

A arquitectura Y86

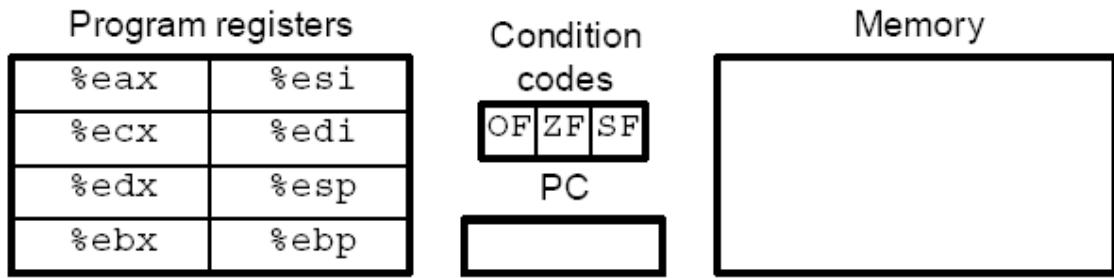


Figura 1 - Estado visível ao programador

8 registos de 32 bits. O registo `%esp` é utilizado implicitamente pelas instruções `push`, `pop`, `call` e `ret`.

O Program Counter (PC) contém o endereço da próxima instrução a ser executada.

Os códigos de condição representam atributos do resultado da última operação executada numa unidade funcional, nomeadamente, a existência de *Overflow* ($OF=1$), resultado nulo ($ZF=1$) e resultado com sinal ($SF=1$).

Byte	0	1	2	3	4	5
<code>nop</code>	0	0				
<code>halt</code>	1	0				
<code>rrmovl rA, rB</code>	2	0	rA	rB		
<code>irmovl V, rB</code>	3	0	8	rB	V	
<code>rmmovl rA, D(rB)</code>	4	0	rA	rB	D	
<code>mrmovl D(rB), rA</code>	5	0	rA	rB	D	
<code>OpI rA, rB</code>	6	fn	rA	rB		
<code>jXX Dest</code>	7	fn	Dest			
<code>call Dest</code>	8	0	Dest			
<code>ret</code>	9	0				
<code>pushl rA</code>	A	0	rA	8		
<code>popl rA</code>	B	0	rA	8		

Figura 2 - Instruções e respectivos formatos (valores em hexadecimal)

O comprimento das instruções varia entre 1 e 6 octetos. Os 4 *bits* mais significativos do octeto 0 contêm o código da operação. Para operações lógicas/aritméticas o campo `fn` indica qual a operação a realizar (ver figura 3).

Integer operations		Branches									
addl	<table border="1"><tr><td>6</td><td>0</td></tr></table>	6	0	jmp	<table border="1"><tr><td>7</td><td>0</td></tr></table>	7	0	jne	<table border="1"><tr><td>7</td><td>4</td></tr></table>	7	4
6	0										
7	0										
7	4										
subl	<table border="1"><tr><td>6</td><td>1</td></tr></table>	6	1	jle	<table border="1"><tr><td>7</td><td>1</td></tr></table>	7	1	jge	<table border="1"><tr><td>7</td><td>5</td></tr></table>	7	5
6	1										
7	1										
7	5										
andl	<table border="1"><tr><td>6</td><td>2</td></tr></table>	6	2	j1	<table border="1"><tr><td>7</td><td>2</td></tr></table>	7	2	jg	<table border="1"><tr><td>7</td><td>6</td></tr></table>	7	6
6	2										
7	2										
7	6										
xorl	<table border="1"><tr><td>6</td><td>3</td></tr></table>	6	3	je	<table border="1"><tr><td>7</td><td>3</td></tr></table>	7	3				
6	3										
7	3										

Figura 3 – Instruções lógicas/aritméticas e saltos

No caso de instruções que referem registos explicitamente o segundo código contém o código dos registos. Cada campo de 4 *bits* contém o código de um registo de acordo com a figura 4. O código 8 é usado quando apenas um registo é indicado explicitamente (caso de `pop`, `push`, `irmovl`).

