Universidade do Minho

Módulo 6

Organização sequencial Y86

Objectivos:

No final desta sessão os alunos deverão ser capazes de:

i) descrever os vários formatos de instrução do Y86;
ii) descrever a organização sequencial do Y86;
iii) instanciar os valores dos sinais relevantes para a execução de cada instrução Y86.

Exercícios:

Considere o seguinte programa *assembly* Y86 que define e invoca uma função de soma de dois inteiros.

```
# Execução começa no endereço 0
  .pos 0
init:
  irmovl stack, %esp          # ***
  irmovl stack, %ebp
  jmp main                    # ***

main:
  irmovl $4, %eax
  pushl %eax
  irmovl data, %ebx           # ***
  mrmovl $4(%ebx), %eax
  pushl %eax                  # ***
  call soma                   # ***
  halt

soma:
  pushl %ebp
  rrmovl %esp, %ebp           # ***
  mrmovl $8(%ebp), %eax
  mrmovl $12(%ebp), %ebx      # ***
  addl %ebx, %eax             # ***
  rrmovl %ebp, %esp
  popl %ebp                   # ***
  ret                         # ***

.pos 0x100
data: .long 10
      .long 24

.pos 0x200
stack: # Inicio da pilha
```

Para cada instrução assinalada com ***, apresente:

a) A codificação da instrução, a sua disposição na memória, indicando o endereço onde está armazenada (lembre-se de que se trata de uma arquitectura *little endian*);
    a. Gere o ficheiro objeto (soma.yo) com o **yas** para verificar as suas respostas.
    b. Pode executar o programa usando o simulador "**ssim –g soma.yo**" (nota: deve copiar a script `seq.tcl` de `/usr/local/bin` para a sua diretoria de trabalho).
b) Uma tabela com os valores dos sinais de controlo da organização sequencial. Apresente os sinais diferenciados pelo estágio em que são relevantes/gerados. Para cada tipo de instrução que surja pela primeira vez apresente os seus valores genéricos e os valores específicos para este programa.

O exemplo seguinte correspondente às 2 primeiras instruções assinaladas (em anexo encontra as tabelas com os formatos de instruções do Y86 e dos sinais do *datapath* Y86 SEQ)

irmovl stack, %esp

Codificação:

30	84	0000 0200

Disposição na memória:

Endereço	Conteúdos
0x0	30 84 00 02 00 00

Tabela com os sinais de controlo relevantes organizados pelas fases de execução (estágios)

Estágio	Genérico	Específico
	irmovl V, rB	irmovl stack, %esp
Extração	icode:ifun = M1[PC] rA:rB = M1[PC+1] valC = M4[PC+2] valP=PC+6	Icode:ifun = M1[0]= 3:0 rA:rB = M1[1] = 8:4 valC = M4[2] = 0x200 valP=0+6=6
Descodificação		
Execução	valE = valC + 0	valE = 0x200
Memória		
Atualização	R[rB] = valE	%esp = 0x200
PC	PC = valP	PC = 6

jmp main

Codificação:

70	0000 0011

Disposição na memória:

Endereço	Conteúdos
0xc	70 11 00 00 00

Fases de execução

Estágio	Genérico	Específico
	jmp dest	jmp main
Extração	icode:ifun = M1[PC] valC = M4[PC+2] valP=PC+5	icode:ifun = M1[0xc]= 7:0 valC = M4[0xd] = 0x11 valP = 0xc + 5 = 0x11
Descodificação		
Execução		
Memória		
Atualização		
PC	PC = valC	PC = 0x11

