Y86:
Arquitetura do conjunto de instruções (ISA)

Arquitetura de Computadores

Lic. em Engenharia Informática

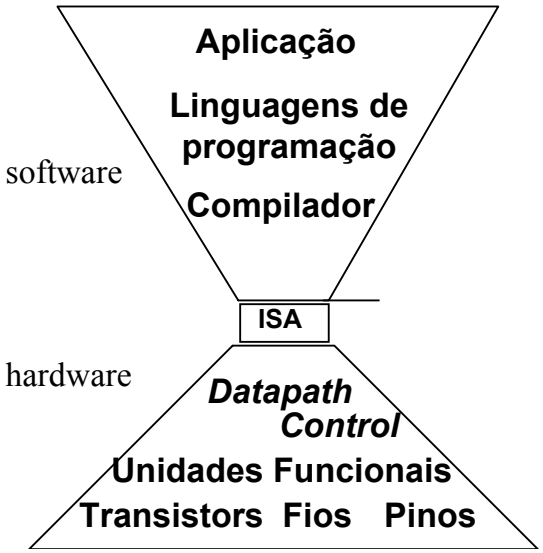
João Luís Sobral

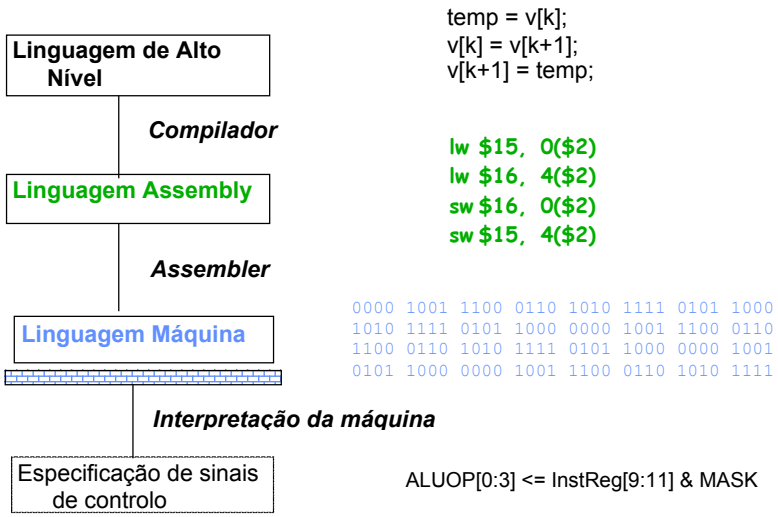
Y86: Arquitectura

Conteúdos	**3 – Organização do Processador**
	3.1 – Conceitos Fundamentais
Resultados de Aprendizagem	R3.1 – Analisar e descrever organizações sequenciais de processadores elementares

Instruction Set Architecture (ISA)

