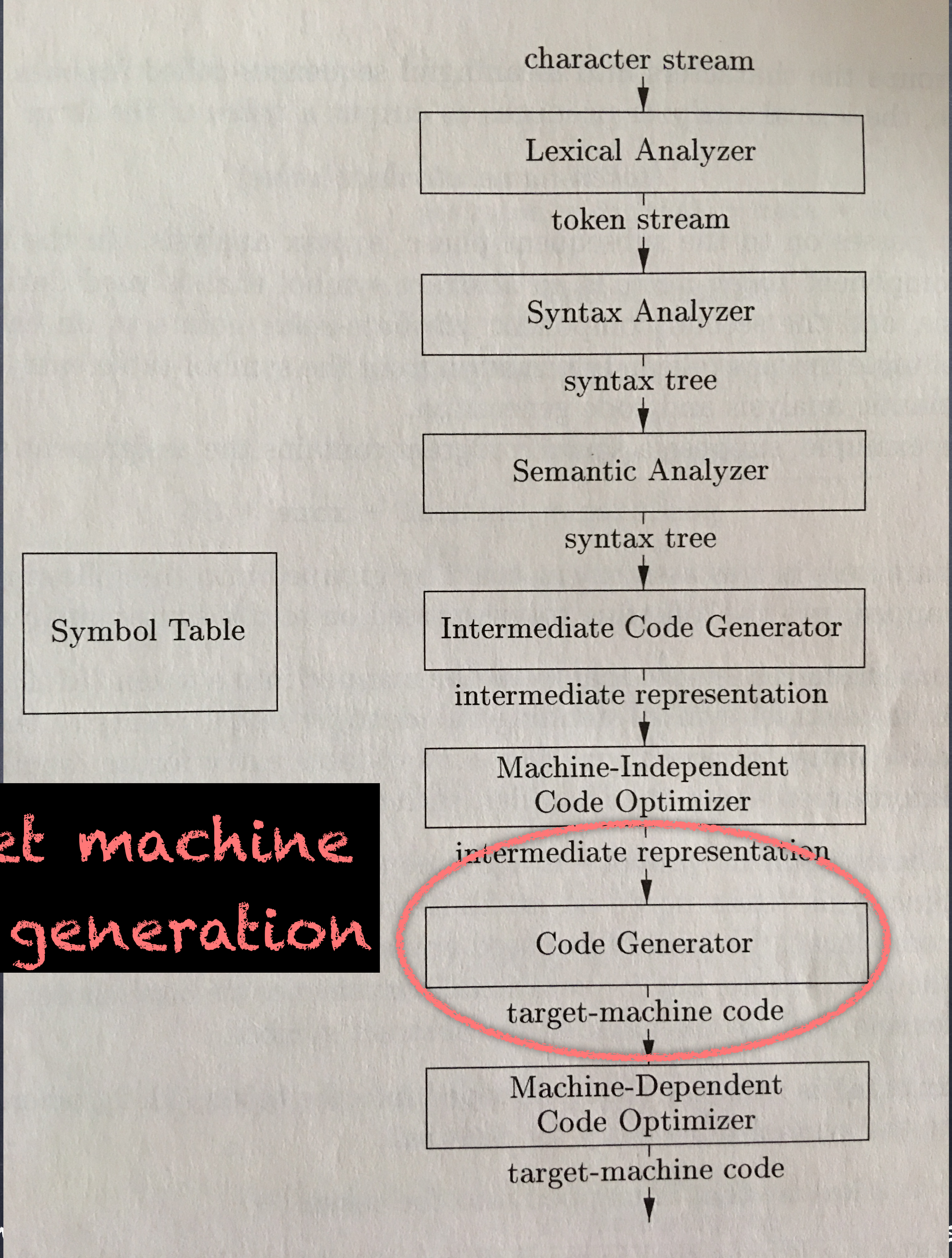
CSE443 Compilers

Dr. Carl Alphonce
alphonce@buffalo.edu
343 Davis Hall

Phases of a compiler



Target machine code generation

Figure 1.6, page 5 of text

Example [p. 546]

```
t = a - b
u = a - c
v = t + u
a = d
d = v + u
```

what does liveness and next use info looking like here?