A arquitectura **Y86**

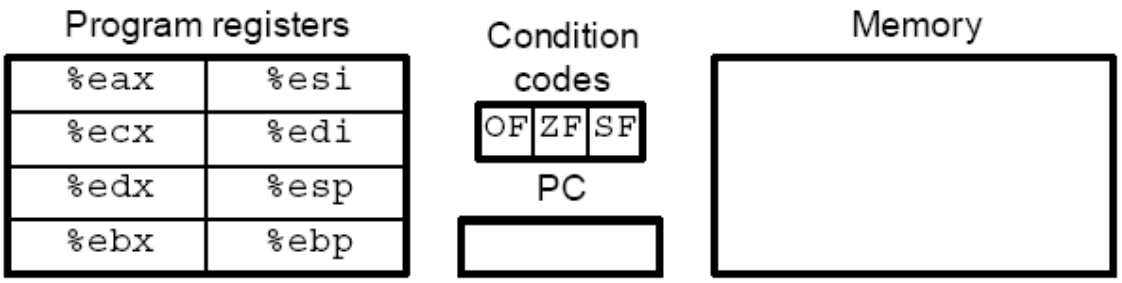

Figura 1 - Estado visível ao programador

8 registos de 32 bits. O registo `%esp` é utilizado implicitamente pelas instruções `pushl`, `popl`, `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).

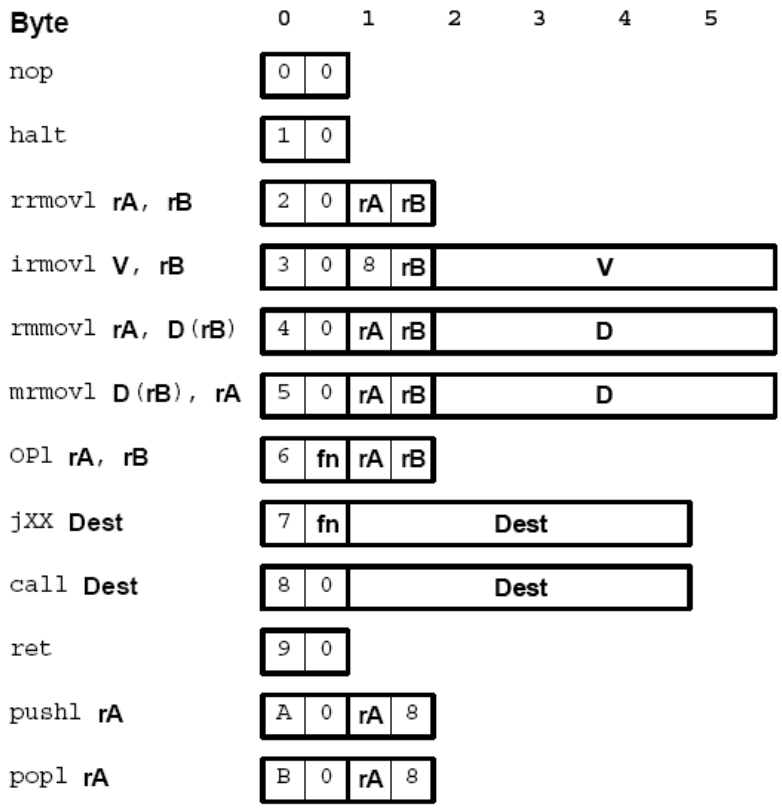

Figura 2 - Instruções e respectivos formatos (valores em hexadecimal)

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).

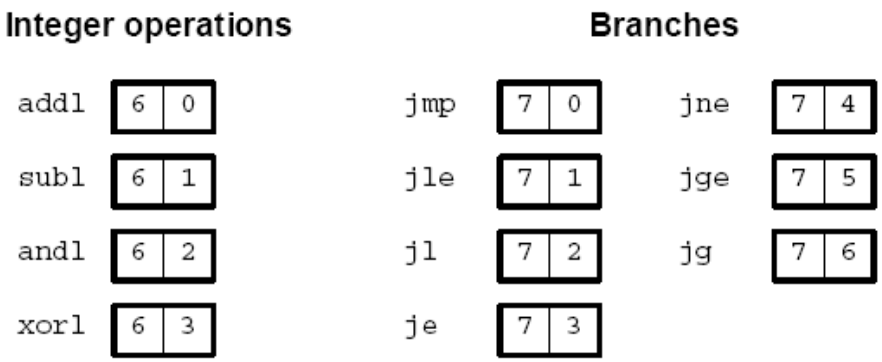

Figura 3 – Instruções lógicas/aritméticas e saltos

No caso de instruções que referem registos explicitamente o segundo código contem 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 `popl`, `pushl`, `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`, `rmmovl`) 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.

