1. Linguagem de Montagem e de Máquina

1.1 Nível de Máquina: Família x86 (IA-32)

Registradores

- **Registradores de Dados (32/16/8 bits)**

	31 16	15 8	7 0	
eax		ax		acumulador
		ah	al	
ebx		bx		base
		bh	bl	
ecx		cx		contador
		ch	cl	
edx		dx		dado
		dh	dl	

- **Registradores de Endereço (32/16 bits)**

esp		sp	ponteiro para pilha
ebp		bp	ponteiro base
esi		si	índice fonte
edi		di	índice destino

- **Flags e Apontador de Instruções (32/16 bits)**

eip		ip	ponteiro de instruções
ef		flags	flags

  - **Detalhes do EFLAGS**

cf	Bit 0	vai-um (carry)
zf	Bit 6	zero
sf	Bit 7	sinal (sign)
of	Bit 11	overflow

1.2 Instruções

movl	f,d	d = f
pushl	f	empilha f (esp = esp - 4; mem[esp] = f)
popl	d	desempilha d (d = mem[esp]; esp = esp + 4)
addl	f,d	d = d + f
subl	f,d	d = d - f
andl	f,d	d = d & f
orl	f,d	d = d | f
xorl	f,d	d = d ^ f
imull	f1,%eax	eax = eax * f1
idivl	f1,%eax	(quociente) eax = edx:eax / f1; (resto) edx = edx:eax % f1
cdq		if (eax < 0) edx = 0FFFFFFFFH else edx = 0 (usar com idiv)
cmpl	f,d	flags(CF,ZF,SF,OF) = d - f (afeta apenas flags)
testl	f,d	ZF = (d & f) == 0 (afeta apenas flag ZF)
jg	label	if (d > f) goto label; if (!ZF && (SF==OF)) goto label
jge	label	if (d >=f) goto label; if (SF= =OF) goto label
jl	label	if (d < f) goto label; if (ZF && (SF!=OF)) goto label
jle	label	if (d <=f) goto label; if (SF!=OF) goto label
je	label	if (d = f) goto label; if (ZF) goto label
jne	label	if (d != f) goto label; if (!ZF) goto label
jmp	label	goto label
call	label	executa subrotina label (empilha eip; goto label)
ret		retorna da subrotina (desempilha eip)

f	f1	d	f,d
$n reg mem	reg mem	reg mem	$n,reg $n,mem reg,reg reg,mem mem,reg

n	reg	mem	index	base	scale	disp
0 .. 2^{32}-1 -2^{31}.. 2^{31}-1	%eax %ebx %ecx %edx %esp %ebp %esi %edi	disp(base) disp(base,index,scale)	%eax %ebx %ecx %edx %ebp %esi %edi	%eax %ebx %ecx %edx %esp %ebp %esi %edi	1 2 4 8	-2^{31}.. 2^{31}-1

endereço(mem) = base + (index * scale) + disp

Observações:

- **f ou f1: operando fonte**
- **d: operando destino ou resultado**
- **Operandos imediatos (constantes numéricas) devem começar com $**
- **Registradores devem iniciar com %**
- **Na mesma instrução, apenas um dos operandos pode ser do tipo mem**
- **O endereço de acesso à memória é calculado pela expressão:**
  - **endereço(mem) = base + (index * scale) + disp**
- **A instrução cdq é usada para estender o sinal de %eax em %edx antes de executar a idivl**
- **Comentários:**
  - **até o término linha: #**
  - **uma ou mais linhas: /* */**

