

Introdução ao Assembly

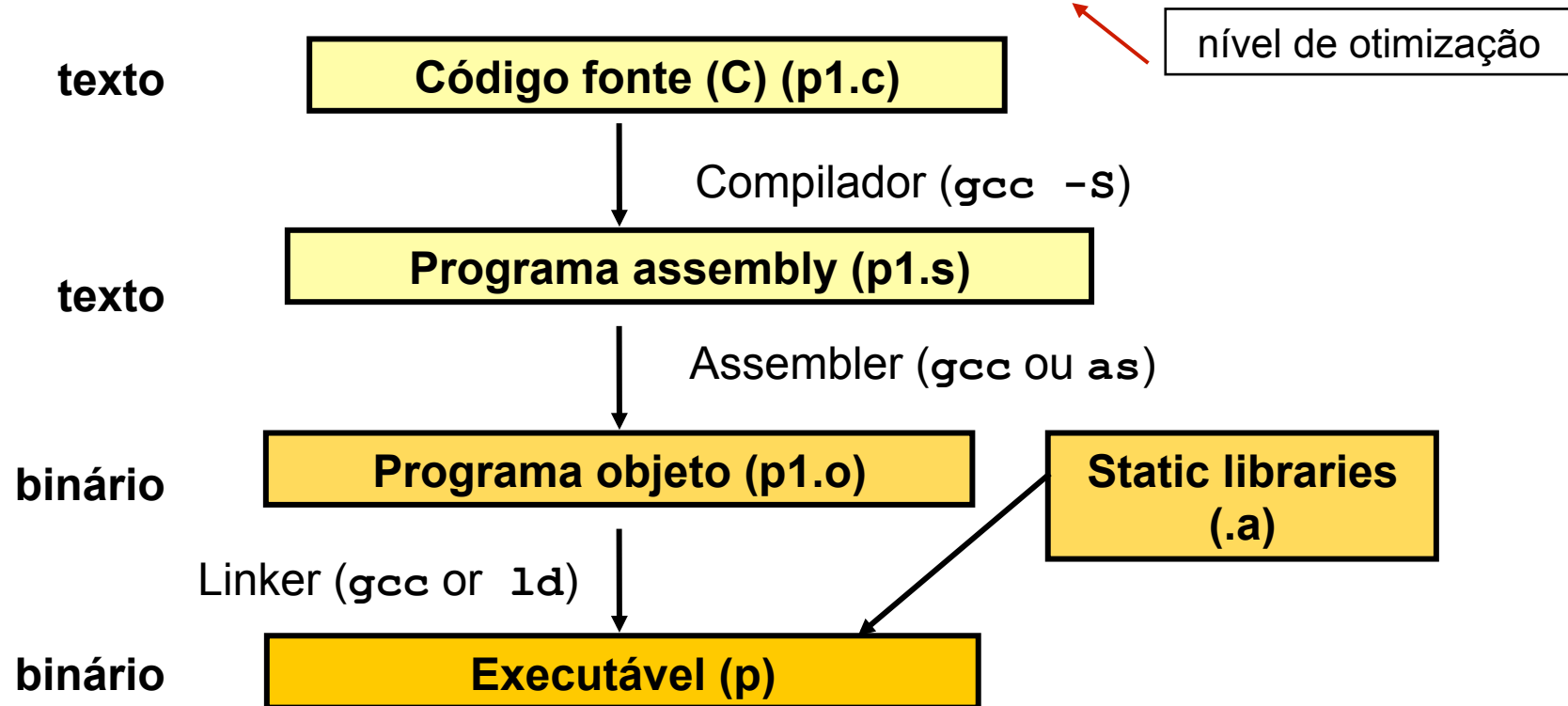
Operações Aritméticas e Lógicas

Representação de Programas

- Computadores executam código de máquina
 - sequências de bytes que codificam operações que manipulam dados, gerenciam memória, realizam E/S, ...
- Um compilador gera o código de um programa conforme
 - o conjunto de instruções da máquina alvo (IA32)
 - as regras estabelecidas pela linguagem de programação (C)
 - as convenções seguidas pelo sistema operacional
- O compilador gcc gera como saída do passo de compilação um programa em linguagem de montagem
 - entrada para o passo de montagem (assembler)

