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 $I_0, I_1, I_2, ..., I_n$ 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 immediately 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
```
A basic block is a *maximal* sequence of instructions \( l_0, l_1, l_2, ..., l_n \) such that if \( l_j \) and \( l_{j+1} \) are two adjacent statements in this sequence, then

- The execution of \( l_j \) is always immediately followed by the execution of \( l_{j+1} \)
- The execution of \( l_{j+1} \) is always immediately preceded by the execution of \( l_j \)

---

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

• A basic block is a \textit{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 immediately preceded by the execution of $I_j$

\begin{verbatim}
ADD t7, t1, t2
Lab1:
  ADD t9, t1, t3
  SUB t2, t7, t9
  BNE t2, t1 Lab1
  ADD t2, t4, t7
\end{verbatim}
more formally

A basic block is a \textit{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 immediately preceded by the execution of $I_j$

\begin{verbatim}
ADD t7, t1, t2
Lab1:
  ADD t9, t1, t3
  SUB t2, t7, t9
  BNE t2, t1 Lab1
ADD t2, t4, t7
\end{verbatim}
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 immediately 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
```
• 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 immediately preceded by the execution of $I_j$
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 targets**: targets mentioned in jump statement
  • **Implicit targets**: 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

repeat {
  t2 = t1/C

  if (t2 ≥ W) {
    M = t1 * k
    t3 = M + I
  }

  H = I

  M = t3 - H

} until (T3 ≥ 0)
finding basic blocks

1. $A = 4$
2. $t1 = A \times B$
3. **L1**: $t2 = t1 / C$
4. if $t2 < W$ goto L2
5. $M = t1 \times k$
6. $t3 = M + I$
7. **L2**: $H = I$
8. $M = t3 - H$
9. if $t3 \geq 0$ goto L3
10. goto L1
11. **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, \( \text{stat}(i) = i^{\text{th}} \) statement in input

• Output: set of \textit{leaders}, set of basic blocks where \( \text{block}(x) \) is the set of statements in the block with leader \( x \)

• Algorithm

\[
\text{leaders} = \{1\} \quad // \text{Leaders always includes first statement}
\]

\[
\text{for } i = 1 \text{ to } |n| \quad // |n| = \text{number of statements}
\]

\[
\text{if } \text{stat}(i) \text{ is a branch, then}
\]

\[
\text{leaders} = \text{leaders} \cup \text{all potential targets}
\]

\text{end for}

\[
\text{worklist} = \text{leaders}
\]

\[
\text{while } \text{worklist} \text{ not empty do}
\]

\[
x = \text{remove earliest statement in worklist}
\]

\[
\text{block}(x) = \{x\}
\]

\[
\text{for } (i = x + 1; i \leq |n| \text{ and } i \not\in \text{leaders}; i++)
\]

\[
\text{block}(x) = \text{block}(x) \cup \{i\}
\]

\text{end for}

\text{end while}
where are the basic blocks?

1. \( A = 4 \)
2. \( t1 = A \times B \)
3. \( \textbf{L1: } t2 = t1 / C \)
4. \( \text{if } t2 < W \text{ goto L2} \)
5. \( M = t1 \times k \)
6. \( t3 = M + I \)
7. \( \textbf{L2: } H = I \)
8. \( M = t3 - H \)
9. \( \text{if } t3 \geq 0 \text{ goto L3} \)
10. \( \text{goto L1} \)
11. \( \textbf{L3: } \text{halt} \)
where are the basic blocks?

```
A = 4

1. t1 = A * B

2. L1: t2 = t1 / C

3. if t2 < W goto L2

4. M = t1 * k

5. t3 = M + I

6. L2: H = I

7. M = t3 - H

8. if t3 ≥ 0 goto L3

9. goto L1

10. L3: halt
```
where are the basic blocks?

| Leader | 1 | A = 4 |
| Leader | 2 | t1 = A * B |
| Leader | 3 | L1: t2 = t1 / C |
| Leader | 4 | if t2 < W goto L2 |
| Leader | 5 | M = t1 * k |
| Leader | 6 | t3 = M + I |
| Leader | 7 | L2: H = I |
| Leader | 8 | M = t3 - H |
| Leader | 9 | if t3 ≥ 0 goto L3 |
| Leader | 10 | goto L1 |
| Leader | 11 | L3: halt |
Control Flow Graphs
what's a control flow graph

• A directed graph $G(V, E)$ where:
  • $V$ (vertices) are the basic blocks in the program
  • $E$ (edges) are control flow edges between basic blocks

• A control flow edge shows that execution may proceed along that edge
  • It is possible (though not always guaranteed) that a program's execution can go from the source of the edge directly to the target of the edge

```
ADD t7, t1, t2
Lab1:
ADD t9, t1, t3
SUB t2, t7, t9
BNE t2, t1 Lab1
ADD t2, t4, t7
```
What's a control flow graph

- A directed graph $G(V, E)$ where:
  - $V$ (vertices) are the basic blocks in the program
  - $E$ (edges) are control flow edges between basic blocks

- A control flow edge shows that execution may proceed along that edge

- It is possible (though not always guaranteed) that a program’s execution can go from the source of the edge directly to the target of the edge

```
ADD t7, t1, t2
Lab1:
ADD t9, t1, t3
SUB t2, t7, t9
BNE t2, t1 Lab1
ADD t2, t4, t7
```
what’s a control flow graph

- A directed graph \( G(V, E) \) where:
  - \( V \) (vertices) are the basic blocks in the program
  - \( E \) (edges) are control flow edges between basic blocks

- A control flow edge shows that execution may proceed along that edge
  - It is possible (though not always guaranteed) that a program’s execution can go from the source of the edge directly to the target of the edge

```
ADD t7, t1, t2
Lab1:
ADD t9, t1, t3
SUB t2, t7, t9
BNE t2, t1 Lab1
ADD t2, t4, t7
```
adding control flow edges

• There is a directed edge from $B_1$ to $B_2$ if
  • There is a branch from the last statement of $B_1$ to the first statement (leader) of $B_2$
  • $B_2$ immediately follows $B_1$ in program order and $B_1$ does not end with an unconditional branch

• Input: block, a sequence of basic blocks

• Output: The CFG

  \[
  \text{for } i = 1 \text{ to } |\text{block}| \\
  x = \text{last statement of } \text{block}(i) \\
  \text{if } \text{stat}(x) \text{ is a branch, then} \\
  \text{for each explicit target } y \text{ of } \text{stat}(x) \\
  \quad \text{create edge from block } i \text{ to block } y \\
  \text{end for} \\
  \text{if } \text{stat}(x) \text{ is not unconditional then} \\
  \quad \text{create edge from block } i \text{ to block } i + 1 \\
  \text{end for}
  \]
A = 4

\[ t_1 = A \times B \]

\[ L_1: t_2 = t_1 / C \]

if \( t_2 < W \) goto L2

\[ M = t_1 \times k \]

\[ t_3 = M + I \]

\[ L_2: H = I \]

\[ M = t_3 - H \]

if \( t_3 \geq 0 \) goto L3

goto L1

\[ L_3: \text{halt} \]
1. A = 4
2. t1 = A * B
3. \(L1: t2 = t1 / C\)
4. if \(t2 < W\) goto \(L2\)
5. M = t1 * k
6. t3 = M + I
7. \(L2: H = I\)
8. M = t3 - H
9. if \(t3 \geq 0\) goto \(L3\)
10. goto \(L1\)
11. \(L3: \text{halt}\)