Parte do processador visível ao programador/compilador

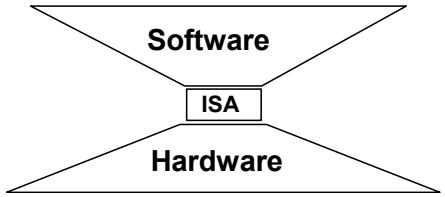


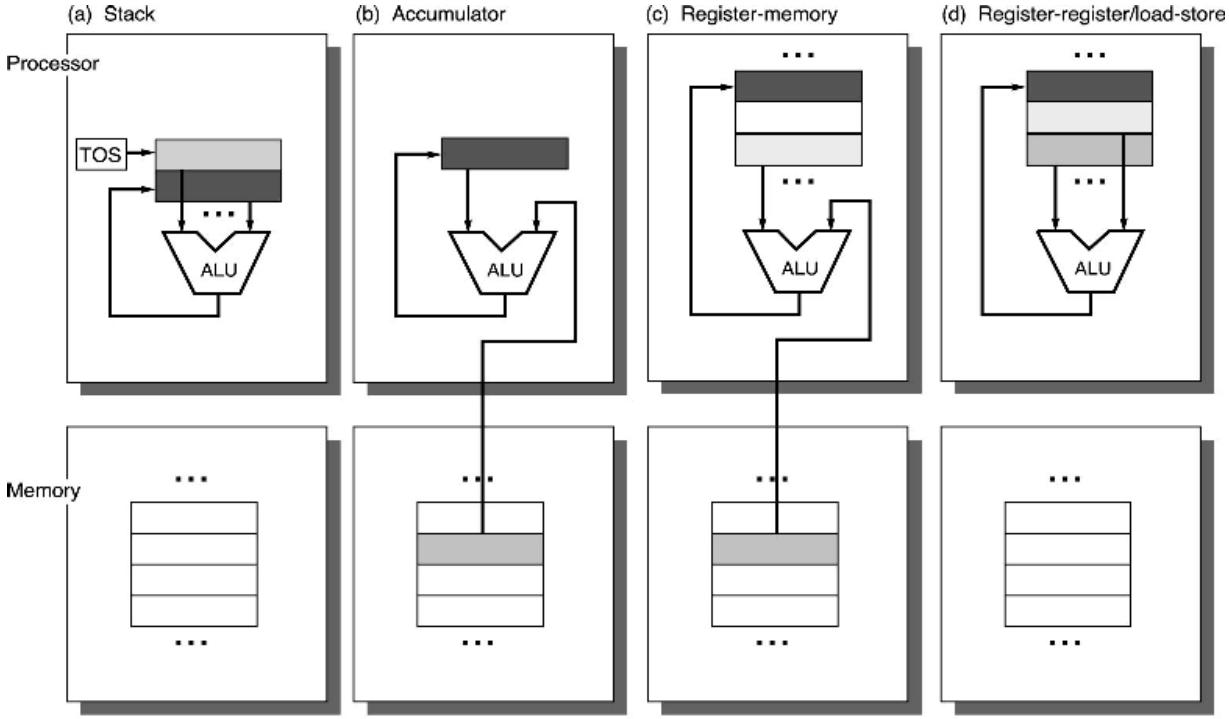


- Serve de Interface entre o software e o hardware
  - Um conjunto de instruções possibilita a separação entre as ferramentas de desenvolvimento (compiladores) e a implementação desse conjunto de instruções (processadores)
- Neste módulo estudaremos a implementação de um conjunto de instruções designado por Y86 (inspirado no IA32 da Intel, mas bastante mais simples)

Instruction Set Architecture (ISA)

Tipos de ISA





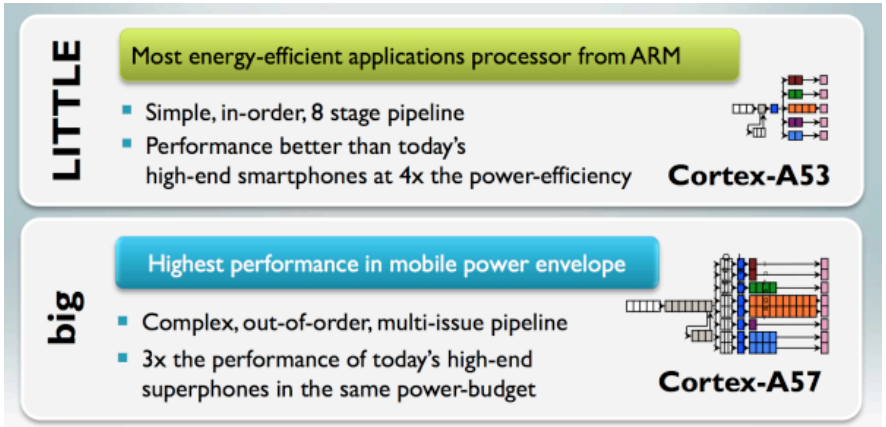
C= A + B

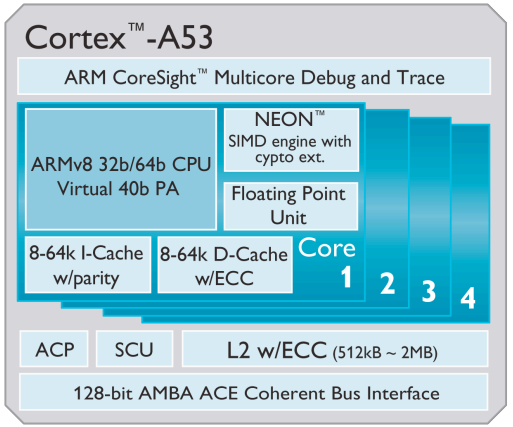
Stack	Acumulador	Reg-Mem	*Load-store*
Push A	Load A	Load R1, A	Load R1, A
Push B	Add B	Add R1, B	Load R2, A
Add	Store C	Store R1, C	Add R3, R1, R2
Pop C			Store R3, A

