

## Síntese da Unidade de Controle (2): **MICROPROGRAMAÇÃO**

ABF - AC1\_Multi-cycle\_CPU

26

## Microprogramação

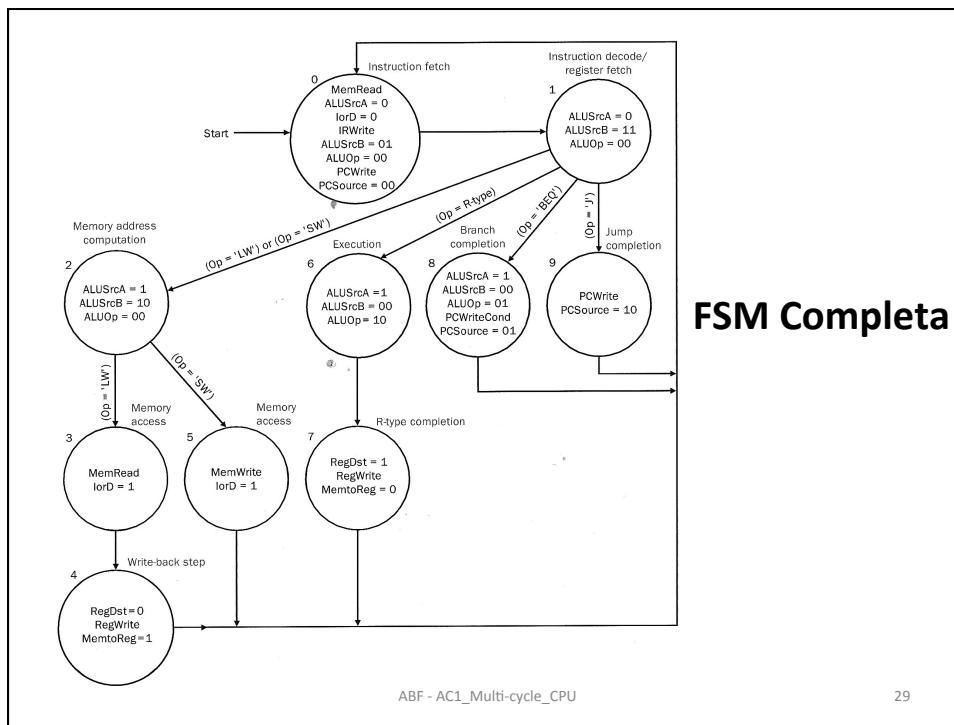
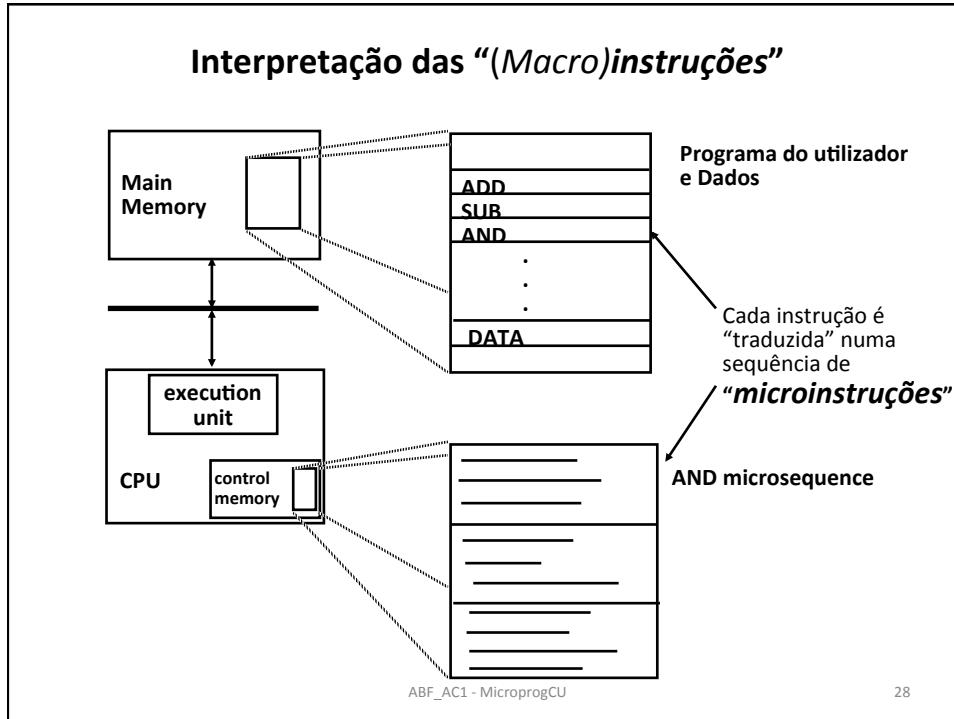
- **M.V.Wilkes, J.B.Stringer (1953), “Microprogramming and the design of the control circuits in an electronic digital computer”:**

