Compiladores 24-2
Back-end y generación de código

Lourdes del Carmen González Huesca
luglzhuesca@ciencias.unam.mx

Facultad de Ciencias, UNAM

27 Mayo 2024

Back-end

generación de código

Para generar código eficiente:

- Selección de instrucciones
- Register Transfer Language
- Explicit Register Transfer Language
- Location Transfer Language
  - Se eliminan los pseudoregistros para usar los registros físicos y los espacios en la pila para preparar la ejecución:
    - analizar la vida útil de los valores
    - construcción de la gráfica de interferencia
    - coloración de la gráfica para la asignación en la pila
- Código Linearizado

Back-end

generación de código

Para generar código eficiente:

- Selección de instrucciones
- Register Transfer Language
- Explicit Register Transfer Language
- Location Transfer Language
  - Se eliminan los pseudoregistros para usar los registros físicos y los espacios en la pila para preparar la ejecución:
    - analizar la vida útil de los valores
    - construcción de la gráfica de interferencia
    - coloración de la gráfica para la asignación en la pila
- Código Linearizado

¿Cómo asignar registros de manera eficiente?

Back-end

Asignación de registros

- Número finito de registros en un procesador que deben ser usados de la forma más eficiente posible: mapear todas las variables de un programa a los registros.

Back-end

Asignación de registros

- Número finito de registros en un procesador que deben ser usados de la forma más eficiente posible: mapear todas las variables de un programa a los registros.
- La asignación de registros maneja los registros para almacenar los valores de la variables y almacenar temporalmente valores directo en la memoria.

Back-end

Asignación de registros

- Número finito de registros en un procesador que deben ser usados de la forma más eficiente posible: mapear todas las variables de un programa a los registros.
- La asignación de registros maneja los registros para almacenar los valores de la variables y almacenar temporalmente valores directo en la memoria.
- La asignación se puede realizar en el middle-end o en el back-end pero en ambos casos se usan las mismas técnicas:

Back-end

Asignación de registros

- Número finito de registros en un procesador que deben ser usados de la forma más eficiente posible: mapear todas las variables de un programa a los registros.
- La asignación de registros maneja los registros para almacenar los valores de la variables y almacenar temporalmente valores directo en la memoria.
- La asignación se puede realizar en el middle-end o en el back-end pero en ambos casos se usan las mismas técnicas:
  - calcular si una variable esta *viva* en cierto punto o estado del programa
  - decidir si dos variables pueden compartir un registro
  - *spilling* (almacenamiento en memoria de ciertos valores)

Asignación de registros

liveness

El estado de un programa o un punto en su ejecución involucra un antes y un después de cierta instrucción cercana a lenguaje máquina.

Asignación de registros

liveness

El estado de un programa o un punto en su ejecución involucra un antes y un después de cierta instrucción cercana a lenguaje máquina.

Una variable está *viva* si el valor que contiene en ese punto o estado puede ser usado en cómputos futuros.
En otro caso se dice que está *muerta*.

Asignación de registros

liveness

El estado de un programa o un punto en su ejecución involucra un antes y un después de cierta instrucción cercana a lenguaje máquina.

Una variable está *viva* si el valor que contiene en ese punto o estado puede ser usado en cómputos futuros.
En otro caso se dice que está *muerta*.

1. si una instrucción usa el contenido de una variable entonces está *viva* al inicio de esa instrucción
2. si la variable se le asigna un valor en la instrucción pero no es usada como operando entonces está *muerta*
3. si la variable está viva al final de una instrucción y ésta no le asigna un valor entonces está viva al inicio de la instrucción
4. una variable está viva si al final de una instrucción si está viva al inicio de cualquier instrucción sucesiva inmediatamente

Asignación de registros

liveness

El estado de un programa o un punto en su ejecución involucra un antes y un después de cierta instrucción cercana a lenguaje máquina.

Una variable está *viva* si el valor que contiene en ese punto o estado puede ser usado en cómputos futuros.
En otro caso se dice que está *muerta*.