Instruction Set Architecture (ISA)

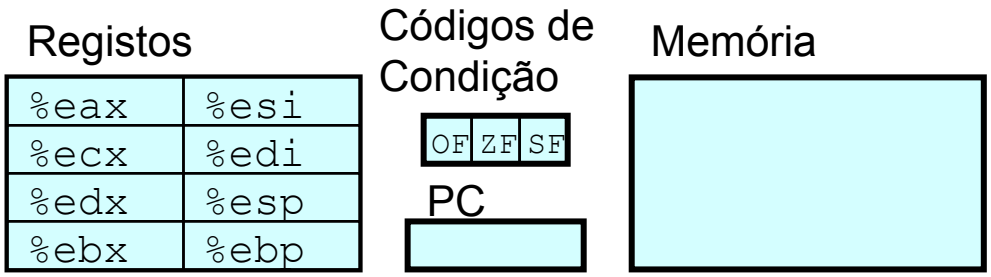
Exemplo de um ISA (recente): ARMv8-A

- Versão mais recente do ISA da ARM (Out/2011, sucessor do ARMv7 32bits)
  - 31 registos genéricos de 64 bits (+PC e SP)
  - Formato de instruções de 32 bits / arquitetura tipo *Load/Store*
  - Instruções para suporte a operandos de 64 bits
  - Endereços de 64 bits
- Implementações do ISA (Out/2012, vendidas pela ARM):





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*

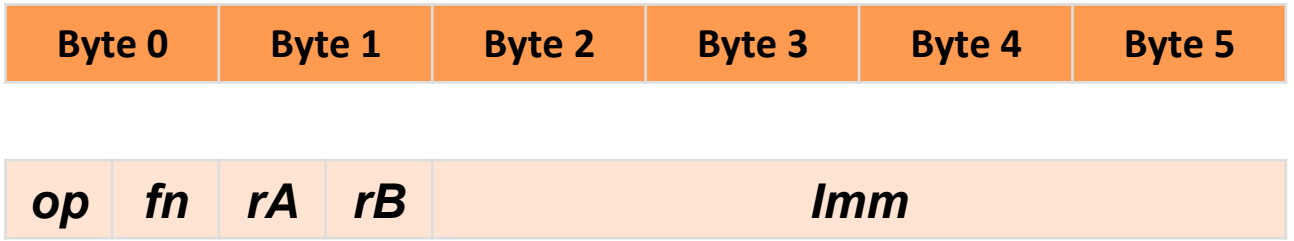
Y86 : Instruction Set Architecture (ISA)

Instrução	Octetos	Comentários
`nop`	1	Nenhuma operação
`halt`	1	Parar execução
`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**
`mrmovl 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
`pushl rA`	2	Incrementa %esp e lê o topo da pilha para **rA**

Jogo de Instruções do Y86

Y86 : Instruction Set Architecture (ISA)

Formato das instruções:



- ***op*** – código de operação (*opcode*): 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:

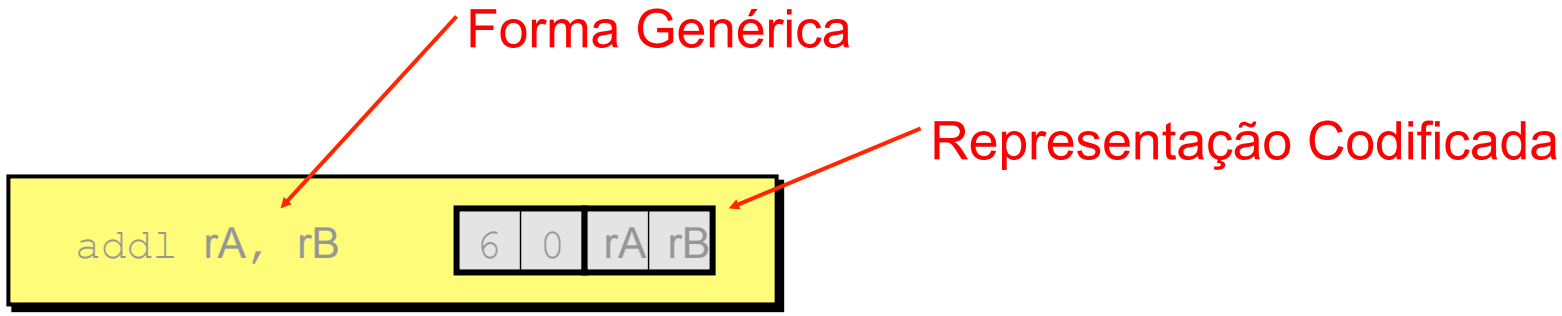
- apenas *op* e *fn* estão sempre presentes pois identificam a instrução

