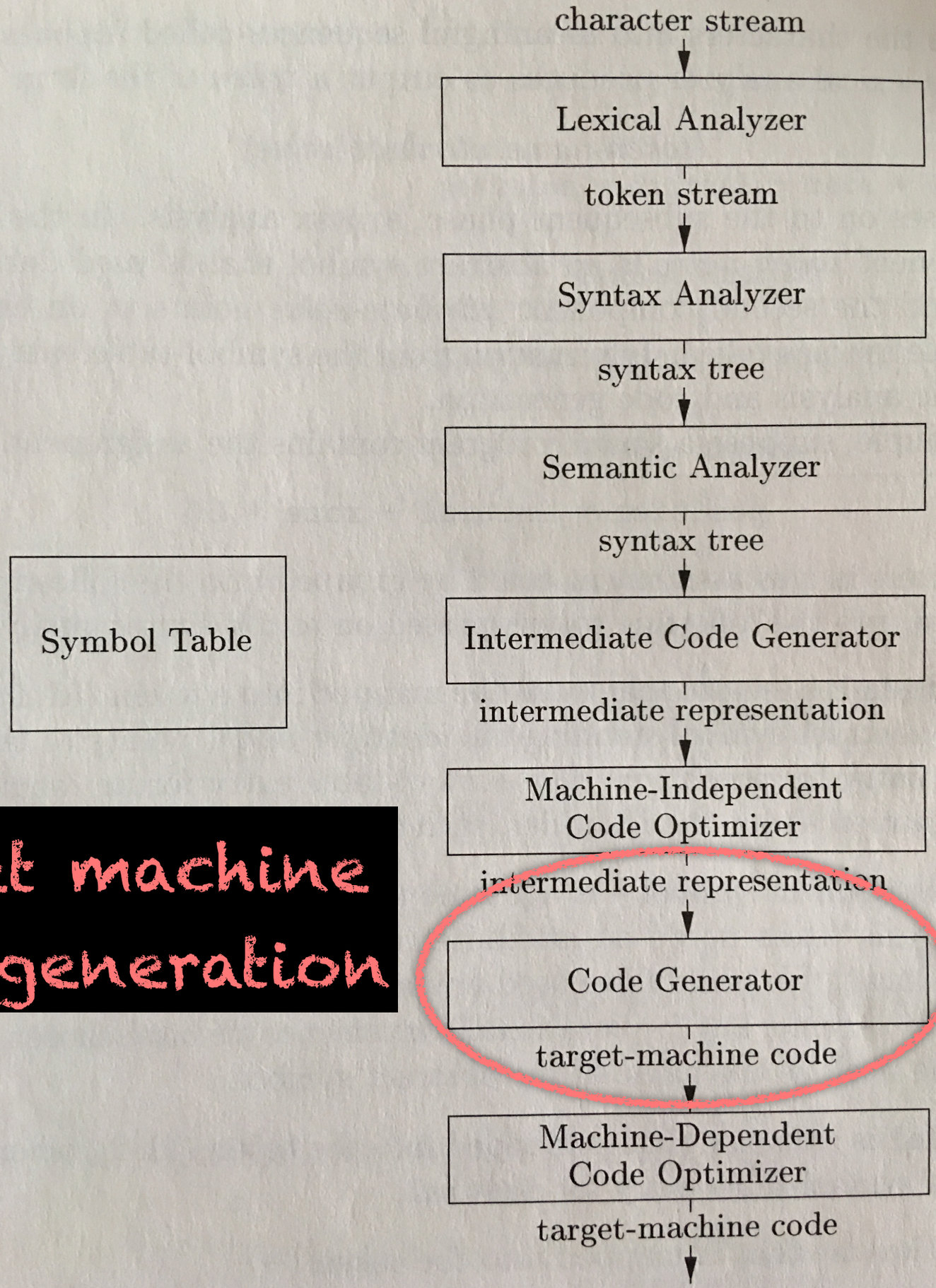
CSE443
Compilers

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

Phases of a compiler



Target machine code generation

Figure 1.6, page 5 of text

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
```

```
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			

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: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

```
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
```

```
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
```

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.