1. si una instrucción usa el contenido de una variable entonces está *viva* al inicio de esa instrucción
2. si la variable se le asigna un valor en la instrucción pero no es usada como operando entonces está *muerta*
3. si la variable está viva al final de una instrucción y ésta no le asigna un valor entonces está viva al inicio de la instrucción
4. una variable está viva si al final de una instrucción si está viva al inicio de cualquier instrucción sucesiva inmediatamente

liveness generated – killed – propagated

Asignación de registros

análisis de *liveness*

Diseñar ecuaciones que describan cuándo se genera *liveness*, cuándo muere y cuándo se propaga una variable

1. enumerar cada instrucción de una IR
2. calcular el conjunto de sucesores para cada instrucción, $suc[i]$, es decir las instrucciones que pueden estar después de ejecutar dicha instrucción
3. para cada instrucción calcular el conjunto de variables que pueden ser leidas o generadas por dicha instrucción $gen[i]$
4. calcular el conjunto de variables que son matadas en una instrucción $kill[i]$
5. para cada instrucción i se tienen dos conjuntos para describir el estado de *liveness* al inicio $in[i]$ y al final $out[i]$

Asignación de registros

análisis de *liveness*

suc Para cada tipo de instrucción se definen reglas para calcular las instrucciones siguientes:

1. la instrucción *j* está en *suc*[*i*] si está enlistada justo después de la instrucción *i*, a menos que *i* sea `GOTO` o `IF-THEN-ELSE`
2. si *i* es un `GOTO l`, entonces el número de la instrucción de `LABEL l` está en *suc*[*i*]
3. si la instrucción *i* es `IF p THEN l1 ELSE l2` entonces las instrucciones de `LABEL l1` y `LABEL l2` están en *suc*[*i*]

Asignación de registros

análisis de *liveness*

Instruction i	$gen[i]$	$kill[i]$
LABEL l	$\emptyset$	$\emptyset$
$x := y$	$\{y\}$	$\{x\}$
$x := k$	$\emptyset$	$\{x\}$
$x :=$ **unop** y	$\{y\}$	$\{x\}$
$x :=$ **unop** k	$\emptyset$	$\{x\}$
$x := y$ **binop** z	$\{y,z\}$	$\{x\}$
$x := y$ **binop** k	$\{y\}$	$\{x\}$
$x := M[y]$	$\{y\}$	$\{x\}$
$x := M[k]$	$\emptyset$	$\{x\}$
$M[x] :=$ y	$\{x,y\}$	$\emptyset$
$M[k] :=$ y	$\{y\}$	$\emptyset$
GOTO l	$\emptyset$	$\emptyset$
IF x **relop** y THEN l_t ELSE l_f	$\{x,y\}$	$\emptyset$
$x :=$ CALL $f(args)$	$args$	$\{x\}$

Conjuntos de variables generadas o muertas en una instrucción.

Asignación de registros

análisis de *liveness*

Ecuaciones para describir la variables vivas al inicio y al final de una instrucción que se resuelven usando puntos fijos

$$in[i] = gen[i] \bigcup \left(out[i] \backslash kill[i]\right)$$

$$out[i] = \bigcup_{j \in suc[i]} in[j]$$

Asignación de registros

análisis de *liveness*

Ecuaciones para describir la variables vivas al inicio y al final de una instrucción que se resuelven usando puntos fijos

$$in[i] = gen[i] \bigcup (out[i] \backslash kill[i])$$

$$out[i] = \bigcup_{j \in suc[i]} in[j]$$

Recursivamente:

- inicializar cada conjunto como vacíos
- repetir el cálculo de nuevos valores hasta que no hay cambios en el conjunto

Asignación de registros

ejemplo: *liveness*

$$in[i] = gen[i] \bigcup (out[i] \backslash kill[i])$$

$$out[i] = \bigcup_{j \in suc[i]} in[j]$$

```
1:  a := 0
2:  b := 1
3:  z := 0
4:  LABEL loop
5:  IF n = z THEN end ELSE body
6:  LABEL body
7:  t := a + b
8:  a := b
9:  b := t
10: n := n − 1
11: z := 0
12: GOTO loop
13: LABEL end
```

Asignación de registros

gráfica de interferencia

- ★ Una vez que se ha determinado el ciclo de *liveness* de cada variable se puede decidir si dos variables están en condición de compartir un registro.
- ✓ Dos variables pueden compartir un registro si ninguna de ellas interfiere con la otra, esto puede ser que no estén vivas en el mismo estado del programa.

Asignación de registros

gráfica de interferencia

★ Una vez que se ha determinado el ciclo de *liveness* de cada variable se puede decidir si dos variables están en condición de compartir un registro.