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***)
- Os dois primeiros caracteres de instruções de "mov" indicam o modo de endereçamento utilizado, respectivamente para a fonte e destino
  - **Ex. irmov $1,eax**

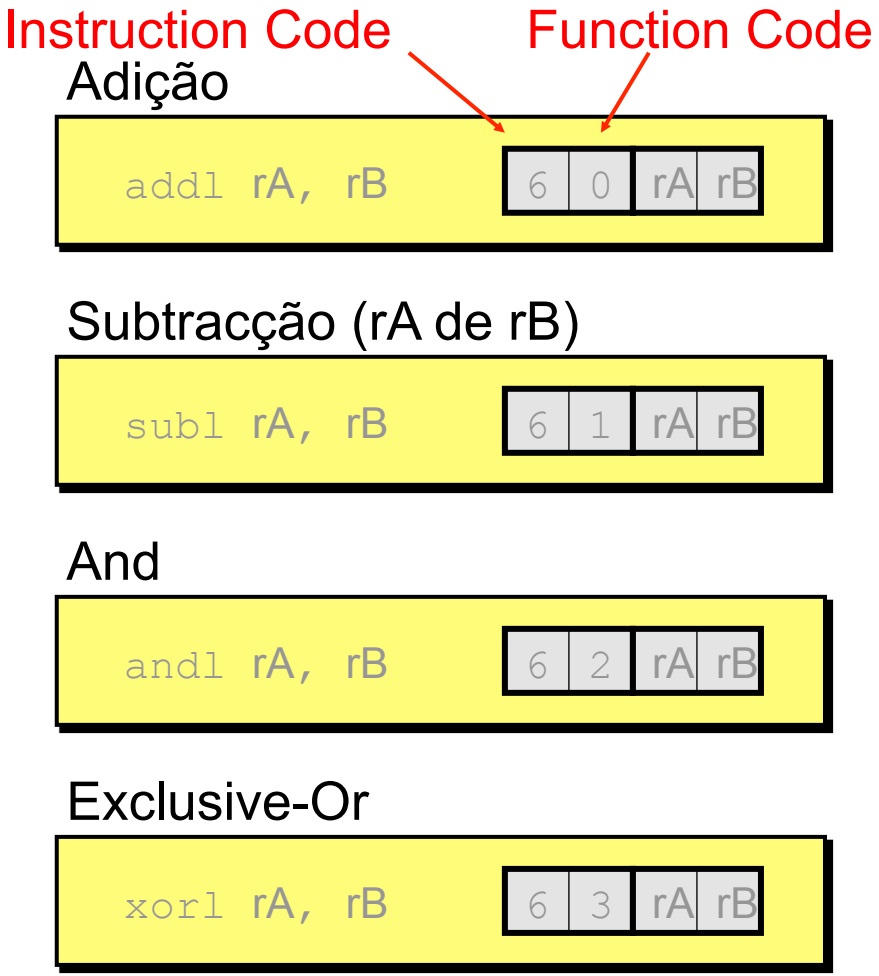
Y86 : Exemplo de Instrução

- Adição



