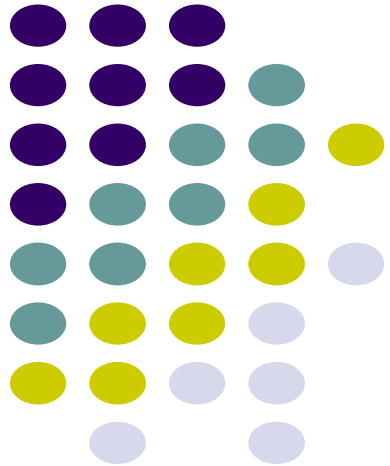
UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
DEPARTAMENTO DE CIÊNCIAS EXATAS E NATURAIS
CURSO DE CIÊNCIA DA COMPUTAÇÃO

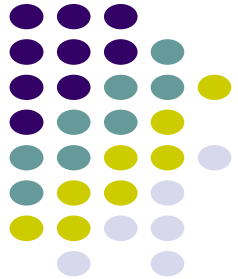
Arquitetura e Organização de Computadores

O Processador: controle



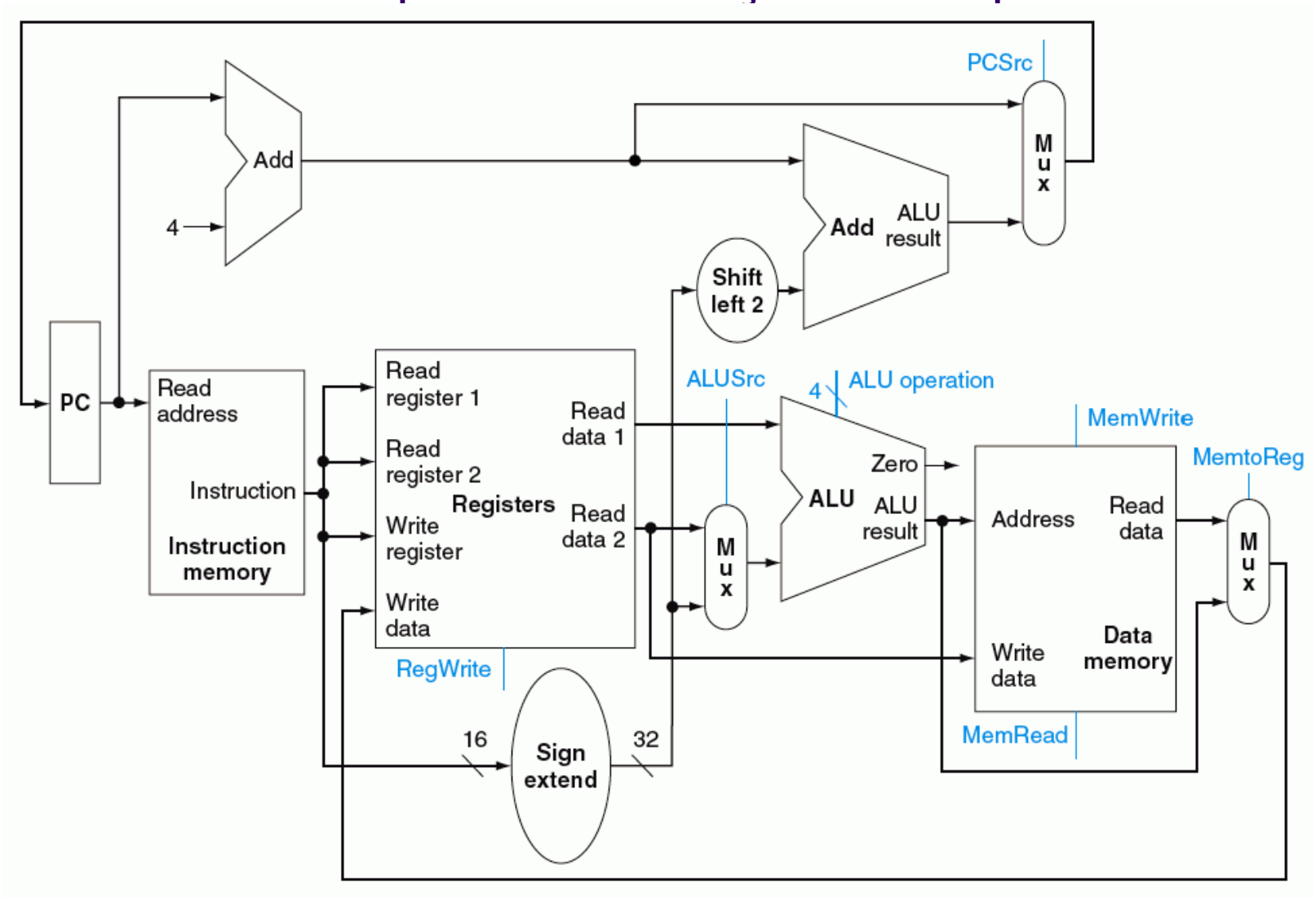
Prof. Sílvio Fernandes

Um esquema de implementação simples

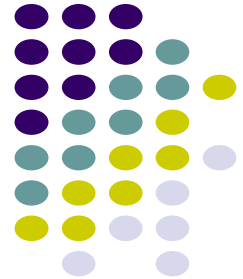


- As operações do caminho de dados das instruções lógicas e aritméticas (ou do tipo R) e das instruções de acesso à memória são muito semelhantes
  - As inst. lógicas e aritméticas usam a ALU com as entradas vindas de 2 registradores. As inst. de acesso à memória também podem usar a ALU para fazer o cálculo do endereço, embora a segunda entrada seja o campo offset de 16 bits com sinal estendido da instrução
  - O valor armazenado em um registrador de destino vem da ALU (para inst. tipo R) ou da memória (para load)