✓ Dos variables pueden compartir un registro si ninguna de ellas interfiere con la otra, esto puede ser que no estén vivas en el mismo estado del programa.

- Decimos que una variable x **interfiere** con una variable y si
  1. $x \neq y$
  2. hay una instrucción i tal que $x \in kill[i]$, $y \in out[i]$ y la instrucción i no es $x := y$

Asignación de registros

gráfica de interferencia

Condiciones para decidir interferencia:

- después de una asignación $x := y$, estas variables pueden estar vivas al mismo tiempo y pueden compartir un registro dado que tienen el mismo valor

Asignación de registros

gráfica de interferencia

Condiciones para decidir interferencia:

- después de una asignación $x := y$, estas variables pueden estar vivas al mismo tiempo y pueden compartir un registro dado que tienen el mismo valor
- si $x \notin out[i]$ a pesar de que $x \in kill[i]$ entonces x no debería de estar viva después de la instrucción i pero puede interferir con cualquier variable en $out[i]$

Asignación de registros

gráfica de interferencia

Condiciones para decidir interferencia:

- después de una asignación $x := y$, estas variables pueden estar vivas al mismo tiempo y pueden compartir un registro dado que tienen el mismo valor
- si $x \notin out[i]$ a pesar de que $x \in kill[i]$ entonces x no debería de estar viva después de la instrucción i pero puede interferir con cualquier variable en $out[i]$

Algunas condiciones pueden servir como optimizaciones al código y otras pueden asegurar la corrección del programa al preservar el comportamiento del programa.

Asignación de registros

gráfica de interferencia

Condiciones para decidir interferencia:

- después de una asignación $x := y$, estas variables pueden estar vivas al mismo tiempo y pueden compartir un registro dado que tienen el mismo valor
- si $x \notin out[i]$ a pesar de que $x \in kill[i]$ entonces x no debería de estar viva después de la instrucción i pero puede interferir con cualquier variable en $out[i]$

Algunas condiciones pueden servir como optimizaciones al código y otras pueden asegurar la corrección del programa al preservar el comportamiento del programa.
Para calcular las variables que interfieren:

1. considerar las instrucciones de asignación y la variable a la izquierda
2. para cada instrucción j en 1., calcular la interferencia al obtener la diferencia entre los conjuntos $out[j]$ y $kill[j]$
3. obtener una gráfica donde los vértices son variables y cada arista indica la interferencia obtenida en 2.

Asignación de registros

ejemplo: cálculo de in & out

```
1:  a := 0
2:  b := 1
3:  z := 0
4:  LABEL loop
5:  IF n = z THEN end ELSE body
6:  LABEL body
7:  t := a + b
8:  a := b
9:  b := t
10: n := n - 1
11: z := 0
12: GOTO loop
13: LABEL end
```

i	$succ[i]$	$gen[i]$	$kill[i]$
1	2		a
2	3		b
3	4		z
4	5		
5	6, 13	n, z	
6	7		
7	8	a, b	t
8	9	b	a
9	10	t	b
10	11	n	n
11	12		z
12	4		
13			

	Initial		Iteration 1		Iteration 2		Iteration 3	
i	$out[i]$	$in[i]$	$out[i]$	$in[i]$	$out[i]$	$in[i]$	$out[i]$	$in[i]$
1			n, a	n	n, a	n	n, a	n
2			n, a, b	n, a	n, a, b	n, a	n, a, b	n, a
3			n, z, a, b	n, a, b	n, z, a, b	n, a, b	n, z, a, b	n, a, b
4			n, z, a, b	n, z, a, b	n, z, a, b	n, z, a, b	n, z, a, b	n, z, a, b
5			a, b, n	n, z, a, b	a, b, n	n, z, a, b	a, b, n	n, z, a, b
6			a, b, n	a, b, n	a, b, n	a, b, n	a, b, n	a, b, n
7			b, t, n	a, b, n	b, t, n	a, b, n	b, t, n	a, b, n
8			t, n	b, t, n	t, n, a	b, t, n	t, n, a	b, t, n
9			n	t, n	n, a, b	t, n, a	n, a, b	t, n, a
10				n	n, a, b	n, a, b	n, a, b	n, a, b
11					n, z, a, b	n, a, b	n, z, a, b	n, a, b
12					n, z, a, b	n, z, a, b	n, z, a, b	n, z, a, b
13			a	a	a	a	a	a

