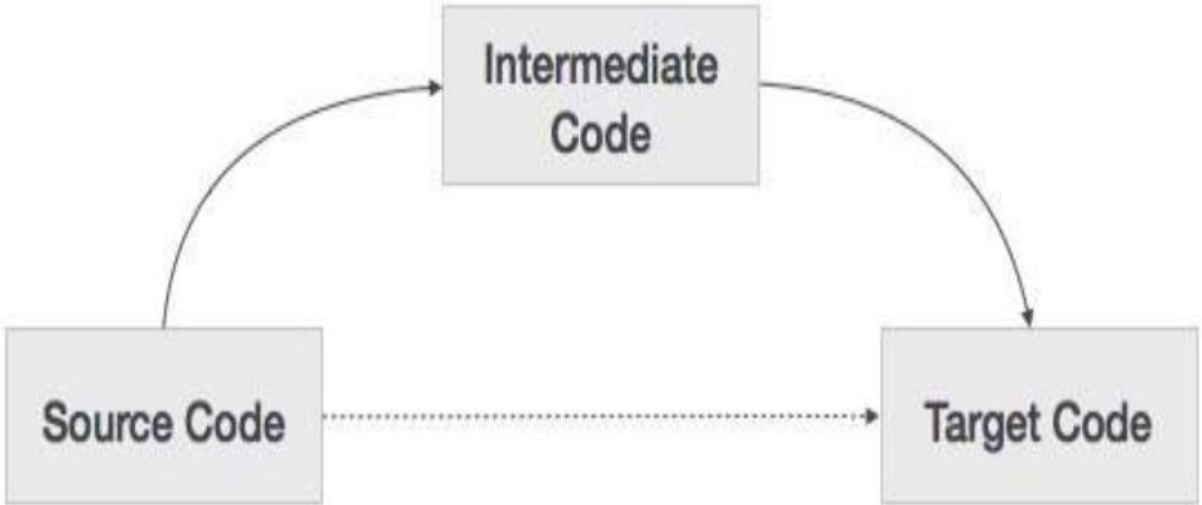
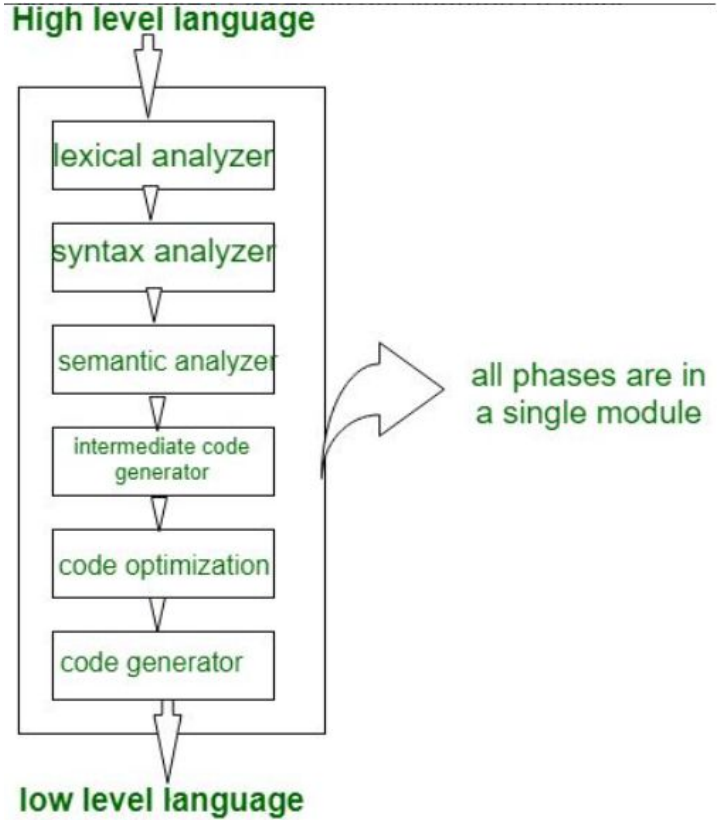
Intermediate Code Generation

- Many Compilers convert the high level language programs to intermediate form and then convert it to machine level language.