1.3 Atribuições

- **int a,b,c,i,v[10];**

a = 10;	movl $10,a
b = a;	movl a,%eax movl %eax,b
a = a + 10;	addl $10,a

b = a - 10;	movl a,%eax
	subl $10,%eax
	movl %eax,b

c = a & b;	movl a,%eax
	andl b,%eax
	movl %eax,c

c = a * b;	movl a,%eax
	imull b,%eax
	movl %eax,c

c = a / b;	movl a,%eax
	cdq
	idivl b,%eax
	movl %eax,c

c = a % b;	movl a,%eax
	cdq
	idivl b,%eax
	movl %edx,c

a = v[i];	movl v,%ecx
	movl i,%edx
	movl 0(%ecx,%edx,4),%eax
	movl %eax,a

1.4 Introdução a Procedimentos e Funções

- Uso da pilha:
  - Endereço de retorno
  - Passagem de parâmetros
  - Armazenamento de variáveis locais
- Uso de registradores:
  - %ebp é usado como apontador para parâmetros e variáveis locais na pilha
    - O valor original do %ebp deve ser preservado
    - Deve ser guardado na pilha no início da rotina e recuperado no final da rotina
    - A instrução movl %esp,%ebp faz o %ebp apontar para os parâmetros e variáveis locais na pilha
  - Seqüência mínima de uma rotina:

```
rotina:
    pushl %ebp          # guarda %ebp original na pilha
    movl %esp,%ebp      # %ebp aponta para parâmetros
                        # e variáveis locais
    subl $N,%esp        # reserva N bytes na pilha
                        # para variáveis locais
    ...
    movl %ebp,%esp      # libera espaço reservado na
                        # pilha para variáveis locais
    popl %ebp           # recupera %ebp original da pilha
ret                     # retorna
```

  - %eax guarda o valor de retorno de uma função

Parâmetros e Retorno de Função

proc()
{
 	
...
 	
}
	proc:
	pushl %ebp
	movl %esp,%ebp
	subl $0,%esp
	...
	movl %ebp,%esp
	popl %ebp
	ret

int func()
{
 	
return(10);
 	
}
	func:
	pushl %ebp
	movl %esp,%ebp
	subl $0,%esp
	movl $10,%eax
	movl %ebp,%esp
	popl %ebp
	ret

int func(int x,y,z)
{
 	
return(x-y+z);
 	
}
	func:
	pushl %ebp
	movl %esp,%ebp
	subl $0,%esp
	movl 8(%ebp),%eax
	subl 12(%ebp),%eax
	addl 16(%ebp),%eax
	movl %ebp,%esp
	popl %ebp
	ret

1.5 Linguagem de Montagem no Linux

- Programas em linguagem de montagem:
  - as -o arquivo.o arquivo.s
  - ld -o arquivo arquivo.o
- Programas em linguagem de montagem e em linguagem C:
  - gcc arquivo.c arquivo.s -o arquivo
- Obtendo mais informações:
  - Linguagem de montagem: info as
  - Linguagem C: info gcc
- Preservar o valor de %ebx, caso seja necessário o seu uso, guarda-lo na pilha:

```
pushl %ebx
...
popl %ebx
```

- Pseudo instruções
  - .text:
    - indica um trecho de programa que será armazenado no segmento de código
  - .globl label
    - declara o label global, visível por outros módulos

Um programa simples em linguagem de montagem e linguagem C para exercício

- simples.s

```
.text
.globl func
func:
   pushl %ebp
   movl %esp,%ebp
   subl $0,%esp
   movl $10,%eax
   movl %ebp,%esp
   popl %ebp
ret
```

- simples.c

```
main()
{
   printf("%d\n",func());
}
```

- Compilação
  - gcc simples.c simples.s -o simples
- Execução
  - ./simples
- Resultado
  - 10

1.6 Estruturas Condicionais

- Teste e instrução de desvio condicional

a==b	a!=b	a<b	a<=b	a>b	a>=b
jne	je	jge	jg	jle	jl

if (a==10)
{
  ...
}
	cmpl $10,a
	jne Lendif
	  ...
	Lendif:

if (a>b)
 	
{
  ...
}
	movl a,%eax
	cmpl b,%eax
	jle Lendif
	  ...
	Lendif:

if (a<=b)
 	
{
  ...
}
else
{
  ...
}
	movl a,%eax
	cmpl b,%eax
	jg Lelse
	  ...
	jmp Lendif
	Lelse:
	
	  ...
	Lendif:

1.7 Estruturas de Repetição

for(i=0; i<10 ; i++)
 	
{
  ...
}
	movl $0,i
	Lfor:
	  cmpl $10,i
	  jge Lendfor
	  ...
	  addl $1,i
	jmp Lfor
	Lendfor:

while (a>=b)
 	