Algorithm 8.7 [p. 528]
Determining the liveness and next-use information for each statement in a basic block.

INPUT: A basic block B of three address instructions. Assume the symbol table initially shows all non-temporary variables in B as being live on exit.

Not this instruction specifically, but instructions of the form x = y op z, x = op y, or x = y.

OUTPUT: At each statement i: x = y + z in B, we attach to i the liveness and next-use information for x, y, and z.

METHOD: We start at the last statement in B and scan backwards to the beginning of B. At each statement i: x = y + z in B do the following:

1) attach to statement i the information currently found in the symbol table regarding the next-use and liveness of x, y, and Z.
2) In the symbol table, set x to "not live" and "no next use".
3) In the symbol table, set y and z to "live" and the next uses of y and z to instruction i.

Example [p. 546]

```
1:t = a - b
2:u = a - c
3:v = t + u
4:a = d
5:d = v + u
```

	a	b	c	d	t	u	v
1:t = a - b							
2:u = a - c							
3:v = t + u							
4:a = d							
5:d = v + u							

INPUT: A basic block B of three address instructions. Assume the symbol table initially shows all non-temporary variables in B as being live on exit.

a	b	c	d	t	u	v
L	L	L	L			

Example [p. 546]

Statement	a	b	c	d	t	u	v
1:t = a - b							
2:u = a - c							
3:v = t + u							
4:a = d							
5:d = v + u				L			

We start at the last statement in B and scan backwards to the beginning of B. At each statement i:

$x = y + z$

in B do the following:

1) attach to statement i the information currently found in the symbol table regarding the next-use and liveness of x, y, and z.

2) In the symbol table, set x to "not live" and "no next use".

3) In the symbol table, set y and z to "live" and the next uses of y and z to instruction i.

a	b	c	d	t	u	v
L	L	L	L			

Example [p. 546]

We start at the last statement in B and scan backwards to the beginning of B. At each statement i:

x = y + z

in B do the following:

1) attach to statement i the information currently found in the symbol table regarding the next-use and liveness of x, y, and z.

2) In the symbol table, set x to "not live" and "no next use".

3) In the symbol table, set y and z to "live" and the next uses of y and z to instruction i.

	a	b	c	d	t	u	v
1:t = a – b							
2:u = a – c							
3:v = t + u							
4:a = d							
5:d = v + u				L			

a	b	c	d	t	u	v
L	L	L	D		L	L
					5	5

Example [p. 546]

We start at the last statement in B and scan backwards to the beginning of B. At each statement i:

$x = y + z$

in B do the following:

1) attach to statement i the information currently found in the symbol table regarding the next-use and liveness of x, y, and z.

2) In the symbol table, set x to "not live" and "no next use".

3) In the symbol table, set y and z to "live" and the next uses of y and z to instruction i.

	a	b	c	d	t	u	v
1:t = a - b							
2:u = a - c							
3:v = t + u							
4:a = d	L			D			
5:d = v + u				L			

a	b	c	d	t	u	v
L	L	L	D		L	L
					5	5

Example [p. 546]

	a	b	c	d	t	u	v
1:t = a - b							
2:u = a - c							
3:v = t + u							
4:a = d	L			D			
5:d = v + u				L			

We start at the last statement in B and scan backwards to the beginning of B. At each statement i:

x = y + z

in B do the following:

1) attach to statement i the information currently found in the symbol table regarding the next-use and liveness of x, y, and z.

2) In the symbol table, set x to "not live" and "no next use".