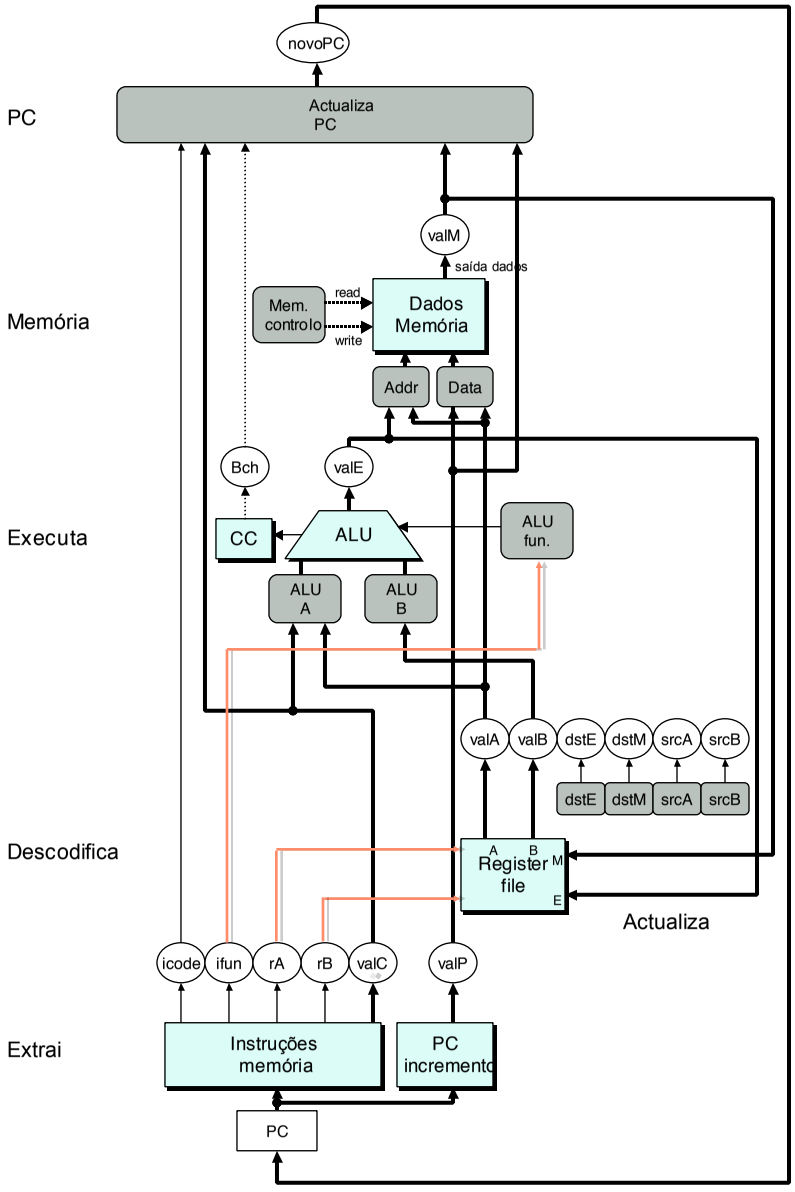

Figure 21: **Hardware structure of SEQ, a sequential implementation.** Some of the control signals, as well as the register and control word connections, are not shown.

3. *Fluxograma arquitectura SEQ Y86*