Geração de um executável

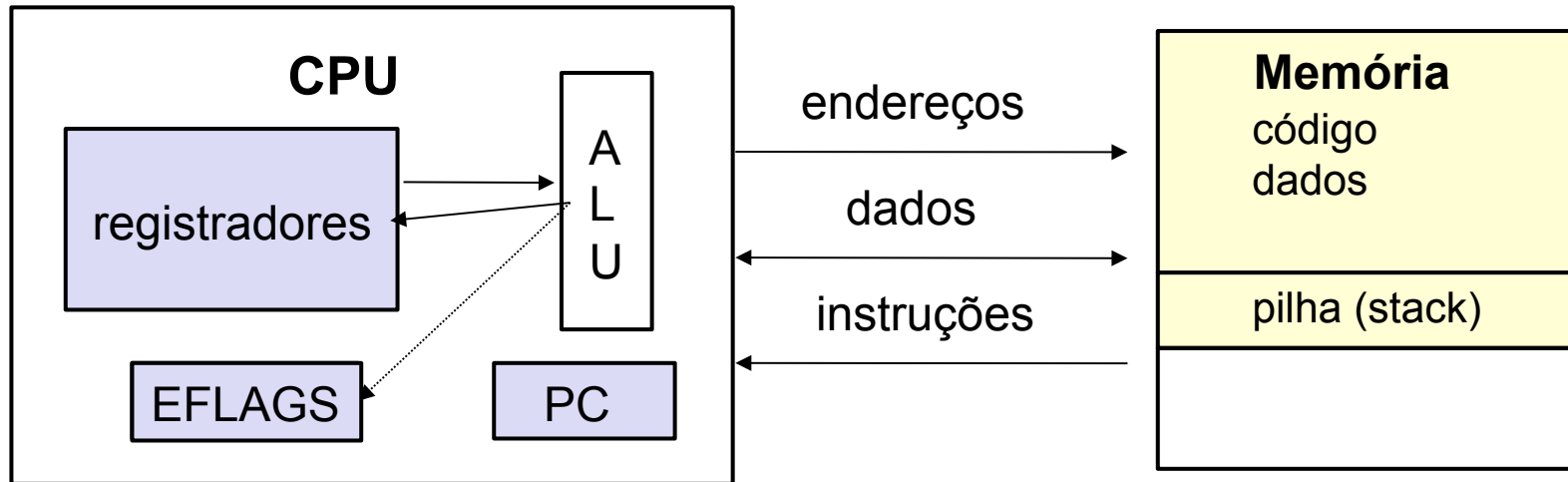
- Arquivo fonte `p1.c`
- Comando de compilação: `gcc -O1 -o p p1.c`



Linguagem de Montagem

- Muito próxima à linguagem de máquina
- Instruções de máquina executam operações bem simples
 - operações aritméticas/lógicas
 - transferência de dados entre memória e registradores
 - controle do fluxo de execução (desvios)
- Tipos de dados básicos
 - valores inteiros de 1, 2, ou 4 bytes
 - endereços de memória
 - dados em ponto flutuante (4 ou 8 bytes)
- Não existem tipos de dados estruturados (arrays ou structs)

Visão do programador assembly



Estado da CPU

- PC: endereço da próxima instrução
- Registradores:
 - valores inteiros, endereços
- EFLAGS: status da última operação
 - (overflow? zero? ...)

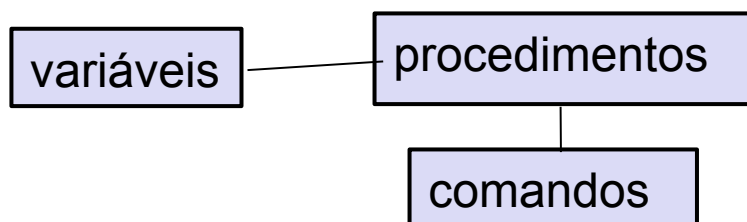
Memória “plana”

- array de bytes endereçáveis
- contém código, dados e pilha
- pilha usada para tratar chamada a procedimentos

