Uma Infra-estrutura Modular e Extensível para Otimização de Binários

Arquitetura MIPS como Estudo de Caso

Ricardo Nabinger Sanchez
rnsanchez@wait4.org

César Augusto Missio Marcon
Orientador

Universidade do Vale do Rio dos Sinos
Centro de Ciências Exatas e Tecnológicas

14 de dezembro de 2006

Roteiro

Introdução I

Compiladores freqüentemente precisam:

- Converter uma expressão em alto nível para uma representação suportada pela arquitetura alvo
- Aplicar técnicas de otimização
- Evitar que elas interfiram negativamente entre si
- Tirar proveito dos recursos disponíves em um *hardware* específico

Introdução II

Diversos trabalhos publicados:

Interação entre técnicas de otimização Pan, Z., Eigenmann, R. *Fast and Effective Orchestration of Compiler Optimizations for Automatic Performance Tuning.* IEEE CGO'06.

Desenvolvimento de técnicas de otimização Tjiang, S., Hennessy, J. *Sharlit—A Tool for Building Optimizers.* ACM SIGPLAN'92.

Introdução III

Otimizações dinâmicas Duesterwald, E. *Design and Engineering of a Dynamic Binary Optimizer.* Proc. of the IEEE, 2005.

Otimizações estáticas Cifuentes, C., et. al. *Experience in the Design, Implementation and Use of a Retargetable Static Binary Translation Framework.* TR, Sun Microsystems, 2002.

Objetivo Geral

Implementar uma infra-estrutura de software que permita avaliar técnicas de otimização, modificando diretamente binários executáveis.

Motivação I

Por que alterar diretamente um binário executável?

Modificar diretamente binários apresenta vantagens como:

- Operar independentemente da linguagem de programação de alto nível usada na compilação do binário
- Possibilitar a (re-)otimização de binários
- Tradução binária entre arquiteturas (*retargeting*)
- Quantidade de dados significativamente menor

Objetivos Específi cos

1. Infra-estrutura de *software* modular e extensível
2. Implementação de módulos para a validação das técnicas adotadas
3. Protótipo para programas ELF/UNIX compilados para MIPS

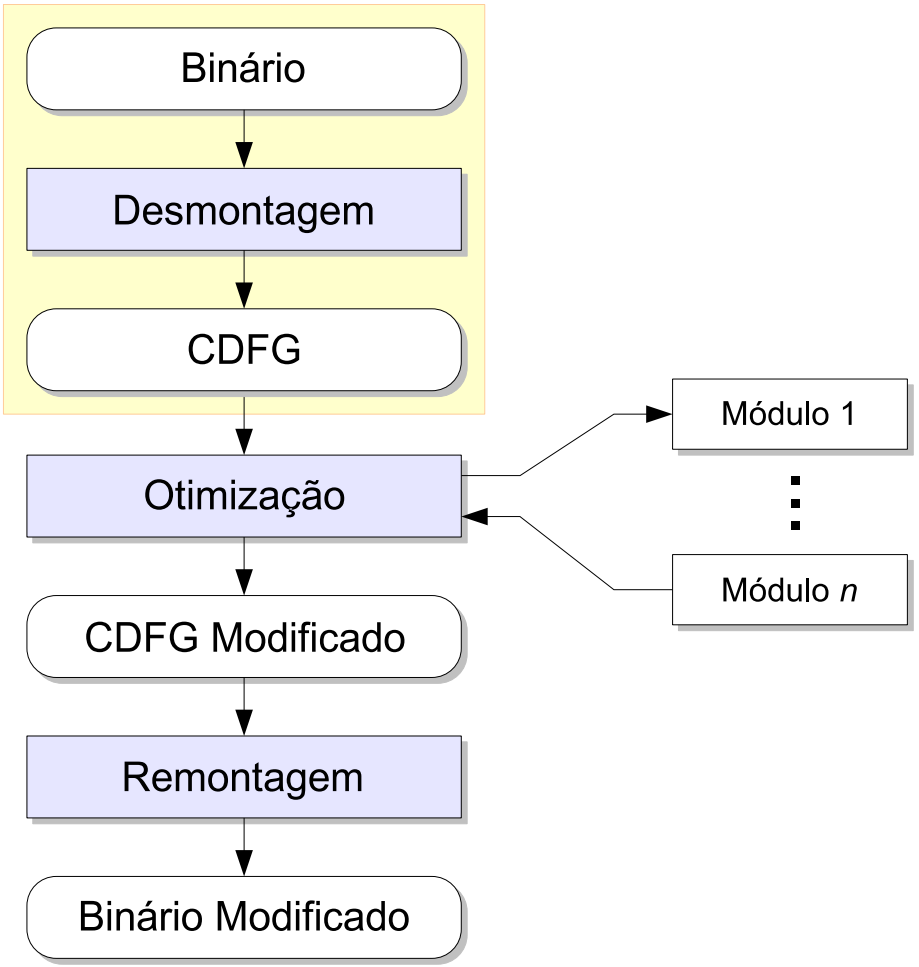
Motivação II

Dentre tantas arquiteturas, por que MIPS?

- Arquitetura RISC[1]
- Semelhanças com outras plataformas RISC
- Existem diversos simuladores e emuladores em *software*
- Possibilidade futura de análise experimental usando *hardware* de prototipação (FPGA[2])

[1] *Reduced Instruction Set Computer*
[2] *Field-programmable Gate Array*

Arquitetura Proposta

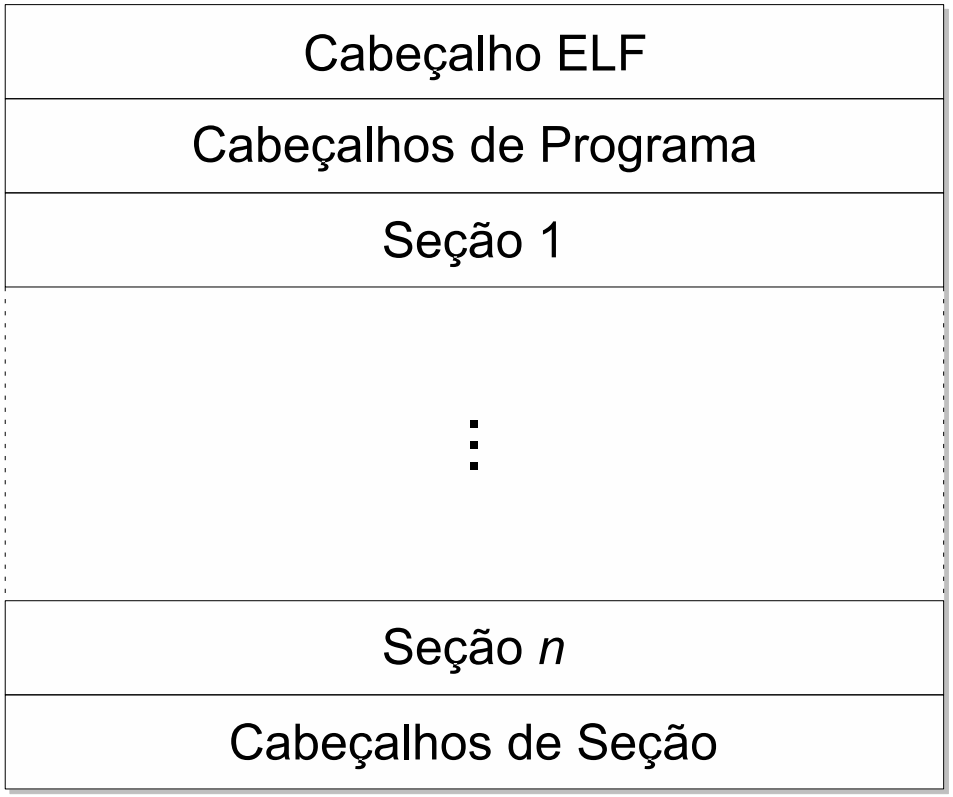


Formato dos Arquivos Binários ELF

Um arquivo binário ELF[3] é composto por:

- Um cabeçalho principal no início do arquivo
- Vetor contendo cabeçalhos de programa
- Vetor contendo cabeçalhos de seção
- Seções

[3] *Executable and Linking Format*


Cabeçalho ELF
Cabeçalhos de Programa
Seção 1
Seção *n*
Cabeçalhos de Seção

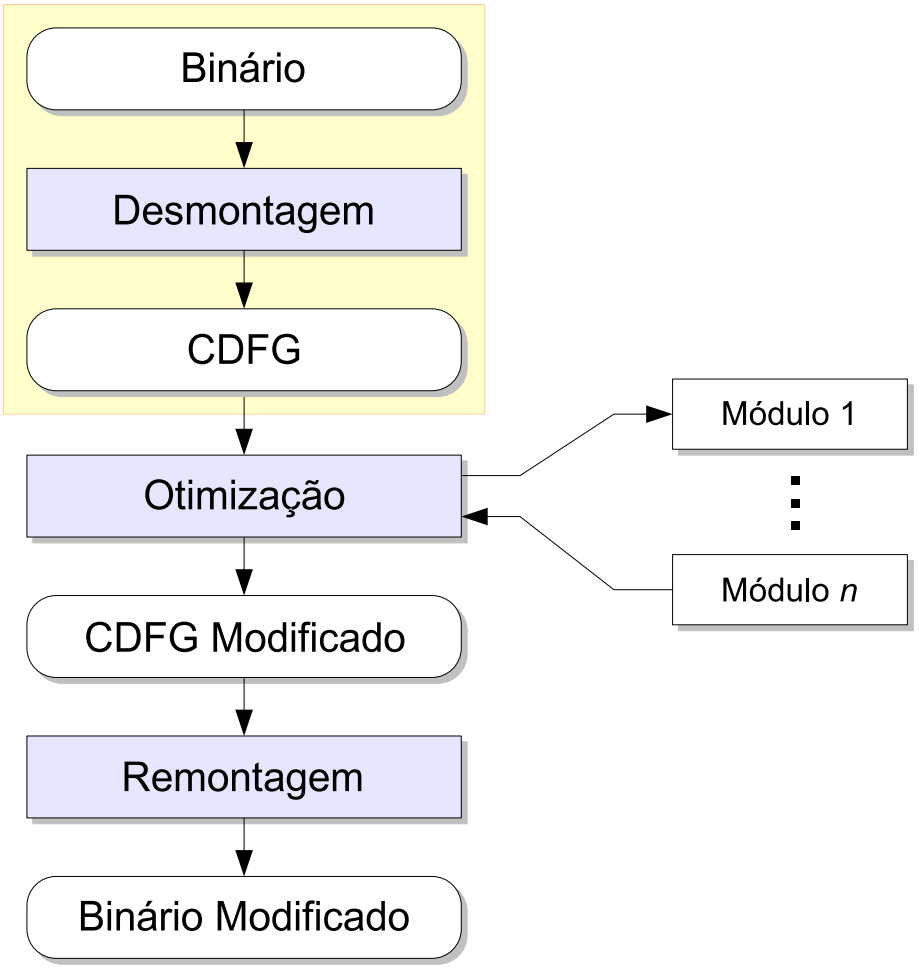
Cabeçalhos de Seção

- Contêm atributos e localização da respectiva seção no arquivo
- Atributo `sh_flags` indica se a seção contém instruções de máquina
- A infra-estrutura analisa todos os cabeçalhos de seção presentes no arquivo em busca de seções executáveis

