Representing Program Control Flow (Objectives)

- Given a function in intermediate form, the student will be able to find the leaders and basic blocks in the function.
- Given a set of basic blocks the student will be able to construct a control-flow graph

Motivation

- Suppose we have the following intermediate code

```
iLDI    2        ⇒ r1
  .
  .
  .
iLDI    2        ⇒ r1
```

Under what conditions can the second instruction be removed?

Control Flow

- To represent control flow we will use a graph where the nodes represent single-entry, single-exit program regions and the edges represent transfers of control
- A single-entry, single-exit region is called a basic block.

```
      brlt   L2 L1     r1
L1: nop
      ...
L2: nop
```

Computing Basic Blocks

- First need to find all intermediate statements that begin basic blocks – leaders
  1. The first instruction in a procedure
  2. a labeled instruction
  3. an instruction immediately after a branch

- What are the leaders?

	loadI	4	⇒ r1
	mul	r2,r1	⇒ r10
L1:	div	r10,r3	⇒ r11
	sub	r11,r4	⇒ r12
	brgt	r12	⇒ L2
	mul	r10,r6	⇒ r5
	add	r5,r7	⇒ r13
L2:	i2i	r7	⇒ r8
	sub	r13,r8	⇒ r5
	brge	r13	⇒ L4
	br	L1	
L4:	halt		

Computing Basic Blocks

```
FindBasicBlocks(P) {
    Leaders ∪= {P.first()}
    for (i = P.first().next(); i ≠
    NULL;
          i = i.next())
      if (i.labeled() ||
    i.prev().isbr())
        Leaders ∪= {i}
    Work = Leaders
    Blocks = ∅
    while (Work ≠ ∅) {
          b = new block()
          Blocks ∪= {b}
          i = Work.smallest()
          Work -= {i}
          b ∪= {i}
          i = i.next()
          while (i ≠ NULL &&
                  i ∉ Leaders) {
             b ∪= {i}
             i = i.next()
          }
    }
}
```

Example

```
1.        loadI   4        ⇒ r1   ←      leader
2.        mul     r2,r1    ⇒ r10  ←
3.   L1:  div     r10,r3   ⇒ r11  ←      leader
4.        sub     r11,r4   ⇒ r12  ←
5.        brgt    r12      ⇒ L2   ←
6.        mul     r10,r6   ⇒ r5   ←      leader
7.        add     r5,r7    ⇒ r13  ←
8.   L2:  i2i     r7       ⇒ r8   ←      leader
9.        sub     r13,r8   ⇒ r5   ←
10.       brge    r13      ⇒ L4   ←
11.       br      L1 ←                   leader
12.  L4:  halt ←                         leader
```

Construction the CFG

- Link the basic blocks so that paths through the program are represented as paths through the CFG

G = (V,E)

