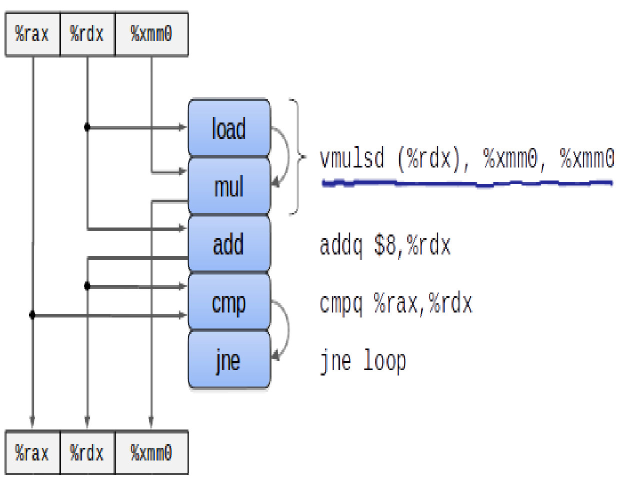
MicroOptimization for Latency and Throughput

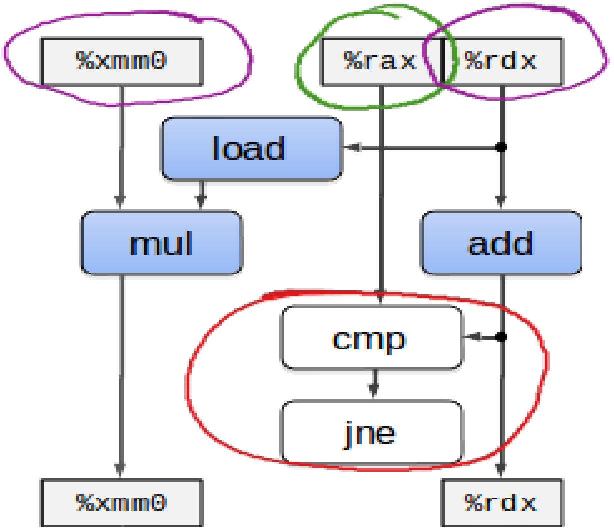
```c
void combine4(vec_ptr v, data_t *dest)
{
    long i;
    long length = vec_length(v);
    data_t *data = get_vec_start(v);
    data_t acc = IDENT;

    for (i = 0; i < length; i++) {
        acc = acc OP data[i];
    }
    *dest = acc;
}
```

```asm
# Inner loop of combine4.  data_t = double, OP = *
# acc in %xmm0, data+i in rdx, data+length in rax
.L25:                                   # loop:
        vmulsd  (%rdx), %xmm0, %xmm0    #   Multiply acc by data[i]
        addq    $8, %rdx                #   Increment data+i
        cmpq    %rax, %rdx              #   Compare to data+length
        jne     .L25                    #   If !=, goto loop
```



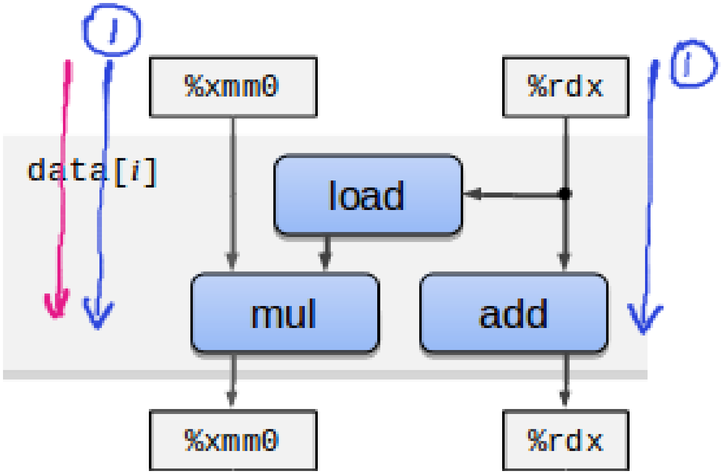
- informally the data flow graph of the code executes on the CPU.
- load the data from the address at rdx and pass it to the multiplication unit.
    - this is also an example of μops generated after decoding.
    - in this case a load unit is being used, and a multiplication unit is being used.
- multiply data[i] to xmm0 & store it in xmm0.

Optimization:ex4:Removing loop inefficiency



- we could classify the registers that are accessed into 4 categories
  - Read Only - Used as source values, not modified within the loop. holds the loop boundary value.
  - Write Only - Destination of data movement. (Nil)
  - Local - used within loop but no dependency from one iteration to another
  - Loop - Used as source and destination values. (rdx for counter and xmm0 for result)
- This checking can be done really quickly because of branch prediction done by the CPU.

