Universidade Federal do Rio Grande do Sul
Instituto de Informática

Organização de Computadores

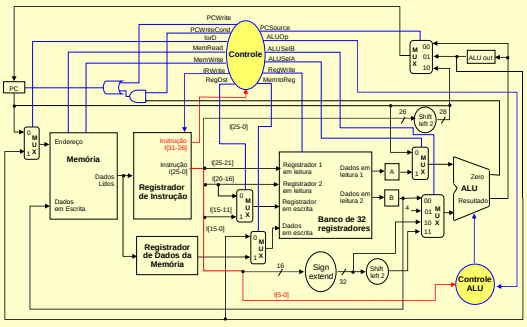
Aula 07

Bloco de controle multi-ciclo
Projeto com FSM

Bloco de controle – projeto com FSM

1. Sinais de controle
2. Máquina de estados finitos
   - Busca e decodificação de instruções
   - Instruções de referência à memória
   - Instruções de tipo R
   - Instrução de desvio condicional (branch)
   - Instrução de desvio incondicional (jump)
3. Implementando a FSM

1. Revisão



Microinstruções tipo R e Lw

• RInstr = Memoria[PC] • PC = PC + 4	• RInstr = Memoria[PC] • PC = PC + 4
• A = Reg[I[25-21]] • B = Reg[I[20-16]] • ALUout = PC + SignExtend (I[15-0]) << 2	• A = Reg[I[25-21]] • B = Reg[I[20-16]] • ALUout = PC + SignExtend (I[15-0]) << 2
• ALUout = A operacao B	• ALUout = A + SignExtend(I[15-0])
• Reg[I[15-11]] = ALUout	• MDR = Memoria [ALUout] • Reg [I[20-16]] = MDR

Microinstruções tipo Sw e Beq

- RInstr = Memoria[PC]
- PC = PC + 4

- A = Reg[I[25-21]]
- B = Reg[I[20-16]]
- ALUout = PC + SignExtend (I[15-0]) << 2

- ALUout = A + SignExtend(I[15-0])

- Memoria [ALUout] = B

- RInstr = Memoria[PC]
- PC = PC + 4

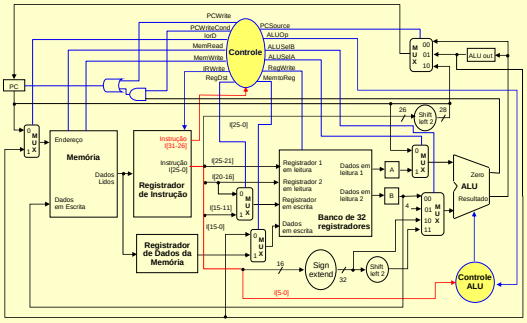
- A = Reg[I[25-21]]
- B = Reg[I[20-16]]
- ALUout = PC + SignExtend (I[15-0]) << 2

- se (A = = B)
- PC = ALUout

Sinais de controle

MemRead = 1	Ler conteúdo da memória
MemWrite = 1	Escrever conteúdo na memória
ALUSelA = 0	PC é primeiro operando para ALU
ALUSelA = 1	Registrador A é primeiro operando para ALU
ALUSelB = 00	Registrador B é segundo operando para ALU
ALUSelB = 01	Constante = 4 é segundo operando para ALU
ALUSelB = 10	Deslocamento (estendido p/ 32 bits) é segundo operando para ALU
ALUSelB = 11	Deslocamento (estendido p/ 32 bits e deslocado 2 bits p/ esq.) é 2º operando p/ ALU
RegDst = 0	Registrador a ser escrito é especificado pelo campo rt
RegDst = 1	Registrador a ser escrito é especificado pelo campo rd
RegWrite = 1	Escrever conteúdo em registrador
MemtoReg = 0	Valor a ser escrito em registrador vem do registrador da saída da ALU (ALUout)
MemtoReg = 1	Valor a ser escrito em registrador vem do registrador de dados da memória (MDR)
IorD = 0	Endereço em memória vem do PC
IorD = 1	Endereço em memória vem da ALU
IRWrite = 1	Armazenar instrução no IR