{
  ...
}
	Lwhile:
	  movl a,%eax
	  cmpl b,%eax
	  jl Lendwhile
	  ...
	jmp Lwhile
	Lendwhile:

do
{
  ...
}while (a<=b);
	Ldo:
	
	  ...
	  movl a,%eax
	  cmpl b,%eax
	jle Ldo

- Para as estruturas for e while, pode-se usar a tabela de teste do desvio condicional
- Para o do-while, deve-se usar o seguinte:

a==b	a!=b	a<b	a<=b	a>b	a>=b
je	jne	jl	jle	jg	jge

1.8 Procedimentos e Funções

int func()
{
 	
  int a,b,c;
  a = 10;
  b = 20;
  c = 30;
 	
}
	func:
	  pushl %ebp
	  movl %esp,%ebp
	  subl $12,%esp
	  movl $10,-4(%ebp)
	  movl $20,-8(%ebp)
	  movl $30,-12(%ebp)
	  movl %ebp,%esp
	  popl %ebp
	ret

int func(int x,y,z)
{
 	
  int a,b,c;
  a = x + y;
 	
  b = y + z;
 	
  c = z + x;
 	
  return(a-b+c);
 	
}
	func:
	  pushl %ebp
	  movl %esp,%ebp
	  subl $12,%esp
	  movl 8(%ebp),%eax
	  addl 12(%ebp),%eax
	  movl %eax,-4(%ebp)
	  movl 12(%ebp),%eax
	  addl 16(%ebp),%eax
	  movl %eax,-8(%ebp)
	  movl 16(%ebp),%eax
	  addl 8(%ebp),%eax
	  movl %eax,-12(%ebp)
	  movl -4(%ebp),%eax
	  subl -8(%ebp),%eax
	  addl -12(%ebp),%eax
	  movl %ebp,%esp
	  popl %ebp
	ret

- Seqüencia mínima de um procedimento ou função:

```
subrotina:
    pushl %ebp          # guarda %ebp original
    movl %esp,%ebp      # aponta % ebp para parâmetros
                        # e variáveis locais
    subl $N,%esp        # reserva N bytes na pilha
                        # para variáveis locais
    ...
    movl %ebp,%esp      # libera espaço reservado na
                        # pilha para variáveis locais
    popl %ebp           # recupera %ebp original
ret                     # retorna
```

- Chamada de procedimento ou de função com passagem de parâmetros:

```
# subrotina(int p1, int p2, int p3, ..., int pn)
pushl pn            # empilha último parâmetro
...
pushl p2            # empilha segundo parâmetro
pushl p1            # empilha primeiro parâmetro
call subrotina      # executa a subrotina
addl $(4*n),%esp    # retira parâmetros da pilha
```

- **Acesso aos parâmetros passados para uma subrotina (números inteiros ou endereços, 4 bytes):**

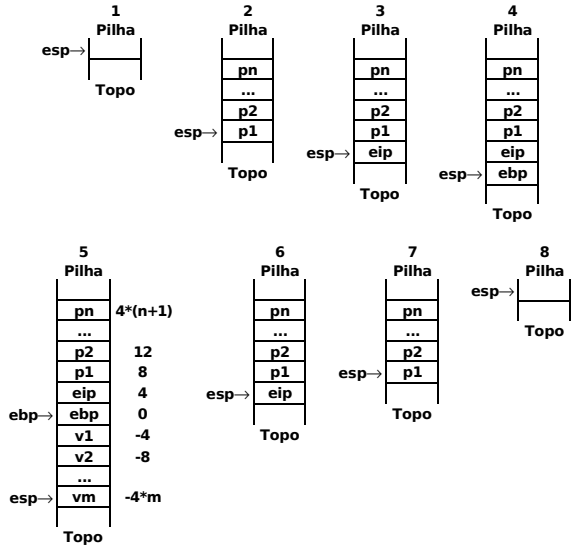
```
# subrotina(int p1, int p2, int p3, ..., int pn)
movl 8(%ebp),%eax          # parâmetro p1
...
movl 12(%ebp),%eax         # parâmetro p2
...
movl 16(%ebp),%eax         # parâmetro p3
...
movl 4*(n+1)(%ebp),%eax # parâmetro pn
```

