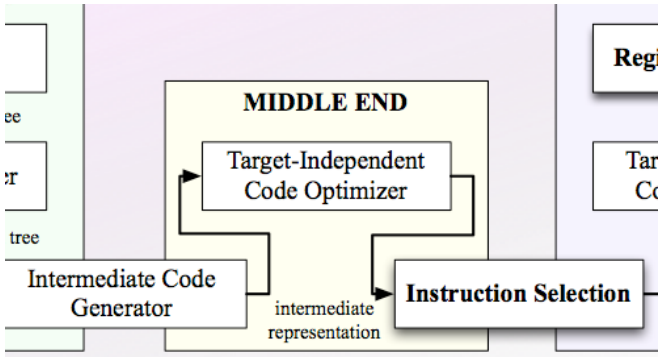
Compiladores 24-2
Back-end

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

Facultad de Ciencias, UNAM

29 abril 2024

Middle-end



- **Forma intermedia**
  representación o representaciones obtenidas del front-end:
  - árbol de sintaxis abstracta
  - árbol de sintaxis concreta
  - gráficas de control de flujo
  - gráficas de dependencias
  - código de 3 direcciones

Un compilador requiere almacenar y representar la información del programa fuente en una o varias representaciones intermedias. Estas representaciones sólo existen en tiempo de compilación.

Transición al back-end

high-level vs low-level

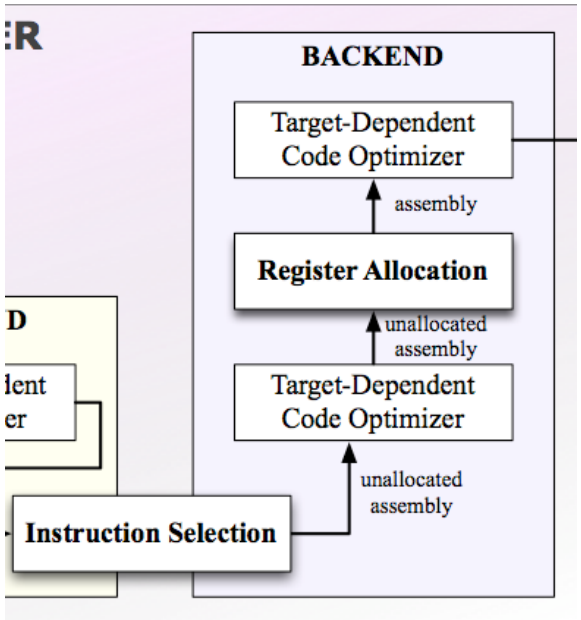
AST

- forma del programa
- fácil identificación de alcances y referencias a variables
- manejo de memoria de alto nivel
- lenguaje de programación de propósito general
- semántica dinámica del lenguaje

IR

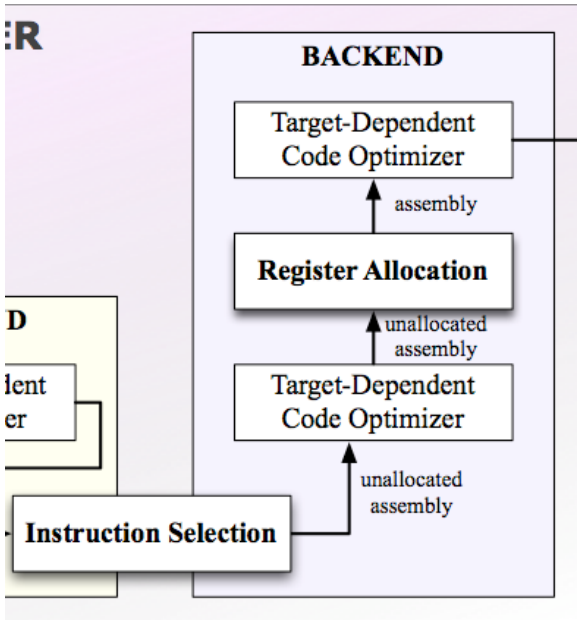
- detalles de cálculos y operaciones
- asignación de nombres a variables internas y resultados temporales (para optimizaciones) en registros
- hacer explícitos los detalles ocultos por el alto nivel
- lenguaje intermedio (Java bytecode, Jimple, Gimple, etc.)
- máquinas abstractas de bajo nivel