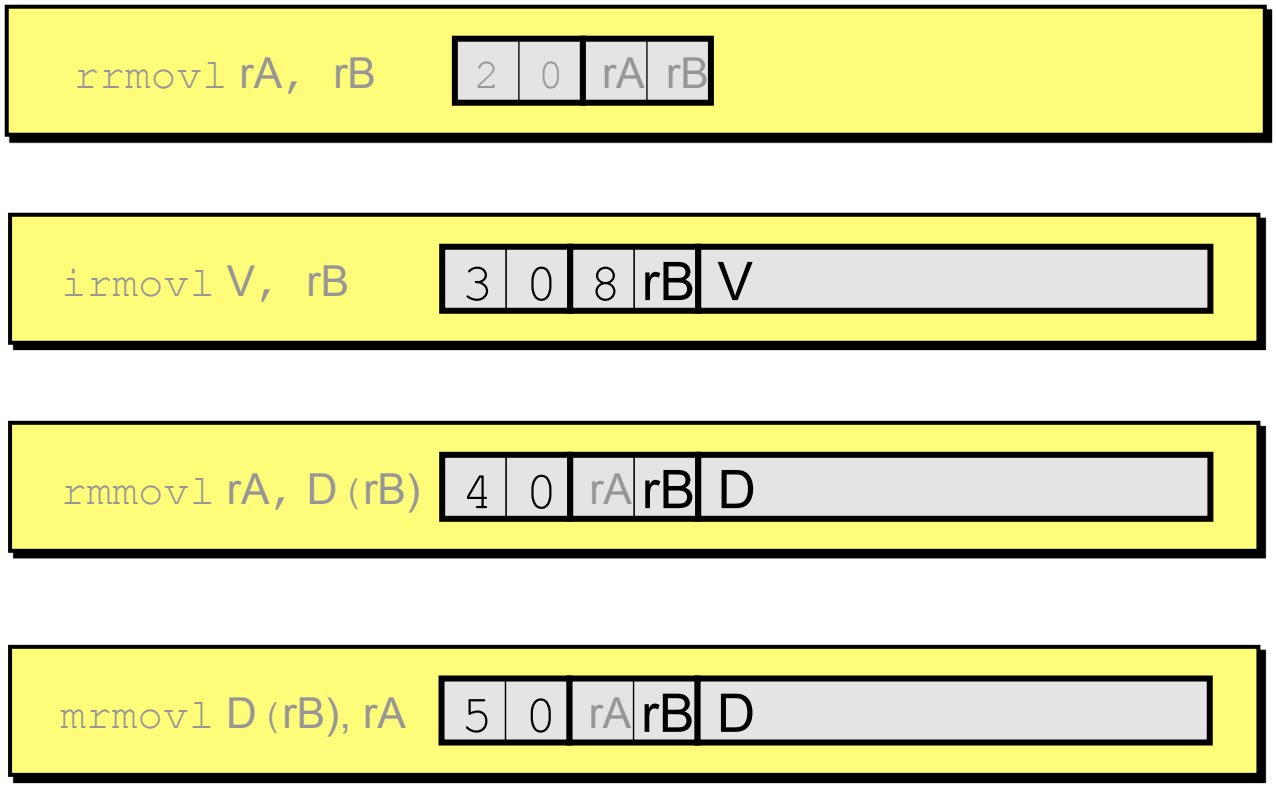
  - Adicionar valor no registo rA ao valor no registo rB
    - Guardar resultado no registo rB
    - Operações Aritméticas ou Lógicas só sobre operandos em registos
  - Códigos de condição dependem do resultado
  - e.g., **`addl %eax,%esi`** Representação Máquina: **`60 06`**

