

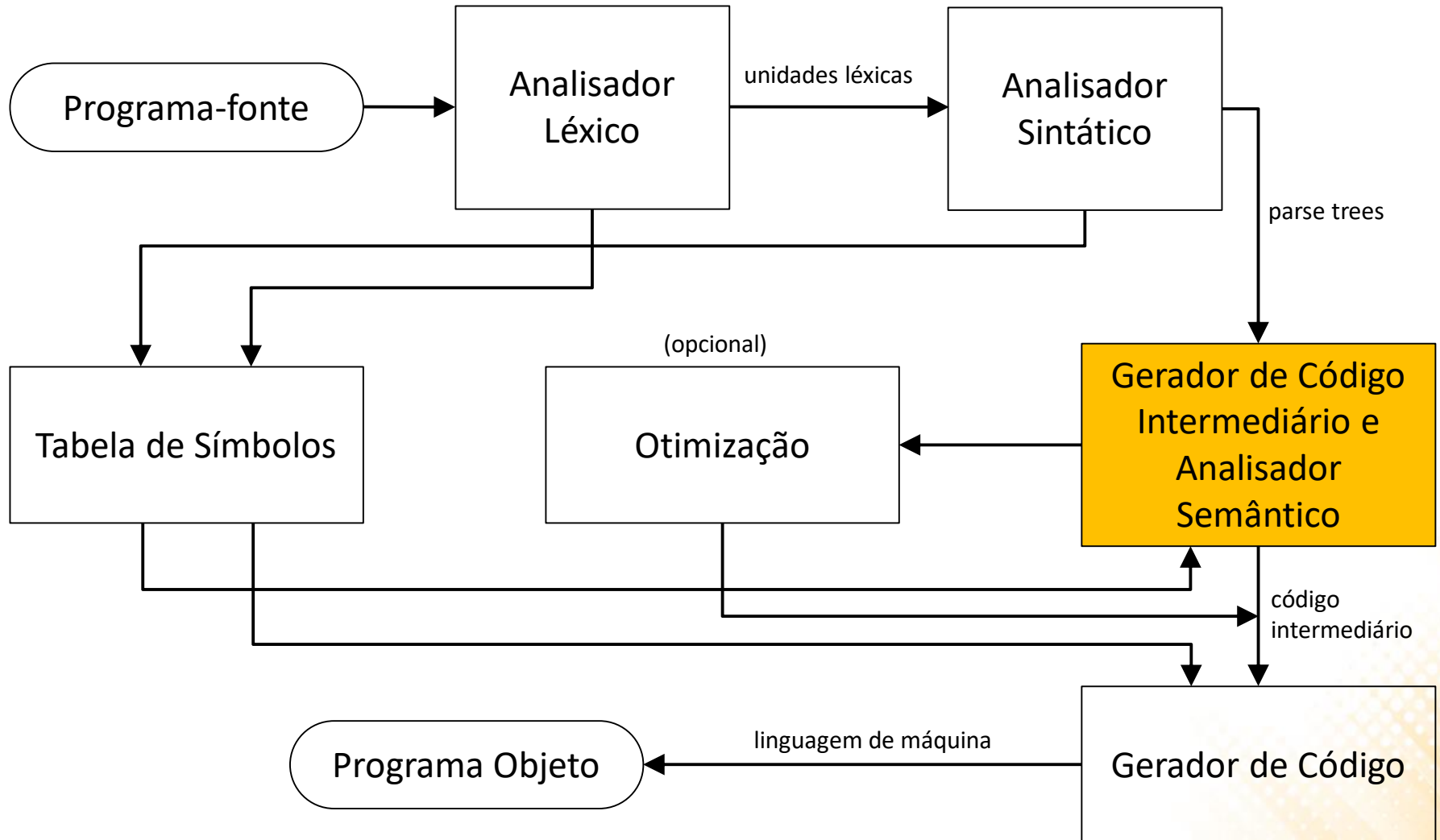
Compiladores

Aula 07 – Analise Semântica e Geração de Código Intermediário

Edirlei Soares de Lima

<edirlei.lima@universidadeeuropeia.pt>

Processo de Compilação



Processo de Compilação

- O **Gerador de Código Intermediário** produz um programa em uma linguagem intermediária entre o programa-fonte e saída final do compilador.
 - As linguagens intermediárias se parecem muito com linguagens de montagem (e muitas vezes são linguagens de montagem (assembly)).