Asignación de registros

ejemplo: gráfica de interferencia

```
1:  a := 0
2:  b := 1
3:  z := 0
4:  LABEL loop
5:  IF n = z THEN end ELSE body
6:  LABEL body
7:  t := a + b
8:  a := b
9:  b := t
10: n := n − 1
11: z := 0
12: GOTO loop
13: LABEL end
```

i	$succ[i]$	$gen[i]$	$kill[i]$
1	2		a
2	3		b
3	4		z
4	5		
5	6, 13	n, z	
6	7		
7	8	a, b	t
8	9	b	a
9	10	t	b
10	11	n	n
11	12		z
12	4		
13			

	Initial		Iteration 1		Iteration 2		Iteration 3	
i	$out[i]$	$in[i]$	$out[i]$	$in[i]$	$out[i]$	$in[i]$	$out[i]$	$in[i]$
1			n, a	n	n, a	n	n, a	n
2			n, a, b	n, a	n, a, b	n, a	n, a, b	n, a
3			n, z, a, b	n, a, b	n, z, a, b	n, a, b	n, z, a, b	n, a, b
4			n, z, a, b	n, z, a, b	n, z, a, b	n, z, a, b	n, z, a, b	n, z, a, b
5			a, b, n	n, z, a, b	a, b, n	n, z, a, b	a, b, n	n, z, a, b
6			a, b, n	a, b, n	a, b, n	a, b, n	a, b, n	a, b, n
7			b, t, n	a, b, n	b, t, n	a, b, n	b, t, n	a, b, n
8			t, n	b, t, n	t, n, a	b, t, n	t, n, a	b, t, n
9			n	t, n	n, a, b	t, n, a	n, a, b	t, n, a
10				n	n, a, b	n, a, b	n, a, b	n, a, b
11					n, z, a, b	n, a, b	n, z, a, b	n, a, b
12					n, z, a, b	n, z, a, b	n, z, a, b	n, z, a, b
13			a	a	a	a	a	a

Asignación de registros

gráfica de interferencia

- ✓ Después de obtener la gráfica de interferencia se decide si dos variables comparten registro si no están conectadas en la gráfica.

Asignación de registros

gráfica de interferencia

- ✓ Después de obtener la gráfica de interferencia se decide si dos variables comparten registro si no están conectadas en la gráfica.

  ¿Cómo asignar un registro a cada nodo de la gráfica?

Asignación de registros

gráfica de interferencia

✓ Después de obtener la gráfica de interferencia se decide si dos variables comparten registro si no están conectadas en la gráfica.

¿Cómo asignar un registro a cada nodo de la gráfica?

- a cada vértice de la gráfica se le asigna un número diferente de registro;
- el número de registros no debe sobrepasar el número de registros disponibles;

Asignación de registros

gráfica de interferencia

✓ Después de obtener la gráfica de interferencia se decide si dos variables comparten registro si no están conectadas en la gráfica.

¿Cómo asignar un registro a cada nodo de la gráfica?

- a cada vértice de la gráfica se le asigna un número diferente de registro;
- el número de registros no debe sobrepasar el número de registros disponibles;

¿Cómo resolver el problema anterior?

Asignación de registros

gráfica de interferencia

- ✓ Después de obtener la gráfica de interferencia se decide si dos variables comparten registro si no están conectadas en la gráfica.

  ¿Cómo asignar un registro a cada nodo de la gráfica?

  - a cada vértice de la gráfica se le asigna un número diferente de registro;
  - el número de registros no debe sobrepasar el número de registros disponibles;

  ¿Cómo resolver el problema anterior?

- ✓ Colorear la gráfica de interferencia con tantos colores como número de registros disponibles.

Asignación de registros

gráfica de interferencia

✓ Después de obtener la gráfica de interferencia se decide si dos variables comparten registro si no están conectadas en la gráfica.

¿Cómo asignar un registro a cada nodo de la gráfica?

- a cada vértice de la gráfica se le asigna un número diferente de registro;
- el número de registros no debe sobrepasar el número de registros disponibles;

¿Cómo resolver el problema anterior?

✓ Colorear la gráfica de interferencia con tantos colores como número de registros disponibles.