Um esquema de implementação simples



Um esquema de implementação simples

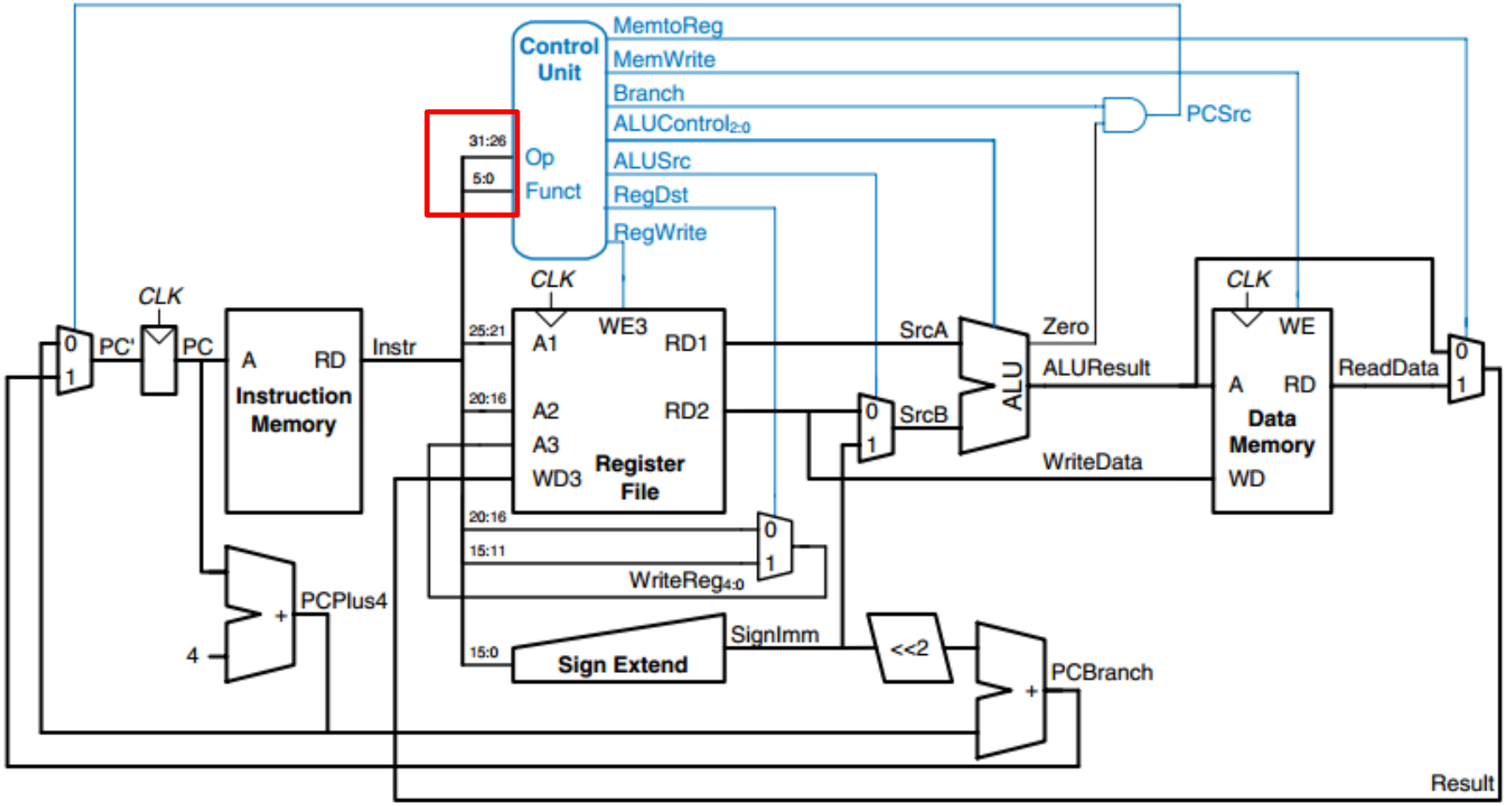


- A unidade de controle atribui os valores dos sinais de controle baseado no *opcode* (Instr 31:26) e campo *funct* (Instr 5:0) das instruções

	31:26					5:0
R	op	rs	rt	rd	shamt	funct
I	op	rs	rt	endereço de 16 bits		
J	op	endereço de 26 bits				

Um esquema de implementação simples

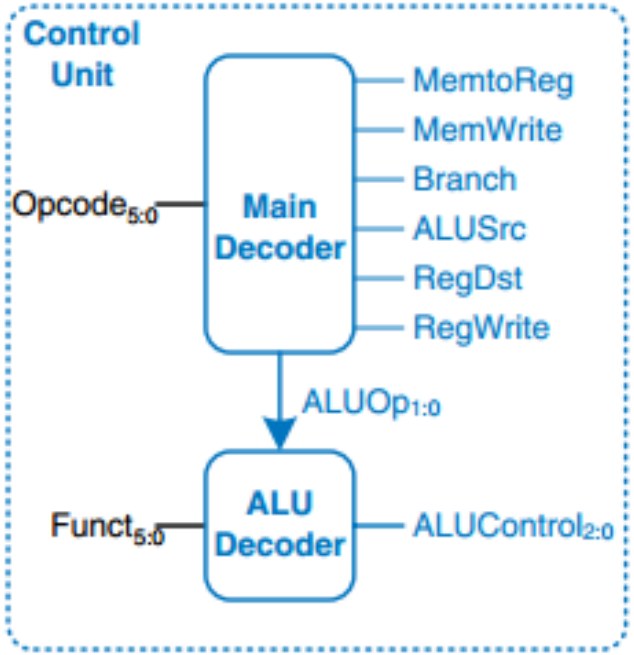
- Controle



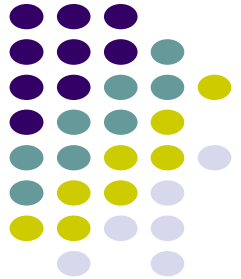
Um esquema de implementação simples

- Controle
  - Instruções do tipo R tem o mesmo *opcode* então *funct* determina a operação da ALU
  - O codificador principal deve computa a maioria dos sinais a partir do *opcode*
  - Também determina 2 bits ALUop
  - ALUDecoder usa os 2 bits com

  Funct para controlar a ALU



Um esquema de implementação simples



- Controle
  - O significado dos 2 bits do ALUOp

ALUOp	Meaning
00	add
01	subtract
10	look at `funct` field
11	n/a

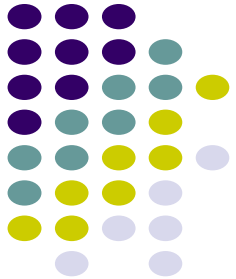
Acesso a memória

Condicionais

Tipo R

Não usada

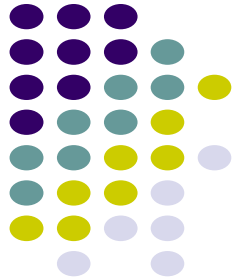
Controle da ALU



- A ALU possui 4 entradas de controle, com bits não codificados (usamos apenas 6 das 16 combinações possíveis)

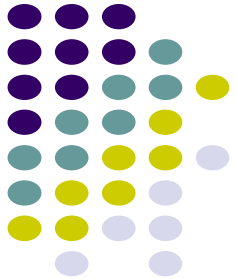
ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

Controle da ALU



- Dependendo da classe de instrução, a ALU precisará realizar uma das funções
  - NOR é usada para outras partes do conjunto de instruções
  - Load e Store usam a ALU para calcular o endereço de memória por adição
  - Instruções do tipo R precisam realizar uma das 5 instruções (AND, OR, sub, add ou set on less than), dependendo do campo funct
  - Branch equal precisa realizar uma subtração

Controle da ALU



- Podemos gerar a entrada do controle da ALU de 4 bits usando como entradas o campo funct da instrução e um campo control de 2 bits (chamaremos de ALUOp)

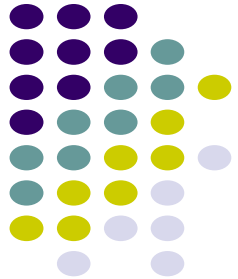
Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	and	0000
R-type	10	OR	100101	or	0001
R-type	10	set on less than	101010	set on less than	0111

Controle da ALU

- Tabela verdade para os 3 bits de controle da ALU (chamados operação)

ALUOp		Funct field						Operation
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	0110
1	X	X	X	0	0	0	0	0010
1	X	X	X	0	0	1	0	0110
1	X	X	X	0	1	0	0	0000
1	X	X	X	0	1	0	1	0001
1	X	X	X	1	0	1	0	0111

Projetando a unidade de controle principal



- Primeiro vamos entender o formato das 3 classes de instruções

Field	0	rs	rt	rd	shamt	funct
Bit positions	31:26	25:21	20:16	15:11	10:6	5:0

a. R-type instruction

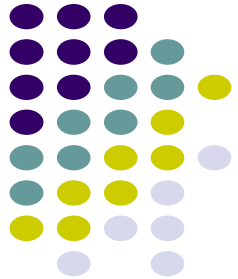
Field	35 or 43	rs	rt	address
Bit positions	31:26	25:21	20:16	15:0

