

Y86:

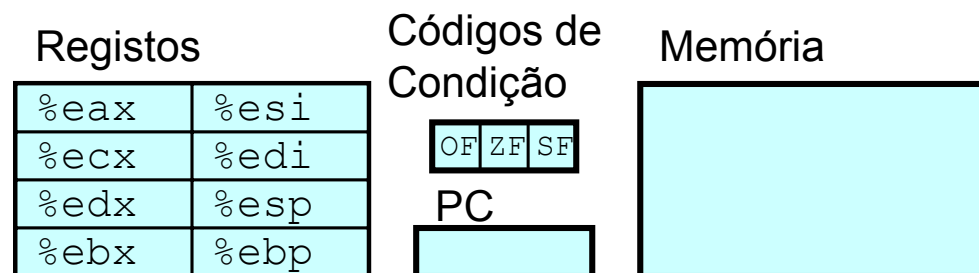
Definição da Arquitectura

Arquitectura de Computadores
Lic. em Engenharia Informática
Luís Paulo Santos

Y86: Arquitectura

9 – Organização do Processador	
Conteúdos	9.1 – Conceitos Fundamentais
Resultados de Aprendizagem	R9.1 – Analisar e descrever organizações sequenciais de processadores elementares

Y86: Estado visível



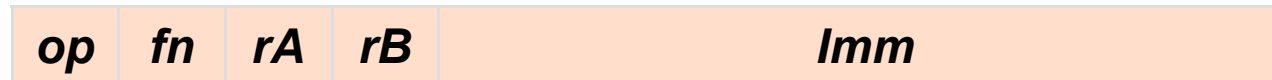
- Registos
 - Os mesmos que o IA32. Cada 32 *bits*
- Códigos de Condição
 - Flags de 1 bit alteradas pelas instruções aritméticas ou lógicas
 - OF: Transporte ZF: Zero SF: Sinal
- Program Counter
 - Indica endereço da instrução a executar
- Memória
 - Vector de *bytes*
 - Palavras armazenadas em ordem *little-endian*

Y86 : Instruction Set Architecture (ISA)

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

Jogo de Instruções do Y86

Y86 : Instruction Set Architecture (ISA)



- ***op*** – código de operação : identifica a instrução
- ***fn*** – função: identifica a operação
- ***rA*, *rB*** – indicam quais os registos a utilizar
- ***Imm*** – valor imediato (constante)

%eax	0	%esi	6
%ecx	1	%edi	7
%edx	2	%esp	4
%ebx	3	%ebp	5

Algumas instruções não necessitam de todos os campos:

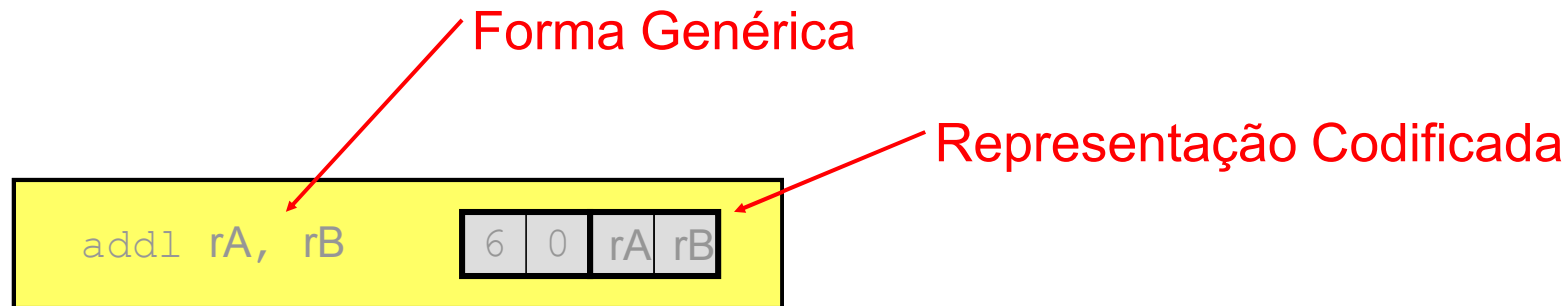
- apenas *op* e *fn* estão sempre presentes pois identificam a instrução