A operação especificada por uma instrução pode ser decomposta numa sequência de operações mais elementares; ... Essas operações elementares serão designadas micro-instruções. As operações básicas da máquina, como a adição, subtração, multiplicação, etc., são pensadas como um *micro-programa* de micro-operações, cada micro-operação sendo especificada por uma micro-instrução.

O processo de escrever um microprograma para uma instrução é muito semelhante ao de escrever um programa. ([tradução adaptada](#))

ABF\_AC1 - MicropogCU

27



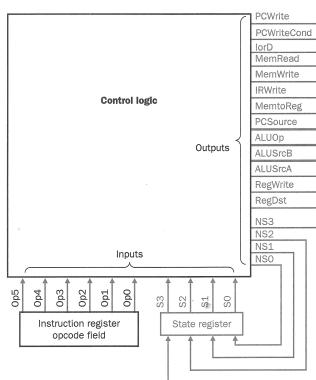
## Unidade de Control – Next State Function

Current state S[3-0]	Op [5-0]					
	000000 (R-format)	000010 (jmp)	000100 (beq)	100011 (lw)	101011 (sw)	Any other value
0000	0001	0001	0001	0001	0001	0001
0001	0110	1001	1000	0010	0010	illegal
0010	XXXX	XXXX	XXXX	0011	0101	illegal
0011	0100	0100	0100	0100	0100	illegal
0100	0000	0000	0000	0000	0000	illegal
0101	0000	0000	0000	0000	0000	illegal
0110	0111	0111	0111	0111	0111	illegal
0111	0000	0000	0000	0000	0000	illegal
1000	0000	0000	0000	0000	0000	illegal
1001	0000	0000	0000	0000	0000	illegal