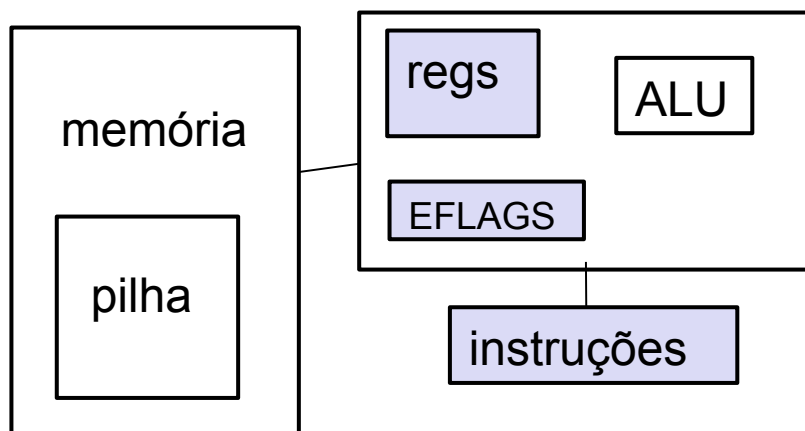
Representação de programas

Modelos de máquinas

“C”



Assembly



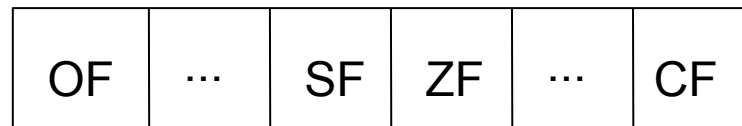
Dados

- char, short, int
- float, double
- struct, array
- pointer

- byte, 2-byte, 4-byte
- 4-byte, 8-byte
- alocação contígua de bytes
- endereço do byte inicial

Registrador EFLAGS

- Seus bits indicam a ocorrência de diferentes eventos
 - associados à última operação executada



- OF (overflow flag) → overflow
 - SF (sign flag) → resultado < 0
 - ZF (zero flag) → resultado == 0
 - CF (carry flag) → unsigned overflow
- Os bits de EFLAGS são consultados por instruções de desvio condicional

Registadores

- A CPU IA32 tem 8 registradores de 32 bits
 - valores inteiros e endereços (ponteiros)
- 6 registradores são de propósito geral
 - algumas instruções usam registradores específicos
- **%ebp** e **%esp** tem ponteiros para a pilha
 - uso de acordo com as convenções para a gerência da pilha
- Todos podem ser acessados como 16 bits (w) ou 32 bits (l)
 - os 4 primeiros registradores permitem acesso independente aos 2 bytes menos significativos (b)

		31	15	7
%eax	%ax	%ah		%al
%ebx	%bx	%bh		%bl
%ecx	%cx	%ch		%cl
%edx	%dx	%dh		%dl
%esi	%si			
%edi	%di			
%esp	%sp			
%ebp	%bp			

Modos de endereçamento

- A maioria das instruções tem um ou mais operandos

movl *fonte, destino*

- constantes (apenas fonte), ou conteúdo de registrador/memória
- Modos de endereçamento (localização dos operandos)
 - Imediato (constante): **\$0xaabb**, **\$1234**
 - Registrador (conteúdo de registrador): **%eax**, **%ax**, **%al**
 - Memória (registrador como ponteiro)
 - direto por registrador (sempre 32 bits!): **(%ebx)**, **(%esi)**
 - base-deslocamento (soma de offset): **8(%ebp)** → soma 8 ao end. em %ebp
 - Modo direto (“nome” de um símbolo/variável): **mov var, %ebx**
 - **ATENÇÃO** → \$var é o **endereço** de var ! (&var em C)

Combinações de operandos

	Fonte	Destino	Exemplo	Equiv. em C
movl	<i>Imed</i>	<i>Reg</i>	movl \$0x04,%eax	temp = 0x04;
		<i>Mem</i>	movl \$-147,(%eax)	*p = -147;
	<i>Reg</i>	<i>Reg</i>	movl %eax,%edx	temp2 = temp1;
		<i>Mem</i>	movl %eax,(%edx)	*p = temp;
	<i>Mem</i>	<i>Reg</i>	movl (%eax),%edx	temp = *p;

Não é possível fazer transferência direta memória-memória com uma única instrução !!!