- **Acesso à variáveis locais de um procedimento ou função (números inteiros ou endereços, 4 bytes):**

```
# int v1,v2,v3, ..., vm;
movl -4(%ebp),%eax              # variável v1
...
movl -8(%ebp),%eax              # variável v2
...
movl -12(%ebp),%eax             # variável v3
...
movl -4*m(%ebp),%eax            # variável vm
```

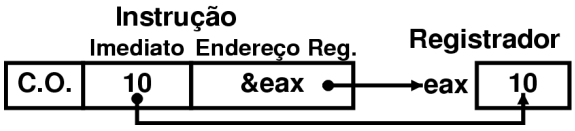

Evolução da Pilha em uma Subrotina

```
subrotina(int p1,p2,...,pn)          subrotina:
{int v1,v2,...,vm;                       pushl %ebp                  3
                                         movl %esp,%ebp              4
                                         subl $(4*m),%esp
   ...                                   ...                         5
                                         movl %ebp,%esp
                                         popl %ebp
}                                    ret                             6
main()                               main:
{  ...                                   ...
   subrotina(p1,p2,...,pn);              pushl pn                    1
                                         ...
                                         pushl p2
                                         pushl p1
                                         call subrotina              2
                                         addl $(4*n),%esp            7
   ...                                   ...                         8
}                                    ret
```

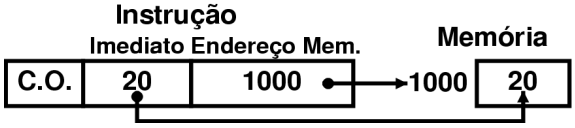


1.9 Endereçamento

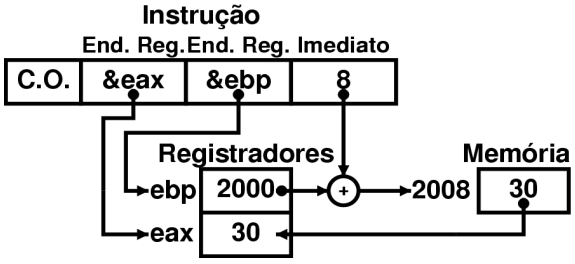
- **Endereçamento Imediato e de Registrador**
  - **movl $10,%eax**



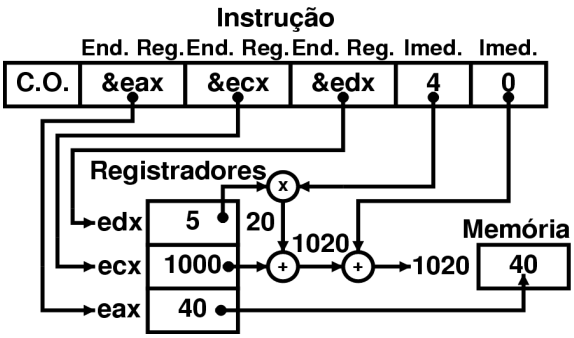
- **Endereçamento Direto**
  - **movl $20,1000**



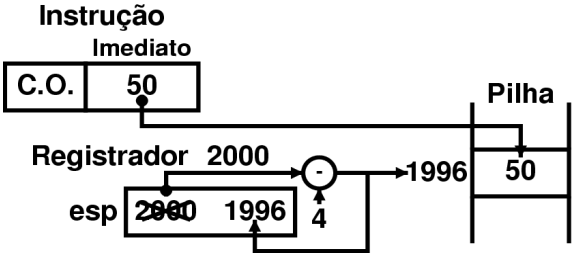
- **Endereçamento Indireto via Registrador**
  - **disp(base): endereço = base + disp**
  - **movl 8(%ebp),%eax**



- **Endereço Indireto Indexado**
  - **disp(base,index,scale):**
    - **endereço = base + index*scale + disp**
  - **movl %eax,0(%ecx,%edx,4)**