Back-end



- **Selección de instrucciones**
  Esta fase se encarga de traducir un código intermedio en un código usando un lenguaje muy cercano a ensamblador o al lenguaje del procesador donde se ejecutará.
- **Planificación**
  Organiza y ordena las operaciones para ser ejecutadas.
- **Asignación de registros**
  Esta fase es la más cercana al código objeto, decide los valores que tendrán los registros y los valores que están en memoria.

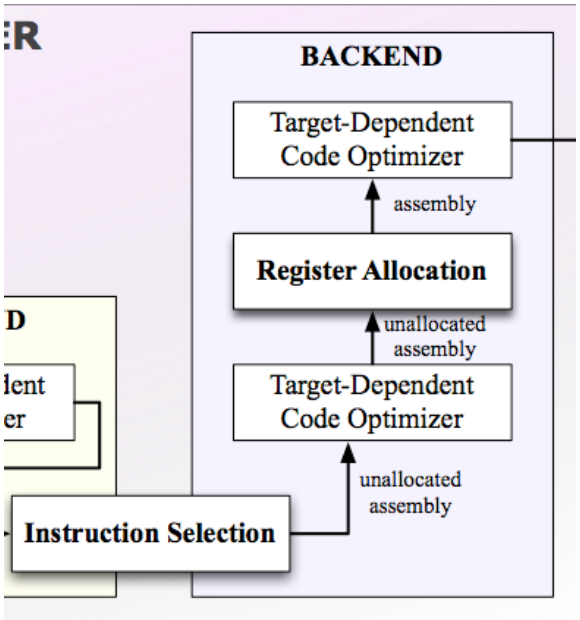
Back-end



- **Selección de instrucciones**
  Esta fase se encarga de traducir un código intermedio en un código usando un lenguaje muy cercano a ensamblador o al lenguaje del procesador donde se ejecutará.
- **Planificación**
  Organiza y ordena las operaciones para ser ejecutadas.
- **Asignación de registros**
  Esta fase es la más cercana al código objeto, decide los valores que tendrán los registros y los valores que están en memoria.

En estas fases se pueden realizar optimizaciones para una ejecución eficiente respetando una traducción fiel del programa fuente.

Back-end



- **Selección de instrucciones**
  Esta fase se encarga de traducir un código intermedio en un código usando un lenguaje muy cercano a ensamblador o al lenguaje del procesador donde se ejecutará.
- **Planificación**
  Organiza y ordena las operaciones para ser ejecutadas.
- **Asignación de registros**
  Esta fase es la más cercana al código objeto, decide los valores que tendrán los registros y los valores que están en memoria.

En estas fases se pueden realizar optimizaciones para una ejecución eficiente respetando una traducción fiel del programa fuente.

Problema: generar código objeto óptimo de un programa fuente.

Back-end o fases de síntesis

- Un compilador debe implementar exactamente las abstracciones incluidas en el código fuente respetando la definición y facilidades del lenguaje de programación.

Back-end o fases de síntesis

- Un compilador debe implementar exactamente las abstracciones incluidas en el código fuente respetando la definición y facilidades del lenguaje de programación.
- En el back-end, además de lo anterior se debe cooperar con el sistema operativo para obtener un código objeto eficiente:
    - crear y manejar un ambiente de ejecución
    - almacenar y manejar las posiciones de memoria
    - acceder a variables
    - ligar procedimientos
    - manejo y paso de parámetros
    - comunicación eon el SO y dispositivos