b. Load or store instruction

Field	4	rs	rt	address
Bit positions	31:26	25:21	20:16	15:0

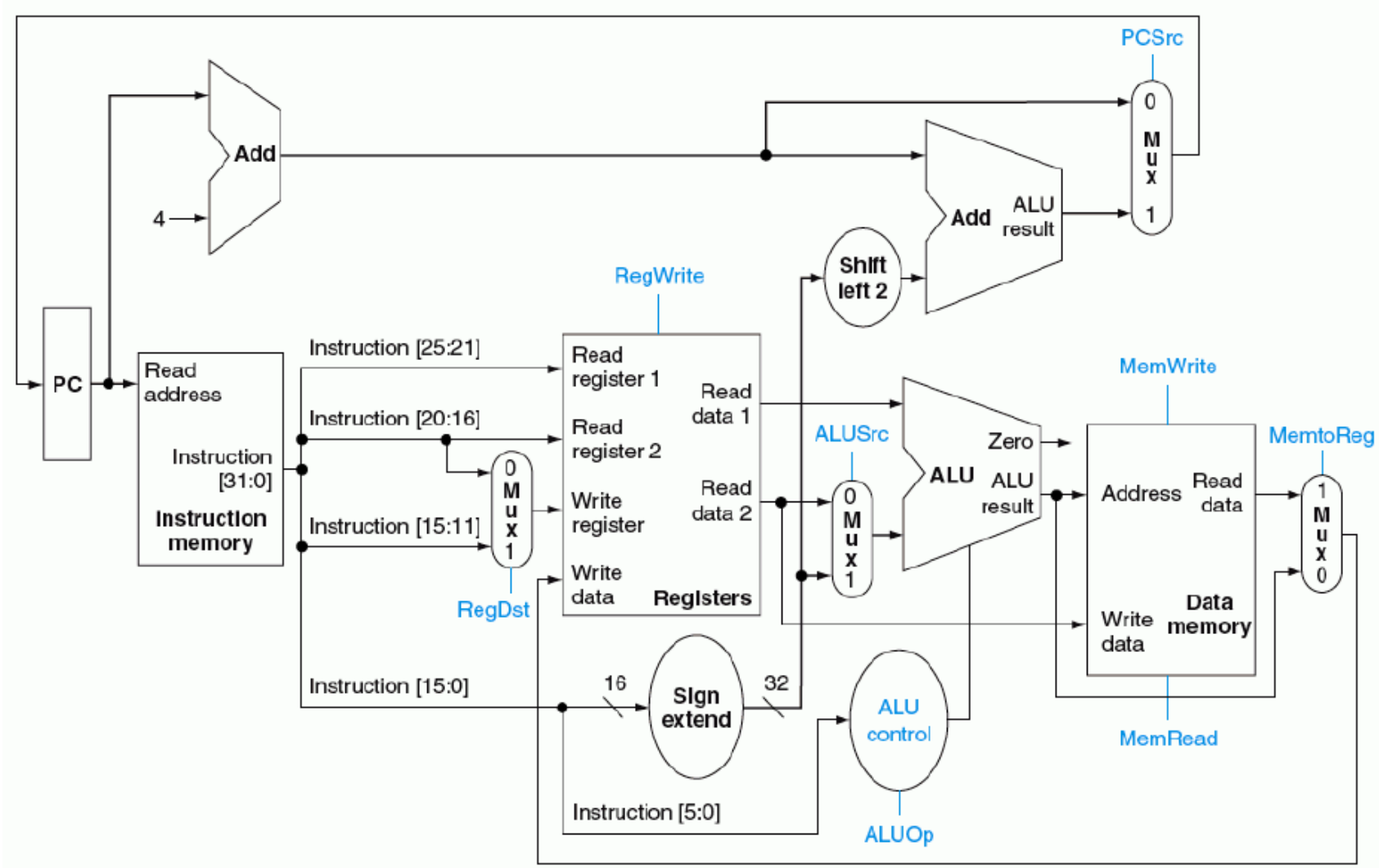
c. Branch instruction

Projetando a unidade de controle principal



- O campo op (opcode), está sempre contido nos bits 31:26. Iremos nos referir a ele como Op[5:0]
- Os 2 registradores a serem lidos sempre são especificados pelos campos rs e rt, nas posições 25:21 e 20:16
- O registrador de base para as instruções load e store está sempre nas posições de bit 25:21
- O offset de 16 bits para branch equal, load e store está sempre nas posições 15:0
- O registrador de destino está nas posições 20:16 (rt) para um load, e nas posições 15:11 (rd) para instr. do tipo R (precisamos de um multiplexador)

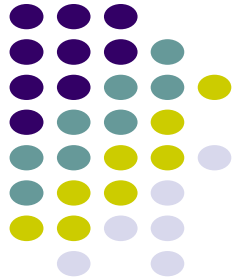
Projetando a unidade de controle principal



Projet

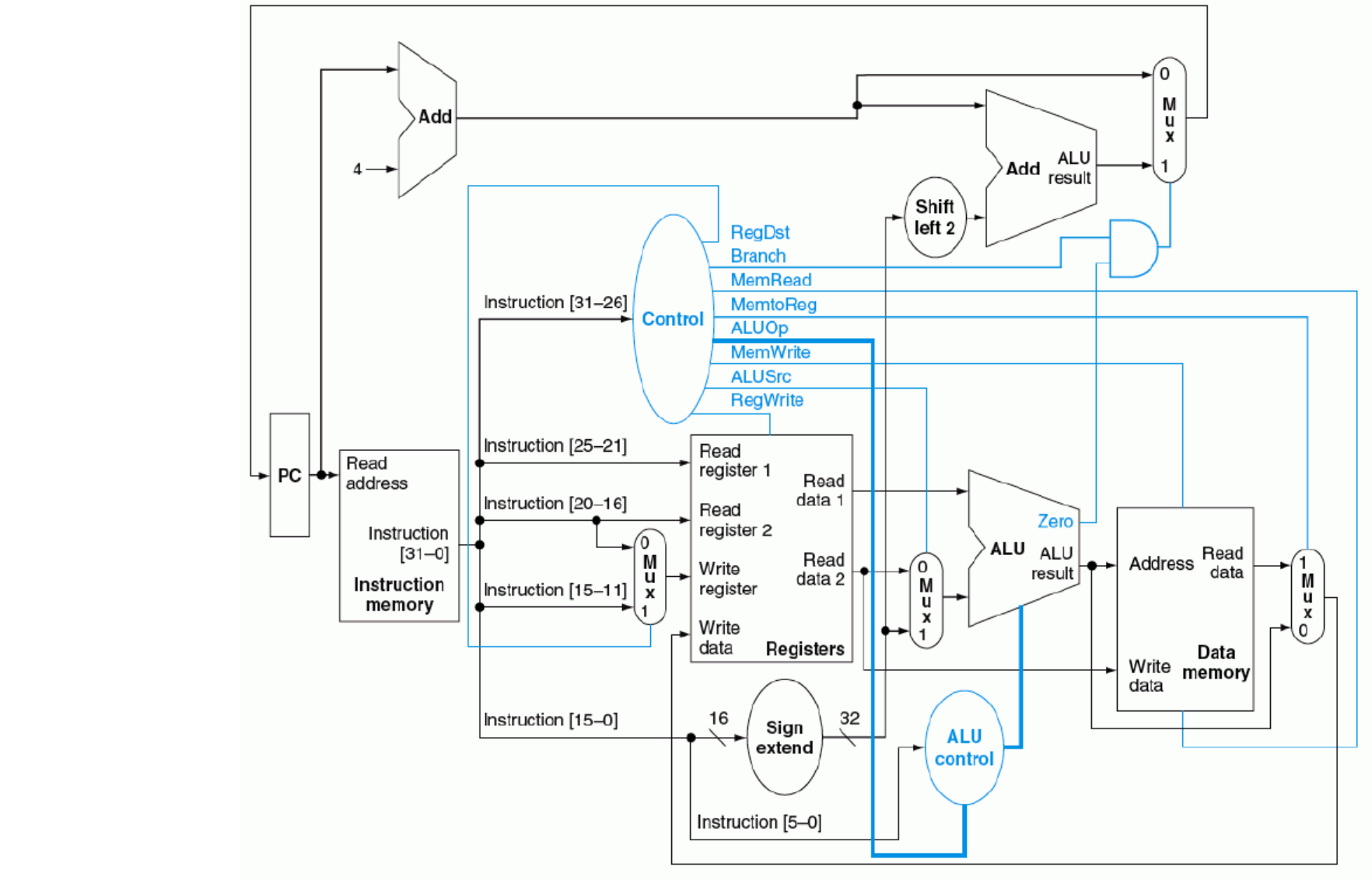
Nome do sinal	Efeito quanto inativo	Efeito quanto ativo
RegDst	O número do registrador destino para a entrada Registrador para escrita vem o campo rt(bits 20:16)	O número do registrador destino para a entrada Registrador para escrita vem do campo rd(bits 15:11)
RegWrite	Nenhum	O registrador na entrada Registrador para escrita é escrito com o valor da entrada Dados para escrita
ALUSrc	O 2º operando da ALU vem da 2ª saída do banco de registradores (Dados da leitura 2)	O 2º operando da ALU consiste nos 16 bits mais baixos da instrução com sinal estendido
PCSrc	O PC é substituído pela saída do somador que calcula o valor de PC+4	O PC é substituído pela saída do somador que calcula o destino do desvio
MemRead	Nenhum	O conteúdo da memória de dados designado pela entrada Endereço é colocado na saída Dados da leitura
MemWrite	Nenhum	O conteúdo da memória de dados designado pela entrada Endereço é substituído pelo valor na entrada Dados para escrita
MemtoReg	O valor enviado para entrada Dados para escrita do banco de registradores vem da ALU	O valor enviado para a entrada Dados para escrita do banco de registradores vem da memória de dados