Seção de um Arquivo ELF

- Seqüência contígua de bytes no arquivo
- Atributos e características especificadas pelo respectivo cabeçalho de seção
- Um arquivo ELF contém diversas seções em geral

Arquitetura Proposta



Desmontagem

Técnicas para decodificar instruções de máquina, podendo ser:

1. Linear
2. Recursiva
3. Especulativa
4. Híbrida

Desmontagem Linear I

- Assume que a seção contém apenas instruções de máquina

```
0x80020000:   addiu   $sp, $sp, -32
0x80020004:   sw      $s1, 20($sp)
0x80020008:   lui     $s1, 0x8002
0x8002000c:   lbu     $v0, 440($s1)
0x80020010:   sw      $ra, 24($sp)
0x80020014:   bnez    $v0, 0x80020060
0x80020018:   sw      $s0, 16($sp)
0x8002001c:   b       0x80020030
0x80020020:   lui     $s0, 0x8002
0x80020024:   addiu   $a1, $a2, 4
0x80020028:   jalr    $v1
0x8002002c:   sw      $a1, 412($s0)
0x80020030:   lw      $a0, 412($s0)
0x80020034:   lw      $v1, 0($a0)
0x80020038:   bnez    $v1, 0x80020024
0x8002003c:   lw      $a2, 412($s0)
0x80020040:   lui     $a3, 0
...
```

Desmontagem Linear II

- O algoritmo é muito simples e rápido

```
0x80020000:   addiu   $sp, $sp, -32
0x80020004:   sw      $s1, 20($sp)
0x80020008:   lui     $s1, 0x8002
0x8002000c:   lbu     $v0, 440($s1)
0x80020010:   sw      $ra, 24($sp)
0x80020014:   bnez    $v0, 0x80020060
0x80020018:   sw      $s0, 16($sp)
0x8002001c:   b       0x80020030
0x80020020:   lui     $s0, 0x8002
0x80020024:   addiu   $a1, $a2, 4
0x80020028:   jalr    $v1
0x8002002c:   sw      $a1, 412($s0)
0x80020030:   lw      $a0, 412($s0)
0x80020034:   lw      $v1, 0($a0)
0x80020038:   bnez    $v1, 0x80020024
0x8002003c:   lw      $a2, 412($s0)
0x80020040:   lui     $a3, 0
...
```

Desmontagem Linear III

- A decodificação *pode* ser incorreta em alguns casos
  - Dados embutidos em uma seção executável
  - Bytes de enchimento (*padding*) ou alinhamento

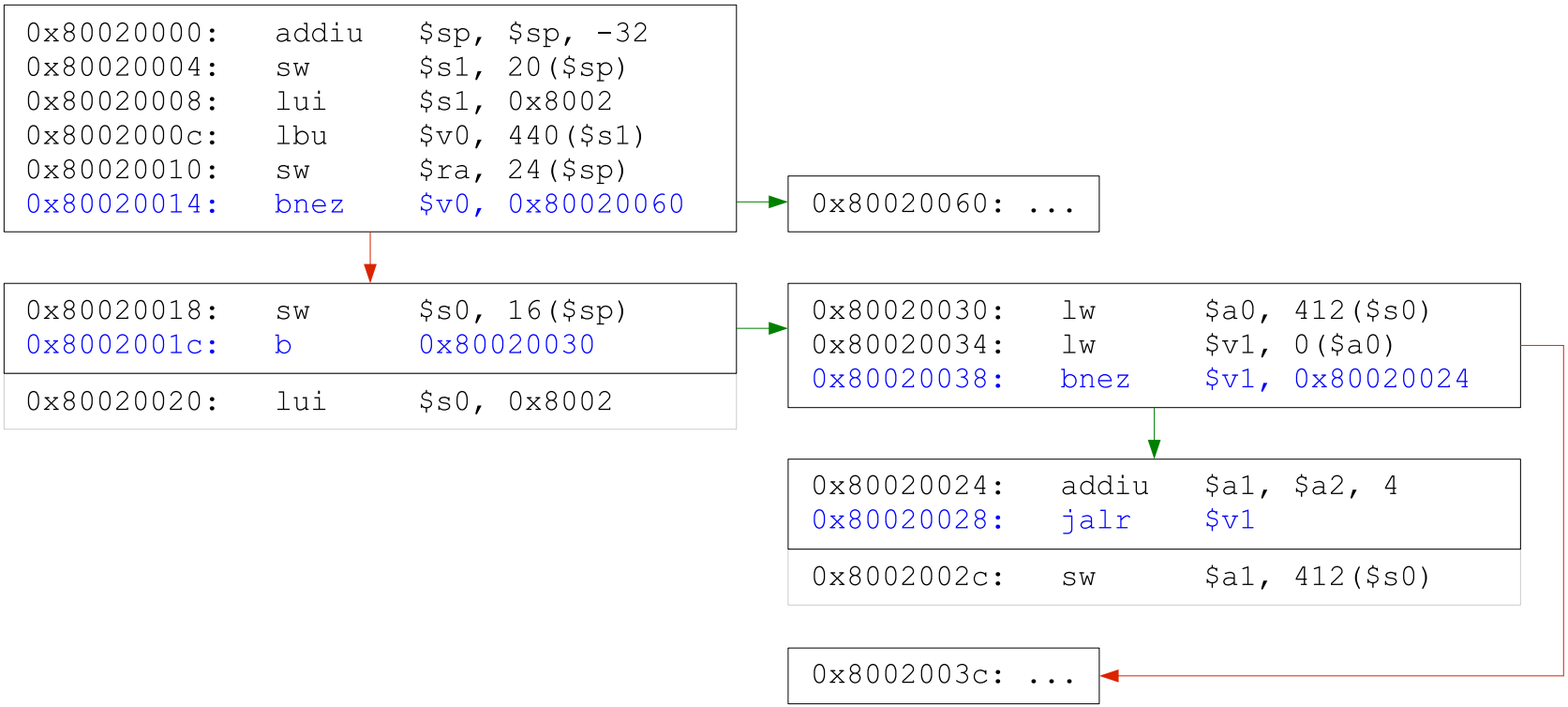
Desmontagem Recursiva I

- Inicia no *ponto de entrada* do programa (indicado no cabeçalho principal do arquivo ELF)

```
0x80020000:   addiu   $sp, $sp, -32
0x80020004:   sw      $s1, 20($sp)
0x80020008:   lui     $s1, 0x8002
0x8002000c:   lbu     $v0, 440($s1)
0x80020010:   sw      $ra, 24($sp)
0x80020014:   bnez    $v0, 0x80020060
0x80020018:   sw      $s0, 16($sp)
0x8002001c:   b       0x80020030
0x80020020:   lui     $s0, 0x8002
0x80020024:   addiu   $a1, $a2, 4
0x80020028:   jalr    $v1
0x8002002c:   sw      $a1, 412($s0)
0x80020030:   lw      $a0, 412($s0)
0x80020034:   lw      $v1, 0($a0)
0x80020038:   bnez    $v1, 0x80020024
0x8002003c:   lw      $a2, 412($s0)
0x80020040:   lui     $a3, 0
...
```