1. Sinais de controle



Sinais de controle para o PC

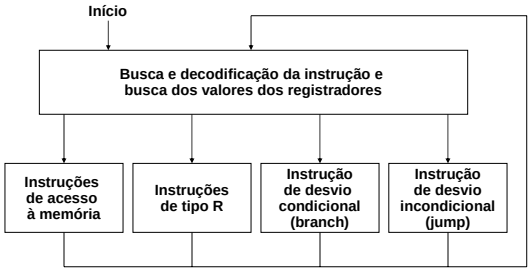
PCWrite = 1	PC é carregado; fonte é controlada por PCSource
PCWriteCond = 1	PC é carregado se saída Zero da ALU está ligada

PCSource	00	Saída da ALU é enviada para o PC
	01	Conteúdo do registrador ALUout é enviado para o PC
	10	Endereço de destino do Jump enviado para o PC

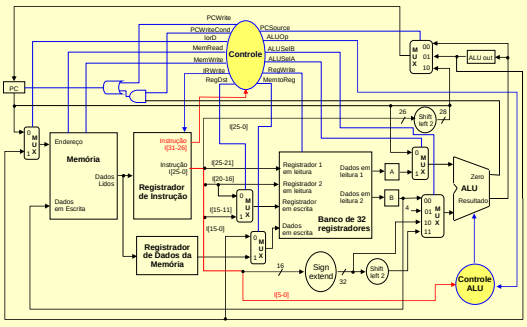
Na seqüência de passos:

- **Existem ações que são sempre feitas?**
- **A partir de qual ciclo há uma distinção entre instruções?**

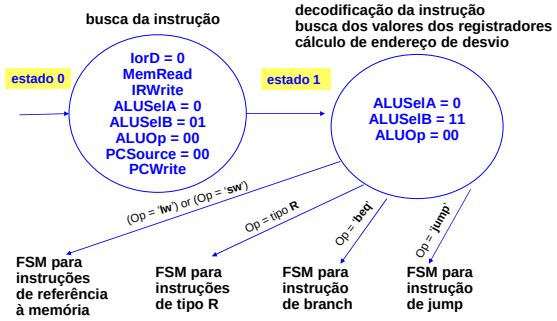
2. Máquina de estados finitos



1. Revisando os sinais de controle



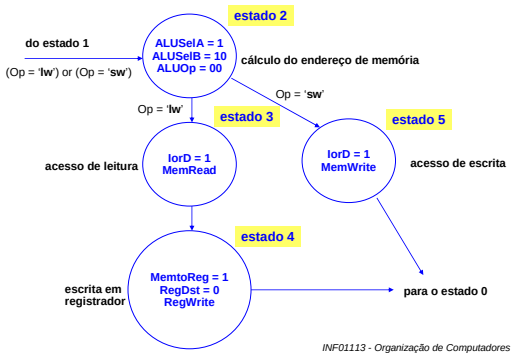
FSM: busca e decodificação de instruções



FSM: busca e decodificação de instruções

- **estado 0 – busca da instrução**
  - **IorD = 0:** PC é fonte de endereço para a memória
  - **MemRead:** ler instrução da memória
  - **IRWrite:** escrever instrução no IR
  - **ALUSelA = 0:** PC é primeiro operando para ALU
  - **ALUSelB = 01:** constante 4 é segundo operando para ALU
  - **ALUOp = 00:** ALU soma PC + 4
  - **PCSource = 00:** Saída da ALU é enviada para o PC
  - **PCWrite:** PC é escrito com PC + 4
- **estado 1 – decodificação da instrução, busca dos valores dos registradores, cálculo do endereço de desvio condicional (branch)**
  - **ALUSelA = 0:** PC é primeiro operando para ALU
  - **ALUSelB = 10:** deslocamento é segundo operando para ALU
  - **ALUOp = 00:** ALU soma PC + deslocamento