Projetando a unidade de controle principal

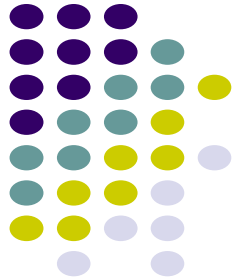


- Os sinais anteriores são gerados unicamente com base no campo opcode da instrução
- A exceção é a linha de controle PCSrc
  - Deve ser ativada se a instrução branch on equal e a saída *Zero* da ALU for verdadeira
  - Para gerar o sinal PCSrc, precisamos realizar um AND

Projetando a unidade de controle principal



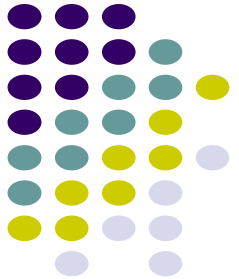
Projetando a unidade de controle principal



- A definição das linhas de controle é completamente determinada pelos campos opcode da instrução

Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

O caminho de dados em operação para uma instrução tipo R

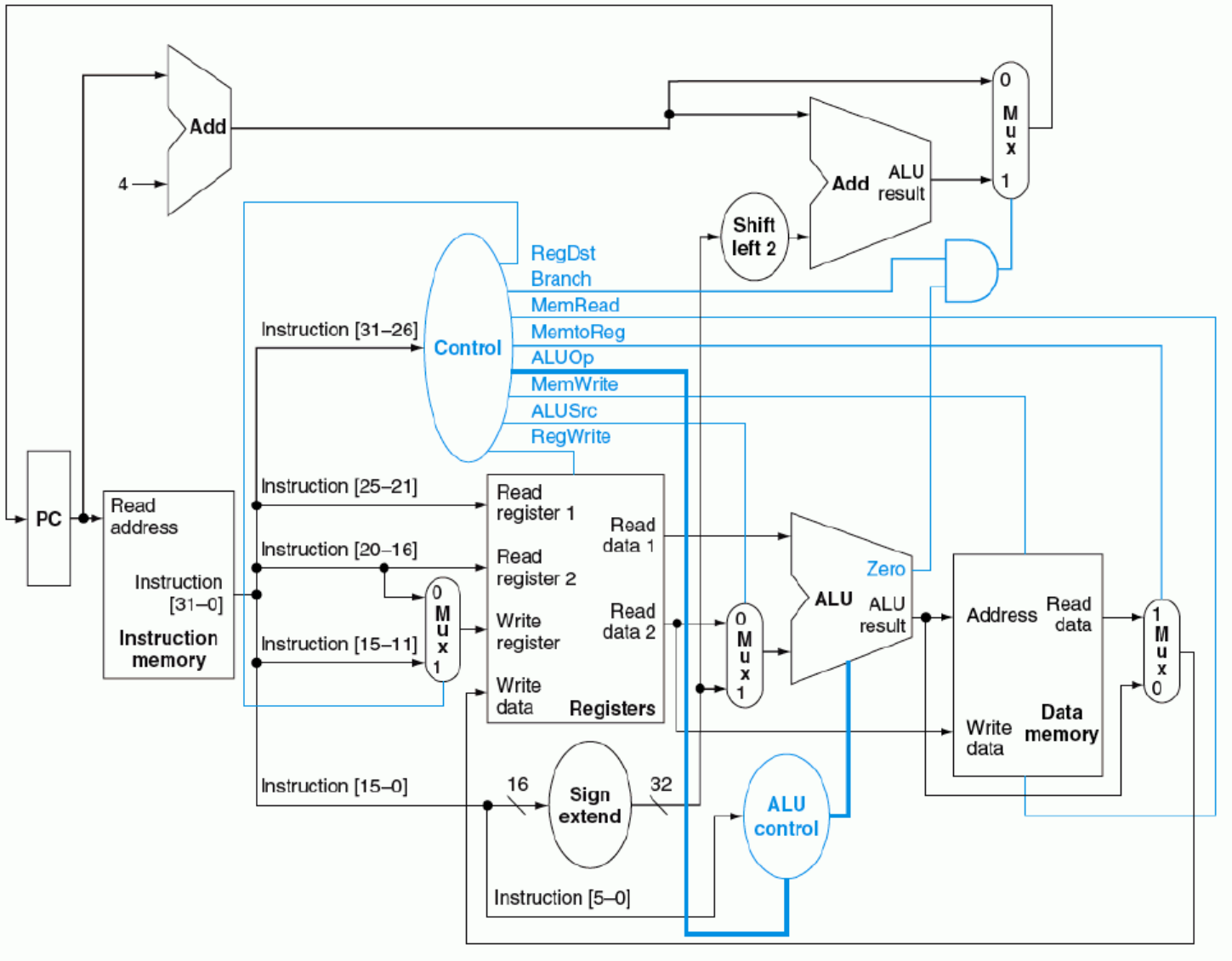


- add $t1, $t2, $t3

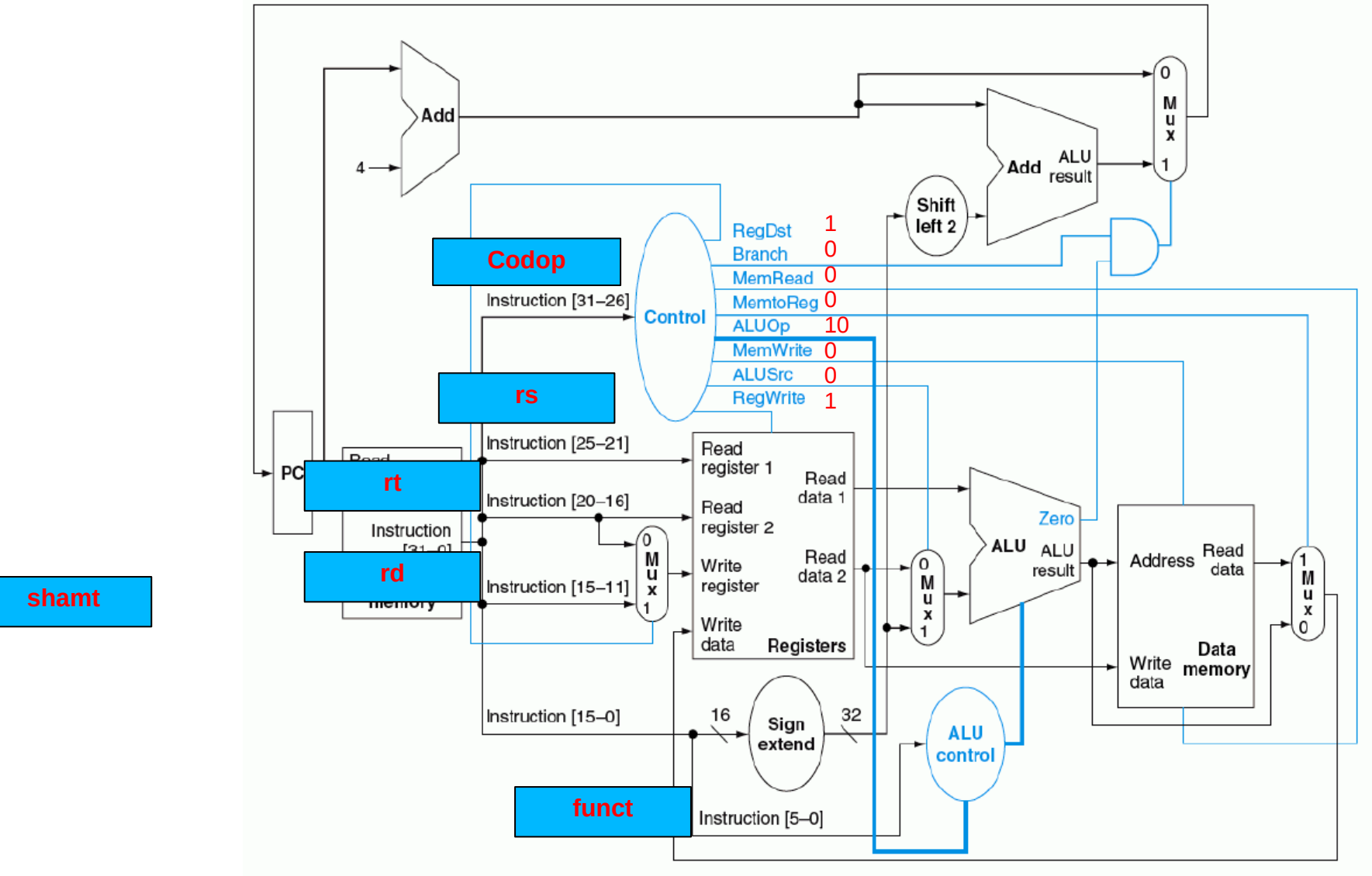
1. A instrução é buscada e o PC é incrementado
2. 2 registradores, $t2 e $t3, são lidos do banco de registradores e a unidade de controle principal calcula a definição das linhas de controle também durante essa etapa
3. A ALU opera nos dados lidos do banco de registradores, usando o código de função (bits 5:0, que é o campo funct, da instrução) para gerar a função da ALU
4. O resultado da ALU é escrito no banco de registradores usando os bits 15:11 da instrução para selecionar o registrador de destino ($t1)