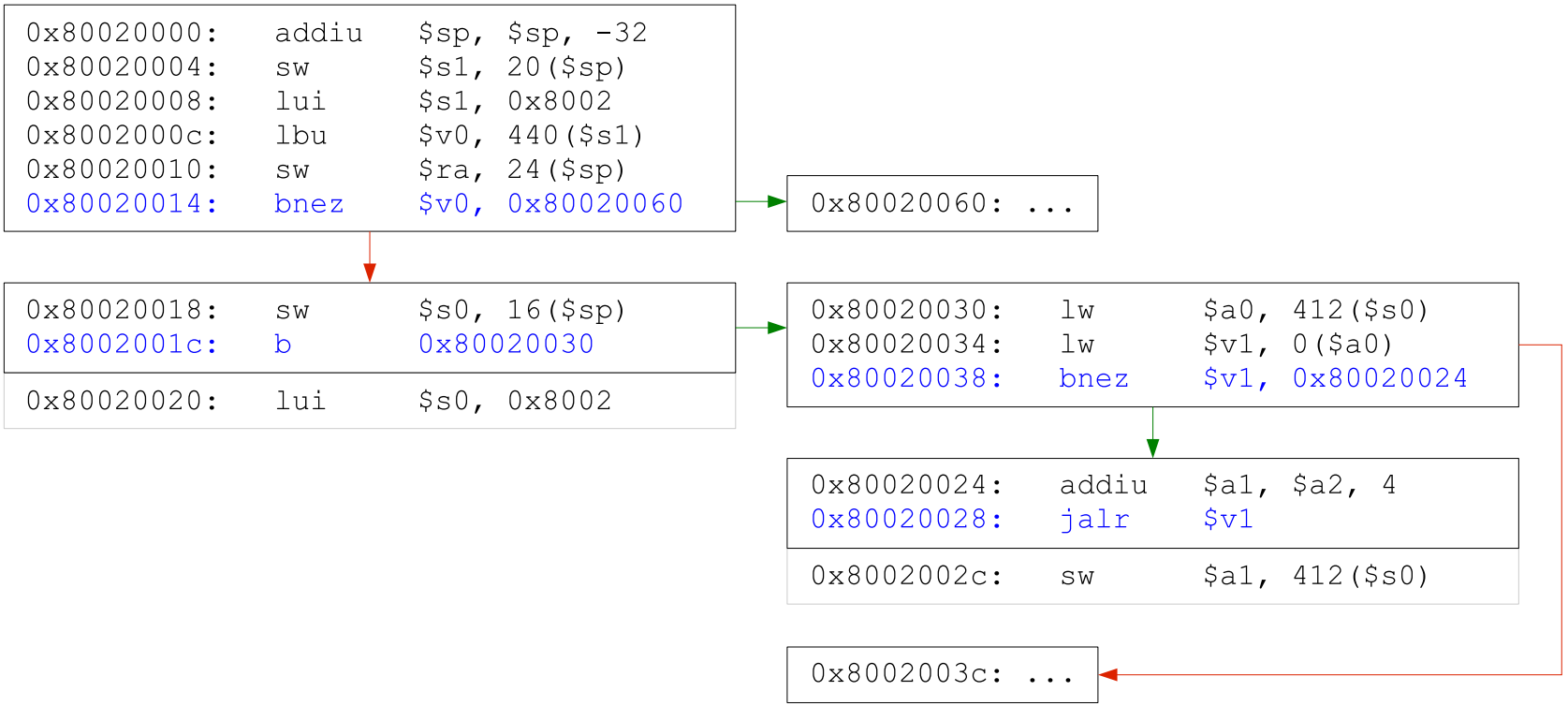
Desmontagem Recursiva II

- Segue o fluxo de controle das instruções



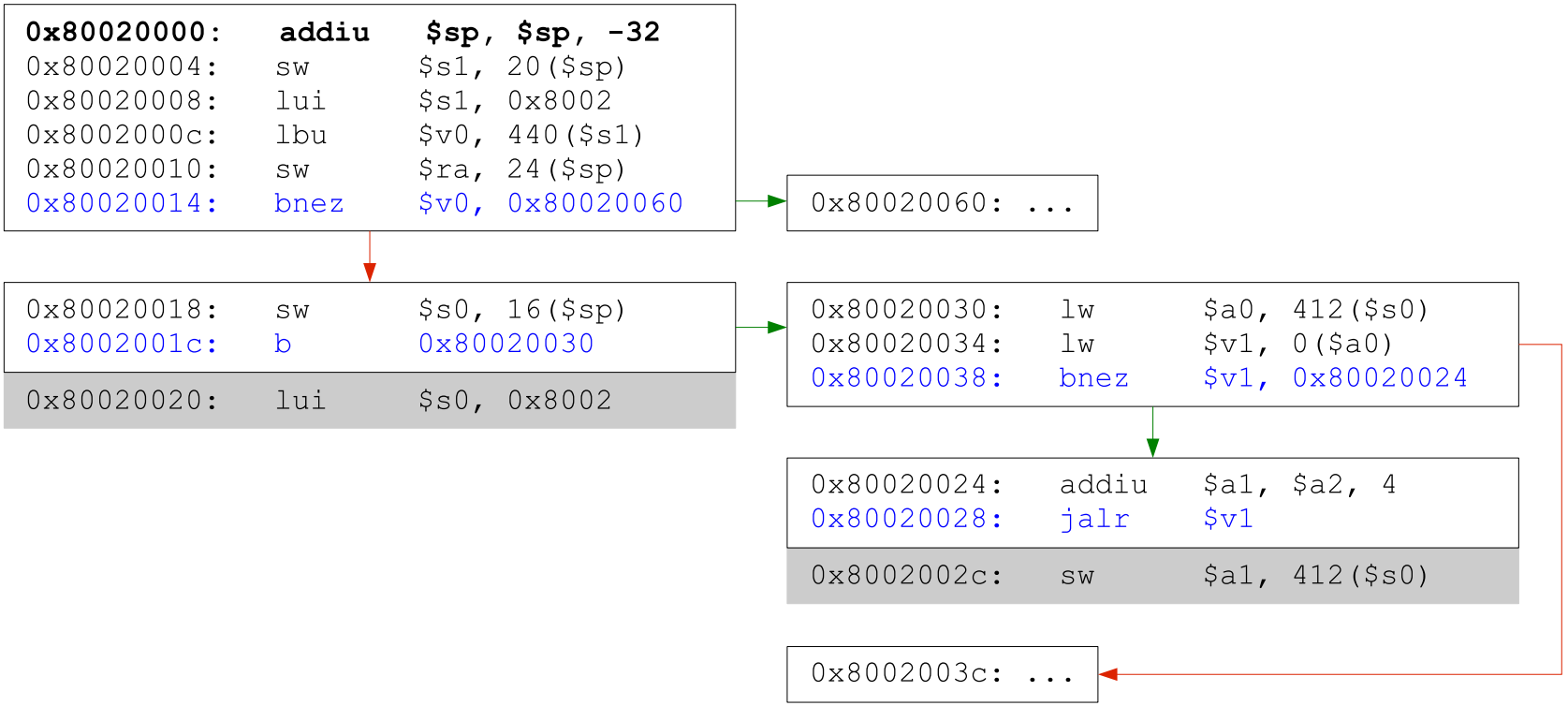
Desmontagem Recursiva III

- A desmontagem não fica limitada a uma determinada seção



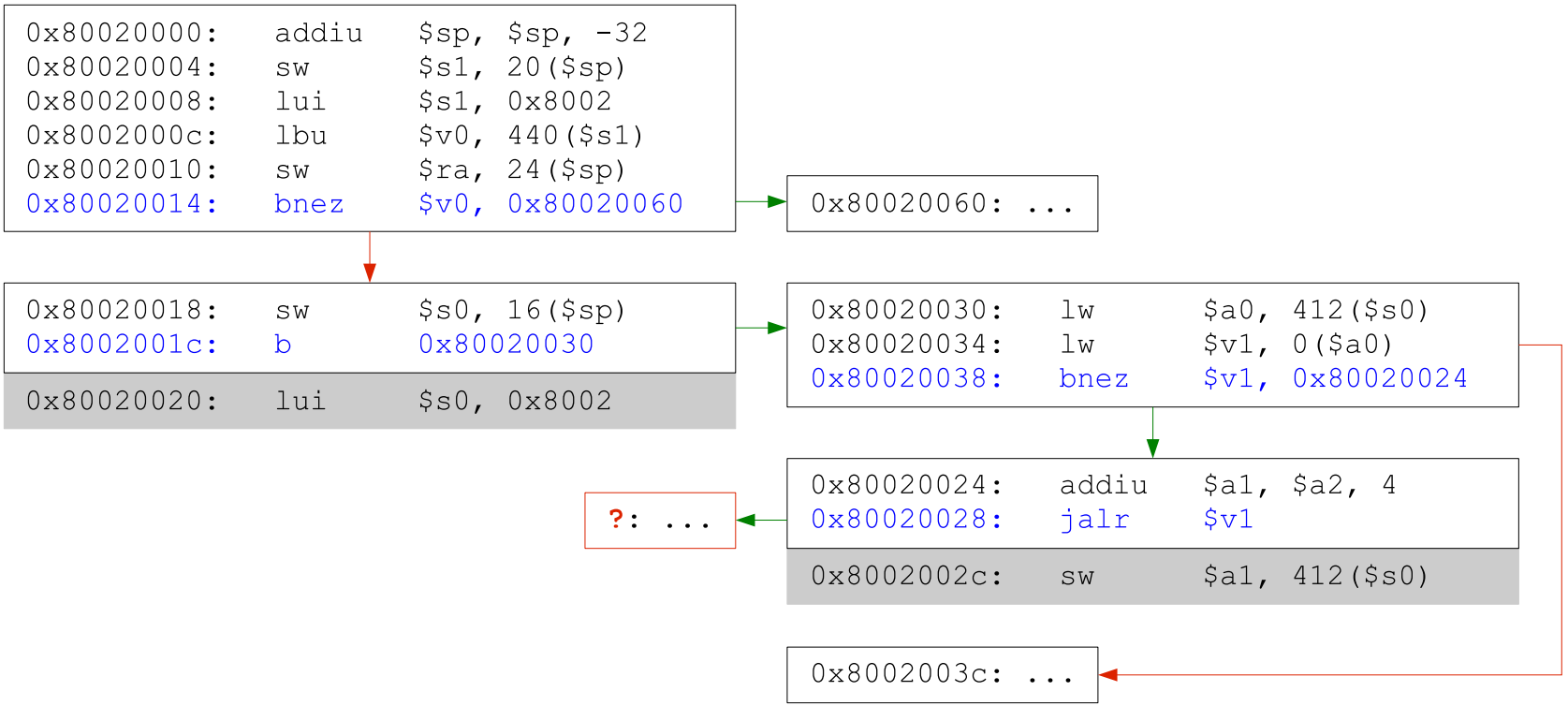
Desmontagem Recursiva IV

- Nem todo arquivo é desmontado, causando o aparecimento de *sombras*



Desmontagem Recursiva V

- Nem todos os fluxos podem ser seguidos



Desmontagem Especulativa

- Geralmente complementa a desmontagem recursiva
- Assume que uma determinada região ainda não desmontada contém instruções
- Baseada em heurísticas, considerando detalhes sobre a linguagem de alto nível ou compilador usados
- Pára em caso de incoerências, como por exemplo sobreposição de instruções

Desmontagem Híbrida

- Geralmente denota a combinação da desmontagem recursiva seguida da linear para complementação mútua
- Não requer heurísticas, ao contrário da especulativa

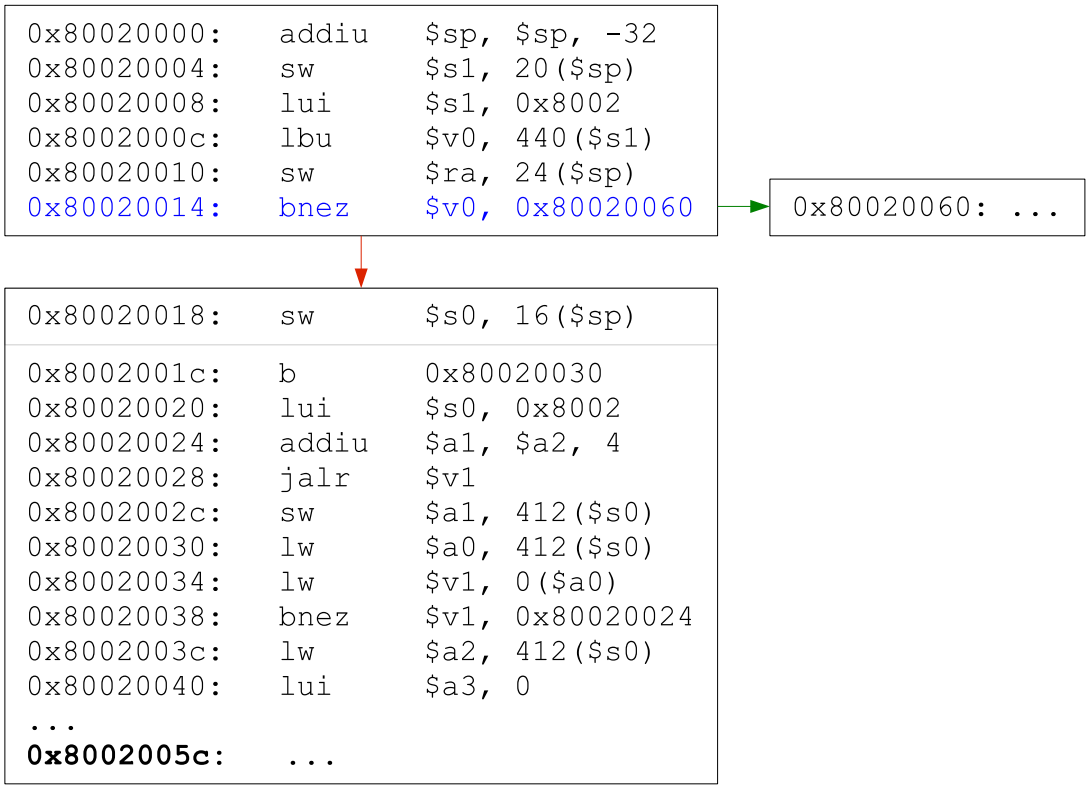
Desmontagem Recursiva Estendida I

- Extensão à desmontagem recursiva “tradicional”
- Inicialmente assume-se um bloco-básico que compreende todo o programa

```
0x80020000:   addiu   $sp, $sp, -32
0x80020004:   sw      $s1, 20($sp)
0x80020008:   lui     $s1, 0x8002
0x8002000c:   lbu     $v0, 440($s1)
0x80020010:   sw      $ra, 24($sp)
0x80020014:   bnez    $v0, 0x80020060
0x80020018:   sw      $s0, 16($sp)
0x8002001c:   b       0x80020030
0x80020020:   lui     $s0, 0x8002
0x80020024:   addiu   $a1, $a2, 4
0x80020028:   jalr    $v1
0x8002002c:   sw      $a1, 412($s0)
0x80020030:   lw      $a0, 412($s0)
0x80020034:   lw      $v1, 0($a0)
0x80020038:   bnez    $v1, 0x80020024
0x8002003c:   lw      $a2, 412($s0)
0x80020040:   lui     $a3, 0
...
```