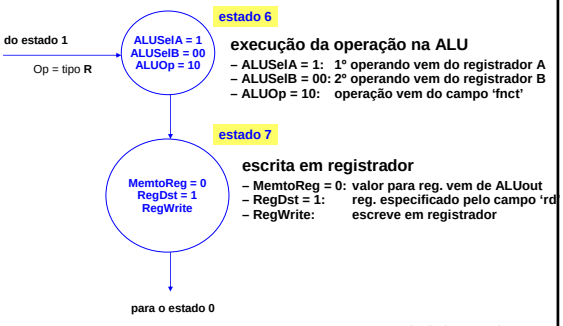
FSM: instruções de referência à memória



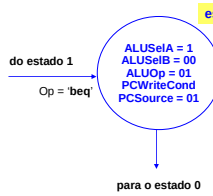
FSM: instruções de referência à memória

- **estado 2 – cálculo do endereço efetivo**
  - **ALUSelA = 1:** registrador base é primeiro operando para ALU
  - **ALUSelB = 10:** deslocamento é segundo operando para ALU
  - **ALUOp = 00:** ALU soma base + deslocamento
- **estado 3 – acesso de leitura na memória**
  - **IorD = 1:** endereço de memória vem da ALU
  - **MemRead:** leitura na memória
- **estado 4 – escrita em registrador**
  - **MemtoReg = 1:** valor para registrador vem da memória
  - **RegDst = 0:** registrador a ser escrito especificado pelo campo rd
  - **RegWrite:** escrita em registrador
- **estado 5 – acesso de escrita na memória**
  - **IorD = 1:** endereço de memória vem da ALU
  - **MemWrite:** escrita na memória

FSM: instruções de tipo R



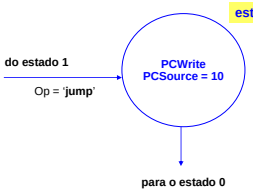
FSM: instrução de desvio condicional (branch)



término do desvio condicional (branch)

- **ALUSelA = 1:** 1º operando vem do registrador A
- **ALUSelB = 00:** 2º operando vem do registrador B
- **ALUOp = 01:** ALU executa subtração
- **PCWriteCond:** PC escrito se saída Zero = 1
- **PCSource = 01:** PC recebe valor do registrador ALUout

FSM: instrução de desvio incondicional (jump)

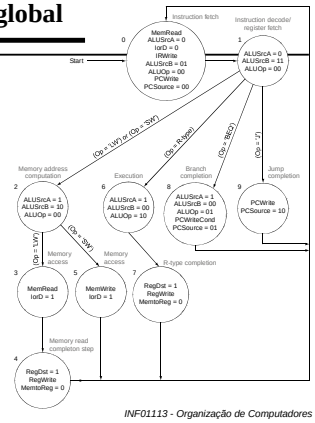


término do desvio incondicional (jump)

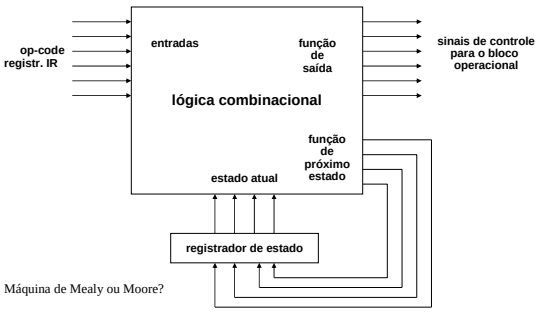
- **PCWrite:** PC escrito
- **PCSource = 10:** PC recebe valor do endereço de desvio

Visão global

- **Quantos FFs são necessários para se implementar a FSM?**

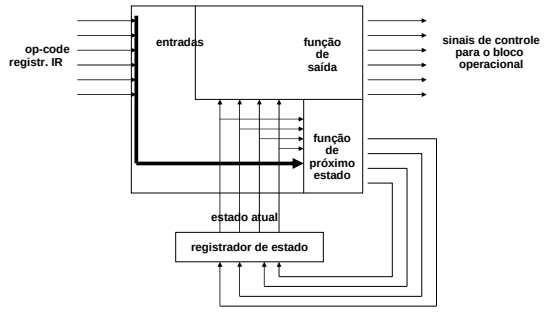


3. Implementando a FSM



Máquina de Mealy ou Moore?