ABF - AC1\_Multi-cycle\_CPU

30

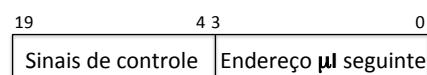
## Unidade de Controle



**Microprogramação:** a execução de cada instrução traduz-se na execução de uma sequência de microinstruções

**State Register: Microprogram Counter  $\mu$ PC**

**Formato das Microinstruções:**



**Endereço  $\mu$ I seguinte:**

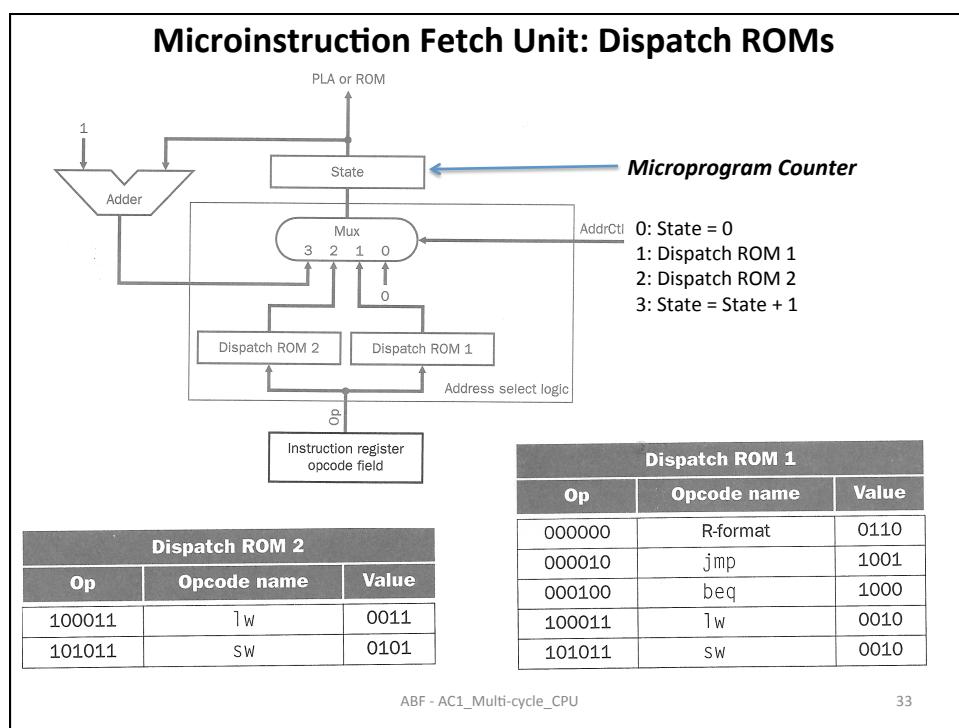
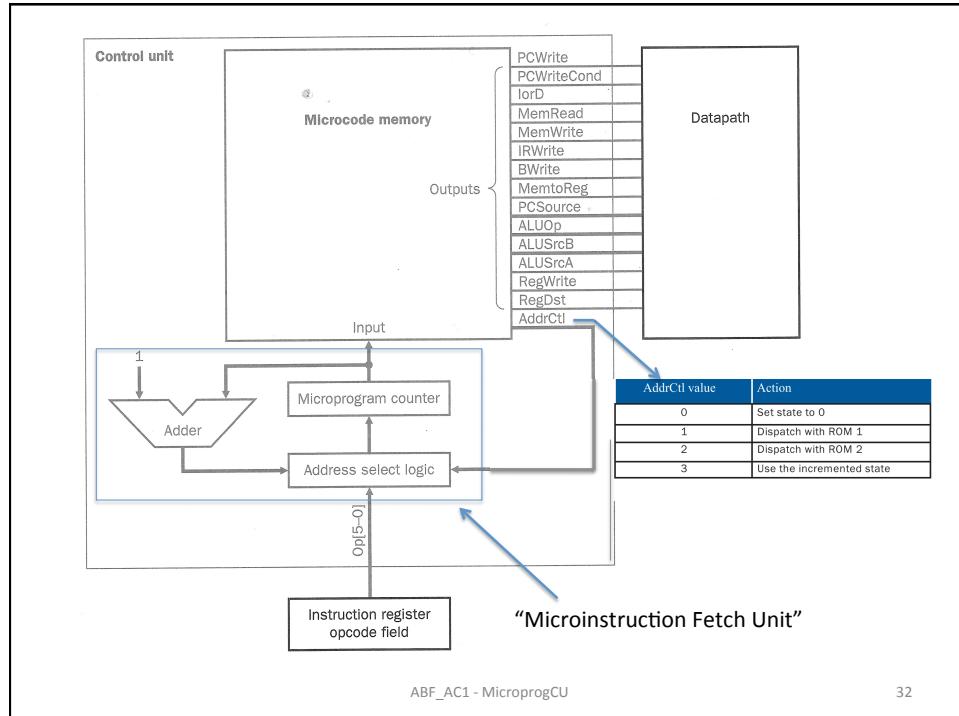
- **( $\mu$ PC + 1)** para os estados em que o sucessor é o estado com o numero seguinte (estados 0, 3 e 6)
- **0** para os estados terminais (estados 4, 5, 7, 8 e 9)
- **Salto determinado pelo OpCode** nos estados com mais do que um sucessor (estados 1 e 2)

### Solução:

em vez do campo “Endereço  $\mu$ I seguinte”, incluir um campo “**Address Control**” que controla uma “microinstruction fetch unit”

ABF\_AC1 - MicroprogCU

31



## AddrCtl

State number	Address-control action	Value of AddrCtl
0	Use incremented state	3
1	Use dispatch ROM 1	1
2	Use dispatch ROM 2	2
3	Use incremented state	3
4	Replace state number by 0	0
5	Replace state number by 0	0
6	Use incremented state	3
7	Replace state number by 0	0
8	Replace state number by 0	0
9	Replace state number by 0	0

## Conceção do Microinstruction Set

- 1) Ponto de partida: lista dos sinais de control
- 2) Agrupar os sinais, de um modo lógico em "**microinstruction fields**"
- 3) Colocar os campos da microinstrução numa ordem lógica (e.g., ALU operation & ALU operands primeiro e microinstruction sequencing por último)
- 4) Criar uma representação simbólica ("**microassembly**") para o formato das microinstruções, dando nome aos valores dos campos e que valores dão aos sinais de controle
- 5) Minimizar o comprimento da microinstrução, codificando as operações que nunca ocorrem simultaneamente

## Formato das Microinstruções

Field name	Function of field
ALU control	Specify the operation being done by the ALU during this clock; the result is always written in ALUOut.
SRC1	Specify the source for the first ALU operand.
SRC2	Specify the source for the second ALU operand.
Register control	Specify read or write for the register file, and the source of the value for a write.
Memory	Specify read or write, and the source for the memory. For a read, specify the destination register.
PCWrite control	Specify the writing of the PC.
Sequencing	Specify how to choose the next microinstruction to be executed.