Back-end o fases de síntesis

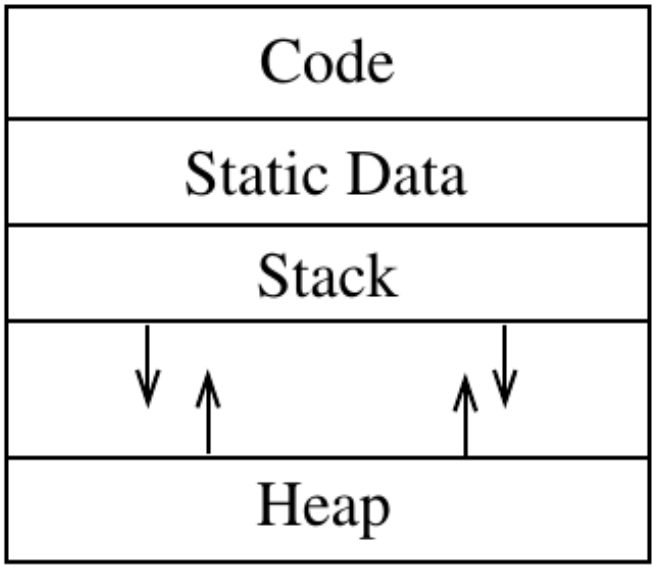
- Un compilador debe implementar exactamente las abstracciones incluidas en el código fuente respetando la definición y facilidades del lenguaje de programación.
- En el back-end, además de lo anterior se debe cooperar con el sistema operativo para obtener un código objeto eficiente:
  - crear y manejar un ambiente de ejecución
  - almacenar y manejar las posiciones de memoria
  - acceder a variables
  - ligar procedimientos
  - manejo y paso de parámetros
  - comunicación eon el SO y dispositivos
- El back-end parte de la representación intermedia y la tabla de símbolos para obtener refinamientos de la representación o representaciones y optimizar el código.
- La generación de código es la última parte del compilador.

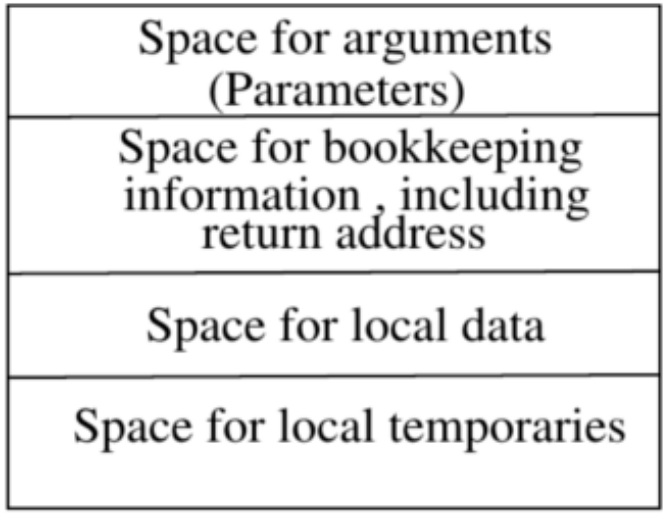
Back-end & Runtime

Ejecución de procesos

- Runtime memory (áreas para código y datos)
- Almacenamiento de datos depende de su tipo.
- Empaquetar información en tiempo de compilación de forma óptima.
- *Padding* empaquetamiento respetando los tamaños estandar de datos:
  eg. un arreglo de caracteres de tamaño 10 ocupará el espacio respectivo pero realmente tendrá 12 bytes para alinear las palabras en bloques.
- Dynamic store allocation (asignación de almacenamiento dinámico)
  - static storage: almacenar nombres locales en un proceso
  - heap storage: datos reusables que 'viven' después de un proceso
  - heap management: garbage collector (información no referenciada)

Back-end & Runtime





Back-end

generación de código

- En esta fase, el compilador debe completar la importante tarea de producir el código objeto, es decir traducir el código fuente en operaciones de máquina.

Back-end

generación de código

- En esta fase, el compilador debe completar la importante tarea de producir el código objeto, es decir traducir el código fuente en operaciones de máquina.
- Se pueden considerar instrucciones particulares del procesador final donde se ejecutará el código o de alguna máquina virtual (eg. LLMV low-level virtual machine).