- **Endereçamento de Pilha**
  - **Uso implícito do %esp**
  - **push $50**



1.10 Formatos de Instruções e Linguagem de Máquina

Instrução	8 bits	8 bits	8 bits	32 bits	32 bits	Tam.
xxxxl %rgf,%rgd	xxxxxx 01	11 rgf rgd				2
xxxxl %reg,mem	xxxxxx 01	10 reg 100	ss iii bbb	disp		7
xxxxl mem,%reg	xxxxxx 11	10 reg 100	ss iii bbb	disp		7
xxxxl $n,%reg	1 xxxxxx 1	11 xxx reg			n	6
xxxxl $n,mem	1 xxxxxx 1	10 xxx 100	ss iii bbb	disp	n	11
pushl %reg	11111111	11110 reg				2
pushl mem	11111111	10110100	ss iii bbb	disp		7
pushl $n	01101000				n	5
popl %reg	10001111	11000 reg				2
popl mem	10001111	10000100	ss iii bbb	disp		7
imull %reg,%eax	11110111	11101 reg				2
imull mem,%eax	11110111	10101100	ss iii bbb	disp		7
idivl %reg,%eax	11110111	11111 reg				2
idivl mem,%eax	11110111	10111100	ss iii bbb	disp		7
cdq	10011001					1
jcc disp8	0111 cccc	disp8				2
jmp disp8	11101011	disp8				2
call disp	11101000			disp		5
ret	11000011					1

xxxxl	rgf,rgd reg,mem mem,reg	$n,reg $n,mem	
movl	100010	100011	000
addl	000000	000000	000
orl	000010	000000	001
andl	001000	000000	100
subl	001010	000000	101
xorl	001100	000000	110
cmpl	001110	000000	111
testl	100001	010100	000

cccc	
je	0100
jne	0101
jl	1100
jge	1101
jle	1110
jg	1111

ss	
1	00
2	01
4	10
8	11

rgf rgd reg bbb	iii	
eax	eax	000
ecx	ecx	001
edx	edx	010
ebx	ebx	011
esp		100
ebp	ebp	101
esi	esi	110
edi	edi	111

mem = disp(bbb,iii,ss)

- **movl %edx,%eax**
  - **xxxx = movl**
  - **rgf = edx**
  - **rgd = eax**

Instrução	8 bits	8 bits	8 bits	32 bits	32 bits
movl %edx,%eax	10001001	11 010 000			
xxxxl rgf,rgd	xxxxxx 01	11 rgf rgd			

xxxxl	rgf,rgd reg,mem mem,reg	$n,reg $n,mem		cccc		ss			rgf rgd reg bbb	iii	
movl	100010	100011	000	e	0100	1	00	rgd	eax	eax	000
addl	000000	000000	000	ne	0101	2	01		ecx	ecx	001
orl	000010	000000	001	l	1100	4	10	rgf	edx	edx	010
andl	001000	000000	100	ge	1101	8	11		ebx	ebx	011
subl	001010	000000	101	le	1110				esp		100
xorl	001100	000000	110	g	1111				ebp	ebp	101
cmpl	001110	000000	111						esi	esi	110
testl	100001	010100	000						edi	edi	111

- **addl %edx,8(%ebp)**
  - **xxxxl = addl**
  - **reg = %edx**
  - **mem = 8(%ebp) = disp(bbb,iii,ss)**
    - **ss = 1**
    - **iii =**
    - **bbb = ebp**
    - **disp = 0x00000008**

Instrução	8 bits	8 bits	8 bits	32 bits	32 bits
addl %edx,8(%ebp)	000000 01	10 010 100	00 100 101	0x8	
xxxxl %reg,mem	xxxxxx 01	10 reg 100	ss iii bbb	disp	

xxxxl	rgf,rgd reg,mem mem,reg	$n,reg $n,mem		cccc		ss			rgf rgd reg bbb	iii	
movl	100010	100011	000	e	0100	1	00		eax	eax	000
addl	000000	000000	000	ne	0101	2	01		ecx	ecx	001
orl	000010	000000	001	l	1100	4	10	reg	edx	edx	010
andl	001000	000000	100	ge	1101	8	11		ebx	ebx	011
subl	001010	000000	101	le	1110			iii	esp		100
xorl	001100	000000	110	g	1111			bbb	ebp	ebp	101
cmpl	001110	000000	111						esi	esi	110
testl	100001	010100	000						edi	edi	111