Instruções do tipo R – Ex: add $t1, $t2, $t3

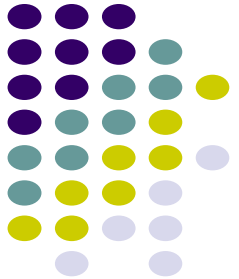
Codop	rs	rt	rd	shamt	funct



Instruções do tipo R – Ex: add $t1, $t2, $t3



O caminho de dados em operação para uma instrução load

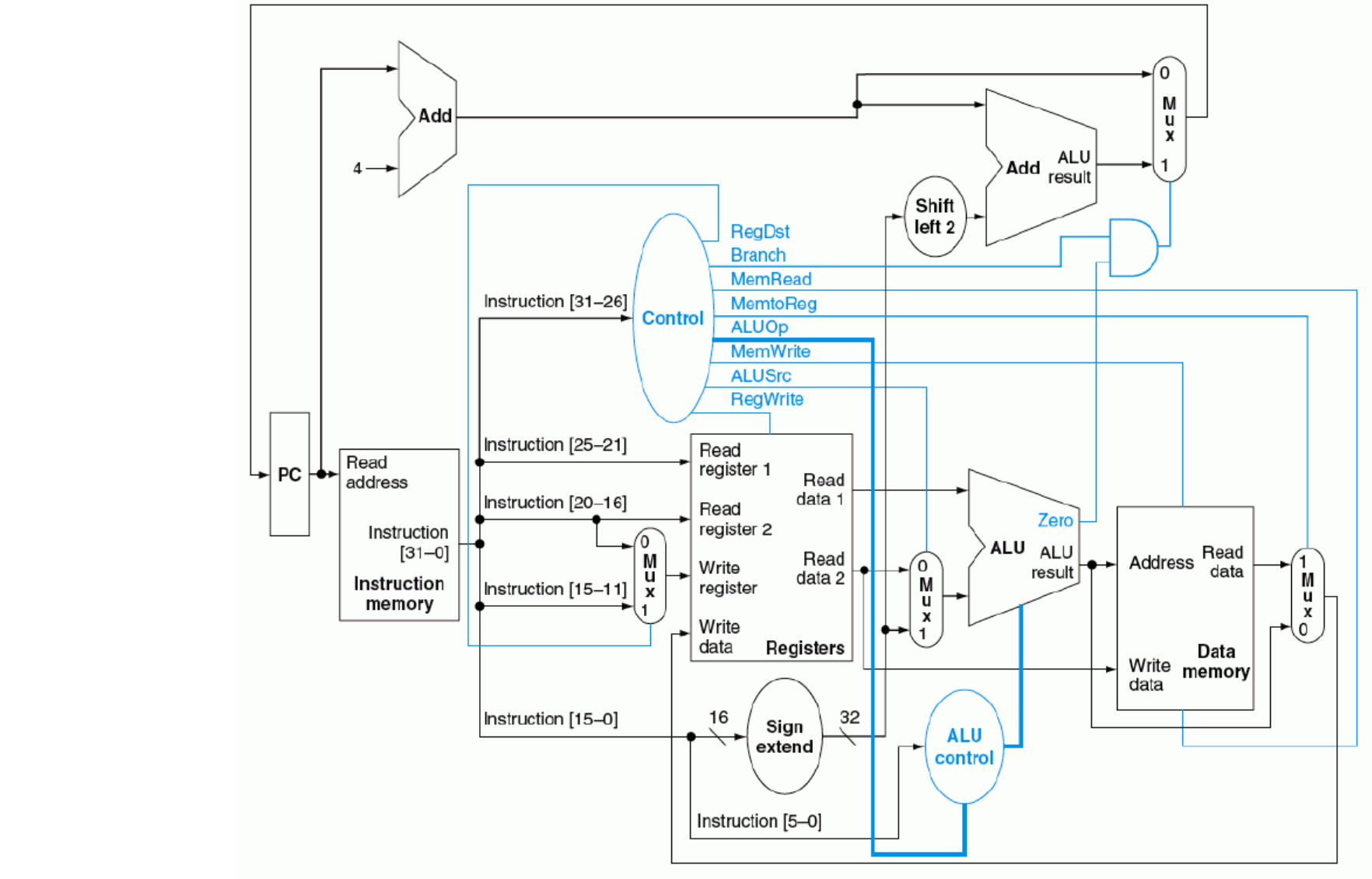


- lw $t1, offset($t2)

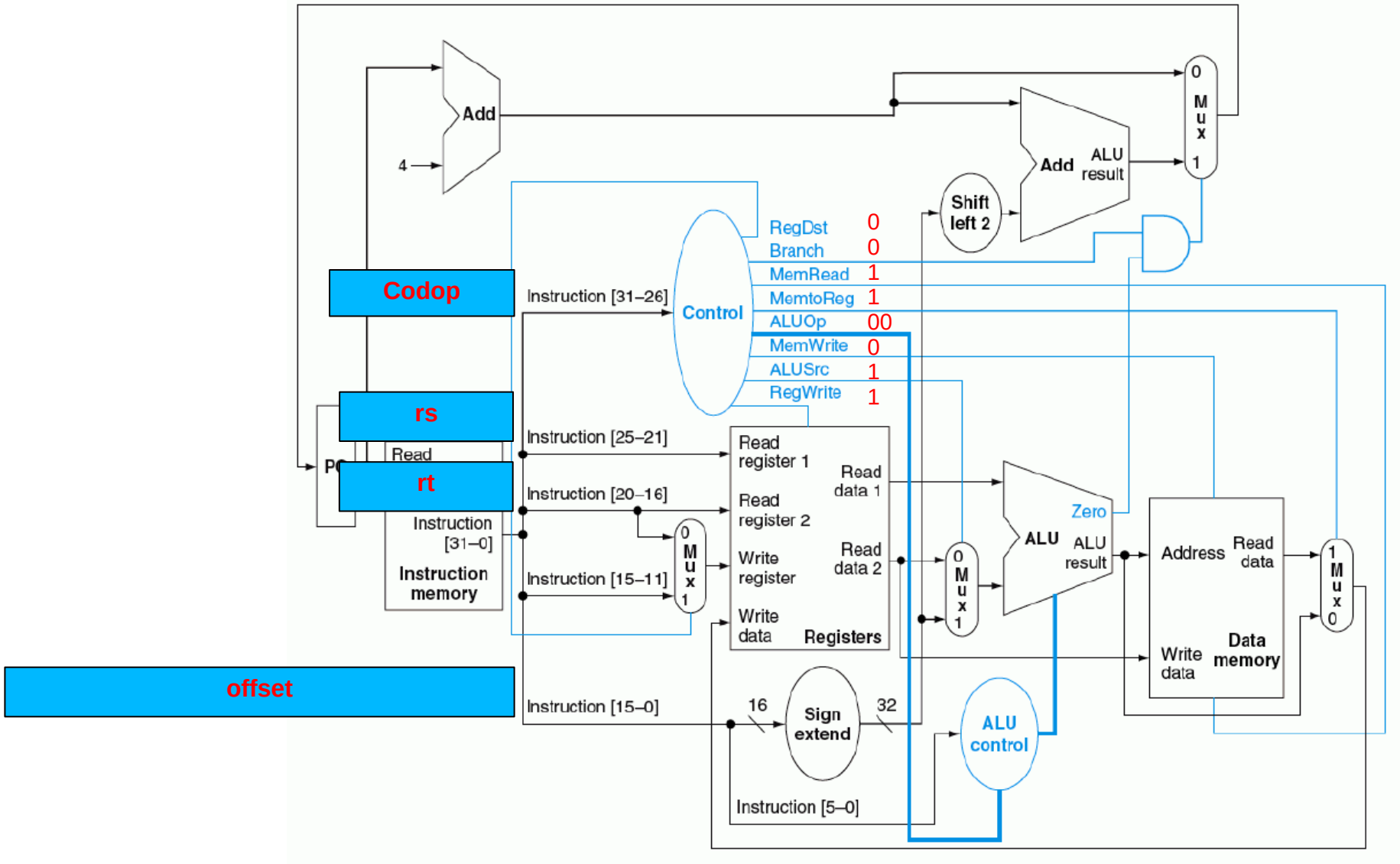
1. Uma instrução é buscada da memória de instruções e o PC é incrementado
2. Um valor de registrador ($t2) é lido do banco de registradores
3. A ALU calcula a soma do valor lido do banco de registradores com os 16 bits com sinal estendido (offset)
4. A soma da ALU é usada como o endereço para a memória de dados
5. Os dados da unidade de memória são escritos no banco de registradores; o registrador de destino é fornecido pelos bits 20:16 ($t1)

