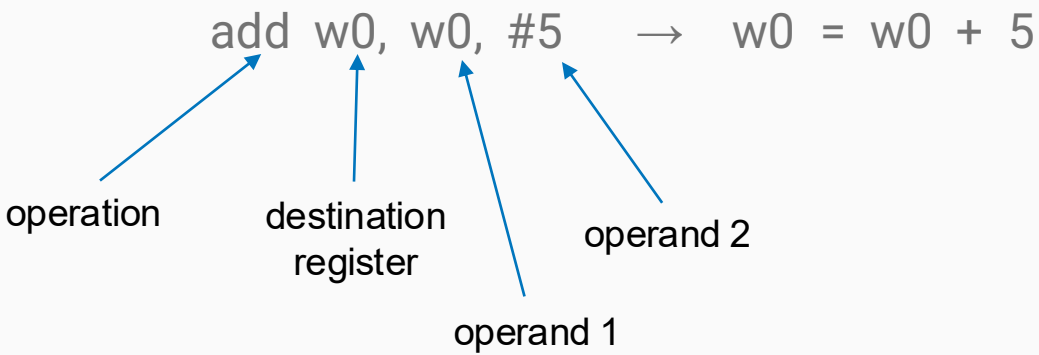
Recitation 11

Important Dates

- Homework 3, Final Submission is due TODAY, April 8, @ 8:00 PM
- Homework 4 will be released soon – get started on it early!

Compiler Level Optimizations

- We will be looking at **ARM64** architecture optimizations
  - Alternatives: **x86**, MIPS, SPARC, etc.
- Assembly Notation:



Loop to Math

- Loops require O(n) complexity, so compiler converts the loop into a simpler formula which takes O(1):

$$sum\_to\_n(n) = \sum_{i=0}^{n-1} i$$

$$sum\_to\_n(n) = \frac{n(n-1)}{2}$$

```
int sum_to_n(int n) {
    int result = 0;
    for (int i = 0; i < n; i++) {
        result += i;
    }
    return result;
}
```

→

```
sub     w8, w0, #1          ; w8 = n - 1
sub     w9, w0, #2          ; w9 = n - 2
umull   x9, w8, w9          ; x9 = (n-1) * (n-2)
lsr     x9, x9, #1          ; x9 = x9 / 2
add     w8, w8, w9          ; w8 = (n-1) + result
cmp     w0, #1              ; check if n < 1
csel    w0, wzr, w8, lt     ; if n < 1 return 0, else return w8
```

Dead Code Elimination

- Compiler identifies and eliminates redundant lines of code

```
int dead_code(int x) {
    int y = x + 5;      // computed
    y = x + 5;          // recomputed (pointless!)
    return y;
}
```

```
add     w0, w0, #5              ; w0 = w0 + 5
```

Pre-Calculating Math

- Programmers may use intermediate variables when coding to maintain logical flow and readability
- Compiler already knows that 2+3=5 and 5x10=50

```
int constant_folding(int x) {
    int a = 2 + 3;      // compiler does: 5
    int b = a * 10;     // compiler does: 50
    return x + b;       // x + 50
}
```

→

```
add     w0, w0, #50          ; w0 = w0 + 50
```

Simplifying Loop Math

- Multiplication is more expensive than addition
- Convert to closed equation where addition is the repeated operation, and multiplication happens once:

$$7 \times \frac{n(n-1)}{2}$$

```c
int repeated_multiplication(int n) {
    int result = 0;
    for (int i = 0; i < n; i++) {
        result += 7 * i;  // 7*0, 7*1, 7*2, ... could be a single multiply
    }
    return result;
}
```

→

```
subs    w8, w0, #1          ; w8 = n - 1, set flags
b.lt    LBB3_2              ; if n < 1, branch to return 0
sub     w9, w0, #2          ; w9 = n - 2
umull   x8, w8, w9          ; x8 = (n-1) * (n-2)
lsr     x8, x8, #1          ; x8 = x8 / 2
```

Reusing Math Results

- The expression (x + 5) is written for two intermediate results
- Compiler performs algebraic simplification so (x + 5) is calculated once:

$$2(x + 5) + 3(x + 5) = \mathbf{5x + 25}$$

```
int common_subexpr(int x) {
    int a = (x + 5) * 2;      // compute this
    int b = (x + 5) * 3;      // compute (x+5) again? or reuse?
    return a + b;
}
```

→

```
add     w8, w0, w0, lsl #2  ; w8 = w0 + (w0 * 4) = w0 * 5
add     w0, w8, #25         ; w0 = w8 + 25
```

Moving Math Outside the Loop

- Some operations don't change between loop iterations
  - If the loop runs 100 times, the CPU calculates the operation 100 times.
- Move the invariant outside the loop
  - Compiler deduced this loop is equal to **m x 5 x n**

```c
int loop_invariant(int n, int m) {
    int result = 0;
    for (int i = 0; i < n; i++) {
        result += m * 5;  // m * 5 doesn't change each iteration
    }
    return result;
}
```

→

```
mul     w8, w1, w0          ; Multiply m * n
add     w8, w8, w8, lsl #2  ; Multiply by 5
cmp     w0, #0              ; check if n > 0
csel    w0, w8, wzr, gt     ; if n > 0 return w8, else return 0
```

Flattening Recursion

- A function calls itself to calculate a sum. Every function call saves state to the “call stack”.
  - If n is too large, you run out of memory (Stack Overflow).
- The compiler rewrites the recursion as a loop or recognizes the sum pattern and uses the closed-form formula directly.

```c
int recursive_sum(int n) {
    if (n <= 0) return 0;
    return n + recursive_sum(n - 1);
}
```

→

```
sub     w8, w0, #1          ; w8 = n - 1
sub     w9, w0, #2          ; w9 = n - 2
umull   x9, w8, w9          ; x9 = (n-1) * (n-2)
lsr     x9, x9, #1          ; x9 = x9 / 2
madd    w8, w8, w8, w0      ; w8 = (n-1)*(n-1) + n
```