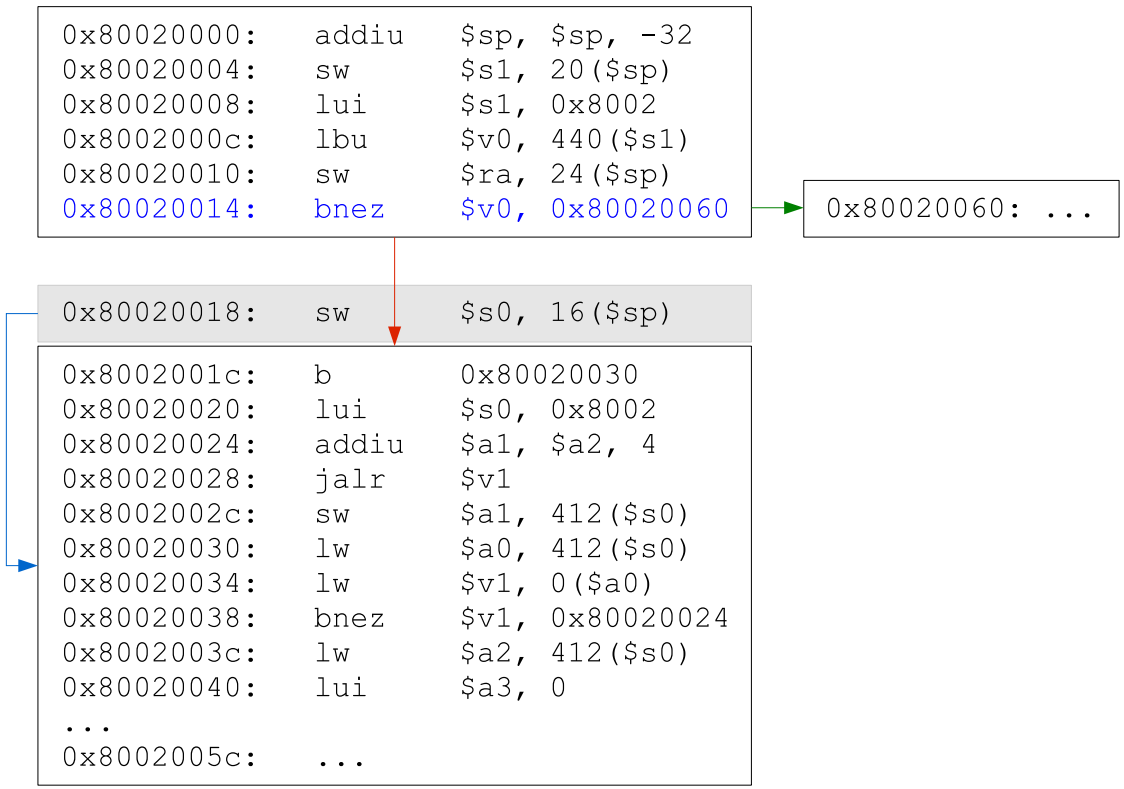
Desmontagem Recursiva Estendida II

- Divide-se sucessivamente o bloco conforme os delimitadores vão sendo encontrados



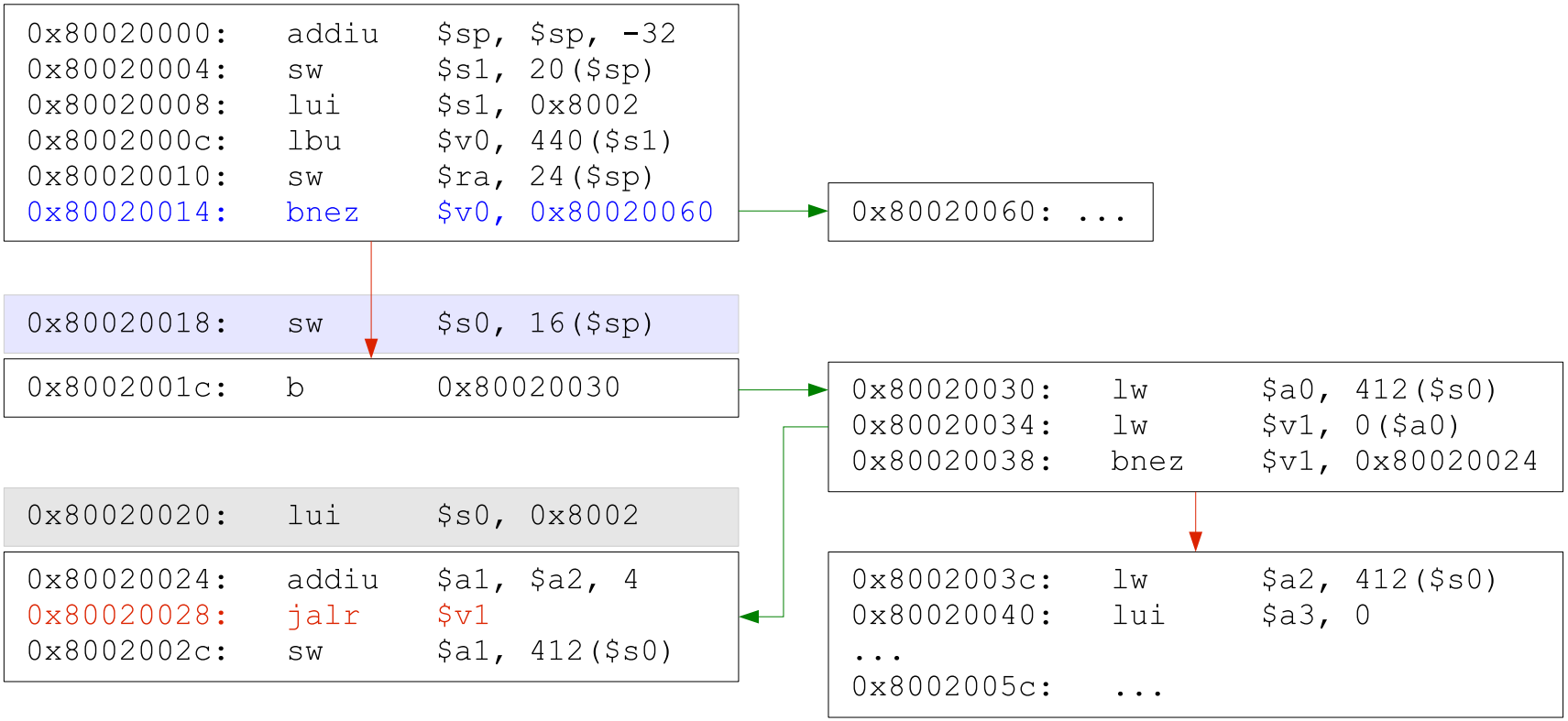
Desmontagem Recursiva Estendida III

- O objetivo desta extensão é lidar com sombras

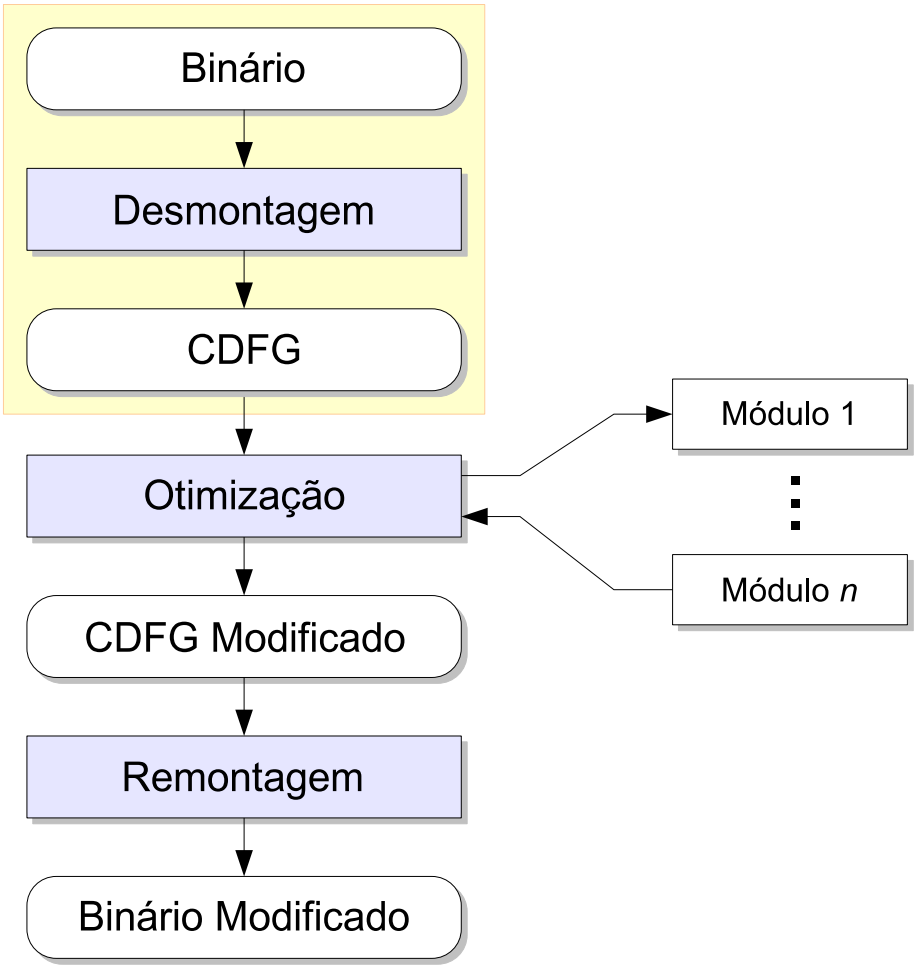


Desmontagem Recursiva Estendida IV

- Ainda assim, nem todos os fluxos podem ser seguidos



Arquitetura Proposta



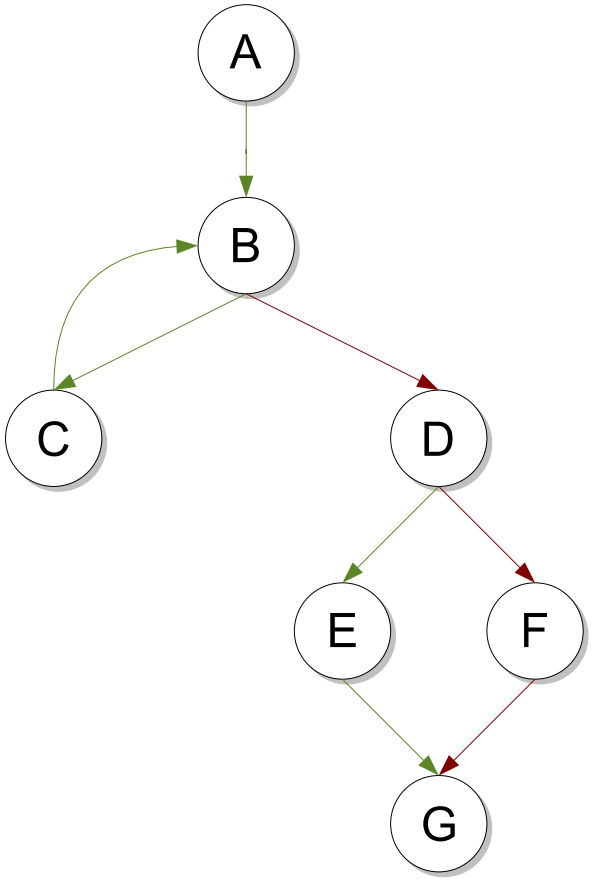
Representação Intermediária

Conforme a infra-estrutura decodifica as instruções do arquivo executável, ela popula diversas estruturas:

- Grafo de controle de fluxo (CFG)
- Blocos-básicos, cada um contendo uma lista de suas respectivas instruções
- Grafo de dependência de dados (DFG)
- Listas auxiliares

