- Lab 8
  - A was due today
  - B due 6am this Wednesday (12/1)
  - C due 6am next Monday (12/6)

- Leave
  - Doesn’t update PC: just esp and ebp
  - Don’t need to pass in anything, just hard-code registers

```
void combine4(vec_ptr v, data_t *dest)
{
  int length = vec_length(v);
  data_t *d = get_vec_start(v);
  data_t x = IDENT;
  int i;
  for (i = 0; i < length; i++)
    t = t OP d[i];
  *dest = t;
}
```

```
void combine4(vec_ptr v, data_t *dest)
{
  int length = vec_length(v);
  data_t *d = get_vec_start(v);
  data_t x = IDENT;
  int i;
  for (i = 0; i < length; i++)
    t = t OP d[i];
  *dest = t;
}
```

```
void unroll2a_combine(vec_ptr v, data_t *dest)
{
    int length = vec_length(v);
    data_t *d = get_vec_start(v);
    data_t x = IDENT;
    int i;
    int limit = length-1;
    for (i = 0; i < limit; i+=2) {
        x = (x OP d[i]) OP d[i+1];
    }
    for (; i < length; i++) {
        x = x OP d[i];
    }
    *dest = x;
}
```

Loop Unrolling

```
void unroll2a_combine(vec_ptr v, data_t *dest)
{
    int length = vec_length(v);
    int limit = length-1;
    data_t *d = get_vec_start(v);
    data_t x = IDENT;
    int i;
    /* Combine 2 elements at a time */
    for (i = 0; i < limit; i+=2) {
        x = (x OP d[i]) OP d[i+1];
    }
    /* Finish any remaining elements */
    for (; i < length; i++) {
        x = x OP d[i];
    }
    *dest = x;
}
```

- Perform 2x more useful work per iteration

Effect of Loop Unrolling

		Integer		Floating Point		
Function	Method	+	*	+	F*	D*
combine 4	accum in local	2.0	3.0	3.0	4.0	5.0
combine5	unroll by x2	2.0	1.50	3.0	4.0	5.0
	unroll by x3	1.0	1.0	3.0	4.0	5.0

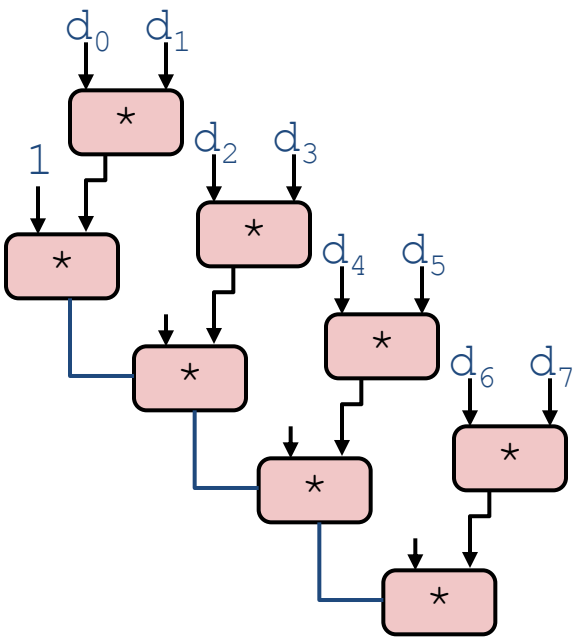
Loop Unrolling with Reassociation

```
void unroll2aa_combine(vec_ptr v, data_t *dest)
{
    int length = vec_length(v);
    int limit = length-1;
    data_t *d = get_vec_start(v);
    data_t x = IDENT;
    int i;
    /* Combine 2 elements at a time */
    for (i = 0; i < limit; i+=2) {
        x = x OP (d[i] OP d[i+1]);
    }
    /* Finish any remaining elements */
    for (; i < length; i++) {
        x = x OP d[i];
    }
    *dest = x;
}
```

- Can this change the result of the computation?
- Yes, for FP.