1. V is the set of basic blocks plus two special blocks Entry and Exit
    1. Entry is an empty basic block such that

       $$\forall v \in V, Entry \xrightarrow{*} v$$

    2. Exit is an empty basic block such that

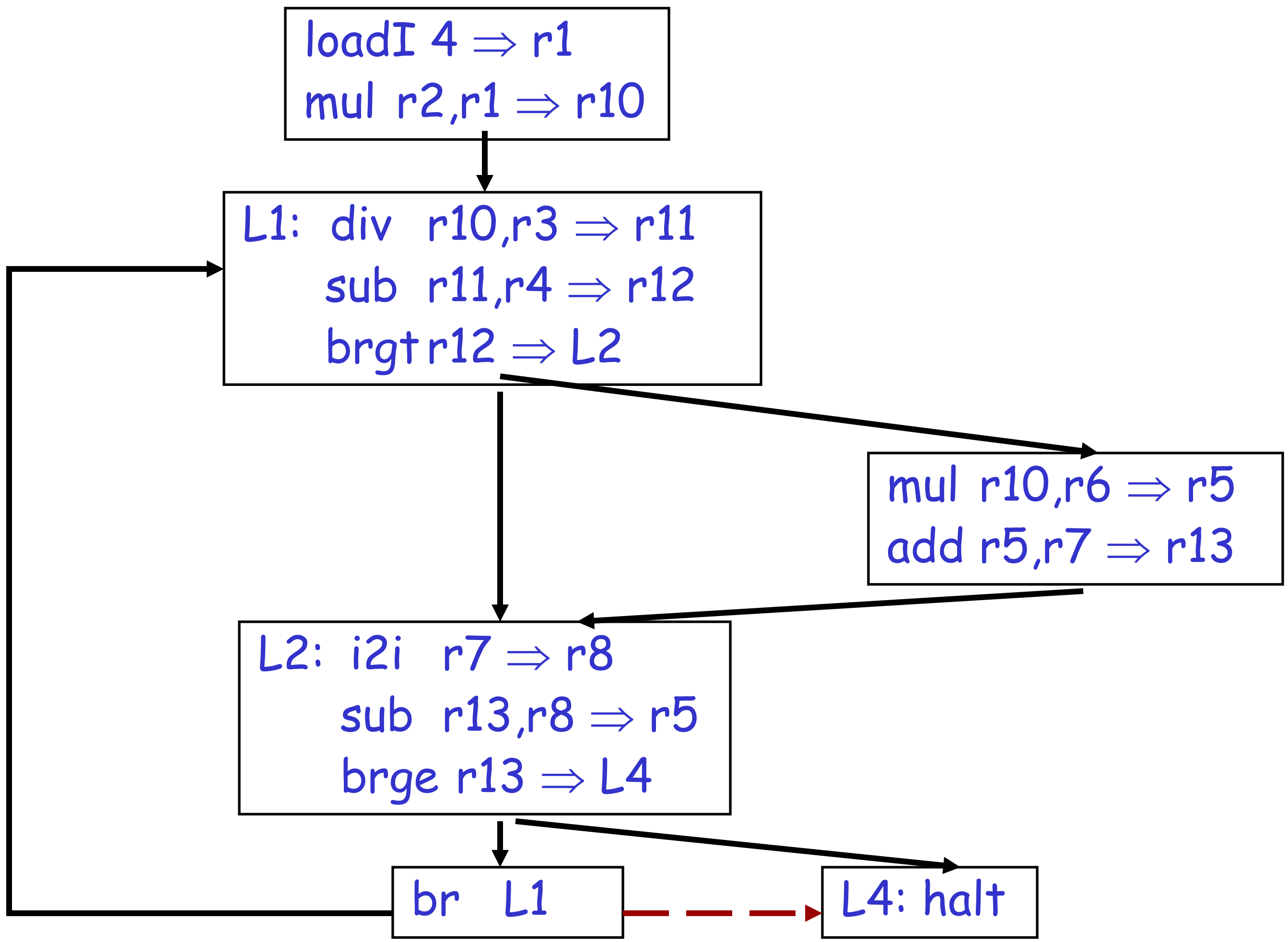
       $$\forall v \in V, v \xrightarrow{*} Exit$$

4. $\exists$ a directed edge from b_1 to b_2 iff
    1. there is a jump from the last instruction of b_1 to the first instruction of b_2
    2. b_2 immediately follows b_1 and b_1 does not end with an unconditional branch

Algorithm

```
ConstructCFG(B) {
   work = B
   while (work ≠ ∅) {
      b = some member of work
      work -= {b}
      if (b.last() is a branch)
         add an edge from b to block(b.last().target())
      if (b.last() is not an unconditional branch)
         add an edge from b to block(b.last().next())
   }
}
```

Example


loadI 4 ⇒ r1
mul r2,r1 ⇒ r10
L1: div r10,r3 ⇒ r11
sub r11,r4 ⇒ r12
brgt r12 ⇒ L2
mul r10,r6 ⇒ r5
add r5,r7 ⇒ r13
L2: i2i r7 ⇒ r8
sub r13,r8 ⇒ r5
brge r13 ⇒ L4
br L1
L4: halt