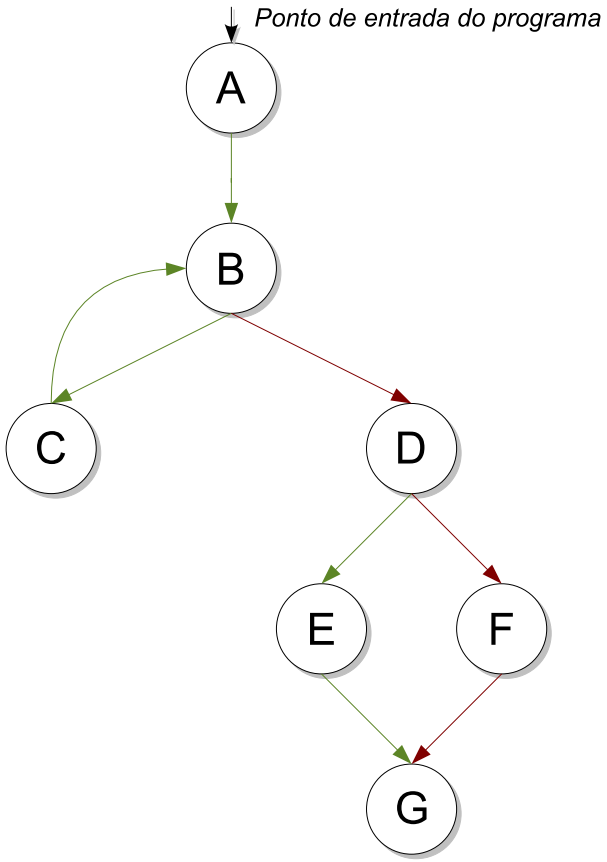
Grafo de Fluxo de Controle I — CFG

- Modela a dependência de controle entre os blocos-básicos



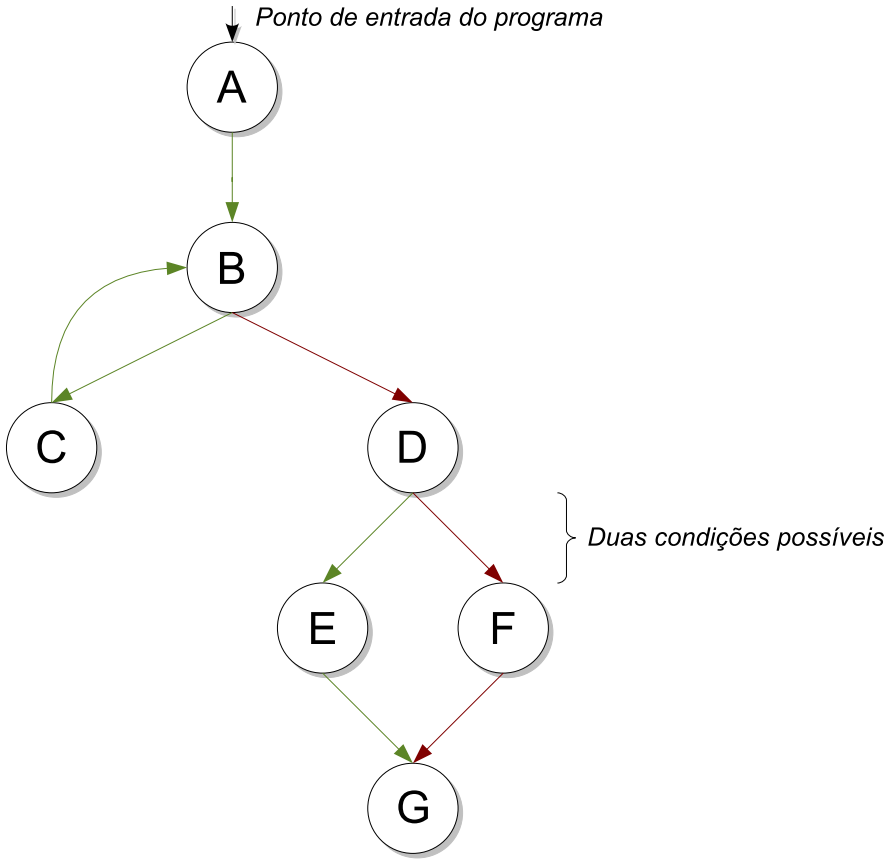
Grafo de Fluxo de Controle II — CFG

- Existe um único nodo raiz, que representa o ponto de entrada do programa



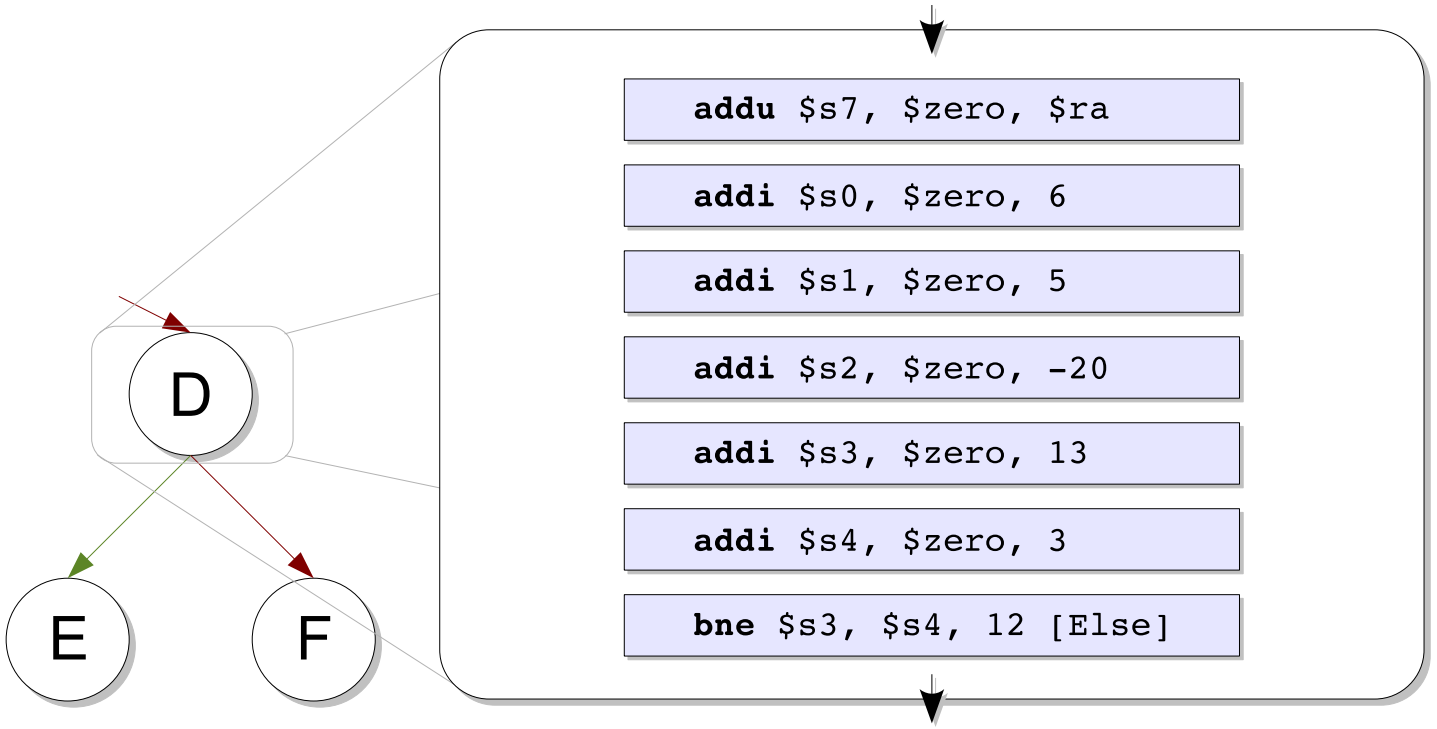
Grafo de Fluxo de Controle III — CFG

- Cada nodo possui até 2 ramos, *tomado* e *não tomado*



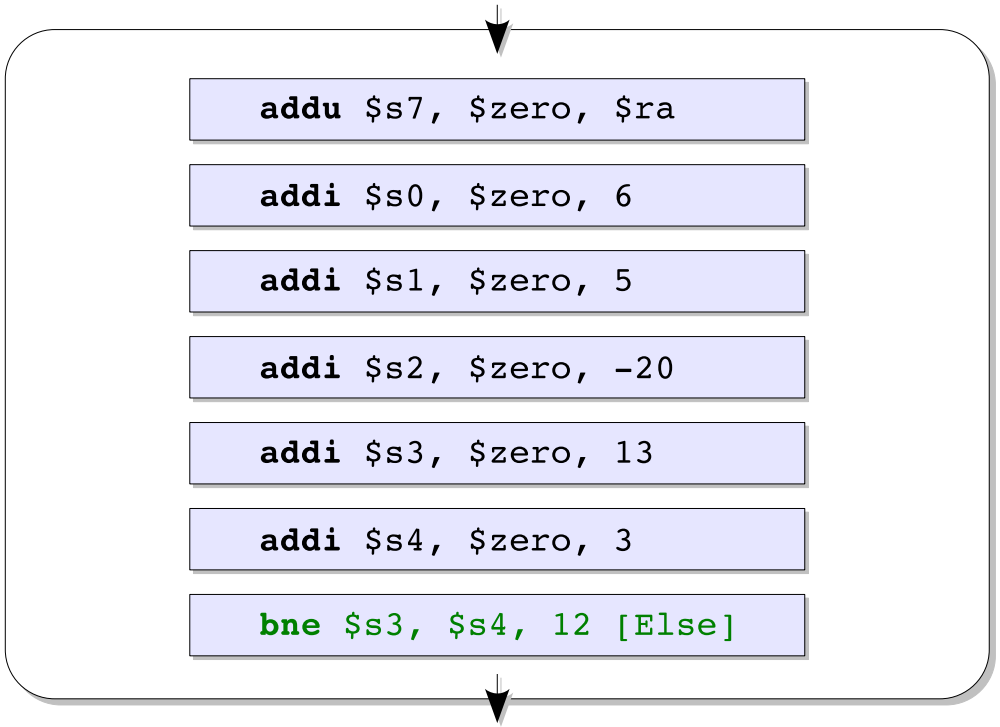
Bloco-básico I

- É um conjunto de instruções com um único ponto de entrada e outro de saída



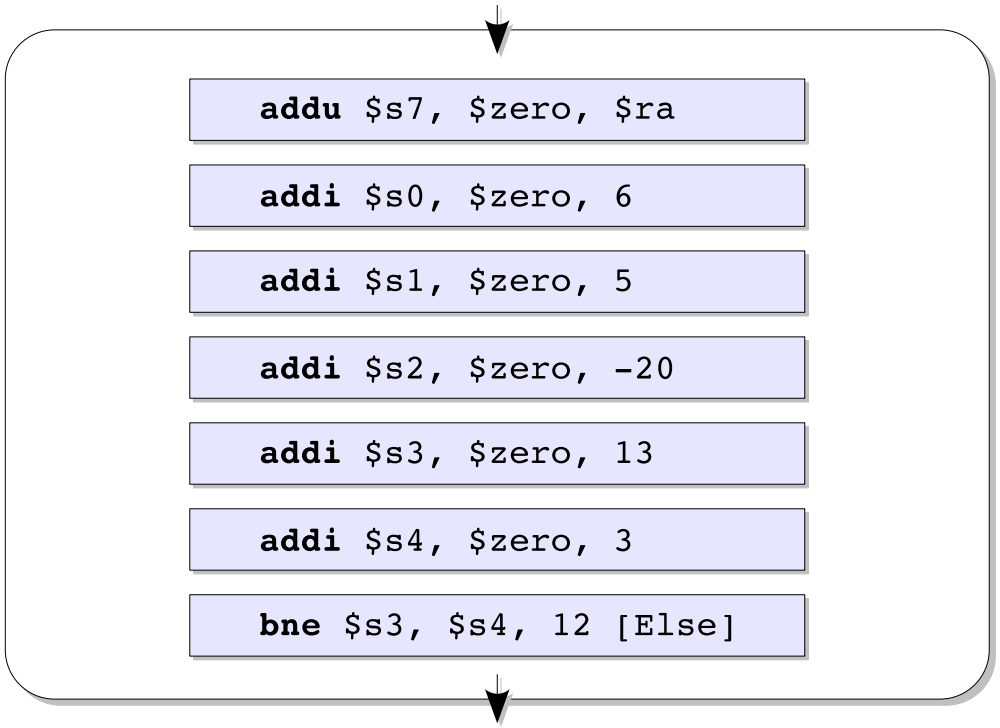