Reassociated Computation

```
x = x OP (d[i] OP d[i+1]);
```



- What changed:
  - Ops in the next iteration can be started early (no dependency)
- Overall Performance
  - N elements, D cycles latency/op
  - Should be (N/2+1)*D cycles: **CPE = D/2**
  - Measured CPE slightly worse for FP

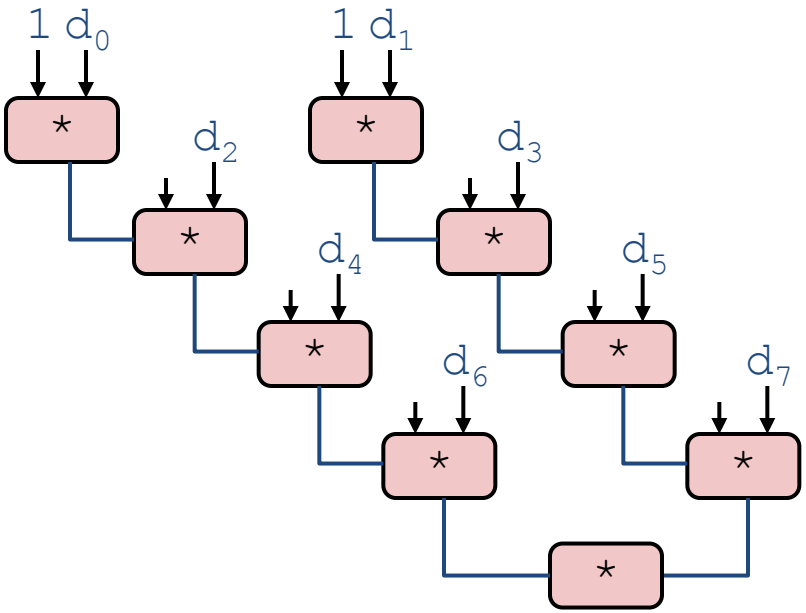
Loop Unrolling with Separate Accumulators

```
void unroll2a_combine(vec_ptr v, data_t *dest)
{
    int length = vec_length(v);
    int limit = length-1;
    data_t *d = get_vec_start(v);
    data_t x0 = IDENT;
    data_t x1 = IDENT;
    int i;
    /* Combine 2 elements at a time */
    for (i = 0; i < limit; i+=2) {
       x0 = x0 OP d[i];
       x1 = x1 OP d[i+1];
    }
    /* Finish any remaining elements */
    for (; i < length; i++) {
        x0 = x0 OP d[i];
    }
    *dest = x0 OP x1;
}
```

- Different form of reassociation

Separate Accumulators

```
x0 = x0 OP d[i];
x1 = x1 OP d[i+1];
```



- **What changed:**
  - Two independent "streams" of operations
- **Overall Performance**
  - N elements, D cycles latency/op
  - Should be (N/2+1)*D cycles: **CPE = D/2**
  - CPE matches prediction!

Unrolling & Accumulating

Idea

- Can unroll to any degree L
- Can accumulate K results in parallel
- L must be multiple of K

Limitations

- Diminishing returns
  - Cannot go beyond throughput limitations (execution units)
- Large overhead for short lengths

Calculating Improvement

- CPE_{old} / CPE_{new}

Calculating Improvement

- CPE_{old} / CPE_{new}
- CPE_{new} / CPE_{old}
- $100 (CPE_{new} – CPE_{old}) / CPE_{new}$
- $100 (CPE_{new} – CEP_{old}) / CEP_{old}$

Calculating Improvement

$CPE_{new} = 1.0$ $CPE_{old} = 2.0$

- **$CPE_{old} / CPE_{new} = 2.0$**
- $CPE_{new} / CPE_{old} = 0.5$
- $100 (CPE_{old} – CPE_{new}) / CPE_{new} = 100\%$
- $100 (CPE_{old} – CEP_{new}) / CEP_{old} = 50\%$

Profiling Demo

Amdahl’s Law