Tamanho dos operandos

- Sufixo da instrução indica o tamanho do(s) operando(s)
 - long (l), word (w), byte (b)
- b - byte (8 bits)
 - `movb $0, (%eax) /* %eax contém end. Memória */`
 - `movb %al, %bl`
- w - word (2 bytes = 16 bits)
 - `movw $0, (%eax)`
 - `movw %bx, %ax`
- l - long (4 bytes = 32 bits)
 - `movl $0, (%eax)`

Grupos de Instruções

- Movimentação de dados
- Aritmética inteira
- Instruções lógicas
- Instruções de controle de fluxo
- Instruções em ponto flutuante

Movimentação de Dados

Instrução	Efeito	Descrição
mov S, D mov[b w l]	S → D move byte, word(16), long(32)	Move
movs S, D movsbw movsbl movswl	signExtend(S) → D move signed-ext byte to word move signed-ext byte to long move signed-ext word to long	Move sign extension
movz S, D movzbw movzbl movzwl	zeroExtend(S) → D move zero-ext byte to word move zero-ext byte to long move zero-ext word to long	Move zero extension ...

Exemplos

- `movl $0x4050, %eax` - imm--reg
- `movw %bx, %ax` - reg--reg
- `movb $-17, 1(%eax)` - imm--mem
- **Movimentação com extensão**
 - assuma que `%dh = 0xCD` e `%eax = 0x98765432`
 - `movb %dh, %al` → `%eax = 987654CD`
 - `movsbl %dh, %eax` → `%eax = FFFFFFFCD`
 - `movzbl %dh, %eax` → `%eax = 000000CD`

Operações aritméticas e lógicas

Instrução	Efeito	Descrição
inc_s D	$D + 1 \rightarrow D$	Incremento (reg ou mem)
dec_s D	$D - 1 \rightarrow D$	Decremento (reg ou mem)
neg_s D	$-D \rightarrow D$	Negação (compl 2)
not_s D	$\sim D \rightarrow D$	NOT (lógico, bit a bit)
add_s S, D	$D + S \rightarrow D$	ADD
sub_s S, D	$D - S \rightarrow D$	Subtract
imul_s S, D	$D * S \rightarrow D$	Signed Multiply
and_s S, D	$D \& S \rightarrow D$	AND
or_s S, D	$D S \rightarrow D$	OR
xor_s S, D	$D \wedge S \rightarrow D$	XOR

Exemplos

- Assuma os valores armazenados em memória e registradores:

Endereço	Valor
0x100	0xFF
0x104	0xAB
0x108	0x13

Registrador	Valor
%eax	0x100
%ecx	0x01
%edx	0x03

Instrução	Destino	Valor
<code>addl %ecx, (%eax)</code>	Mem em 0x100	0x100
<code>subl %edx, 4(%eax)</code>	Mem em 0x104	0xA8
<code>subl %edx, %eax</code>	%eax	0xFD
<code>incl 8(%eax)</code>	Mem em 0x108	0x14

Operações de deslocamento

Instrução	Efeito	Descrição
sals k, D	$D \ll k \rightarrow D$	Left shift (preenche com 0)
shls k, D	$D \ll k \rightarrow D$	Left shift (preenche com 0)
sars k, D	$D \gg k \rightarrow D$	Arithmetic Right shift (ext sinal)
shrs k, D	$D \gg k \rightarrow D$	Logical Right shift (preenche com 0)

- k de 0 a 31 (codificação em 1 byte)
 - valor imediato ou conteúdo de %cl
 - **shl \$2, %eax**
 - **sarl %cl, %ebx**

Declaração de Dados

- Assembler do gcc permite declaração de variáveis de vários tipos
 - na seção de dados `.data`
- Exemplos:

```
vet: .byte 48, 0b00110000, 0x30, '0'  
s1:  .string "resultado igual a %d\n"  
i:   .int 1000
```

Exemplo do laboratório

```
.data
S2: .int 10, 20, 30, 40
.text
.globl main
main:
    pushl %ebx                /* salva ebx (na pilha) */
    movl $0, %ebx            /* ebx = 0; */
    movl $S2,%ecx            /* ecx = &S2; */
L1:  cmpl $4, %ebx           /* if (ebx == 4) ? */
    je L2                    /* goto L2 */
    movl (%ecx), %eax        /* eax = *ecx; */
    call printnum            /* imprime valor de eax */
    addl $1, %ebx            /* ebx += 1; */
    addl $4, %ecx            /* ecx += 4; */
    jmp L1                    /* goto L1; */
L2:  movl $0, %eax           /* eax = 0; (valor de retorno) */
    popl %ebx                /* restaura ebx */
    ret                       /* retorna */
```