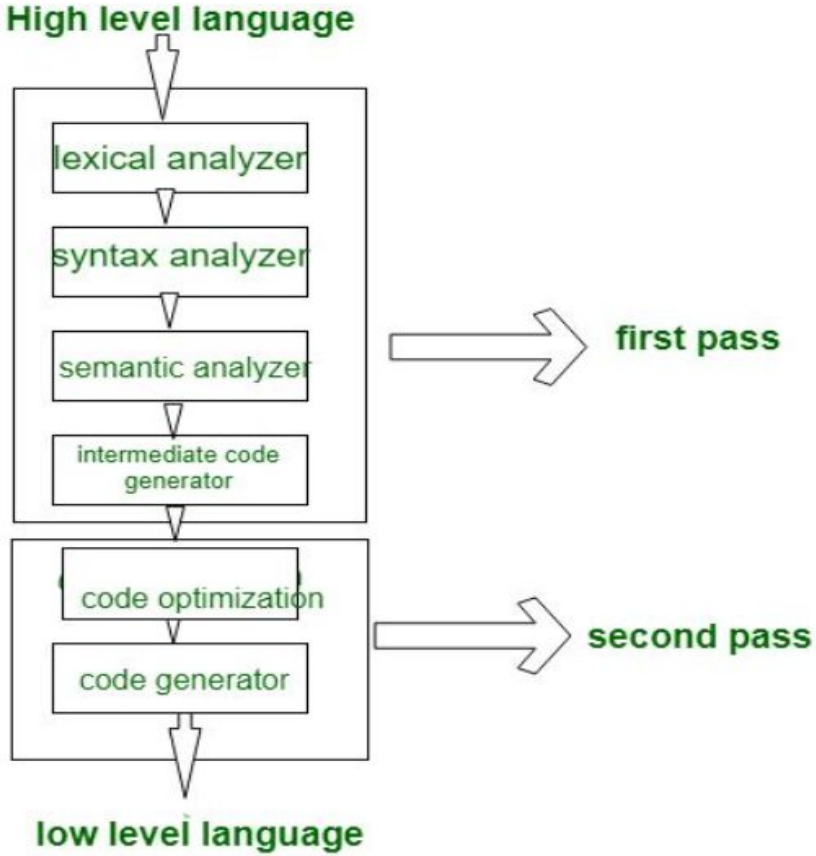


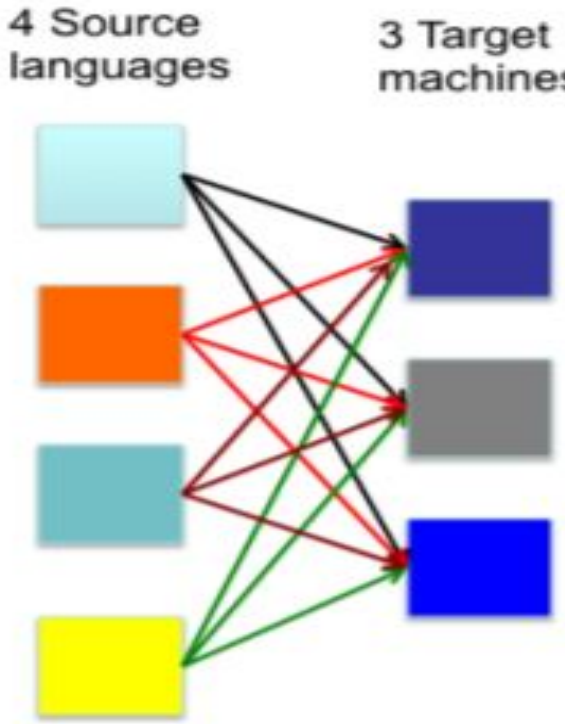
Compiler Passes

Single Pass

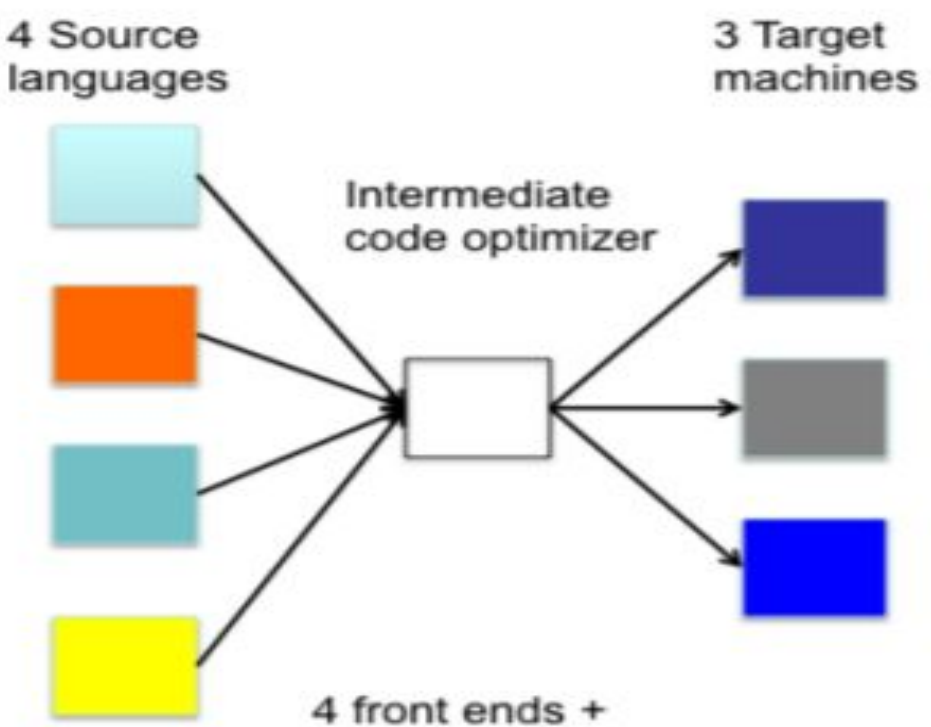


Two pass/ Multi pass compiler





4 front ends +
4x3 optimizers +
4x3 code generators



4 front ends +
1 optimizer +
3 code generators

Why Intermediate code ?

While generating machine code directly from source code is possible, but it entails two problems

- With m languages and n target machines, we need to write m front ends, m × n optimizers, and m × n code generators.
- The code optimizer which is one of the largest and very-difficult-to-write components of a compiler, cannot be reused.

This means just m front ends, n code generators and 1 optimizer

Intermediate code representations:

- Three-address code
- Postfix notation
- Syntax tree
- Directed acyclic graph

Three-address code

- A statement involving no more than three references(two for operands and one for result) is known as three address statement.
- A sequence of three address statements is known as three address code.
- Three address statement is of the form x = y op z , here x, y, z will have address (memory location). Sometimes a statement might contain less than three references but it is still called three address statement.

Three Address statements:

- x = y op z → Logical or binary operator
- x = op y → Unary operator
- x = y → Copy statement
- param x, param y → Procedure call
- x = y[i] → Index assignment
- if x relational op. y goto label → Conditional jump
- x = &y, x = *y → Pointer and address

Example:

a = b * c - d

r1 = b * c
r2 = r1 - d
a = r2

A three-address code has at most three address locations to calculate the expression. A three- address code can be represented in two forms :

- Quadruples
- Triples

Quadruples

Quadruples presentation is divided into four fields: operator, arg1, arg2, and result.

Example:

a = b * – c + b * – c

a = b * – c + b * – c

0. t1 = -c
1. t2 = b * t1
2. t3 = -c
3. t4 = b * t3
4. t5 = t2 + t4
5. a = t5

	Op	Arg. 1	Arg. 2	Result
0	Unary (-)	c		t1
1	*	b	t1	t2
2	Unary (-)	c		t3
3	*	b	t3	t4
4	+	t2	t4	t5
5	Assignment op. =	t5		a

Triples

- Each instruction in triples presentation has three fields : op, arg1, and arg2.

a = b * – c + b * – c

0. t1 = -c
1. t2 = b * t1
2. t3 = -c
3. t4 = b * t3
4. t5 = t2 + t4
5. a = t5

	Op	Arg. 1	Arg. 2
0	Unary (-)	c	
1	*	b	0
2	Unary (-)	c	
3	*	b	2
4	+	1	3
5	Assignment op. =	a	4

Indirect triples

This representation makes use of pointer to the listing of all references to computations which is made separately and stored.

a = b * – c + b * – c

0. t1 = -c
1. t2 = b * t1
2. t3 = -c
3. t4 = b * t3
4. t5 = t2 + t4
5. a = t5

#	Statement
0	14
1	15
2	16
3	17
4	18
5	19

#	Op	Arg. 1	Arg. 2
14	Unary (-)	c	
15	*	14	b
16	Unary (-)	c	
17	*	16	b
18	+	15	17
19	Assignment op. =	a	18

Questions

1. a + a * (b – c) + (b - c) * d
2. (x + y) * (y + z) + (x + y + z)

References

- https://youtube.com/playlist?list=PL-JvKqQx2Ate5DWhppx-MUOtGNA4S3spT
- https://www.gatevidyalay.com/implementation-of-three-address-code/
- NPTEL video: https://youtube.com/playlist?list=PLbMVogVj5nJTmKzaSlCpGgi7qxgcRRs8h