Y86: Operações Lógicas e Aritméticas



- 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



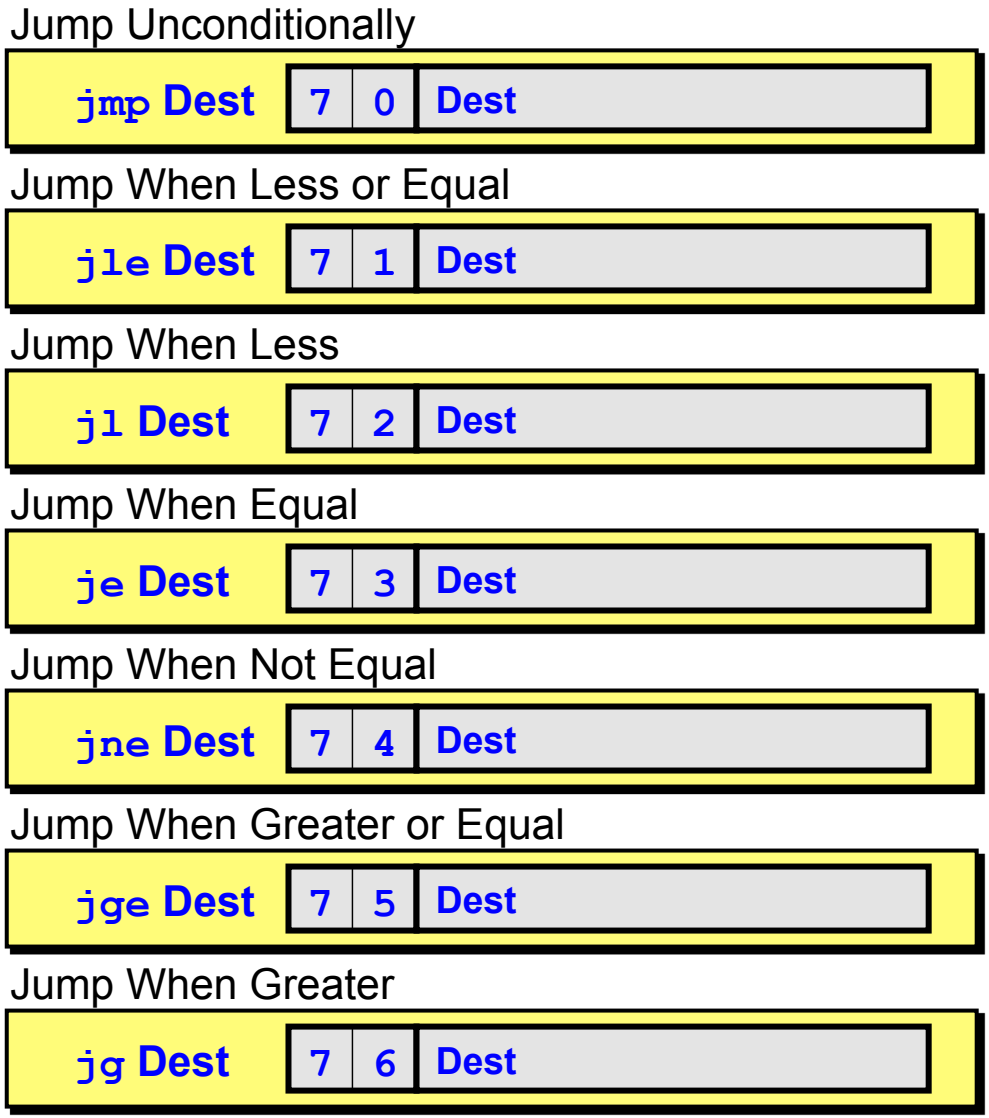
- Semelhante à instrução IA32 `movl`
- Modo de endereçamento mais simples: Registo+Deslocamento

Y86: Transferência de Dados (exemplos)

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

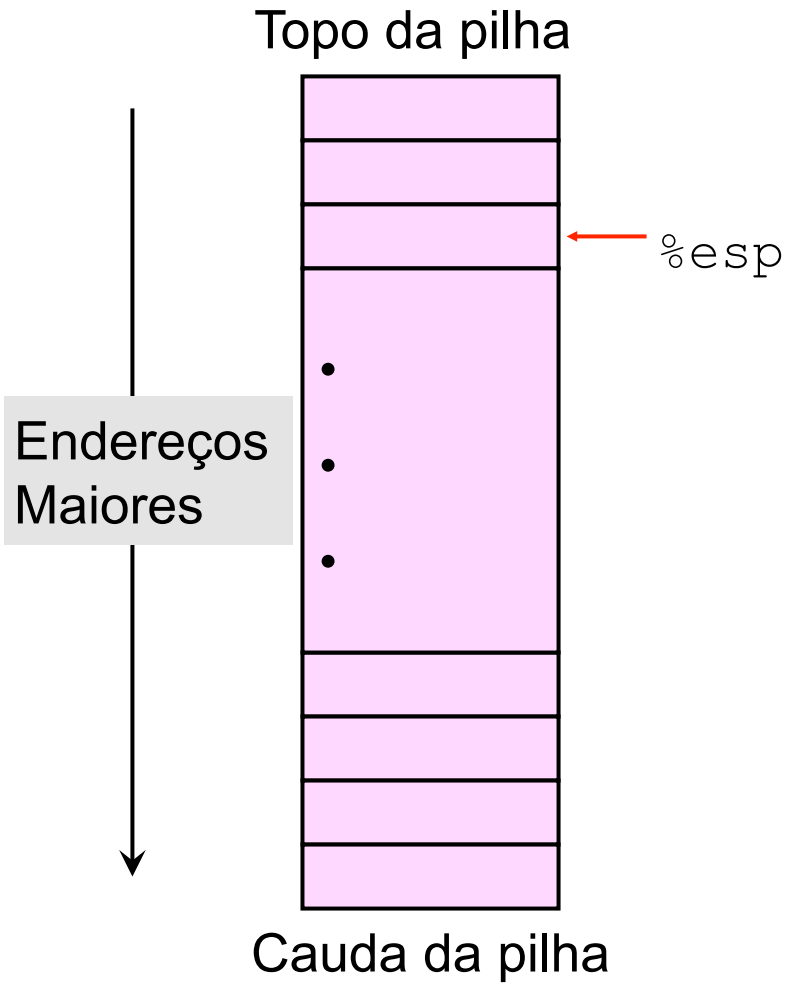
`movl $0xabcd, (%eax)`	—
`movl %eax, 12(%eax,%edx)`	—
`movl (%ebp,%eax,4),%ecx`	—