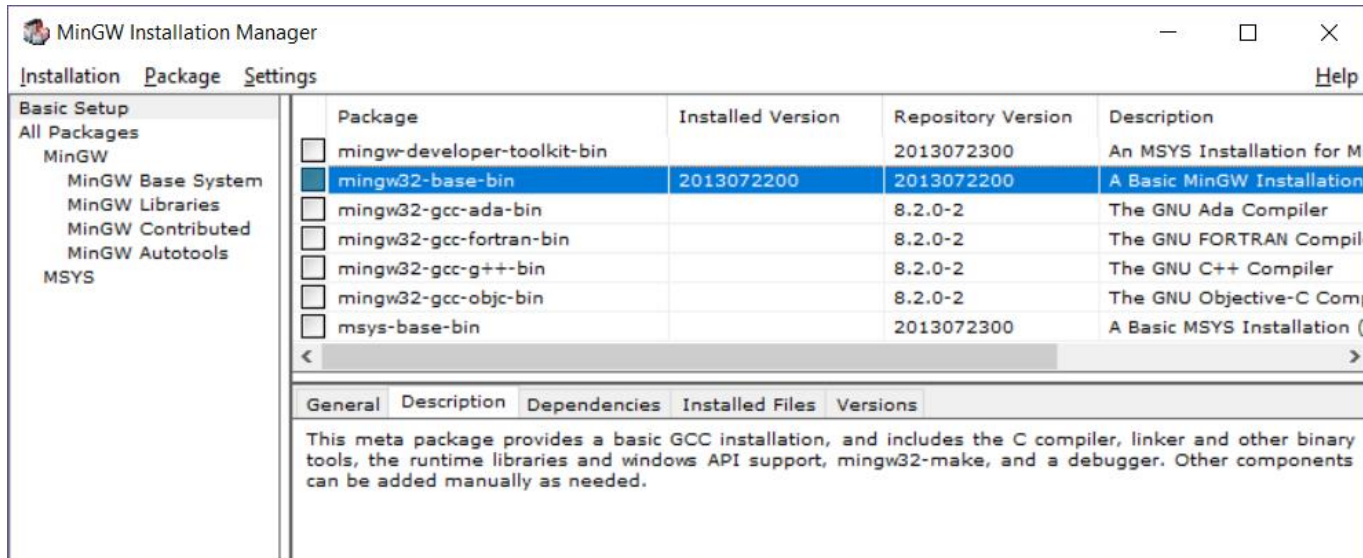
```
gcd:  pushl %ebp           % Save FP
      movl %esp,%ebp
      movl 8(%ebp),%eax  % Load a from stack
      movl 12(%ebp),%edx % Load b from stack
.L8:  cmpl %edx,%eax
      je   .L3           % while (a != b)
      jle .L5           % if (a < b)
      subl %edx,%eax    % a -= b
      jmp .L8
.L5:  subl %eax,%edx    % b -= a
      jmp .L8
.L3:  leave             % Restore SP, BP
      ret
```

Linguagem Assembly

- Assembly é uma linguagem legível por humanos para o código de máquina de uma arquitetura de computador específica.
- Instruções executam operações simples:
 - Operações aritméticas/lógicas;
 - Transferência de dados;
 - Controle do fluxo de execução (desvios, chamadas de função).
- Tipos de dados básicos:
 - Valores inteiros (1,2,4,8 bytes);
 - Endereços de memória;
 - Valores em ponto flutuante.

Software – Windows

- NASM:
 - <https://www.nasm.us/pub/nasm/releasebuilds/2.14.03rc2/win32/nasm-2.14.03rc2-win32.zip>
- GCC (MinGW):
 - <https://sourceforge.net/projects/mingw/files/latest/download>



Exemplo de Programa Assembly

```
SECTION .data
    formatout: db "%d", 10, 0
    variavell: dd 0
```

Dados Globais

```
SECTION .text
    global _main
    extern _printf
```

Código do Programa

```
_main :
    mov eax, 2
    mov ebx, 4
    add eax, ebx
    mov dword [variavell], eax

    push dword[variavell]
    push formatout
    call _printf
    add esp, 8

    mov eax, 0
    ret
```