Bloco-básico II

- Geralmente delimitado por instrução de desvio



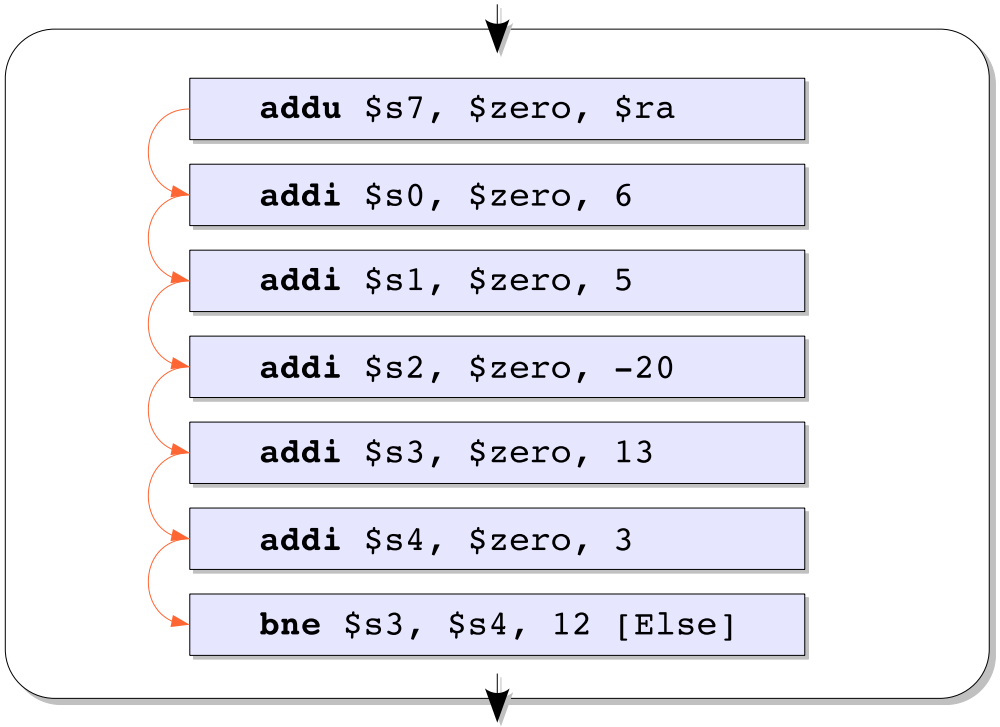
Bloco-básico III

- Estima-se que um bloco-básico contenha até 16 instruções, em média



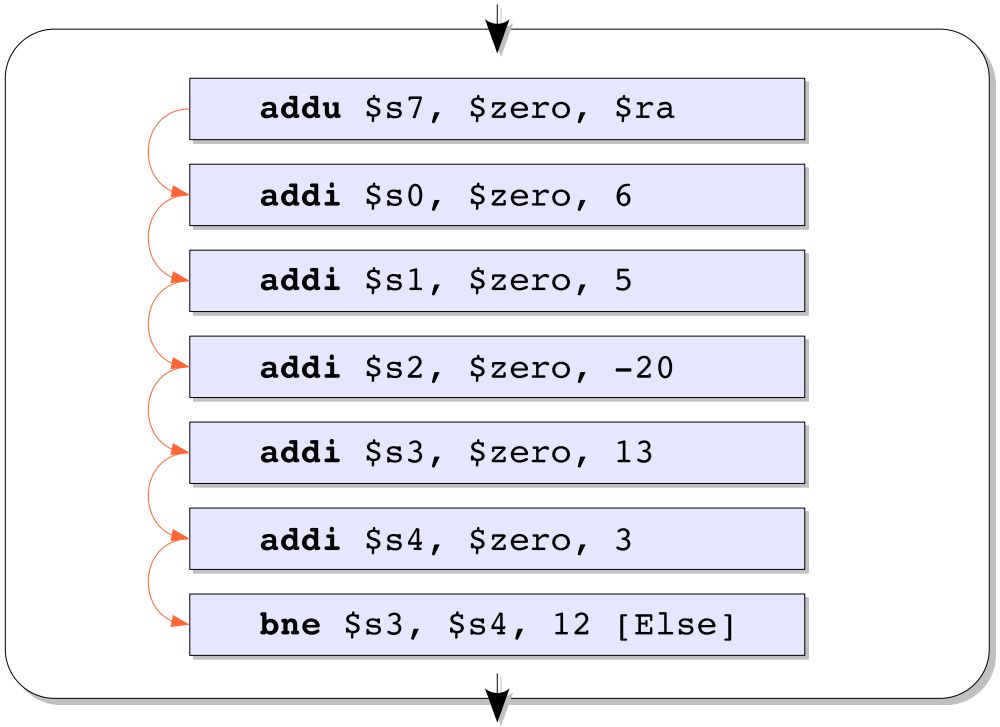
Lista de Instruções I

- Modela a dependência de controle dentro do bloco-básico



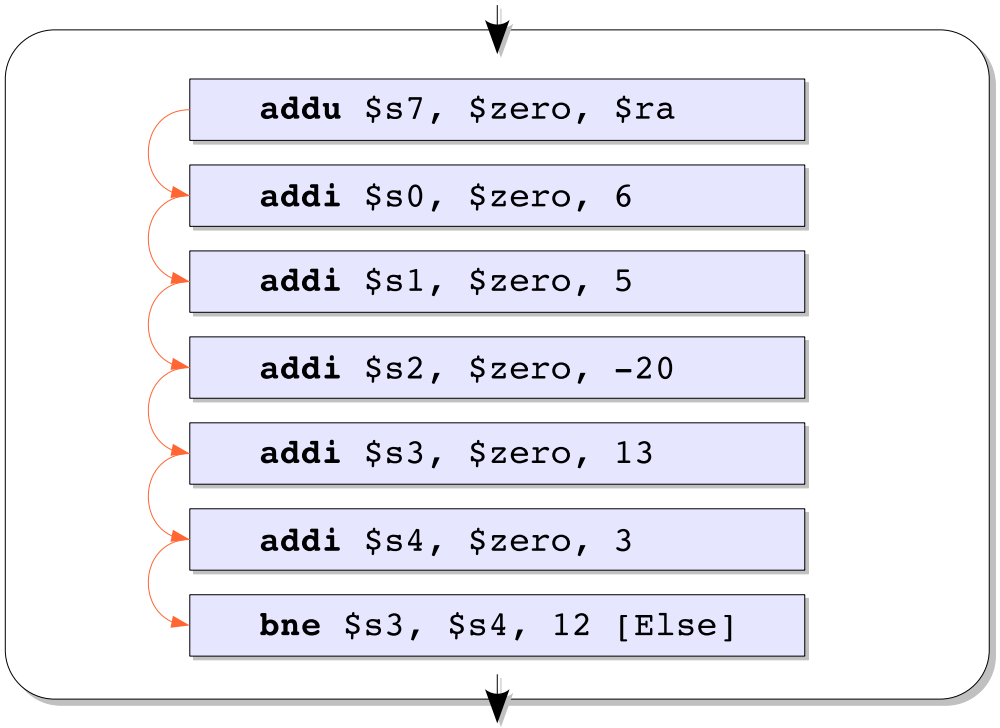
Lista de Instruções II

- Implementada como uma lista simplesmente encadeada



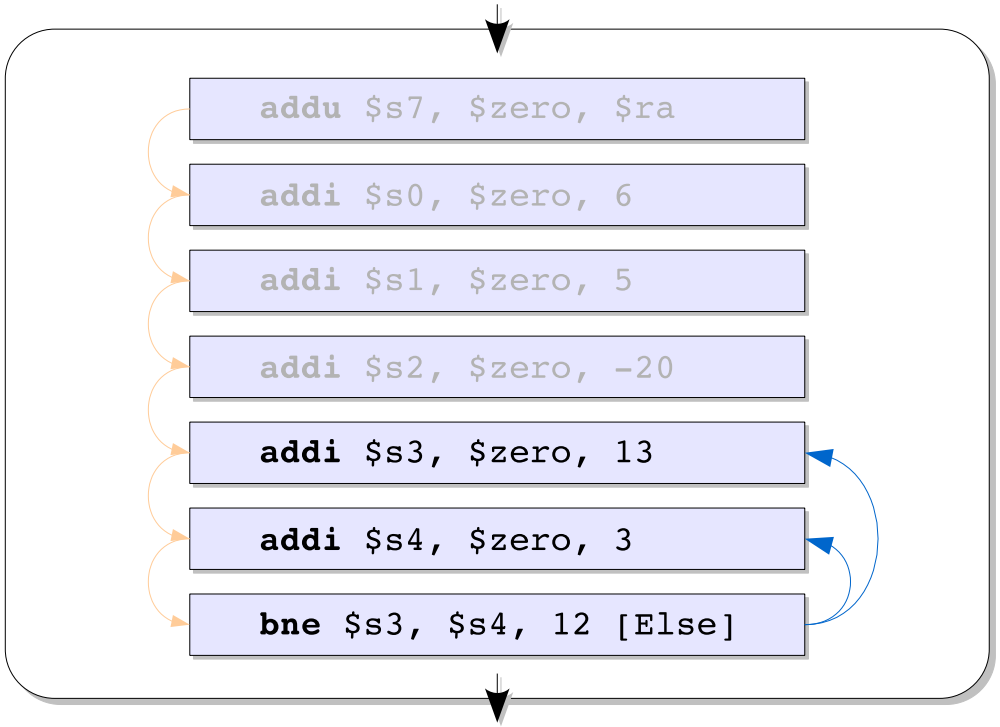
Lista de Instruções III

- Permite que as instruções sejam facilmente manipuladas, inclusive inserções e remoções