Optimization:ex4:Removing loop inefficiency

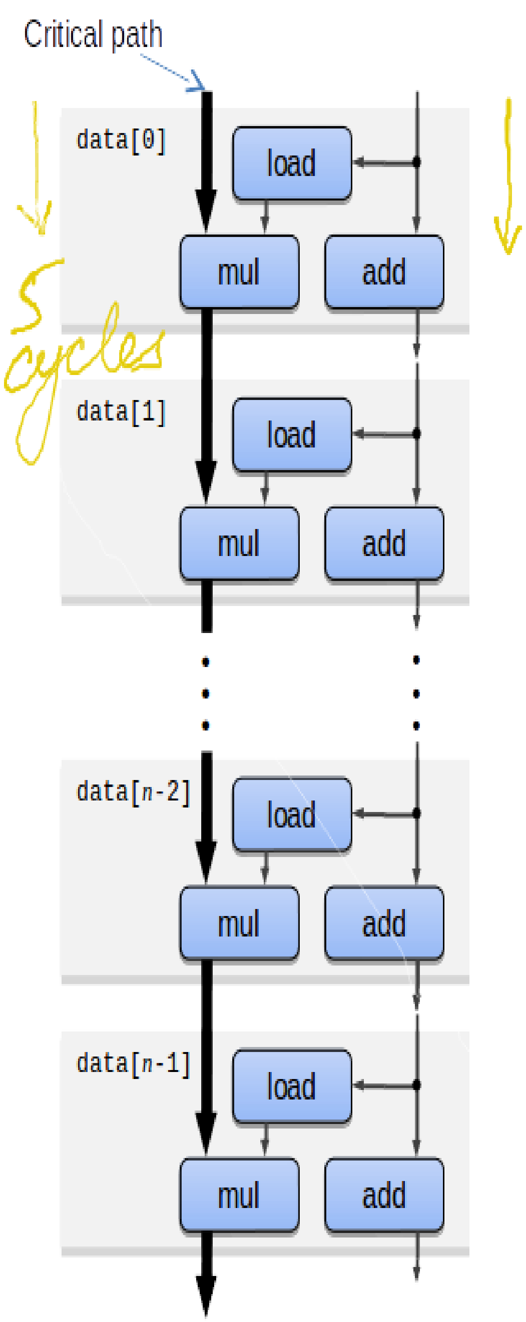


Method	Integer		Double FP	
Operation	Add	Mult	Add	Mult
Combine4	1.27	3.01	3.01	5.01
Latency Bound	1.00	3.00	3.00	5.00
Throughput Bound(C/I)	0.50	1.00	1.00	0.50

- Taking out the not critical operations we see that there are 2 dependencies
  1. should have a latency of 1 cycle (integer)
  2. should have a latency of 5 cycles (float)

  This will form the critical path.
- This figure also shows why we achieve a CPE of latency bound except for the integer.
- For all cases, where operation has a latency L greater than 1, measured CPE is L.

Optimization:ex4:Removing loop inefficiency



- The (data+i) reading data from memory addition and the test can proceed in parallel.
- As each successive value is computed, it is fed back around to compute the next value, but this will not occur until 5 cycles later.
- for integer why it takes 1.27 instead of 1 will require a much more detailed hardware knowledge than is publicly available.

Optimization:ex5:loop unrolling

```
void combine5(vec_ptr v, data_t *dest)
{
    long i;
    long length = vec_length(v);
    long limit = length-1;
    data_t *data = get_vec_start(v);
    data_t acc = IDENT;

    /* Combine 2 elements at a time */
    for (i = 0; i < limit; i+=2) {
      /* $begin combine5-update */
        acc = (acc OP data[i]) OP data[i+1];
      /* $end combine5-update */
    }

    /* Finish any remaining elements */
    for (; i < length; i++) {
        acc = acc OP data[i];
    }
    *dest = acc;
}
```

Function	Method	Integer +	Integer *	Floating point +	Floating point *
combine1	Abstract unoptimized	22.68	20.02	19.98	20.18
combine1	Abstract -O1	10.65	10.44	11.07	11.38
combine2	Move Vector length	07.03	09.03	08.85	10.98
combine3	Direct data access	07.27	09.02	08.82	10.98
combine4	Accumalate in temoporary	01.27	03.01	02.95	04.98
combine5	2X1 unroll loop	01.01	03.01	02.95	04.98
Latency Bound		1.00	3.00	3.00	5.00
Throughput Bound(C/I)		0.50	1.00	1.00	0.50

- CPE for integer addition improves, to achieve the latency bound of 1.00.
- This can be attributed to reducing the loop overhead operation relative to the number of additions required to compute the sum.
- On the other hand none of the other cases improve. (already at latency bound)