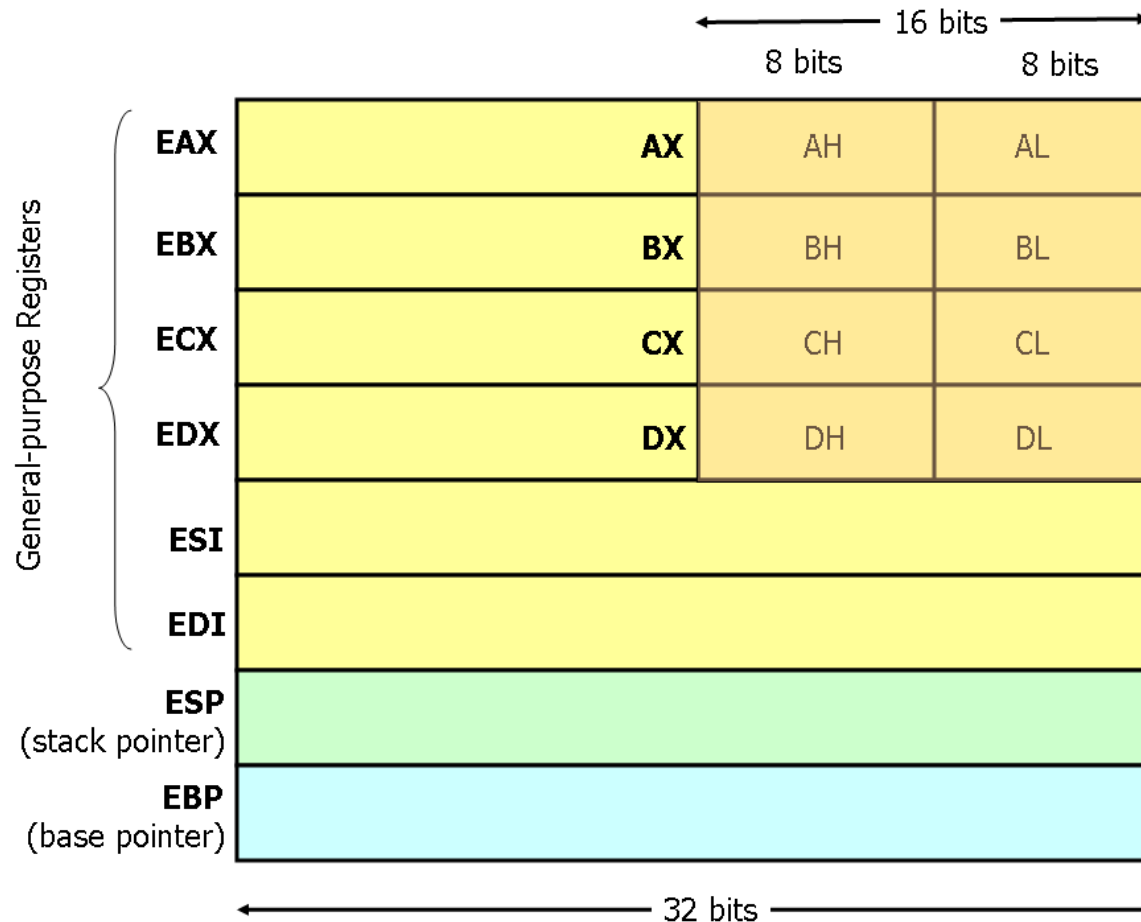
Compilar:

- `nasm -f win32 code.asm`
- `gcc code.obj -o code.exe`

Assembly – Tamanhos de Dados

Diretiva	Proposito	Espaço
db	Define Byte	Allocates 1 byte
dw	Define Word	Allocates 2 bytes
dd	Define Doubleword	Allocates 4 bytes
dq	Define Quadword	Allocates 8 bytes
dt	Define Ten Bytes	Allocates 10 bytes