R1	R2	R3
a	t	

a	b	c	d	t	u	v
a				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	

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
```

3:v = t + u

a	b	c	d	t	u	v
					L S	L S

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

```
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
```

```
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.

```
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

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.

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
```

R1	R2	R3
d	a	v

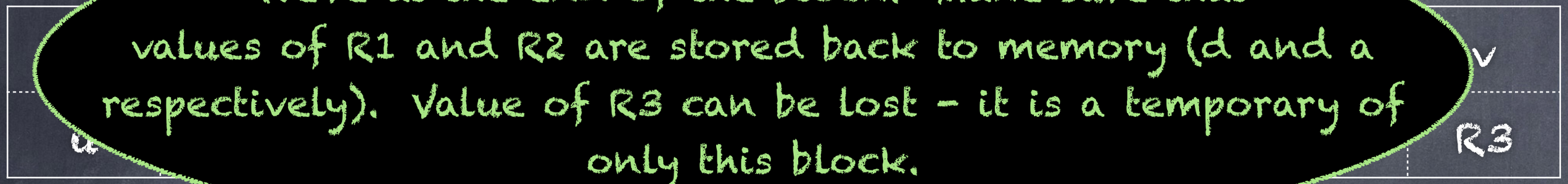
a	b	c	d	t	u	v
R2	b	c	R1			R3

```
exit
```

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

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

a = d



d = v + u

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

exit

ST a, R2
ST d, R1

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

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.

getReg function

x = y op z

How do we do this?

getReg function

x = y op z

"1. If y is currently in a register, pick a register already containing y as Ry. Do not issue a machine instruction to load this register, as none is needed.

2. If y is not in a register, but there is a register currently empty, pick one such register as Ry. [LD Ry, y]

getReg function

x = y op z

3. The difficult case occurs when y is not in a register, and there is no register that is currently empty. We need to pick one of the allowable registers anyway, and we need to make it safe to reuse. Let R be a candidate register, and suppose v is one of the variables that the register descriptor for R says is in R. We need to make sure that v's value either is not really needed, or that there is somewhere else we can go to get the value of v. The possibilities are:

getReg function

x = y op z

(a) If the address descriptor for v says that v is somewhere else besides R, then we are OK.

getReg function

x = y op z

(b) If v is x, the variable being computed by instruction I, and x is not also one of the other operands of instruction I (z in this example), then we are OK. The reason is that in this case we know this value of x is never again going to be used, so we are free to ignore it.

getReg function

x = y op z

(c) Otherwise, if v is not used later (that is, after the instruction I, there are no further uses of v, and if v is live on exit from the block, then v is recomputed within the block), then we are OK.

getReg function

x = y op z

(d) If we are not OK by one of the first three cases, then we need to generate the store instruction ST v, R to place a copy of v in its own memory location. This operation is called a spill." [p. 547–548]

getReg function

x = y op z

Repeat the above (a) - (d) steps for each variable v currently in R.

Let the score of R be the number of ST instructions generated. Choose the R with lowest score to actually use.

getReg function

x = y op z

We also need a register for the result, Rx. "The issues and options are almost as for y, so we shall only mention the differences.

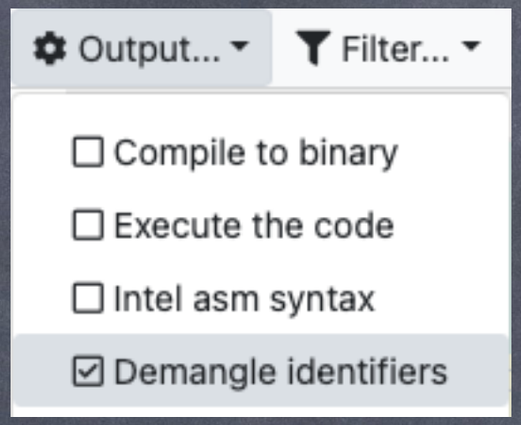
1. Since a new value of x is being computed, a register that holds only x is always an acceptable choice for Rx. This statement holds even if x is one of y and z, since our machine instructions allow two registers to be the same in one instruction.

getReg function

x = y op z

2. If y is not used after instruction I, in the sense described for variable v in item (3c), and Ry holds only y after being loaded, if necessary then Ry can also be used as Rx. A similar option holds regarding z and Rz." [p. 548]

godbolt.org



cerf.cse.buffalo.edu

```
gcc -fno-asynchronous-unwind-tables -O0 -S foo.c
```