## Os diversos campos das microinstruções

Field name	Values for field	Function of field with specific value
Label	Any string	Used to specify labels to control microcode sequencing. Labels that end in a 1 or 2 are used for dispatching with a jump table that is indexed based on the opcode. Other labels are used as direct targets in the microinstruction sequencing. Labels do not generate control signals directly but are used to define the contents of dispatch tables and generate control for the Sequencing field.
ALU control	Add	Cause the ALU to add.
	Subt	Cause the ALU to subtract; this implements the compare for branches.
	Func code	Use the instruction's funct field to determine ALU control.
SRC1	PC	Use the PC as the first ALU input.
	A	Register A is the first ALU input.
SRC2	B	Register B is the second ALU input.
	4	Use 4 for the second ALU input.
	Extend	Use output of the sign extension unit as the second ALU input.
	Extshft	Use the output of the shift-by-two unit as the second ALU input.
	Read	Read two registers using the rs and rt fields of the IR as the register numbers, putting the data into registers A and B.
Register control	Write ALU	Write the register file using the rd field of the IR as the register number and the contents of ALUOut as the data.
	Write MDR	Write the register file using the rt field of the IR as the register number and the contents of the MDR as the data.
	Read PC	Read memory using the PC as address; write result into IR (and the MDR).
Memory	Read ALU	Read memory using ALUOut as address; write result into MDR.
	Write ALU	Write memory using the ALUOut as address; contents of B as the data.
PCWrite control	ALU	Write the output of the ALU into the PC.
	ALUOut-cond	If the Zero output of the ALU is active, write the PC with the contents of the register ALUOut.
	Jump.address	Write the PC with the jump address from the instruction.
Sequencing	Seq	Choose the next microinstruction sequentially.
	Fetch	Go to the first microinstruction to begin a new instruction.
	Dispatch i	Dispatch using the ROM specified by i(1 or 2).

## Microinstruções Fetch e Decode

Label	ALU control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
Fetch	Add	PC	4		Read PC	ALU	Seq
	Add	PC	Extshift	Read			Dispatch 1

### 1ª microinstrução: Fetch

Fields	Effect
ALU control, SRC1, SRC2	Compute PC + 4. (The value is also written into ALUOut, though it will never be read from there.)
Memory	Fetch instruction into IR.
PCWrite control	Causes the output of the ALU to be written into the PC.
Sequencing	Go to the next microinstruction.

### 2ª microinstrução: Decode, Read Registers

Fields	Effect
ALU control, SRC1, SRC2	Store PC + sign extension (IR[15:0]) << 2 into ALUOut.
Register control	Use the rs and rt fields to read the registers placing the data in A and B.
Sequencing	Use dispatch table 1 to choose the next microinstruction address.

ABF\_AC1 - MicroprogCU

38

## Instruções com referência à memória

Label	ALU control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
Mem1	Add	A	Extend				Dispatch 2
LW2					Read ALU		Seq
				Write MDR			Fetch
SW2					Write ALU		Fetch

### Mem1:

Fields	Effect
ALU control, SRC1, SRC2	Compute the memory address: Register (rs) + sign-extend (IR[15:0]), writing the result into ALUOut.
Sequencing	Use the second dispatch table to jump to the microinstruction labeled either LW2 or SW2.

### LW2:

Fields	Effect
Memory	Read memory using the ALUOut as the address and writing the data into the MDR.
Sequencing	Go to the next microinstruction.

### SW2:

Fields	Effect
Memory	Write memory using contents of ALUOut as the address and the contents of B as the value.
Sequencing	Go to the microinstruction labeled Fetch.

39

## Instruções Tipo-R

Label	ALU control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
Rformat1	Func code	A	B				Seq
				Write ALU			Fetch

### Rformat1:

Fields	Effect
ALU control, SRC1, SRC2	The ALU operates on the contents of the A and B registers, using the function field to specify the ALU operation.
Sequencing	Go to the next microinstruction.