Y86: Controlo de Fluxo (saltos)



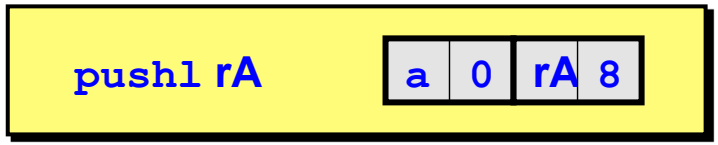
- Referidas genericamente como "`jXX`"
- Códigos variam apenas no "function code"
- Decisão baseada nos valores dos códigos de condição
- Endereçamento absoluto

Y86: Pilha

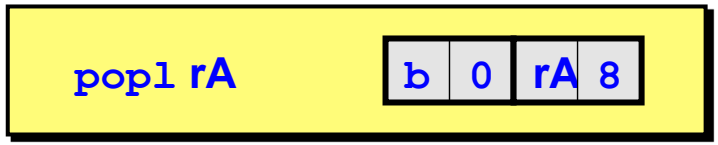


- Região da memória com dados do programa
- Usada no Y86 (e IA32) para suportar invocação de procedimentos
- Topo da pilha : `%esp`
- Pilha cresce para endereços menores
  - Topo no endereço menor
  - `Push` – primeiro subtrair 4 ao *%esp*
  - `Pop` – adicionar 4 ao *%esp* após leitura

Y86: Transferência de Dados (pilha)

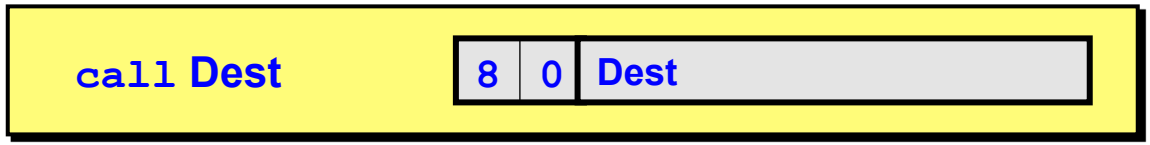


- Decrementar `%esp` por 4
- Armazenar palavra de rA na memória apontada por `%esp`

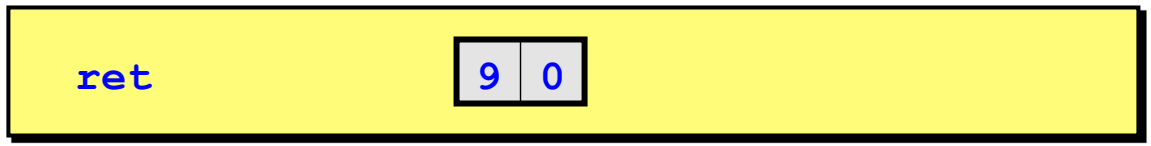


- Ler palavra da memória apontada por `%esp`
- Guardar em rA
- Incrementar `%esp` por 4

Y86: Procedimentos



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



- 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: 8028000000   | 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
```