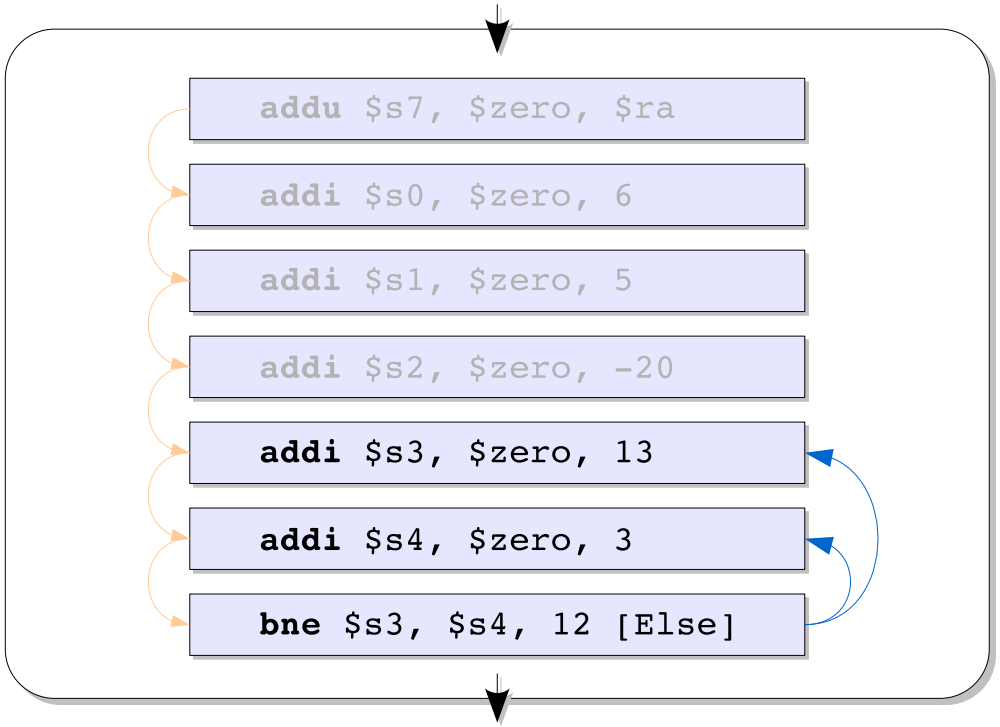
Grafo de Fluxo de Dados I — DFG

- Computado apenas dentro do bloco-básico



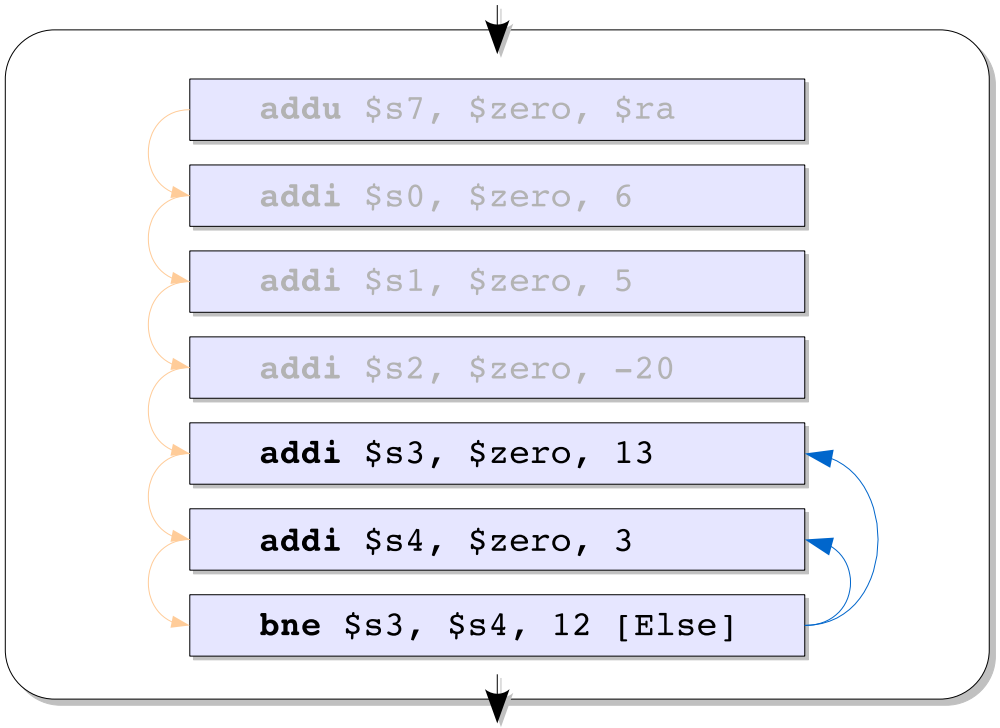
Grafo de Fluxo de Dados II — DFG

- Indica a dependência de seus operandos



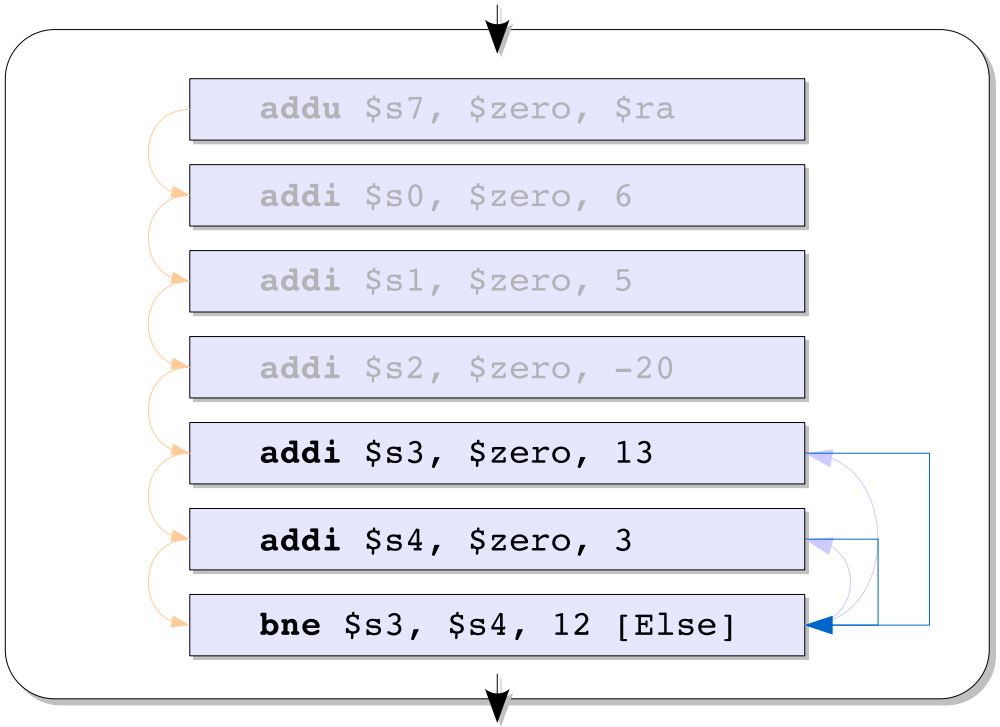
Grafo de Fluxo de Dados III — DFG

- Permite modificar o fluxo de controle sem alterar a semântica do programa



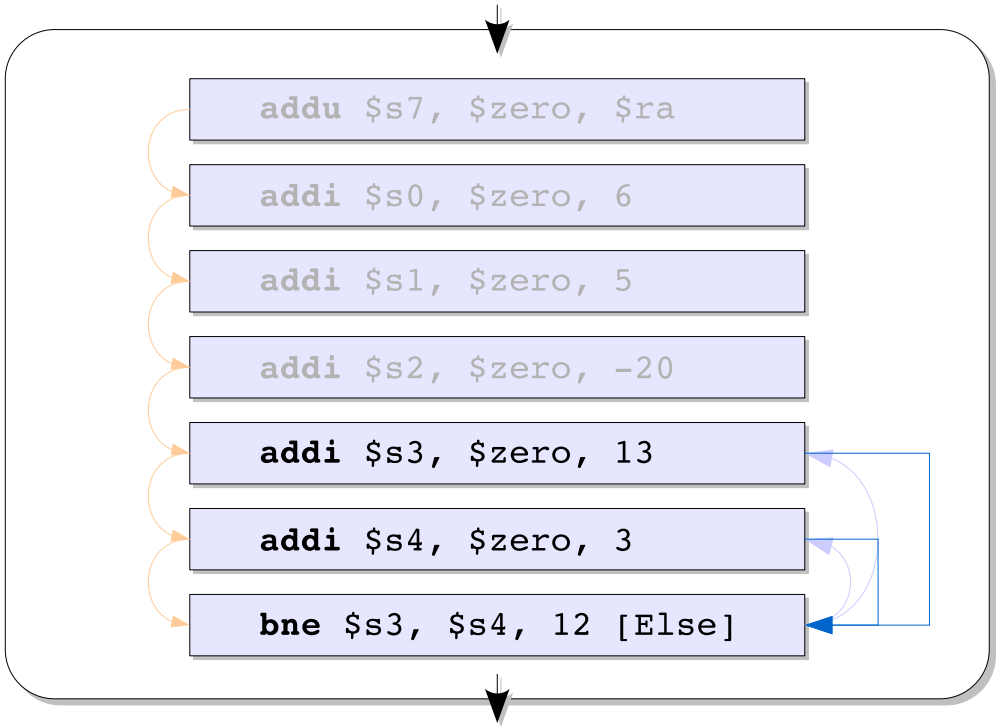
Lista de Dependências Reversas I

- Modela os dependentes de determinada instrução



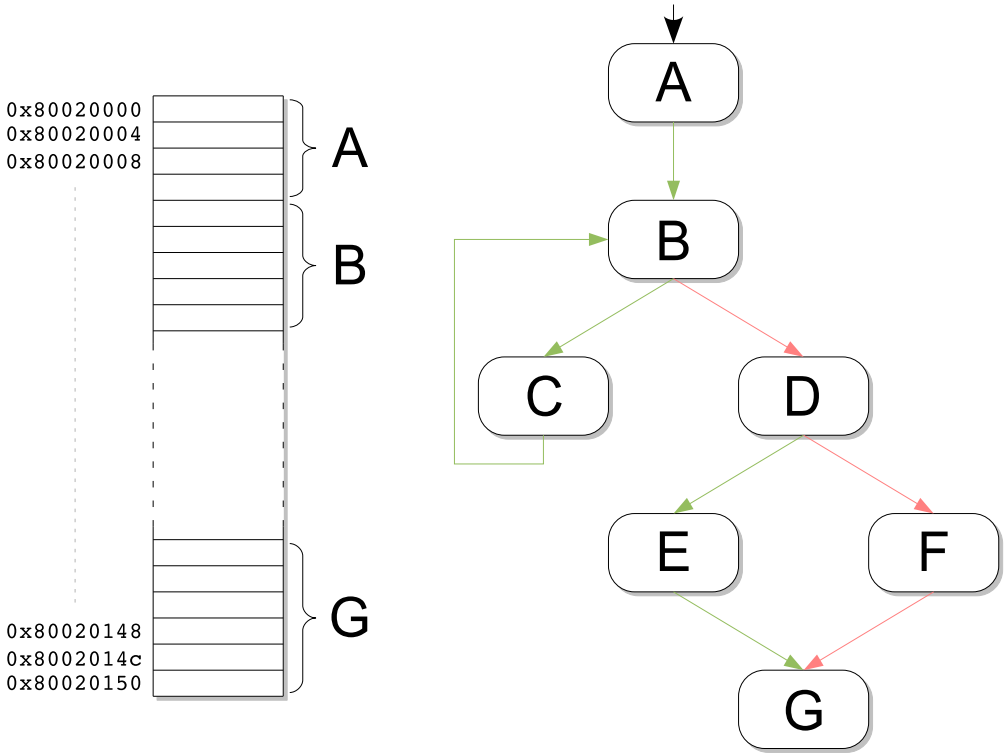
Lista de Dependências Reversas II

- Permite a atualização do DFG com menor esforço



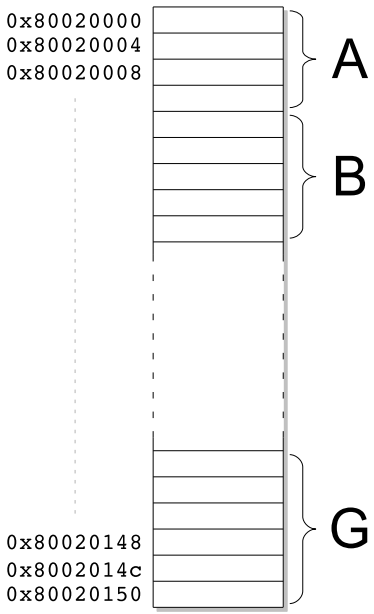
Mapeamento de Endereços Virtuais I

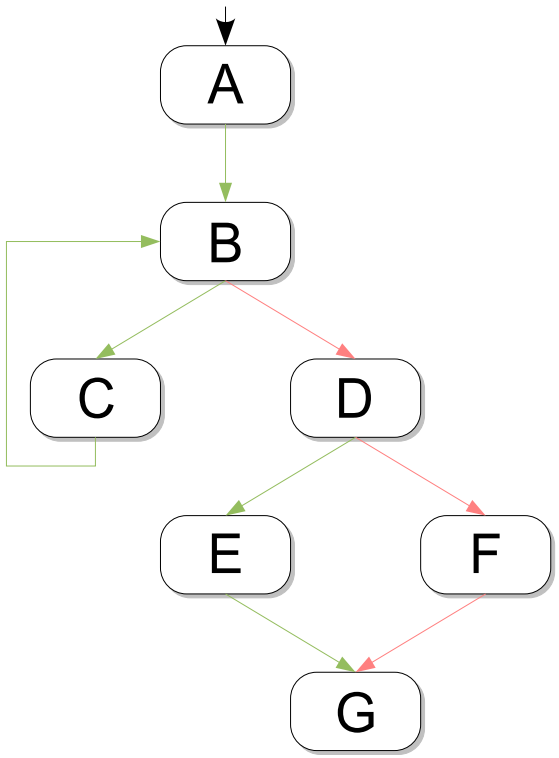
- Mapeia os endereços virtuais compreendidos por blocos-básicos



Mapeamento de Endereços Virtuais II

- Permite localizar rapidamente uma instrução, dado seu endereço virtual





Mapeamento de Endereços Virtuais III

- Permite verificar a consistência do programa e da desmontagem

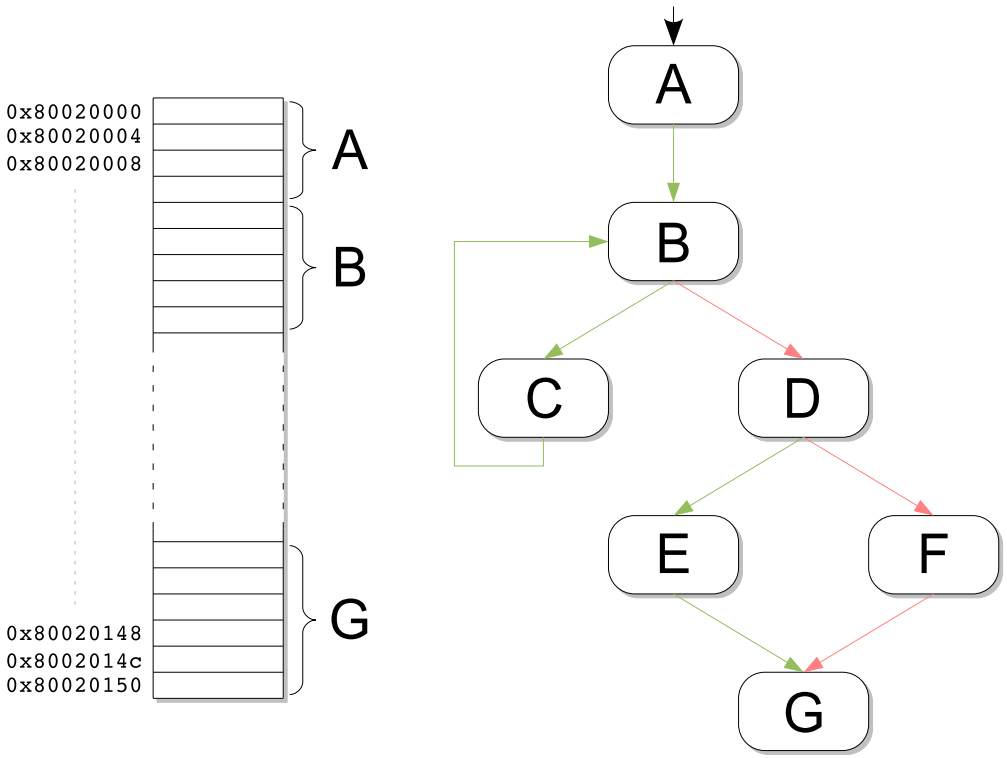


Tabela de Saltos

- Definição ainda em fase de de estudo
- Elencará todas as origens e destinos de desvio encontrados
- Facilitará a atualização dos desvios afetados por uma modificação no CDFG

Resultados Parciais

A infra-estrutura já é capaz de:

- Carregar seções executáveis de um arquivo ELF/UNIX para MIPS R3000
- Decodificar recursivamente (empregando a extensão)
- Popular o CFG, listas de instruções, tabela de endereços virtuais

Difi culdades Encontradas

- Desmontagem completa geralmente não é possível usando apenas abordagens estáticas