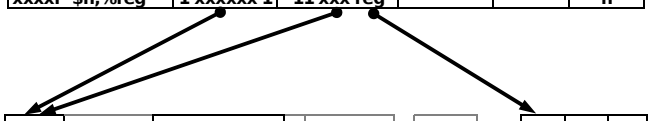
- **cmpl 0(%ecx,%edx,4),%eax**
  - **xxxxl = cmpl**
  - **mem = 0(%ecx,%edx,4) = disp(bbb,iii,ss)**
    - **ss = 4**
    - **iii = edx**
    - **bbb = ecx**
    - **disp = 0x00000000**
  - **reg = %eax**

Instrução	8 bits	8 bits	8 bits	32 bits
cmpl 0(%ecx,%edx,4),%eax	001110 11	10 000 100	10 010 001	0x0
xxxxl mem,%reg	xxxxxx 11	10 reg 100	ss iii bbb	disp

xxxxl	rgf,rgd reg,mem mem,reg	$n,reg $n,mem		cccc		ss			rgf rgd reg bbb	iii	
movl	100010	100011	000	e	0100	1	00	reg	eax	eax	000
addl	000000	000000	000	ne	0101	2	01	bbb	ecx	ecx	001
orl	000010	000000	001	l	1100	4	10	iii	edx	edx	010
andl	001000	000000	100	ge	1101	8	11		ebx	ebx	011
subl	001010	000000	101	le	1110				esp		100
xorl	001100	000000	110	g	1111				ebp	ebp	101
cmpl	001110	000000	111						esi	esi	110
testl	100001	010100	000						edi	edi	111

- **testl $1,%ecx**
  - **xxxxl = testl**
  - **n = 0x00000001**
  - **reg = ecx**

Instrução	8 bits	8 bits	8 bits	32 bits	32 bits
testl $1,%ecx	1 0101001	11 000 001			0x1
xxxxl $n,%reg	1 xxxxxx 1	11 xxx reg			n



xxxxl	rgf,rgd reg,mem mem,reg	$n,reg $n,mem		cccc	
movl	100010	100011	000	e	0100
addl	000000	000000	000	ne	0101
orl	000010	000000	001	l	1100
andl	001000	000000	100	ge	1101
subl	001010	000000	101	le	1110
xorl	001100	000000	110	g	1111
cmpl	001110	000000	111		
testl	100001	010100	000		

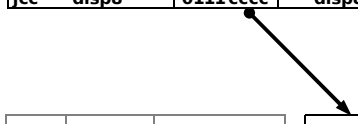
ss	
1	00
2	01
4	10
8	11

reg

rgf rgd reg bbb	iii	
eax	eax	000
ecx	ecx	001
edx	edx	010
ebx	ebx	011
esp		100
ebp	ebp	101
esi	esi	110
edi	edi	111

- **jl 10**
  - **cc = l**
  - **disp8 = end. desvio - (end. jcc + 2) = 0x0A**

Instrução	8 bits	8 bits	8 bits	32 bits	32 bits
jl 10	0111 1100	0x0A			
jcc disp8	0111 cccc	disp8			



xxxxl	rgf,rgd reg,mem mem,reg	$n,reg $n,mem		cccc	
movl	100010	100011	000	e	0100
addl	000000	000000	000	ne	0101
orl	000010	000000	001	l	1100
andl	001000	000000	100	ge	1101
subl	001010	000000	101	le	1110
xorl	001100	000000	110	g	1111
cmpl	001110	000000	111		
testl	100001	010100	000		

ss			rgf rgd reg bbb	iii	
1	00		eax	eax	000
2	01		ecx	ecx	001
4	10		edx	edx	010
8	11		ebx	ebx	011
			esp		100
			ebp	ebp	101
			esi	esi	110
			edi	edi	111

  - **Exemplo de cálculo do disp:**

    1000 jl lendfor

    ...

    1020 lendfor:

    - **disp = 1020 - (1000 + 2) = 18 = 0x12**

- **call 100**
  - **disp = end. rotina - (end. call + 5) = 0x00000064**

Instrução	8 bits	8 bits	8 bits	32 bits	32 bits
call 100	11101000			0x64	
call disp	11101000			disp	