✗ Problema NP-completo

Asignación de registros

coloración de la gráfica

Usar una heurística para la coloración:

- si un nodo de la gráfica tiene menos de n aristas, con n el número de colores, entonces se puede omitir el nodo y colorear el resto de la gráfica
- las aristas de ese nodo pueden ser coloreadas con a lo más $n - 1$ colores
- asegurando que se puede colorear el nodo usando otro color

Asignación de registros

coloración de la gráfica

Usar una heurística para la coloración:

- si un nodo de la gráfica tiene menos de n aristas, con n el número de colores, entonces se puede omitir el nodo y colorear el resto de la gráfica
- las aristas de ese nodo pueden ser coloreadas con a lo más $n - 1$ colores
- asegurando que se puede colorear el nodo usando otro color

Si no existe un nodo con menos de n aristas, usar otra heurística.

Asignación de registros

spilling

- Si la coloración de la gráfica no se logra, entonces no se podrán almacenar los valores de las variables en los registros.
- Se deben seleccionar algunas variables para almacenar su valor en memoria.

Asignación de registros

spilling

- Si la coloración de la gráfica no se logra, entonces no se podrán almacenar los valores de las variables en los registros.
- Se deben seleccionar algunas variables para almacenar su valor en memoria.

  ¿Cuáles variables seleccionar para hacer spilling?

Asignación de registros

spilling

- Si la coloración de la gráfica no se logra, entonces no se podrán almacenar los valores de las variables en los registros.
- Se deben seleccionar algunas variables para almacenar su valor en memoria.

¿Cuáles variables seleccionar para hacer spilling?

✓ Serán las variables que no se les ha podido asignar un color al usar algún algoritmo que implemente una heurística para coloración.

- Se guardará en memoria a las variables que tengan más interferencias.
- Se modificará el código para incorporar estos registros (spilling).

Asignación de registros

spilling

x es una variable con muchas interferencias (inclusive mayor al número de registros)

- Escoger una dirección en memoria `addr_x` para almacenar el valor de la variable *x*
- Para cada instrucción *i* que lea o asigne a *x* se actualiza *x* a x_i
- Antes de cada instrucción *i* que use x_i se incluye `x_i := M[addr_x]`
- Después de cada instrucción *i* que asigne x_i se incluye
  `M[addr_x] := x_i`
- Si *x* es una variable viva al inicio del programa agregar `M[addr_x] := x` al inicio
- Si *x* es una variable viva al final del programa, agregar
  `x_i := M[addr_x]`

Asignación de registros

spilling

x es una variable con muchas interferencias (inclusive mayor al número de registros)

- Escoger una dirección en memoria `addr_x` para almacenar el valor de la variable *x*
- Para cada instrucción *i* que lea o asigne a *x* se actualiza *x* a x_i
- Antes de cada instrucción *i* que use x_i se incluye `x_i := M[addr_x]`
- Después de cada instrucción *i* que asigne x_i se incluye
  `M[addr_x] := x_i`
- Si *x* es una variable viva al inicio del programa agregar `M[addr_x] := x` al inicio
- Si *x* es una variable viva al final del programa, agregar
  `x_i := M[addr_x]`

Una vez que se ha actualizado el programa, se debe repetir el proceso de asignación de registros (cálculo de *liveness*, etc.).

Back-end

optimizaciones

- ✓ Eliminar redundancias en el código, mejorar la velocidad de ejecución y minimizar el tamaño del código.

Back-end

optimizaciones

- ✓ Eliminar redundancias en el código, mejorar la velocidad de ejecución y minimizar el tamaño del código.
- Optimizaciones en fases particulares del compilador: representaciones intermedias

Back-end

optimizaciones

- ✓ Eliminar redundancias en el código, mejorar la velocidad de ejecución y minimizar el tamaño del código.
- Optimizaciones en fases particulares del compilador: representaciones intermedias eliminar código muerto, propagar el uso de constantes, visibilizar los saltos o las excepciones, calcular invariantes de ciclos, tomar ventaja de las caracterísicas del procesador o del hardware
- Alcance de la optimización: local, global, interprocedural