3) In the symbol table, set y and z to "live" and the next uses of y and z to instruction i.

a	b	c	d	t	u	v
D	L	L	L		L	L
			4		5	5

Example [p. 546]

```
1:t = a - b
2:u = a - c
3:v = t + u
4:a = d
5:d = v + u
```

a	b	c	d	t	u	v
					L 5	L 5
L			D			
			L			

We start at the last statement in B and scan backwards to the beginning of B. At each statement i:

x = y + z

in B do the following:

1) attach to statement i the information currently found in the symbol table regarding the next-use and liveness of x, y, and z.
2) In the symbol table, set x to "not live" and "no next use".
3) In the symbol table, set y and z to "live" and the next uses of y and z to instruction i.

a	b	c	d	t	u	v
D	L	L	L		L	L
			4		5	5

Example [p. 546]

We start at the last statement in B and scan backwards to the beginning of B. At each statement i:

x = y + z

in B do the following:

1) attach to statement i the information currently found in the symbol table regarding the next-use and liveness of x, y, and z.

2) In the symbol table, set x to "not live" and "no next use".

3) In the symbol table, set y and z to "live" and the next uses of y and z to instruction i.

```
1:t = a - b
2:u = a - c
3:v = t + u
4:a = d
5:d = v + u
```

a	b	c	d	t	u	v
					L 5	L 5
L			D			
			L			

a	b	c	d	t	u	v
D	L	L	L	L	L	D
			4	3	3	

Example [p. 546]

```
1:t = a - b
2:u = a - c
3:v = t + u
4:a = d
5:d = v + u
```

a	b	c	d	t	u	v
D		L			L 3	
					L 5	L 5
L			D			
			L			

We start at the last statement in B and scan backwards to the beginning of B. At each statement i:

x = y + z

in B do the following:

1) attach to statement i the information currently found in the symbol table regarding the next-use and liveness of x, y, and z.
2) In the symbol table, set x to "not live" and "no next use".
3) In the symbol table, set y and z to "live" and the next uses of y and z to instruction i.

a	b	c	d	t	u	v
D	L	L	L	L	L	D
			4	3	3	

Example [p. 546]

```
1:t = a - b
2:u = a - c
3:v = t + u
4:a = d
5:d = v + u
```

a	b	c	d	t	u	v
D		L			L 3	
					L 5	L 5
L			D			
			L			

We start at the last statement in B and scan backwards to the beginning of B. At each statement i:

$x = y + z$

in B do the following:

1) attach to statement i the information currently found in the symbol table regarding the next-use and liveness of x, y, and Z.

2) In the symbol table, set x to "not live" and "no next use".

3) In the symbol table, set y and z to "live" and the next uses of y and z to instruction i.

a	b	c	d	t	u	v
L	L	L	L	L	D	D
2		2	4	3		

Example [p. 546]

We start at the last statement in B and scan backwards to the beginning of B. At each statement i:

$x = y + z$

in B do the following:

1) attach to statement i the information currently found in the symbol table regarding the next-use and liveness of x, y, and z.
2) In the symbol table, set x to "not live" and "no next use".
3) In the symbol table, set y and z to "live" and the next uses of y and z to instruction i.

```
1:t = a - b
2:u = a - c
3:v = t + u
4:a = d
5:d = v + u
```

a	b	c	d	t	u	v
L 2	L			L 3		
D		L			L 3	
					L 5	L 5
L			D			
			L			

a	b	c	d	t	u	v
L	L	L	L	L	D	D
2		2	4	3		

Example [p. 546]

1:t = a – b

2:u = a – c

3:v = t + u

4:a = d

5:d = v + u

a	b	c	d	t	u	v
L 2	L			L 3		
D		L			L 3	
					L 5	L 5
L			D			
			L			

We start at the last statement in B and scan backwards to the beginning of B. At each statement i:

$x = y + z$

in B do the following:

1) attach to statement i the information currently found in the symbol table regarding the next-use and liveness of x, y, and z.

2) In the symbol table, set x to "not live" and "no next use".