Y86: Modos de endereçamento

- Imediato:
 - constante especificada na própria instrução no campo *Imm*
- Registo:
 - registo(s) a utilizar especificados na instrução nos campos *rA*, *rB*
- Base+Deslocamento:
 - Endereço da posição de memória a ler/escrever especificado como a soma do valor imediato (*Imm*) com o conteúdo do registo (*rB*)

Y86 : Exemplo de Instrução

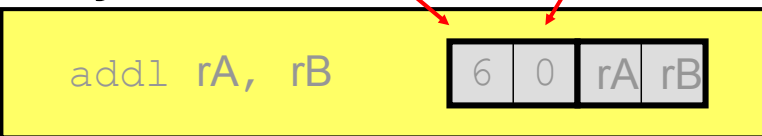
- Adição



- Adicionar valor no registo rA ao valor no registo rB
 - Guardar resultado no registo rB
 - Operações Aritméticas ou Lógicas só sobre operandos em registos
- Códigos de condição dependem do resultado
- e.g., **addl %eax,%esi** Representação Máquina: **60 06**

Y86: Operações Lógicas e Aritméticas

Instruction Code Function Code



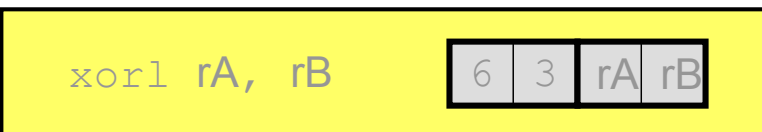
Subtracção (rA de rB)



And



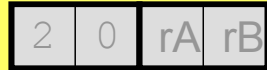
Exclusive-Or



- Referidas genericamente como “OPl”
- Códigos variam apenas no “function code”
 - 4 bits menos significativos do 1º byte
- Altera os códigos de condição

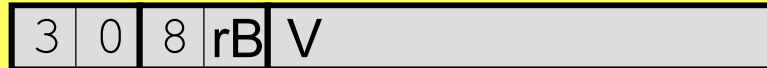
Y86: Transferência de Dados

`rrmovl rA, rB`



Registro --> Registro

`irmovl V, rB`



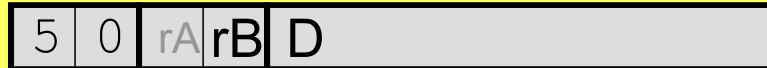
Imediato --> Registro

`rmmovl rA, D(rB)`



Registro --> Memória

`mrmovl D(rB), rA`



Memória --> Registro

- Semelhante à instrução IA32 `movl`
- Modo de endereçamento mais simples: Registro+Deslocamento

Y86: Transferência de Dados (exemplos)

IA32	Y86	Encoding
<code>movl \$0xabcd, %edx</code>	<code>irmovl \$0xabcd, %edx</code>	30 82 cd ab 00 00
<code>movl %esp, %ebx</code>	<code>rrmovl %esp, %ebx</code>	20 43
<code>movl -12(%ebp), %ecx</code>	<code>mrmovl -12(%ebp), %ecx</code>	50 15 f4 ff ff ff
<code>movl %esi, 0x41c(%esp)</code>	<code>rmmovl %esi, 0x41c(%esp)</code>	40 64 1c 04 00 00

<code>movl \$0xabcd, (%eax)</code>	—
<code>movl %eax, 12(%eax, %edx)</code>	—
<code>movl (%ebp, %eax, 4), %ecx</code>	—

Y86: Controlo de Fluxo (saltos)