Number	Register name
0	%eax
1	%ecx
2	%edx
3	%ebx
6	%esi
7	%edi
4	%esp
5	%ebp
8	No register

Figura 4 – Códigos dos registos

As instruções com valores imediatos (`irmovl`), deslocamentos (`mrmovl`, `rmovl`) e endereços (`jXX`) têm constantes de 4 octetos representados como *little endian* (octeto menos significativo primeiro).

De destacar as seguintes diferenças relativamente ao subconjunto relevante do IA32:

- instrução `movl` dividida em 4 instruções que indicam explicitamente o modo de endereçamento: *i* – imediato, *m* – memória, *r* – registo.
- Existe um único modo de endereçamento em memória: base + deslocamento. Não é suportado nem o factor de escala, nem o segundo registo de índice existente no IA32.
- Todas as operações lógico-aritméticas têm como operandos registos. Não é possível ter nenhum operando imediato nem em memória.
- Não existem instruções de multiplicação nem divisão e são excluídas muitas operações lógicas.

A arquitectura Y86

Instrução	Octetos	Comentários
nop	1	Nenhuma operação
halt	1	Parar operação
rrmovl rA, rB	2	Mover conteúdo de registo rA para registo rB
irmovl V, rB	6	Mover valor imediato V para registo rB
rmmovl rA, D(rB)	6	Mover conteúdo de rA para o endereço de memória rB+D
mrmmovl D(rB), rA	6	Mover o conteúdo da posição de memória rb+D para rA
addl rA, rB	2	Adicionar rB com rA colocando o resultado em rB
subl rA, rB	2	A rB subtrair rA , colocando o resultado em rB
andl rA, rB	2	Conjunção de rA com rB , resultado em rB
xorl rA, rB	2	Disjunção exclusiva de rA com rB , resultado em rB
jmp Dest	5	Salto incondicional para Dest
jle Dest	5	Salto se menor ou igual (SF=1 ou ZF=1) para Dest
jl Dest	5	Salto se menor (SF=1) para Dest
je Dest	5	Salto se igual (ZF=1) para Dest
jne Dest	5	Salto se diferente (ZF=0) para Dest
jge Dest	5	Salto se maior ou igual (SF=0 ou ZF=1) para Dest
jg Dest	5	Salto se maior (SF=0) para Dest
call Dest	5	Salta para Dest , guarda o endereço de retorno no topo da pilha
ret	1	Salta para o endereço que se encontra no topo da pilha
pushl rA	2	Guarda o conteúdo de rA na pilha e decrementa %esp
popl rA	2	Incrementa %esp e lê o topo da pilha para rA

A arquitectura Y86

A execução de programas no Y86 começa sempre no endereço 0. Tipicamente neste endereço inicializa-se o apontador para a pilha (`%esp`), seguido de um salto para o início do programa propriamente dito. Estas são funcionalidades que num sistema real estão tipicamente reservadas para o *linker* e sistema operativo.

Uma vez que a pilha cresce no sentido de endereços menores (a operação de *push* subtrai 4 valores ao apontador (`%esp`)), o endereço inicial da pilha deve ser um valor alto. Ainda assim é necessário garantir que o número de escritas na pilha não faz com que estes dados comecem a ser escritos em cima do código. A directiva `.pos` permite indicar ao *assembler* qual o endereço em que deve ser colocado aquilo que aparece a seguir a esta directiva. Usando um valor elevado para o topo da pilha tenta-se evitar sobreposições entre esta e o código.

A directiva `.align n` indica ao *assembler* que os dados imediatamente a seguir devem ser colocados em endereços múltiplos de `n`.

A directiva `.long valor` permite reservar 4 *bytes* para variáveis inteiras. O valor imediatamente a seguir a esta directiva é usado para inicializar estas 4 posições de memória.

```
.pos 0
irmovl Stack, %esp      # inicializa pilha
jmp main

.align 4               #dados alinhados em múltiplos de 4
t:
.long 10               # reserva 4 bytes para um inteiro com o valor inicial 10

main:
...                   # instruções
halt

.pos 0x100            # stack a começar no addr 0x100 (256 em decimal)
Stack:
```

Estrutura de um programa do Y86