Fields	Effect
Register control	The value in ALUOut is written into the register file entry specified by the rd field.
Sequencing	Go to the microinstruction labeled Fetch.

## Branch

Label	ALU control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
BEQ1	Subt	A	B			ALUOut-cond	Fetch

### BEQ1:

Fields	Effect
ALU control, SRC1, SRC2	The ALU subtracts the operands in A and B to generate the Zero output.
PCWrite control	Causes the PC to be written using the value already in ALUOut, if the Zero output of the ALU is true.
Sequencing	Go to the microinstruction labeled Fetch.

# Jump

Label	ALU control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
JUMP1						Jump address	Fetch

## JUMP1:

Fields	Effect
PCWrite control	Causes the PC to be written using the jump target address.
Sequencing	Go to the microinstruction labeled Fetch.

# Microprogramas

Label	ALU control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
Fetch	Add	PC	4		Read PC	ALU	Seq
	Add	PC	Extshft	Read			Dispatch 1
Mem1	Add	A	Extend				Dispatch 2
LW2					Read ALU		Seq
					Write MDR		Fetch
SW2					Write ALU		Fetch
Rformat1	Func code	A	B				Seq
					Write ALU		Fetch
BEQ1	Subt	A	B			ALUOut-cond	Fetch
JUMP1						Jump address	Fetch

## Horizontal vs. Vertical Microprogramming

Unidades de controle microprogramadas:

- **Microprogramação horizontal** (1 bit por cada ponto do datapath a controlar)
- **Microprogramação vertical** (os diferentes campos da microinstrução têm de ser descodificados para gerar os sinais de control)

<u>Horizontal</u>	<u>Vertical</u>
+ mais control sobre o potencial paralelismo das operações no datapath	+ mais fácil de (micro)programar (não muito diferente de programar um processador RISC em assembly)
- consome muita memória (control store)	- um nível extra de descodificação pode tornar o CPU mais lento

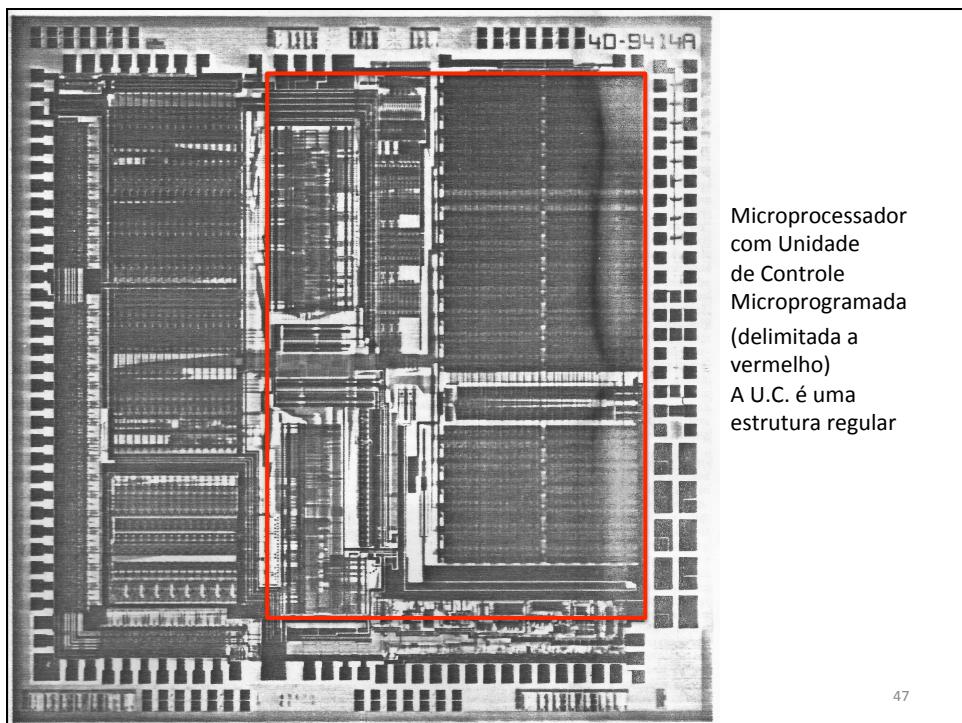
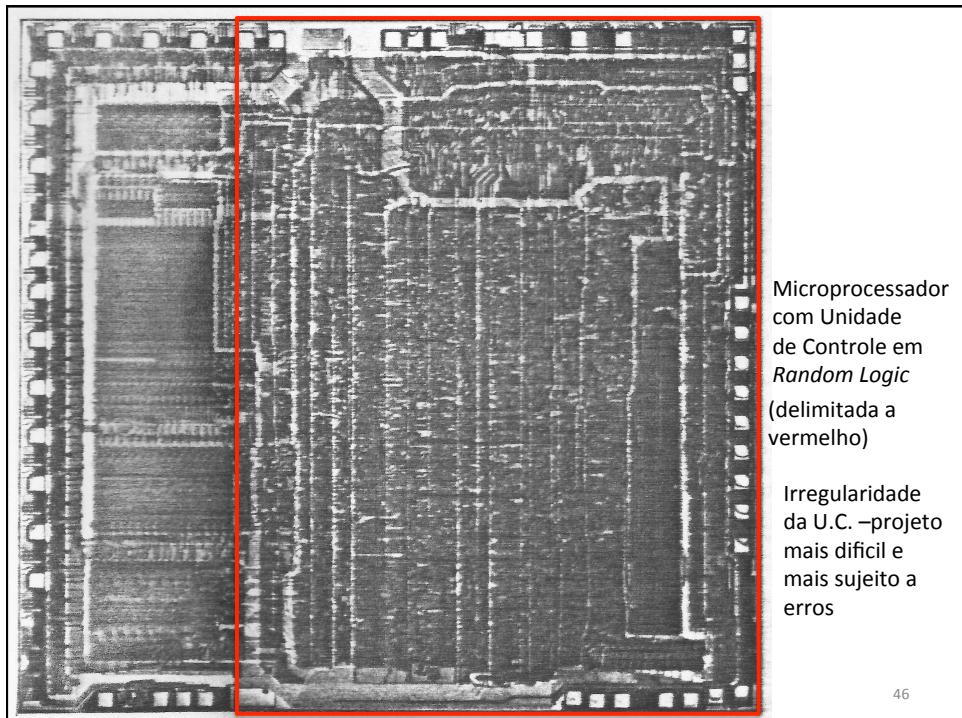
## Microprogramação