Instruções de acesso mem. – Ex: lw $t1, offset($t2)

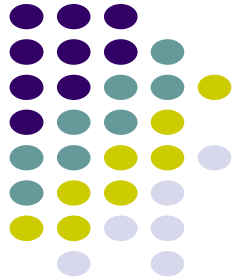
Codop	rs	rt	offset



Instruções de acesso mem. – Ex: lw $t1, offset($t2)



O caminho de dados em operação para uma instrução branch equal

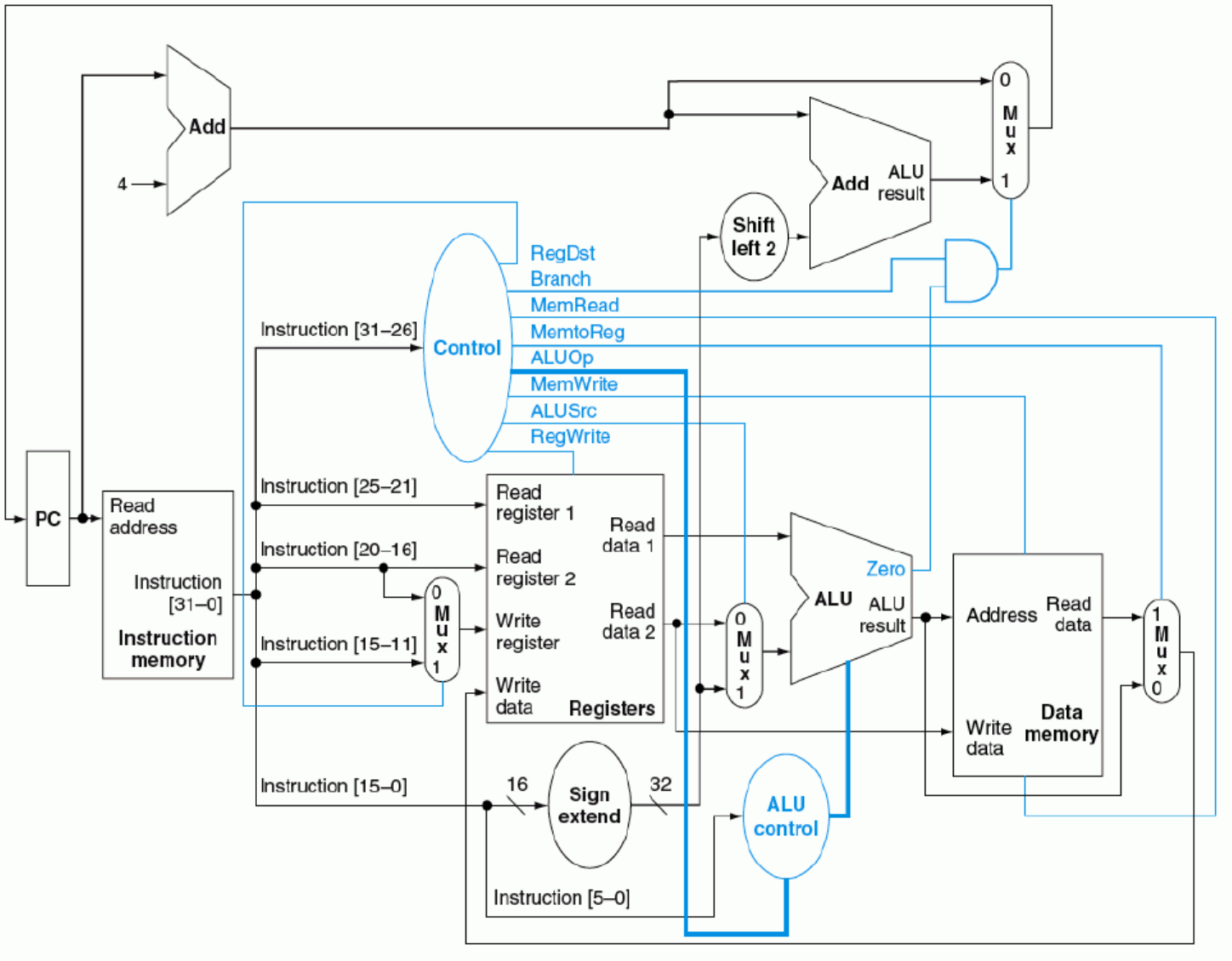


- beq $t1, $t2, offset

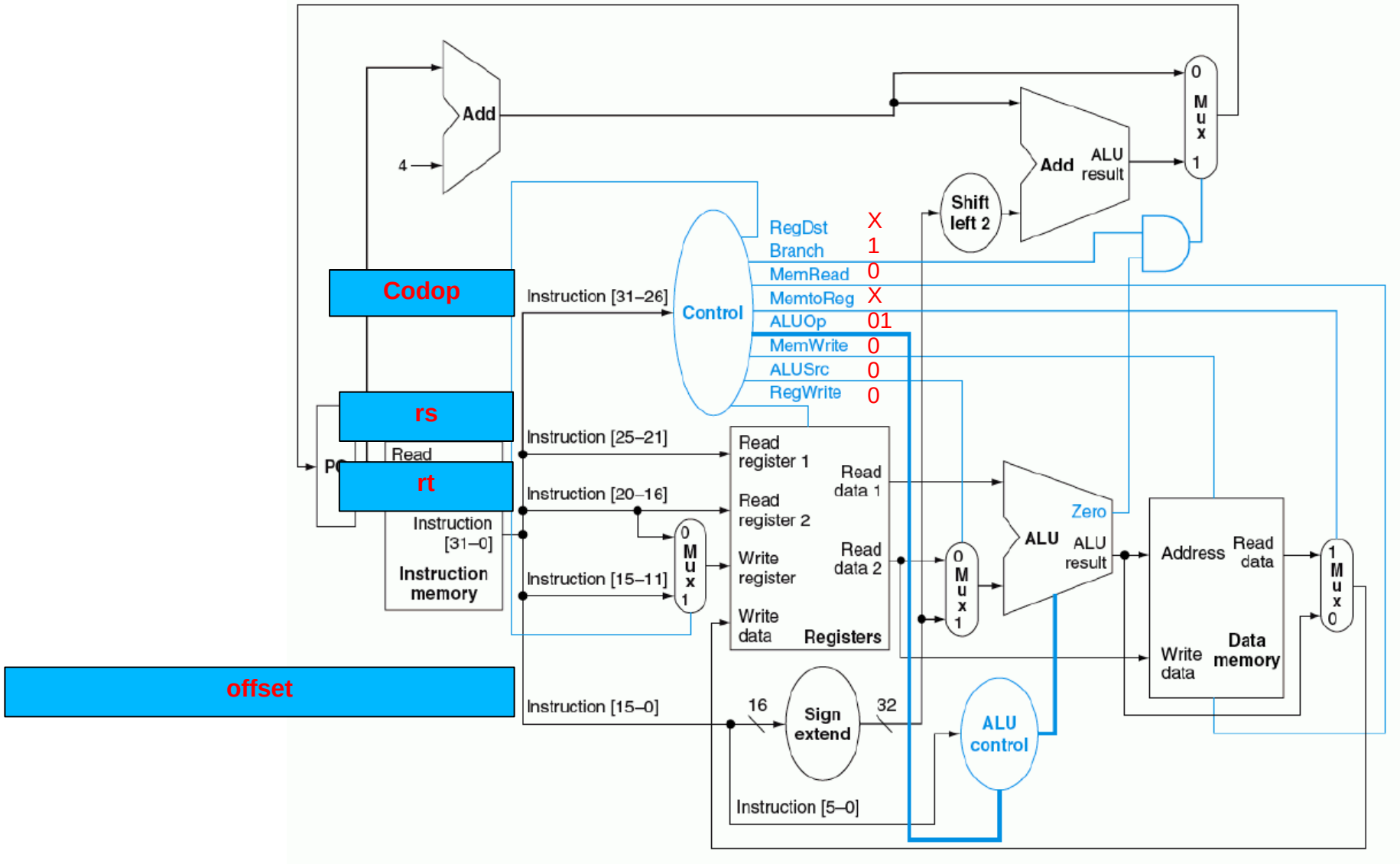
1. Uma instrução é buscada da memória de instruções e o PC é incrementado
2. 2 registradores, $t1 e $t2, são lidos do banco de registradores
3. A ALU realiza uma subtração dos valores lidos; o valor de PC+4 é somado aos 16 bits com sinal estendido (offset) deslocados de 2 para esquerda; o resultado é o endereço de destino do desvio
4. O resultado Zero da ALU é usado para decidir o resultado de que somador deve ser armazenado no PC

