrinsics

```
    .cpu    generic
    .file   "t.c"
    .text
    .align  2
    .global add
    .type   add, %function
```

Header

```
add:
    cbz     w3, .L1
    mov     w4, 0
```

Function Setup

```
.L3:
    ld1     {v1.4s}, [x1],16
```

Vector Loop

```
    ld1     {v0.4s}, [x2],16
    add     v0.4s, v1.4s, v0.4s
    add     w4, w4, 4
    st1     {v0.4s}, [x0],16
    cmp     w3, w4
    bhi     .L3
.L1:
    ret
```

Vector Loop

```
    .size   add, .-add
    .ident  "GCC: (GNU) 4.9.0 20130902 (experimental)"
```

Footer

Neon Intrinsics

```
#include <arm_neon.h>

void add (int * restrict a, const int * restrict b, const int * restrict c, unsigned n)
{
  int32x4_t* va = (int32x4_t*)a;
  const int32x4_t* vb = (const int32x4_t*)b;
  const int32x4_t* vc = (const int32x4_t*)c;

  for ( ; n != 0; n -= 4)
    *(va++) = vaddq_s32 (*(vb++), *(vc++));
}
```

Neon Intrinsics

```
    .cpu generic
    .file  "t.c"
    .text
    .align      2
    .global     add
    .type       add, %function
add:
    cbz  w3, .L1
.L3:
    ld1  {v1.4s}, [x1],16
    ld1  {v0.4s}, [x2],16
    add v0.4s, v1.4s, v0.4s
    st1  {v0.4s}, [x0],16
    subs       w3, w3, #4
    bne .L3
.L1:
    ret
    .size       add, .-add
    .ident      "GCC: (GNU) 4.9.0
    20130902 (experimental)"
```

Neon Intrinsics

```
.cpu generic
.file  "t.c"
.text
.align      2
.global     add
.type       add, %function
```

Header

```
add:
    cbz  w3, .L1
```

Function Setup

```
.L3:
    ld1  {v1.4s}, [x1],16
    ld1  {v0.4s}, [x2],16
    add v0.4s, v1.4s, v0.4s
    st1  {v0.4s}, [x0],16
    subs       w3, w3, #4
    bne .L3
```

Vector Loop

```
.L1:
    ret
```

```
    .size       add, .-add
    .ident      "GCC: (GNU) 4.9.0
20130902 (experimental)"
```

Footer

Neon Intrinsics

```
        .cpu cortex-a9
        .eabi_attribute 27, 3
        .eabi_attribute 28, 1
        .fpu neon
        .eabi_attribute 20, 1
        .eabi_attribute 21, 1
        .eabi_attribute 23, 3
        .eabi_attribute 24, 1
        .eabi_attribute 25, 1
        .eabi_attribute 26, 1
        .eabi_attribute 30, 2
        .eabi_attribute 34, 1
        .eabi_attribute 18, 4
        .file   "t.c"
        .text
        .align  2
        .global add
        .type   add, %function
add:
        @ args = 0, pretend = 0, frame = 0
        @ frame_needed = 0, uses_anonymous_args = 0
        @ link register save eliminated.
        cmp             r3, #0
        bxeq            lr
.L3:
        subs            r3, r3, #4
        vld1.64         {d18-d19}, [r1:64]!
        vld1.64         {d16-d17}, [r2:64]!
        vadd.i32        q8, q9, q8
        vst1.64         {d16-d17}, [r0:64]!
        bne             .L3
        bx              lr
        .size   add, .-add
        .ident  "GCC: (GNU) 4.9.0 20130902 (experimental)"
```

Autovectorisation

```
        .cpu generic
        .file   "t.c"
        .text
        .align  2
        .global add
        .type   add, %function
```

Header

```
add:
        cbz     w3, .L1
        lsr     w6, w3, 2
        lsl     w4, w6, 2
        cbz     w4, .L10
        cmp     w3, 3
        bls     .L10
        mov     x5, 0
        mov     w7, w5
```

Function Setup

```
.L9:
        add     x8, x2, x5
        add     x9, x1, x5
        ld1     {v0.4s}, [x8]
        ld1     {v1.4s}, [x9]
        add     x8, x0, x5
        add     v0.4s, v1.4s, v0.4s
        add     w7, w7, 1
        st1     {v0.4s}, [x8]
        cmp     w6, w7
        add     x5, x5, 16
        bhi     .L9
        cmp     w3, w4
        beq     .L1
```

Vector Loop

```
.L3:
        uxtw    x6, w4
        lsl     x6, x6, 2
        ldr     w8, [x1,x6]
        ldr     w7, [x2,x6]
        add     w5, w4, 1
        add     w7, w8, w7
        str     w7, [x0,x6]
        cmp     w3, w5
        bls     .L1
        ubfiz   x5, x5, 2, 32
        ldr     w7, [x1,x5]
        ldr     w6, [x2,x5]
        add     w4, w4, 2
        add     w6, w7, w6
        str     w6, [x0,x5]
        cmp     w3, w4
        bls     .L1
        ubfiz   x4, x4, 2, 32
        ldr     w3, [x1,x4]
        ldr     w1, [x2,x4]
        add     w1, w3, w1
        str     w1, [x0,x4]
.L1:
        ret
```

Vector Tidy up

```
.L10:
        mov     w4, 0
        b       .L3
```

Function Setup

```
        .size   add, .-add
        .ident  "GCC: (msgd) 4.8.2 20130805 (prerelease)"
```

Footer

Neon Intrinsics

```
#include <arm_neon.h>

void add (int * restrict a, const int * restrict b, const int * restrict c, unsigned n)
{
  int32x4_t* va = (int32x4_t*)a;
  const int32x4_t* vb = (const int32x4_t*)b;
  const int32x4_t* vc = (const int32x4_t*)c;

  for ( ; n != 0; n -= 4)
    *(va++) = vaddq_s32 (*(vb++), *(vc++));
}
```

More about Linaro: http://www.linaro.org/about/
More about Linaro engineering: http://www.linaro.org/engineering/
How to join: http://www.linaro.org/about/how-to-join
Linaro members: www.linaro.org/members