Back-end

generación de código

- En esta fase, el compilador debe completar la importante tarea de producir el código objeto, es decir traducir el código fuente en operaciones de máquina.
- Se pueden considerar instrucciones particulares del procesador final donde se ejecutará el código o de alguna máquina virtual (eg. LLMV low-level virtual machine).
- A partir de una representación intermedia y la información en la tabla de símbolos se puede obtener un refinamiento de la IR para dar forma al código objeto, incorporar optimizaciones para mejorar el consumo de espacio, uso de instrucciones, etc.

Back-end

generación de código

- En esta fase, el compilador debe completar la importante tarea de producir el código objeto, es decir traducir el código fuente en operaciones de máquina.
- Se pueden considerar instrucciones particulares del procesador final donde se ejecutará el código o de alguna máquina virtual (eg. LLMV low-level virtual machine).
- A partir de una representación intermedia y la información en la tabla de símbolos se puede obtener un refinamiento de la IR para dar forma al código objeto, incorporar optimizaciones para mejorar el consumo de espacio, uso de instrucciones, etc.

  Lectura complementaria: representación intermedia (código tres direcciones) de construcciones usuales y asignación de localidades de memoria.
  Capítulo 6 del libro [3].

Back-end

Selección de instrucciones

- ★ Reemplazar operaciones aritméticas del lenguaje de la IR por las del procesador $a + b$ `add a b` considerando los tipos de los argumentos.
- ★ Reemplazar acceso a campos de estructuras por operaciones explícitas en la memoria.

Back-end

Selección de instrucciones

- ★ Reemplazar operaciones aritméticas del lenguaje de la IR por las del procesador $a + b$ `add a b` considerando los tipos de los argumentos.
- ★ Reemplazar acceso a campos de estructuras por operaciones explícitas en la memoria.
  - Se debe asegurar la uniformidad y un conjunto de instrucciones completo:
    para cada instrucción de la IR escoger una implementación adecuada y eficiente.

Back-end

Selección de instrucciones

- ★ Reemplazar operaciones aritméticas del lenguaje de la IR por las del procesador $a + b$ `add a b` considerando los tipos de los argumentos.
- ★ Reemplazar acceso a campos de estructuras por operaciones explícitas en la memoria.
- Se debe asegurar la uniformidad y un conjunto de instrucciones completo:
  para cada instrucción de la IR escoger una implementación adecuada y eficiente.

  Recordatorio:

  - una dirección en memoria es una expresión (dirección) y un corrimiento (desplazamiento)
  - acciones en memoria: `load` y `store` de palabras

Back-end

Selección de instrucciones, optimizaciones

Eficiencia y preservación del código

- ✓ escoger la mejor implementación para cada instrucción si el lenguaje objetivo tiene diferentes
- ✓ evaluar algunas expresiones y simplificar algunas expresiones aritméticas
- ✓ realizar otras optimizaciones de código intermedio:
  - evaluar y propagar valores de constantes
  - eliminar código muerto o código que nunca se alcanzará
  - remover expresiones constantes en ciclos
  - eliminar subexpresiones comunes

Back-end

Selección de instrucciones, ejemplos

```
x := y + z
```

```
lw $19, y
lw $20, z
add $21, $19, $20
sw $21, x
```

```
a := b + c;
```

```
d := a + e;
```

Back-end

subfases para la producción de código

Para generar código eficiente:

- Selección de Instrucciones
- Register Transfer Language

  convenciones de llamadas usando pseudo-registros
- Explicit Register Trasfer Language

  hacer explícitas las llamadas en los registros
- Location Transfer Language

  asignación de registros físicos del procesador mediante el análisis de vida de las variables
- Código Linearizado

Referencias

[1] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman.
Compilers, Principles, Techniques and Tools.
Pearson Education Inc., Second edition, 2007.

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

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

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

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