3. Implementando a FSM



Questões básicas

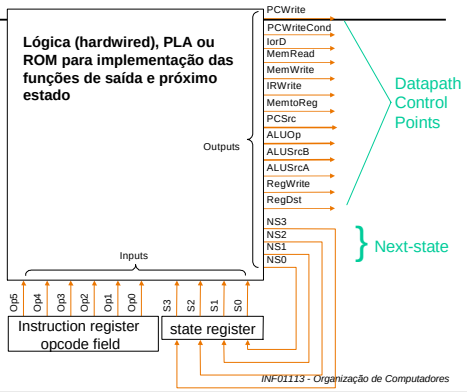
- Quantos ciclos são necessários para executar o código abaixo?

```
lw $t2, 0($t3)
lw $t3, 4($t3)
beq $t2, $t3, Label #assumir não iguais
add $t5, $t2, $t3
sw $t5, 8($t3)
Label:  ...
```

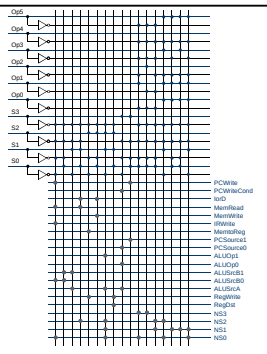
O que está acontecendo no oitavo ciclo de execução?

Em qual ciclo a soma de `$t2` e `$t3` efetivamente acontece?

Implementing the FSM controller

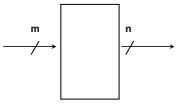


PLA Implementation



ROM Implementation

- **ROM = "Read Only Memory"**
  - **values of memory locations are fixed ahead of time**
- **A ROM can be used to implement a truth table**
  - **if the address is m-bits, we can address 2^m entries in the ROM.**
  - **our outputs are the bits of data that the address points to.**
    m is the "height", and n is the "width" equal to number of outputs.



ROM Implementation

- **How many inputs are there?**
  6 bits for opcode, 4 bits for state = 10 address lines
  (i.e., 2^{10} = 1024 different addresses)
- **How many outputs are there?**
  16 datapath-control outputs, 4 state bits = 20 outputs
- **ROM is 2^{10} x 20 = 20K bits (and a rather unusual size, so go for next size chip)**
- **Rather wasteful due to lots of don't care situations => the outputs only depend on states, not opcodes.**

ROM vs PLA

- **Break up the table into two parts**
  — 4 state bits tell you the 16 outputs, 2^4 x 16 bits of ROM
  — 10 bits tell you the 4 next state bits, 2^{10} x 4 bits of ROM
  — Total: 4.3K bits of ROM => Lots of savings.
- **PLA is much smaller**
  — can share product terms
  — only need entries that produce an active output
  — can take into account don't cares
- **Size is (#inputs × #product-terms) + (#outputs × #product-terms)**
  For this example = (10x17)+(20x17) = 510 PLA cells
  PLA cells usually about the size of a ROM cell (slightly bigger)

E lógica hardwired?

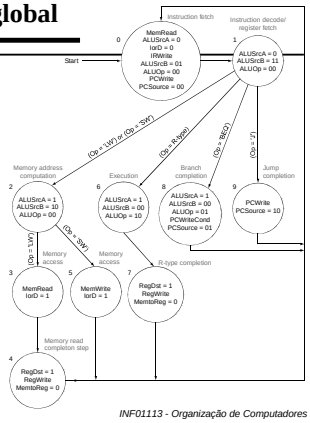
- **O próximo estado é função de quantos bits?**
- **A saída é função de quantos bits?**
- **Qual o tamanho do mapa de karnaugh para o próximo estado?**
- **Quão complicado é o mapa em**
  - **1970?**
  - **1985?**
- **Obs: PLA: dois níveis. Hardwired: multinível. Níveis atuais (profundidade da lógica): entre 5 e 8.**

Para os famintos por Saber

- **O que deve ser alterado no controle do MIPS multiciclo para realizar a instrução LUI?**
- **O que deve ser alterado no controle do MIPS multiciclo para realizar a instrução de load com pós-incremento?**

Visão global

- **RInstr = Memoria[PC]**
- **PC = PC + 4**
- **A = Reg[I[25-21]]**
- **B = Reg[I[20-16]]**
- **ALUout = PC + SignExtend (I[15-0]) << 2**
- **ALUout = A + SignExtend(I[15-0])**
- **MDR = Memoria [ALUout]**
- **Reg [I[20-16]] = MDR**



3. Bloco operacional completo