- Vantagens:

- Permite a implementação de um mesmo reportório de instruções (arquitetura) em recursos de hardware muito diferentes (p.ex. IBM S/360 – arquitetura de 32-bits implementada, simultaneamente, em processadores com níveis de desempenho e custos muito diversos)
- Facilidade de projeto
- Flexibilidade – p.ex. fácil acrescentar instruções ao reportório
- Possível fazer mudanças em fases avançadas do projeto
- Capacidade de implementar instruction sets muito complexos (aumentando a memória de control)
- Generalidade - permite implementar diferentes instruction sets na mesma máquina

- Desvantagens:

- Processador mais lento



## Processadores Microprogramáveis

- Usar RAM em vez de ROM para a memória de controle
  - Microprogramas carregados na memória de controle quando o computador é ligado.
  - Conteúdo da memória de controle pode ser atualizado (*microcode updates*)
- IBM S/360 Instruction Set Architecture (ISA): o mesmo reportório de instruções para máquinas muito diferentes em preço e desempenho – datapaths de 8-bit a 32-bit consoante o modelo – o mesmo instruction set executado por “microarquiteturas” muito diferentes
- **Emulação:** executar diferentes Instruction Sets (Arquiteturas) num mesmo hardware
  - Usado pela IBM para executar código máquina de arquiteturas anteriores em computadores S/360, garantindo “software compatibility”
  - A emulação teve um papel importante na aceitação da nova (e incompatível com as anteriores) arquitetura da IBM.

ABF\_AC1 - MicroprogCU

48

## Sumário: Microprogramação e arquiteturas RISC

- Se instruções simples (*RISC*) podem executar com uma elevada frequência de relógio...
- Se é possível escrever compiladores para gerarem microinstruções...
- Se a maioria dos programas usa instruções e modos de endereçagem simples...
- Se o microcódigo residir em RAM e não em ROM para poder corrigir erros...
- Então porque não dispensar a interpretação das instruções por um microprograma e compilar diretamente para a linguagem máquina?

### ➤ Arquiteturas RISC

ABF\_AC1 - MicroprogCU

49

## Bibliografia

**MICROPROGRAMMING: SIMPLIFYING  
CONTROL DESIGN**  
P&H - SECTION 5.7 (disponível no MOODLE)