3) In the symbol table, set y and z to "live" and the next uses of y and z to instruction i.

a	b	c	d	t	u	v
L	L	L	L	D	D	D
1	1	2	4			

Updating register descriptors (RD) and address descriptors (AD)

1. LD R, x
   (a) Set RD of R to only x
   (b) Add R to AD of x
   (c) Remove R from the AD of any variable other than x
2. ST x, R
   (a) Add &x to AD of x
3. OP Rx, Ry, Rz for x = y op z
   (a) Set RD of Rx to only x
   (b) Set AD of x to only Rx (&x not in AD of x !)
   (c) Remove Rx from the AD of any variable other than x
4. "When we process a copy statement x = y, after generating the load for y into register Ry, if needed, and after managing descriptors as for all load statement (per rule 1):" [p. 545]
   (a) Add x to the RD of Ry
   (b) Set AD of x to only Ry

R1	R2	R3

a	b	c	d	t	u	v

Register descriptor

Assume just 3 registers are available for the sake of this example.

Address descriptor

Variables t, u, and v are compiler-generated temporary variables.

```
t = a - b
u = a - c
v = t + u
a = d
d = v + u
```

R1	R2	R3

a	b	c	d	t	u	v
a	b	c	d			

```
t = a - b
u = a - c
v = t + u
a = d
d = v + u
```

At start of block, assume the values of variables a, b, c, and d are in main memory.

Variables t, u, and v are compiler-generated temporary variables.

R1	R2	R3

a	b	c	d	t	u	v
a	b	c	d			

t = a - b

```
LD R1, a
LD R2, b
SUB R2, R1, R2
```

	a	b	c	d	t	u	v
1:t = a - b	L 2	L			L 3		
2:u = a - c	D		L			L 3	
3:v = t + u						L 5	L 5
4:a = d	L			D			
5:d = v + u				L			

R1	R2	R3

a	b	c	d	t	u	v
a	b	c	d			

t = a - b

```
LD R1, a
LD R2, b
SUB R2, R1, R2
```

1. LD R, x
   (a) Set RD of R to only x
   (b) Add R to AD of x
   (c) Remove R from the AD of any var other than x
2. ST x, R
   (a) Add &x to AD of x
3. OP Rx, Ry, Rz for x = y op z
   (a) Set RD of Rx to only x
   (b) Set AD of x to only Rx (&x not in AD of x !)
   (c) Remove Rx from the AD of any var other than x
4. x = y (after generating LD Ry y (if needed) and using rule 1)
   (a) Add x to the RD of Ry
   (b) Set AD of x to only Ry

1:t = a - b

a	b	c	d	t	u	v
L 2	L			L 3		

R1	R2	R3

a	b	c	d	t	u	v
a	b	c	d			

t = a - b

```
LD R1, a
LD R2, b
SUB R2, R1, R2
```

R1	R2	R3
a	t	

a	b	c	d	t	u	v
a, R1	b	c	d	R2		

No registers are in use - pick the first two available for a and b. Choose to put t in R2 because b is not used again in this block.

```
1. LD R, x
    (a) Set RD of R to only x
    (b) Add R to AD of x
    (c) Remove R from the AD of any var other than x
2. ST x, R
    (a) Add &x to AD of x
3. OP Rx, Ry, Rz for x = y op z
    (a) Set RD of Rx to only x
    (b) Set AD of x to only Rx (&x not in AD of x !)
    (c) Remove Rx from the AD of any var other than x
4. x = y (after generating LD Ry y (if needed) and using rule 1)
    (a) Add x to the RD of Ry
    (b) Set AD of x to only Ry
```

1:t = a - b

a	b	c	d	t	u	v
L 2	L			L 3		

R1	R2	R3

a	b	c	d	t	u	v
a	b	c	d			

t = a - b

```
LD R1, a
LD R2, b
SUB R2, R1, R2
```

R1	R2	R3
a	t	

a	b	c	d	t	u	v
a, R1	b	c	d	R2		

u = a - c

```
LD R3, c
SUB R1, R1, R3
```

1. LD R, x
   (a) Set RD of R to only x
   (b) Add R to AD of x
   (c) Remove R from the AD of any var other than x
2. ST x, R
   (a) Add &x to AD of x
3. OP Rx, Ry, Rz for x = y op z
   (a) Set RD of Rx to only x
   (b) Set AD of x to only Rx (&x not in AD of x !)
   (c) Remove Rx from the AD of any var other than x
4. x = y (after generating LD Ry y (if needed) and using rule 1)
   (a) Add x to the RD of Ry
   (b) Set AD of x to only Ry