Compilación

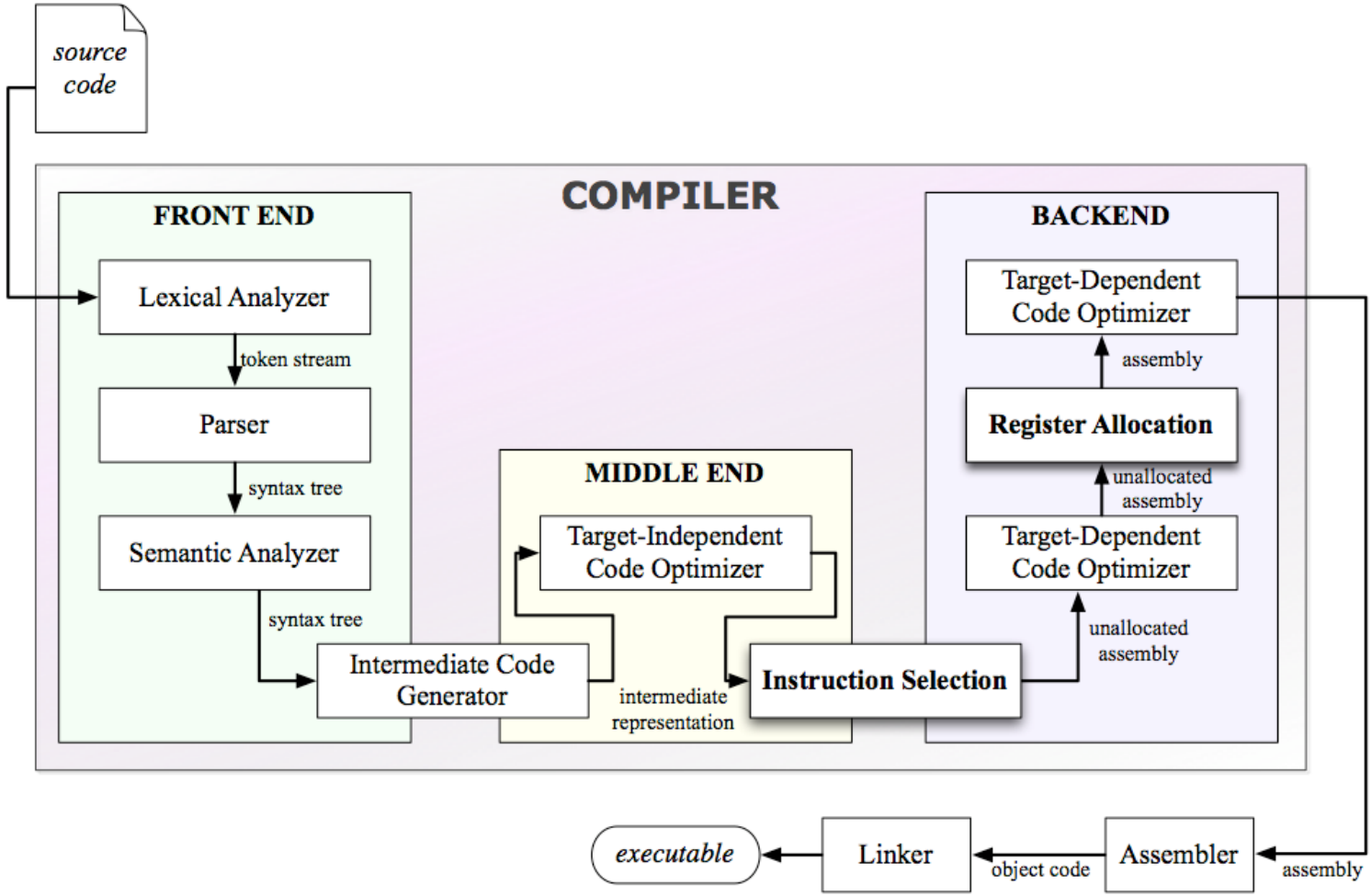

Imagen tomada del curso de F. Pfenning https://www.cs.cmu.edu/~fp/courses/15411-f14/

Referencias

[1] Jean-Christophe Filliâtre.
Curso Compilation (inf564) école Polytechnique, Palaiseau, Francia.
http://www.enseignement.polytechnique.fr/informatique/INF564/, 2018.
Material en francés.

[2] Torben Ægidius Mogensen.
Basics of Compiler Design.
Lulu Press, 2010.

[3] Frank Pfenning.
Notas del curso (15-411) Compiler Design.
https://www.cs.cmu.edu/~fp/courses/15411-f14/, 2014.

[4] Linda Torczon and Keith Cooper.
Engineering A Compiler.
Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 2011.