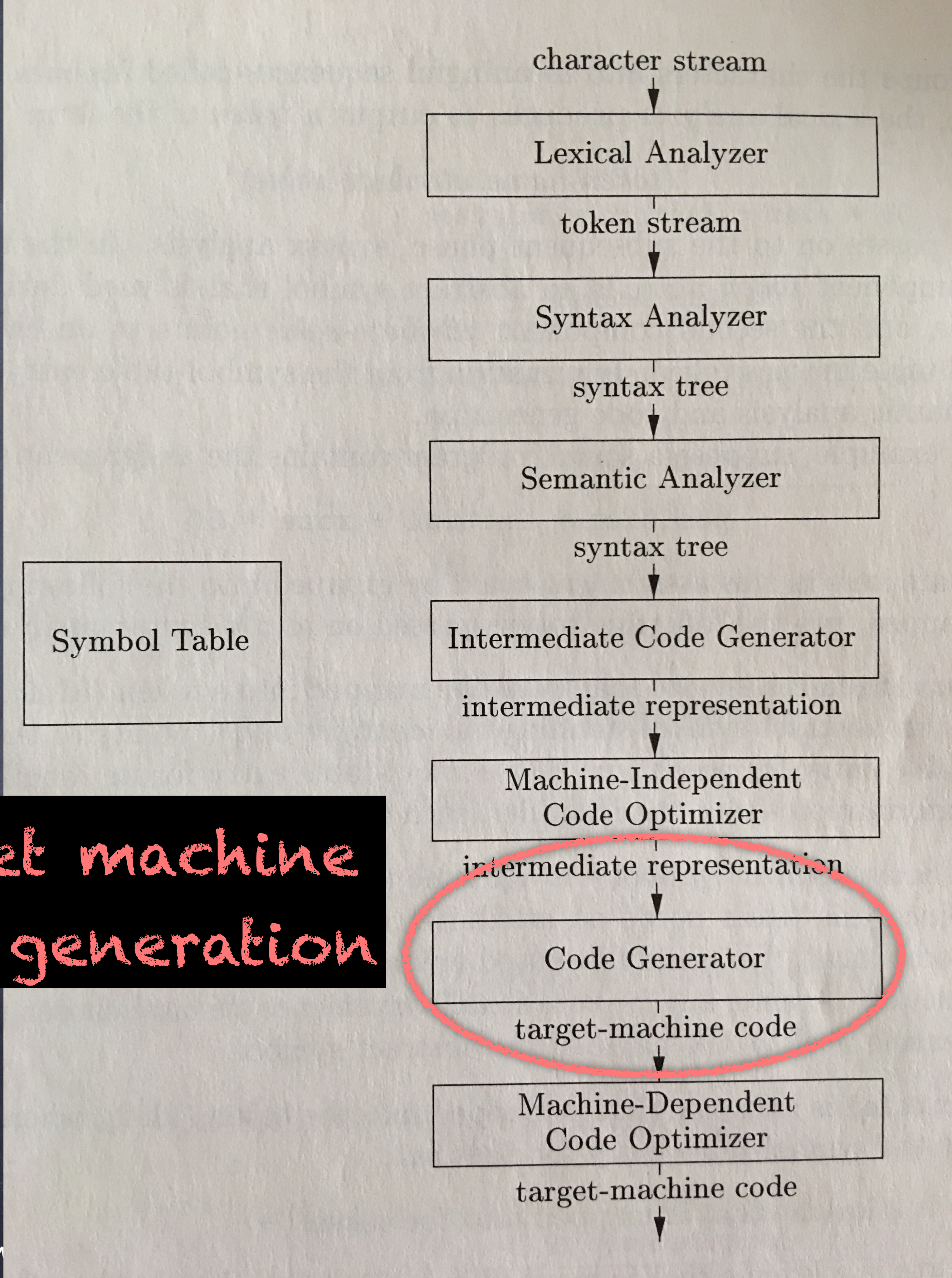
CSE443 Compilers

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

Phases of a compiler



Target machine code generation

Figure 1.6, page 5 of text

8.6 A Simple Code Generator [p. 542]

- algorithm focuses on generation of code for a single basic block
- generates code for each three address code instruction
- manages register allocations/assignment to avoid redundant loads/stores

Principal uses of registers

- operator operands must be in registers
- temporaries needed within block
- variables that span multiple blocks
- stack pointer
- function arguments

"We [...] assume that for each operator, there is exactly one machine instruction that takes the necessary operands in registers and performs that operation, leaving the result in a register. The machine instructions are of the form:

- LD reg, mem
- ST mem, reg
- OP reg, reg, reg" [p. 543]

```
movl    MEM, REG
movl    REG, MEM
addl    REG, REG
```

x86 assembly resources (will add more as we go along)
https://en.wikipedia.org/wiki/X86_assembly_language
https://gcc-renesas.com/pdf/manuals/Assembler.pdf
man as   <-- at OS prompt

8.6.1 Register and Address Descriptors

A three-address instruction of the form:

```
v = a op b
```

we generate:

```
LD Rx, a
LD Ry, b
OP Rx, Rx, Ry
ST Rx, v
```

8.6.1 Register and Address Descriptors

A three-address instruction of the form:

v = a op b

we generate:

```
LD Rx, a
LD Ry, b
OP Rx, Rx, Ry
ST Rx, v
```

where a, b, and v are int

```
v = a + b
```

in x86 asm

```
movl    -4(%rbp), %edx
movl    -8(%rbp), %eax
addl    %edx, %eax
movl    %eax, -12(%rbp)
```

- This results in many redundant loads and stores and may not make effective use of available registers.

- To better manage register use, employ two data structures:

  - register descriptor

  - address descriptor

register descriptor

"For each available register, a register descriptor keeps track of the variable names whose current value is in that register." [p. 543]

address descriptor

"For each program variable, an address descriptor keeps track of the location or locations where the current value of that variable can be found." [p. 543]

getReg function

"...getReg(I)...selects registers for each memory location associated with the three-address instruction I." [p. 544]

Note that I is an instruction, not a variable!

Example
(paraphrased from 8.6.2, page 544)

A three-address instruction of the form:

v = a op b

1. Use getReg(v = a op b) to select registers for v, a and b: Rv, Ra, and Rb respectively
2. If a is not already in Ra, generate LD Ra, a' (where a' is one of the possibly many current locations of a)
3. Similarly for b.
4. Generate OP Rv, Ra, Rb

copy instructions

$x = y$

"We assume getReg will always choose the same register for both x and y. If y is not already in that register Ry, then generate the machine instruction LD Ry, y. If y was already in Ry, we do nothing. It is only necessary that we adjust the register descriptor for Ry so that it includes x as one of the values found there." [p. 544]

Writing back to memory at end of block

At the end of a basic block we must ensure that live variables are stored back into memory.

"...for each variable x whose address descriptor does not say that its value is located in the memory location for x, we must generate the instruction ST x, R, where R is a register in which x's value exists at the end of the block." [p. 545]

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 Rx 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

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]

	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]

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			

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]

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
1							
2							
3							
4	L			D			
5				L			

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
```

Statement	a	b	c	d	t	u	v
1							
2							
3						L 5	L 5
4	L			D			
5				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
1							
2	D		L			L 3	
3						L 5	L 5
4	L			D			
5				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.

	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			

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

Example [p. 546]

Statement	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			

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			