Jump Unconditionally



Jump When Less or Equal



Jump When Less



Jump When Equal



Jump When Not Equal



Jump When Greater or Equal

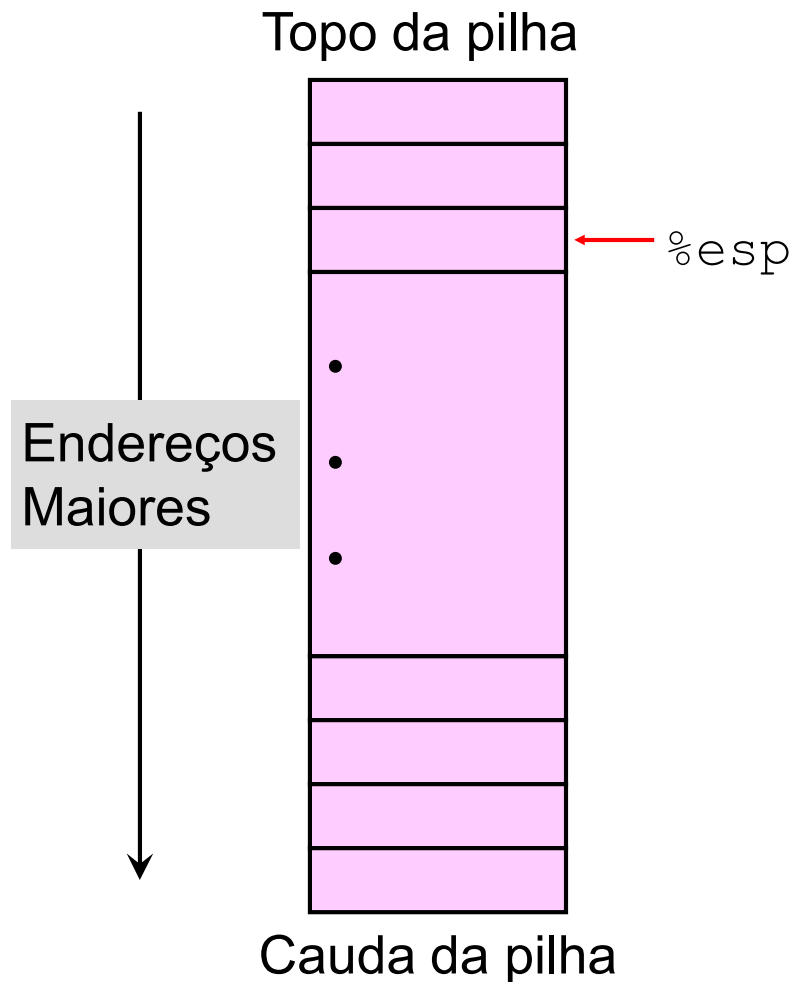


Jump When Greater



- Referidas genericamente como “jXX”
- Códigos variam apenas no “function code”
- Decisão baseada nos valores dos códigos de condição
- Endereçamento absoluto

Y86: Pilha



- Região da memória com dados do programa
- Usada no Y86 (e IA32) para suportar invocação de procedimentos
- Topo da pilha : `%esp`
- Pilha cresce para endereços menores
 - Topo no endereço menor
 - `Push` – primeiro subtrair 4 ao `%esp`
 - `Pop` – adicionar 4 ao `%esp` após leitura

Y86: Transferência de Dados (pilha)

`pushl rA`

a	0	rA	8
---	---	----	---

- Decrementar `%esp` por 4
- Armazenar palavra de `rA` na memória apontada por `%esp`

`popl rA`

b	0	rA	8
---	---	----	---

- Ler palavra da memória apontada por `%esp`
- Guardar em `rA`
- Incrementar `%esp` por 4

Y86: Procedimentos

`call Dest`

8 0 Dest

- Endereço da próxima instrução para a pilha (`push`)
- Iniciar execução a partir do endereço Dest (escrita no PC)

`ret`

9 0

- Ler valor do topo da pilha (`pop`)
- Iniciar execução a partir deste endereço (escrita no PC)

Y86: Instruções Adicionais



- *No Operation*: não altera o estado, excepto para o PC



- Pára a execução de instruções
- IA32 tem uma instrução semelhante, mas não pode ser executada em modo utilizador

Y86: Estrutura dos Programas

```
.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 t com o valor
                          # inicial 10

main:
    ...                  # instruções
    halt

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

- Programas começam no endereço 0
- Tem que se inicializar a pilha
 - Não sobrepor ao código!
- Inicializar dados

Y86: Assembler

```
unix> yas eg.ys
```

- Gera “código objecto” ficheiro **ASCII** `eg.yo`
 - Idêntico ao *output* de um *disassembler*

```
0x000: 308400010000 | irmovl Stack,%esp      # Set up stack
0x006: 2045         | rrmovl %esp,%ebp      # Set up frame
0x008: 308218000000 | irmovl List,%edx      #
0x00e: a028         | pushl %edx            # Push argument
0x010: 802800000000 | call len2             # Call Function
0x015: 10           | halt                  # Halt
0x018:              | .align 4
0x018:              | List:                 # List of elements
0x018: b3130000      | .long 5043
0x01c: ed170000      | .long 6125
0x020: e31c0000      | .long 7395
0x024: 00000000      | .long 0
```