# Basic Blocks

# what's a basic block?

- A basic block is a straight-line piece of code with no control flow
- Basic rule: once you execute the first instruction of the basic block, you are guaranteed to execute all the instructions of the basic block
    - No way to exit out of the basic block before the end (no jump statements)
    - No way to enter the basic block after the beginning (no labels you can jump to)
- Control transfers occur between basic blocks

```
  ADD t7, t1, t2
Lab1:
  ADD t9, t1, t3
  SUB t2, t7, t9
  BNE t2, t1 Lab1
  ADD t2, t4, t7
```

# why a basic block?

- When we are optimizing code a key question we want to answer is: *will the transformed code behave the same as the original code?*
  - Must be true no matter how the program executes, no matter what input the program sees
  - If I can't guarantee this, I can't do the transformation!
- It is much easier to reason about the behavior of straight-line code than it is to reason about code with jumps and branches

```
  ADD t7, t1, t2
Lab1:
  ADD t9, t1, t3
  SUB t2, t7, t9
  BNE t2, t1 Lab1
  ADD t2, t4, t7
```

# more formally

- A basic block is a maximal sequence of instructions $I_0, I_1, I_2, \ldots, I_n$ such that if $I_j$ and $I_{j+1}$ are two adjacent statements in this sequence, then
  - The execution of $I_j$ is always immediately followed by the execution of $I_{j+1}$
  - The execution of $I_{j+1}$ is always immediate preceded by the execution of $I_j$

```
  ADD t7, t1, t2
Lab1:
  ADD t9, t1, t3
  SUB t2, t7, t9
  BNE t2, t1 Lab1
  ADD t2, t4, t7
```

# more formally

- A basic block is a maximal sequence of instructions $I_0, I_1, I_2, \ldots, I_n$ such that if $I_j$ and $I_{j+1}$ are two adjacent statements in this sequence, then
  - The execution of $I_j$ is always immediately followed by the execution of $I_{j+1}$
  - The execution of $I_{j+1}$ is always immediate preceded by the execution of $I_j$

```
  ADD t7, t1, t2
Lab1:
  ADD t9, t1, t3
  SUB t2, t7, t9
  BNE t2, t1 Lab1
  ADD t2, t4, t7
```

# more formally

- A basic block is a maximal sequence of instructions $I_0$, $I_1$, $I_2$, ..., $I_n$ such that if $I_j$ and $I_{j+1}$ are two adjacent statements in this sequence, then
  - The execution of $I_j$ is always immediately followed by the execution of $I_{j+1}$
  - The execution of $I_{j+1}$ is always immediate preceded by the execution of $I_j$

```
  ADD t7, t1, t2
Lab1:
  ADD t9, t1, t3
  SUB t2, t7, t9
  BNE t2, t1 Lab1
  ADD t2, t4, t7
```

# more formally

- A basic block is a maximal sequence of instructions $I_0$, $I_1$, $I_2$, ..., $I_n$ such that if $I_j$ and $I_{j+1}$ are two adjacent statements in this sequence, then
  - The execution of $I_j$ is always immediately followed by the execution of $I_{j+1}$
  - The execution of $I_{j+1}$ is always immediate preceded by the execution of $I_j$

```
  ADD t7, t1, t2
Lab1:
  ADD t9, t1, t3
  SUB t2, t7, t9
  BNE t2, t1 Lab1
  ADD t2, t4, t7
```

# more formally

- A basic block is a maximal sequence of instructions $I_0, I_1, I_2, ..., I_n$ such that if $I_j$ and $I_{j+1}$ are two adjacent statements in this sequence, then
  - The execution of $I_j$ is always immediately followed by the execution of $I_{j+1}$
  - The execution of $I_{j+1}$ is always immediate preceded by the execution of $I_j$

```
  ADD t7, t1, t2
Lab1:
  ADD t9, t1, t3
  SUB t2, t7, t9
  BNE t2, t1 Lab1
  ADD t2, t4, t7
```

# finding basic blocks

- Use three-address code
- Jump targets are labeled
- Also label beginning/end of functions
- Want to keep track of *targets of jump statements*
  - Any statement whose execution may immediately follow execution of jump statement
  - **Explicit target:** targets mentioned in jump statement
  - **Implicit target:** statements that follow conditional jump statements
    - The statement that gets executed if the branch is not taken

# finding basic blocks

```
A = 4
t1 = A * B
do {
  t2 = t1/C
  if (t2 ≥ W) {
    M = t1 * k
    t3 = M + I
  }
  H = I
  M = t3 - H
} while (T3 ≥ 0)
```

# finding basic blocks

```
    A = 4
    t1 = A * B
L1: t2 = t1 / C
    if t2 < W goto L2
    M = t1 * k
    t3 = M + I
L2: H = I
    M = t3 - H
    if t3 ≥ 0 goto L3
    goto L1
L3: halt
```

# finding basic blocks

- Step 1: Identify **leaders**: first statement of a basic block
- Step 2: In program order, construct a block by appending subsequent statements up to, but not including, the next leader

- Identifying leaders
  - First statement in the program
  - Explicit target of any conditional or unconditional branch
  - Implicit target of any branch

# partitioning algorithm

- Input: set of statements, *stat(i)* = i[th] statement in input
- Output: set of *leaders*, set of basic blocks where *block(x)* is the set of statements in the block with leader x
- Algorithm

*leaders* = {1} //Leaders always includes first statement
**for** i = 1 to |*n*| //|*n*| = number of statements
  **if** *stat(i)* is a branch, **then**
    *leaders* = *leaders* ∪ all potential targets
**end for**
*worklist* = *leaders*
**while** *worklist* not empty **do**
  x = remove earliest statement in *worklist*
  *block(x)* = {*x*}
  **for** (*i* = *x* + *1*; *i* ≤ |*n*| and *i* ∉ *leaders*; *i*++)
    *block(x)* = *block(x)* ∪ {*i*}
  **end for**
**end while**

# where are the basic blocks?

```
    A = 4
    t1 = A * B
L1: t2 = t1 / C
    if t2 < W goto L2
    M = t1 * k
    t3 = M + I
L2: H = I
    M = t3 - H
    if t3 ≥ 0 goto L3
    goto L1
L3: halt
```

# where are the basic blocks?

```
leader       A = 4
             t1 = A * B
leader   L1: t2 = t1 / C
             if t2 < W goto L2
leader       M = t1 * k
             t3 = M + I
leader   L2: H = I
             M = t3 - H
             if t3 ≥ 0 goto L3
leader       goto L1
leader   L3: halt
```

# where are the basic blocks?

| | |
|---|---|
| leader | `A = 4`<br>`t1 = A * B` |
| leader | `L1: t2 = t1 / C`<br>`if t2 < W goto L2` |
| leader | `M = t1 * k`<br>`t3 = M + I` |
| leader | `L2: H = I`<br>`M = t3 - H`<br>`if t3 ≥ 0 goto L3` |
| leader | `goto L1` |
| leader | `L3: halt` |

# next: control flow graphs