- Peculiaridades das arquiteturas dificultam a definição de estruturas de dados genéricas e a modularização

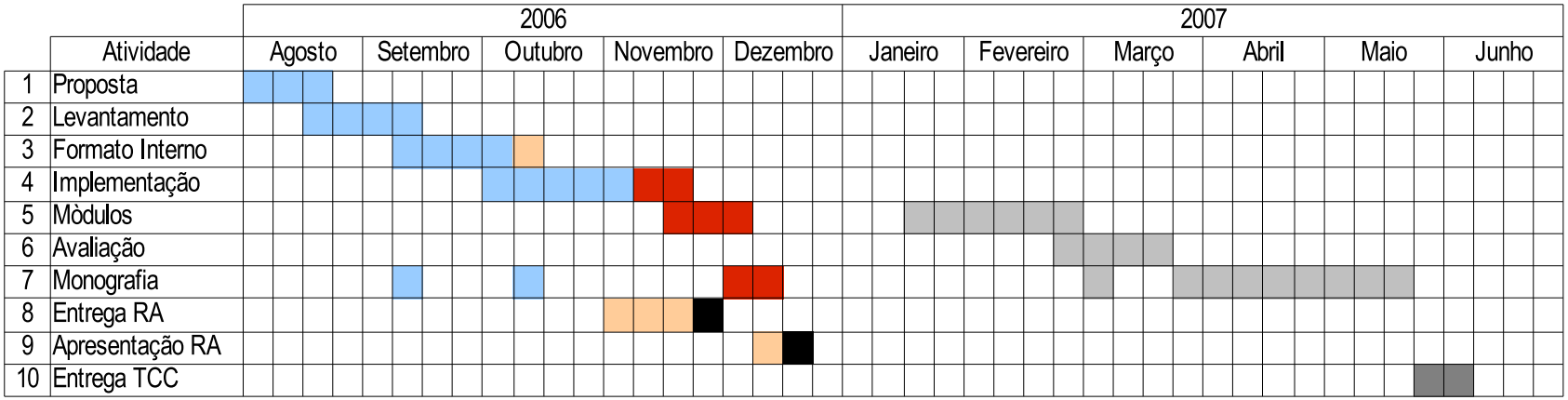
Problemas em Aberto

- Suporte a outras arquiteturas e sistemas operacionais
- Suporte a outros formatos de binários (ex: imagem de *kernel*)
- Binários modificados por técnicas de ofuscamento

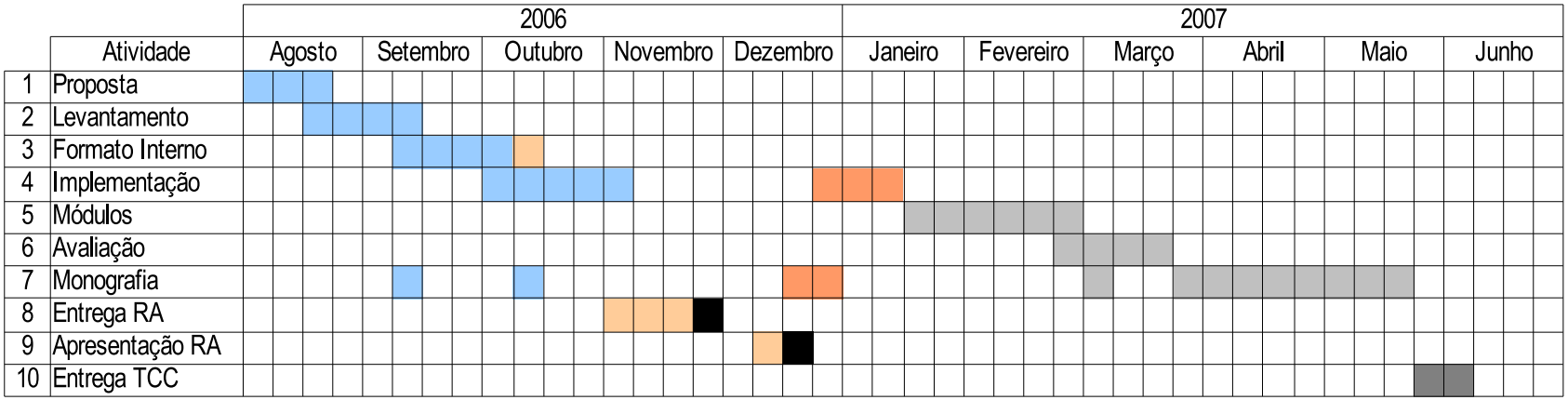
Trabalhos Futuros

- Definir e implementar a lista de dependências reversas e tabela de saltos
- Finalizar a implementação da computação dos DFGs
- Verificar a infra-estrutura
- Atividades previstas para o segundo semestre

Cronograma Estimado (2006/2)



Cronograma Atualizado (2007/1)



Fim da Apresentação

Obrigado!

Perguntas?

Conceituação

Binário Executável

No contexto deste trabalho, um arquivo binário executável é aquele que contém a imagem de um programa, composto por:

- Informações de carregamento para que o sistema operacional possa preparar o ambiente de execução
- Dados inicializados que o programa necessita para executar
- Instruções de máquina que serão executadas nativamente pelo procesador

Conceituação

Desmontagem

É o processo de decodificação onde:

1. A entrada consiste em instruções de máquina codificadas
2. A saída consiste numa representação de nível mais elevado, onde a instrução e seus operandos podem ser identificados

É o inverso da *montagem*

Conceituação

Representação Intermediária

- Estrutura de dados que representa o programa internamente à infra-estrutura proposta
- Permite que o programa tenha sua estrutura modificada de forma progressiva ?