2:u = a - c

a	b	c	d	t	u	v
D		L			L 3	

R1	R2	R3

a	b	c	d	t	u	v

t = a - b

R1	R2	
a	t	

						v
a				R2		

a is already in R1, so no load needed.
t is used later, so don't overwrite R2.
load c into R3.
Put result into R1 since a is not needed again in this block.

u = a - c

```
LD R3, c
SUB R1, R1, R3
```

R1	R2	R3
u	t	c

a	b	c	d	t	u	v
a	b	c, R3	d	R2	R1	

```
1. LD R, x
    (a) Set RD of R to only x
    (b) Add R to AD of x
    (c) Remove R from the AD of any var other than x
2. ST x, R
    (a) Add &x to AD of x
3. OP Rx, Ry, Rz for x = y op z
    (a) Set RD of Rx to only x
    (b) Set AD of x to only Rx (&x not in AD of x !)
    (c) Remove Rx from the AD of any var other than x
4. x = y (after generating LD Ry y (if needed) and using rule 1)
    (a) Add x to the RD of Ry
    (b) Set AD of x to only Ry
```

2:u = a - c

a	b	c	d	t	u	v
D		L			L 3	

R1	R2	R3

a	b	c	d	t	u	v
a	b	c	d			

t = a - b

```
LD R1, a
LD R2, b
SUB R2, R1, R2
```

R1	R2	R3
a	t	

a	b	c	d	t	u	v
a, R1	b	c	d	R2		

u = a - c

```
LD R3, c
SUB R1, R1, R3
```

R1	R2	R3
u	t	c

a	b	c	d	t	u	v
a	b	c, R3	d	R2	R1	

v = t + u

```
ADD R3, R2, R1
```

a	b	c	d	t	u	v
					L S	L S

$\boxed{\text{3:v = t + u}}$

R1	R2	R3

a	b	c	d	t	u	v
a	b	c	d			

```
t = a - b
```

```
LD R1, a
LD R2, b
SUB R2, R1, R2
```

R1	R2	R3
a	t	

					u	v

t and u are already in registers - no loads needed.
Perform addition, putting the result into R3; c is no longer needed in this block.

```
u = a -
```

R1	R2	R3
u	t	c

					u	v
a	b	c, R3	d	R2	R1	

```
v = t + u
```

```
ADD R3, R2, R1
```

R1	R2	R3
u	t	v

a	b	c	d	t	u	v
a	b	c	d	R2	R1	R3

R1	R2	R3
u	t	v

a	b	c	d	t	u	v
a	b	c	d	R2	R1	R3

Same state as at end of previous slide

```
1. LD R, x
    (a) Set RD of R to only x
    (b) Add R to AD of x
    (c) Remove R from the AD of any var other than x
2. ST x, R
    (a) Add &x to AD of x
3. OP Rx, Ry, Rz for x = y op z
    (a) Set RD of Rx to only x
    (b) Set AD of x to only Rx (&x not in AD of x !)
    (c) Remove Rx from the AD of any var other than x
4. x = y (after generating LD Ry y (if needed) and using rule 1)
    (a) Add x to the RD of Ry
    (b) Set AD of x to only Ry
```

3:v = t + u

a	b	c	d	t	u	v
					L S	L S

R1	R2	R3
u	t	v

a	b	c	d	t	u	v
a	b	c	d	R2	R1	R3

a = d

LD R2, d

```
1. LD R, x
    (a) Set RD of R to only x
    (b) Add R to AD of x
    (c) Remove R from the AD of any var other than x
2. ST x, R
    (a) Add &x to AD of x
3. OP Rx, Ry, Rz for x = y op z
    (a) Set RD of Rx to only x
    (b) Set AD of x to only Rx (&x not in AD of x !)
    (c) Remove Rx from the AD of any var other than x
4. x = y (after generating LD Ry y (if needed) and using rule 1)
    (a) Add x to the RD of Ry
    (b) Set AD of x to only Ry
```