Instruções de desvio cond. Ex: beq $t1, $t2, offset

Codop	rs	rt	offset



Instruções de desvio cond. Ex: beq $t1, $t2, offset

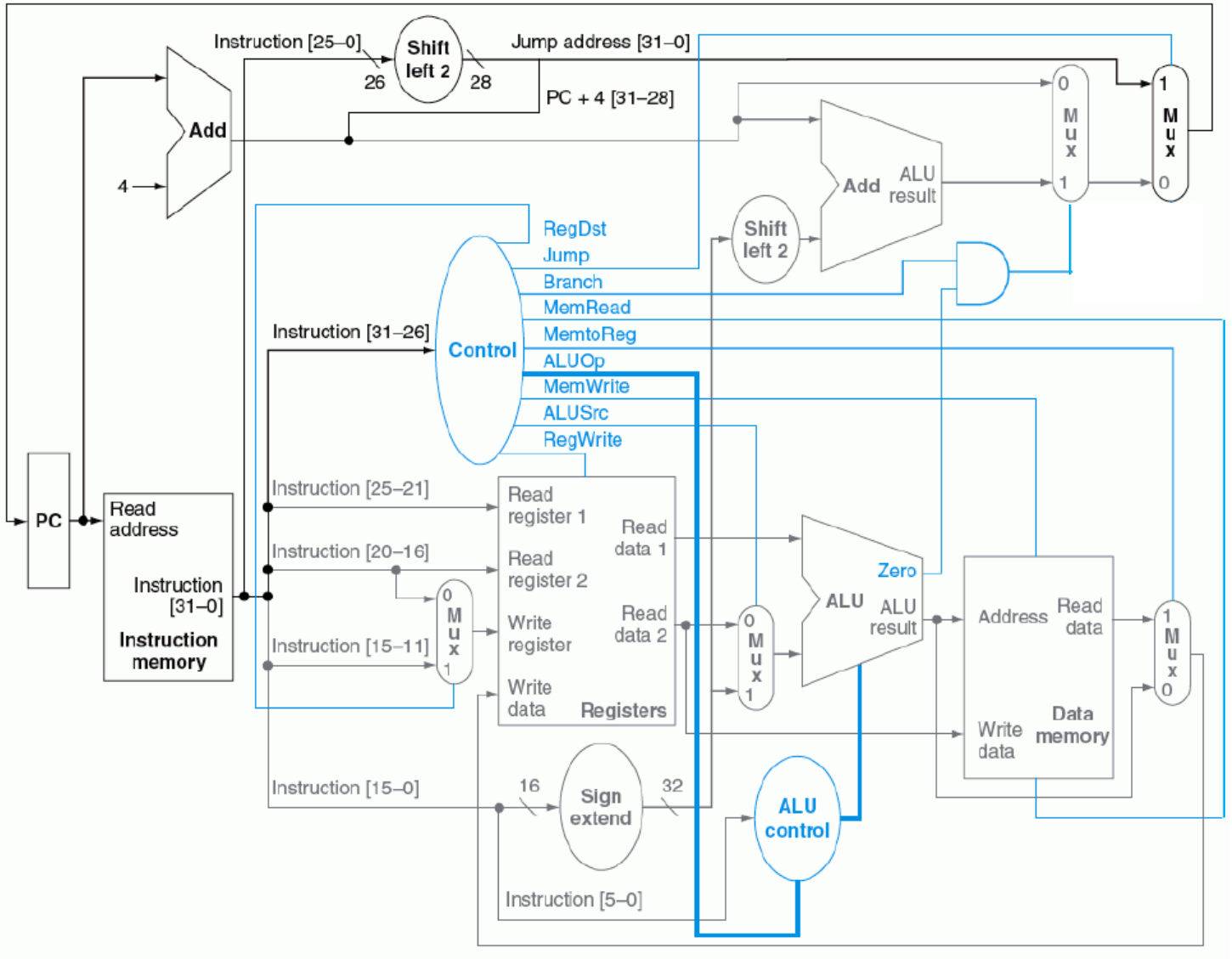


Implementando Jumps

- Podemos implementar um jump armazenando no PC a concatenação de
  - os 4 bits superiores do PC atual + 4 (esses são bits 31:28 do endereço da instrução imediatamente seguinte)
  - o campo de 26 bits imediato da instrução jump
  - os bits 00_{bin}