Assembly – Registradores

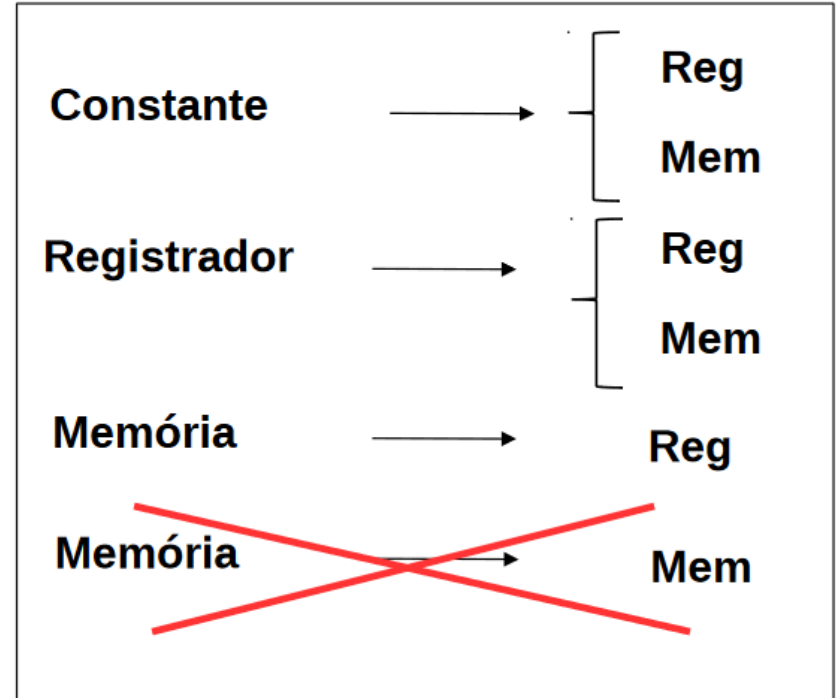


Assembly – Movimentação de Dados

- `mov <op1> <op2>`: copia os dados de `op2` para `op1`.
 - Movimentos de registrador para registrador são possíveis, mas os movimentos diretos de memória para memória não são.
 - É necessário primeiro carregar o conteúdo em um registrador e depois move-lo para a memória.

Exemplos:

```
mov eax, 2
mov ebx, 4
add eax, ebx
mov dword [variavel1], eax
```



Assembly – Movimentação de Dados

- `push <op>`: coloca `op` na parte superior da pilha de memória.
- `pop <op>`: remove 4 bytes da parte superior da pilha e os armazena em `op` (registrador ou memória).
- Exemplo:

```
push dword[variavel1]
push formatout
call _printf
add esp, 8
```

Assembly – Operações Aritméticas Inteiras

- `add <op1> <op2>`: soma `op1` e `op2` e armazena o resultado em `op1`.
- `sub <op1> <op2>`: subtrai `op1` e `op2` e armazena o resultado em `op1`.
- `imul <op1> <op2>`: multiplica `op1` por `op2` e armazena o resultado em `op1`.
- `idiv <op>`: divide o conteúdo do registrador `eax` por `op` e armazena o resultado em `eax`.

Assembly – Usando Funções C Externas

- Função printf:

```
SECTION .data
    formatout: db "%d", 10, 0
    variavel1: dd 0

SECTION .text
    global _main
    extern _printf

    _main :
    mov eax, 2
    mov ebx, 4
    add eax, ebx
    mov dword [variavel1], eax

    push dword[variavel1]
    push formatout
    call _printf
    add esp, 8

    mov eax, 0
    ret
```