4:a = d

a	b	c	d	t	u	v
L			D			

R1	R2	R3
u	t	v

a	b	c	d	t	u	v
a	b	c	d	R2	R1	R3

a = d

LD R2, d

R1	R2	R3
u	a,d	v

a	b	c	d	t	u	v
R2	b	c	d,R2		R1	R3

Load d into R2, attach a to R2 as well.

```
1. LD R, x
    (a) Set RD of R to only x
    (b) Add R to AD of x
    (c) Remove R from the AD of any var other than x
2. ST x, R
    (a) Add &x to AD of x
3. OP Rx, Ry, Rz for x = y op z
    (a) Set RD of Rx to only x
    (b) Set AD of x to only Rx (&x not in AD of x !)
    (c) Remove Rx from the AD of any var other than x
4. x = y (after generating LD Ry y (if needed) and using rule 1)
    (a) Add x to the RD of Ry
    (b) Set AD of x to only Ry
```

4:a = d

a	b	c	d	t	u	v
L			D			

R1	R2	R3
u	t	v

a	b	c	d	t	u	v
a	b	c	d	R2	R1	R3

a = d

LD R2, d

R1	R2	R3
u	a,d	v

a	b	c	d	t	u	v
R2	b	c	d,R2		R1	R3

d = v + u

ADD R1, R3, R1

```
1. LD R, x
    (a) Set RD of R to only x
    (b) Add R to AD of x
    (c) Remove R from the AD of any var other than x
2. ST x, R
    (a) Add &x to AD of x
3. OP Rx, Ry, Rz for x = y op z
    (a) Set RD of Rx to only x
    (b) Set AD of x to only Rx (&x not in AD of x !)
    (c) Remove Rx from the AD of any var other than x
4. x = y (after generating LD Ry y (if needed) and using rule 1)
    (a) Add x to the RD of Ry
    (b) Set AD of x to only Ry
```

5:d = v + u

a	b	c	d	t	u	v
			L			

R1	R2	R3
u	t	v

a	b	c	d	t	u	v
a	b	c	d	R2	R1	R3

a = d

LD R2, d

R1	R2	R3
u	a,d	v

a	b	c	d	t	u	v
R2	b	c	d,R2		R1	R3

d = v + u

ADD R1, R3, R1

R1	R2	R3
d	a	v

a	b	c	d	t	u	v
R2	b	c	R1			R3

u and v are in
registers, so no loads needed.
Cannot destroy a (exists only in R2) without
storing back to memory, so use R1 for result.
Move d to R1 from R2.

d

a	b	c	d	t	u	v
			L			

R1	R2	R3
u	t	v

a	b	c	d	t	u	v
a	b	c	d	R2	R1	R3

a = d

```
LD R2, d
```

R1	R2	R3
u	a,d	v

a	b	c	d	t	u	v
R2	b	c	d,R2		R1	R3

d = v + u

```
ADD R1, R3, R1
```

R1	R2	R3
d	a	v

a	b	c	d	t	u	v
R2	b	c	R1			R3

exit

```
ST a, R2
ST d, R1
```

R1	R2	R3
u	t	v

a	b	c	d	t	u	v
a	b	c	d	R2	R1	R3

a = d

We're at the end of the block. Make sure that values of R1 and R2 are stored back to memory (d and a respectively). Value of R3 can be lost - it is a temporary of only this block.

d = v + u

R1	R2	R3
d	a	v

a	b	c	d	t	u	v
R2	b	c	R1			R3

exit

```
ST a, R2
ST d, R1
```

R1	R2	R3
d	a	v

a	b	c	d	t	u	v
a,R2	b	c	d,R1			R3