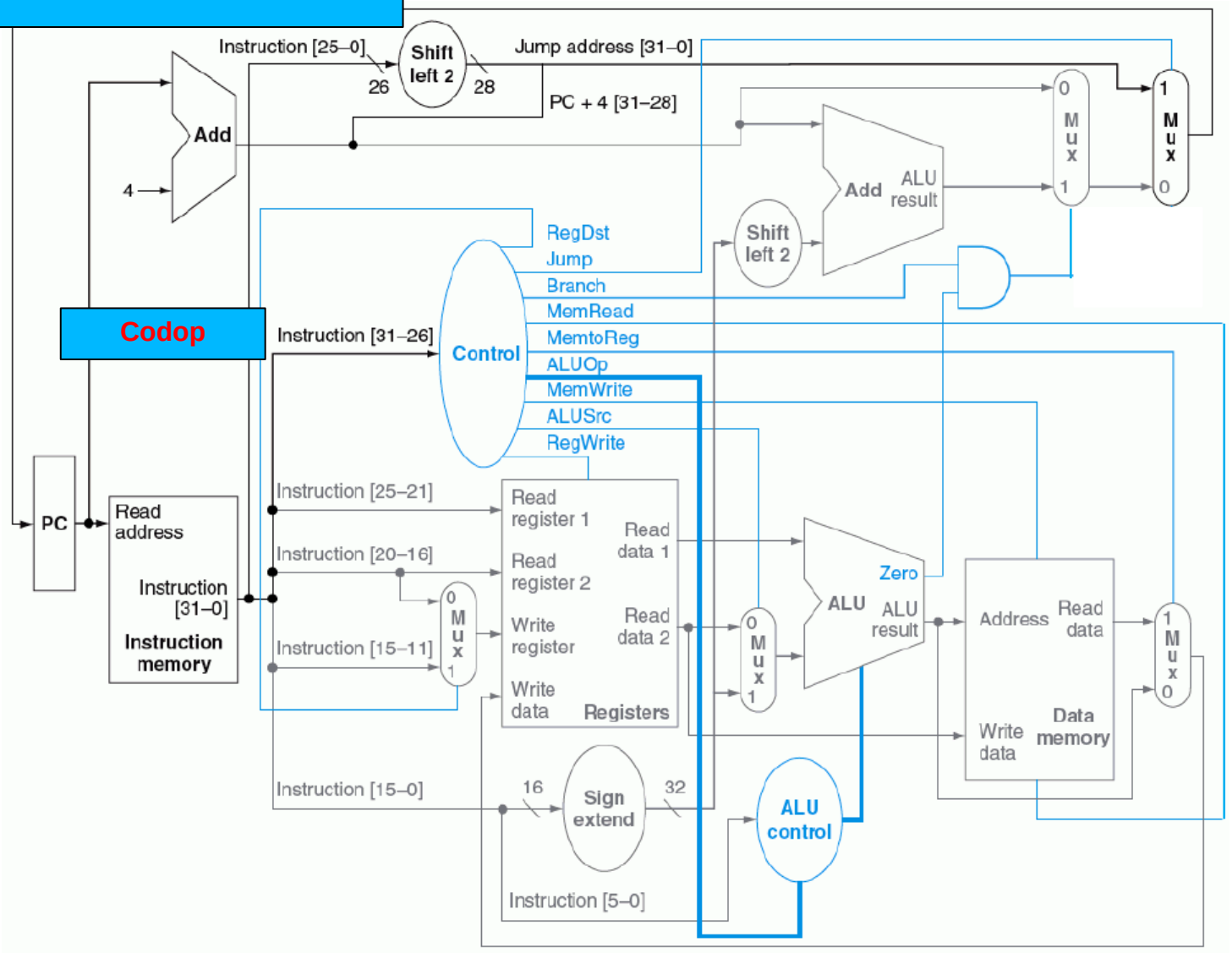
Implementando Jumps

Codop	offset



Implementando Jumps

offset

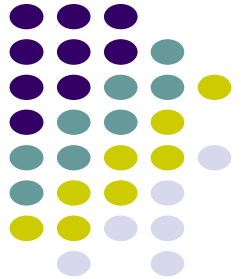


Finalizando o controle

- Implementação simples de ciclo único

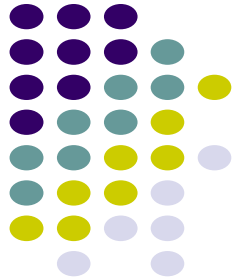
Input or output	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

Por que uma implementação de ciclo único não é usada hoje?



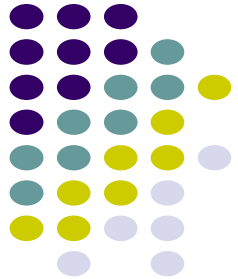
- Porque é ineficiente
  - O ciclo de clock precisa ter a mesma duração para cada instrução nesse projeto de ciclo único e o CPI será 1
  - É claro que o ciclo de clock é determinado pelo caminho mais longo possível na máquina
  - Esse caminho, é quase certamente, uma instrução load, que usa 5 unidades funcionais em série
  - O desempenho de uma implementação não será muito bom, já que várias das classes de instrução poderiam ficar em um ciclo de clock mais curto

Desempenho das máquinas de ciclo único



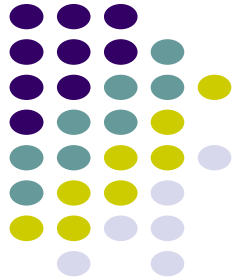
- Suponha que os tempos de operação para as principais unidades funcionais nessa implementação sejam os seguintes
  - Unidades de memória: 200 picossegundos (ps)
  - ALU e somadores: 100 ps
  - Banco de registradores (leitura ou escrita): 50 ps
- Considerando que os multiplexadores, a unidade de controle, os acessos do PC, a unidade de extensão de sinal e os fios não possuem atraso, qual das seguintes implementações seria mais rápida e por quanto?

Desempenho das máquinas de ciclo único



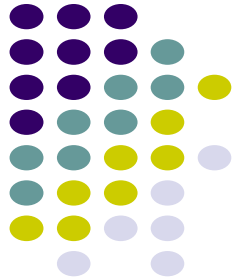
1. Uma implementação em que toda instrução opera em 1 ciclo de clock em uma duração fixa
2. Uma implementação em que toda instrução é executada em 1 ciclo de clock usando um clock de duração variável, que, para cada instrução, tem apenas a duração necessária. (Esse método não é incrivelmente prático, mas permitirá ver o que está sendo sacrificado quando todas as instruções precisam ser executadas com um clock único de mesma duração)

Desempenho das máquinas de ciclo único



- Para comparar o desempenho, considere o seguinte mix de instruções: 25% loads, 10% stores, 45% instruções da ALU, 15% desvios e 5% jumps
- Vamos começar comparando os tempos de execução da CPU
  - Tempo CPU = Contagem instruções x CPI x Tempo do ciclo de clock
- Como CPI é 1, podemos simplificar
  - Tempo CPU = Contagem instruções x Tempo do ciclo de clock

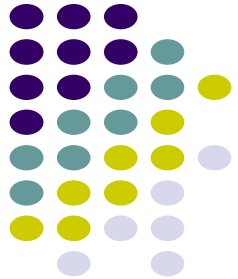
Desempenho das máquinas de ciclo único



- Resposta:
  - Precisamos encontrar o tempo de ciclo de clock para as 2 implementações, já que a contagem de instruções e CPI são iguais

Classe de instrução	Unidades funcionais usadas pela classe de instrução				
Tipo R	Busca de instrução	Acesso a registrador	ALU	Acesso a registrador	
Load Word	Busca de instrução	Acesso a registrador	ALU	Acesso à memória	Acesso a registrador
Store Word	Busca de instrução	Acesso a registrador	ALU	Acesso à memória	
Branch	Busca de instrução	Acesso a registrador	ALU		
Jump	Busca de instrução				

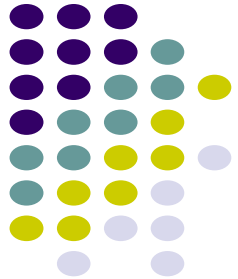
Desempenho das máquinas de ciclo único



- Resposta:
  - Usando esses caminhos críticos, podemos calcular o tamanho exigido para cada classe de instrução:

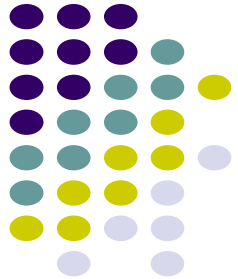
Classe de instrução	Memória de instrução	Leitura de registrador	Operação ALU	Memória de dados	Leitura de registrador	Total
Tipo R	200	50	100	0	50	400 ps
Load Word	200	50	100	200	50	600 ps
Store Word	200	50	100	200		550 ps
Branch	200	50	100	0		350 ps
Jump	200					200 ps

Desempenho das máquinas de ciclo único



- Resposta:
  - Já uma máquina com um clock variável terá um ciclo de clock que varia entre 200ps e 600ps.
  - Podemos encontrar a duração média do ciclo de clock usando essas informações e a distribuição da frequência das instruções
  - **Ciclo de clock da CPU** = 600 x 25% + 550 x 10% + 400 x 45% + 350 x 15% + 200 x 5% = 447,5 ps

Desempenho das máquinas de ciclo único



- Resposta:

$$\frac{Desempenho_{clock\ variável}}{Desemepnho_{clock\ único}} = \frac{Tempo\ de\ Execuçãoclock_{\ único}}{Tempo\ de\ Execuçãoclock_{\ variável}} =$$

$$\frac{CI\ x\ Ciclo\ de\ clock\ da\ CPUclocl_{\ variável}}{CI\ x\ Ciclo\ de\ clock\ da\ CPUclock_{\ único}} = \frac{Ciclo\ de\ clock\ da\ CPU_{único}}{Ciclo\ de\ clock\ da\ CPUvari_{ável}}$$

$$= \frac{600}{447{,}5} = 1{,}34$$

Referências

- PATTERSON, D. A. ; HENNESSY, J.L. Organização e projeto de computadores – a interface hardware software. 3. ed. Editora Campus, 2005.
- HARRIS, David M; HARRIS, Sarah L. **Digital Design and Computer Architecture.** 2ed. Elsevier, 2013.
- STALLINGS, W. Arquitetura e organização de computadores: projeto para o desempenho. 8. ed. Prentice Hall, 2009.