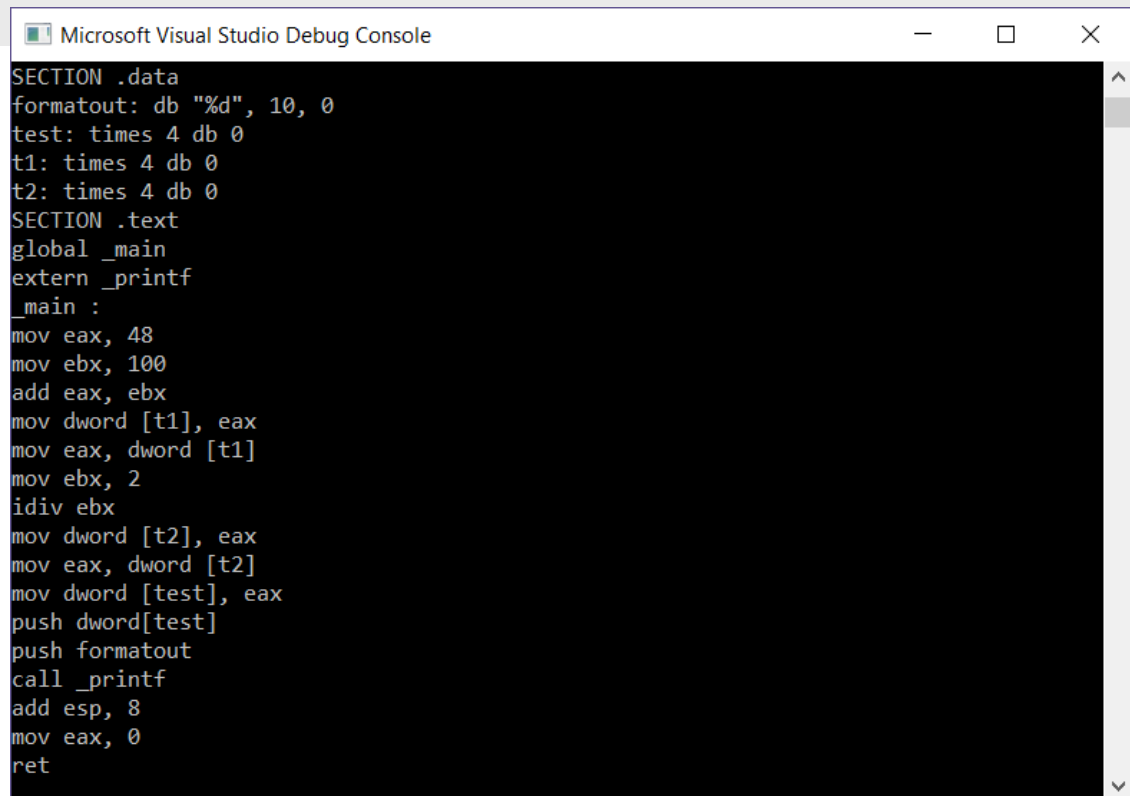
Geração de Código Intermediário (Implementação)

```
...  
  
void GenerateCode(Node * ast)  
{  
    int i = 0;  
    if (ast == NULL)  
        return;  
    while (ast->children[i] != NULL)  
    {  
        GenerateCode(ast->children[i]);  
        i++;  
    }  
    if ((ast->type == ADD_OP) || (ast->type == SUB_OP) ||  
        (ast->type == DIV_OP) || (ast->type == MULT_OP)){  
        if ((ast->children[0]->type == INT_LIT) &&  
            (ast->children[1]->type == INT_LIT)){  
            printf("mov eax, %s\n", ast->children[0]->info);  
            printf("mov ebx, %s\n", ast->children[1]->info);  
        }  
    }  
    ...  
}
```

Geração de Código Intermediário (Implementação)

- Entrada:

```
int test;  
test = (45 + 100) / 2;  
print(test);
```



```
Microsoft Visual Studio Debug Console  
SECTION .data  
formatout: db "%d", 10, 0  
test: times 4 db 0  
t1: times 4 db 0  
t2: times 4 db 0  
SECTION .text  
global _main  
extern _printf  
_main :  
mov eax, 48  
mov ebx, 100  
add eax, ebx  
mov dword [t1], eax  
mov eax, dword [t1]  
mov ebx, 2  
idiv ebx  
mov dword [t2], eax  
mov eax, dword [t2]  
mov dword [test], eax  
push dword[test]  
push formatout  
call _printf  
add esp, 8  
mov eax, 0  
ret
```

Exercício 01

- 1) Continue a implementação do gerador de código intermediário para dar suporte a geração de código para declaração e utilização de variáveis float.
 - Atenção: add, sub, idiv e imul são operações para números inteiros.
 - Na internet existem documentos e exemplos de códigos ilustrando como utilizar números float em Assembly.
 - Algumas referências:
 - https://www.csee.umbc.edu/courses/undergraduate/313/spring05/burt_katz/lectures/Lect12/floatingpoint.html
 - <https://www.nasm.us/doc/nasmdoc3.html>
 - <http://www.cs.virginia.edu/~evans/cs216/guides/x86.html>
 - <https://www.csee.umbc.edu/portal/help/nasm/sample.shtml>

Leitura Complementar

- Aho, A. V., Lam, M. S., Jeffrey, R. S.
Compiladores: Princípios, Técnicas e Ferramentas. 2ª edição, Pearson, 2007.
ISBN: 978-8588639249.
 - Capítulo 6: Intermediate-Code Generation

