Introdução à Arquitetura de Computadores

Aula21

µArquitetura MIPS: Single-cycle - I

Introdução

µArquitetura: Definição e Tipos

A µArquitetura dum CPU

Máquina Síncrona; Arq. Harvard

Datapath e Controlo

Fases de projeto

Datapath

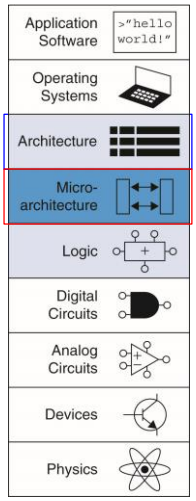
Subconjunto de Instruções (ISA)

Execução de Instruções:

Acesso a Dados (lw e sw)

Tipo-R (*add, sub*)

Branch (beq)



µArquitetura (1) - Introdução

1. Introdução

µArquitetura

- Como implementar o *hardware* da Arquitetura dum Processador (Arquitetura = Visão do programador do conjunto instruções, registos e memória).

Processador

- Datapath: Blocos Funcionais

  Memórias, registos, ALUs e multiplexers que operam sobre dados (*words* de 32-bits)

- Unidade de Controlo:

  Determina como a instrução deve ser executada no *Datapath*, gerando sinais de seleção de multiplexers, de *enable* de registos, de *write* de memórias, etc.

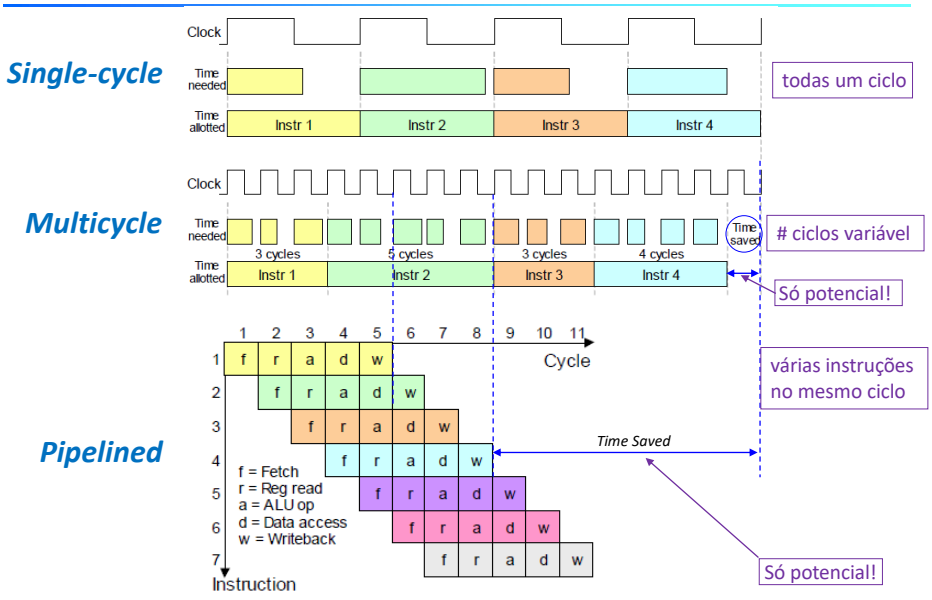
Datapath = Caminho de Dados.

Os Blocos Funcionais estão interligados por *Buses* controlados pela Unidade de Controlo.

µArquitetura (2) - Três tipos de Implementações

- ***Single-cycle***: A execução de cada instrução é efectuada num único ciclo de relógio (*clock*). Todas as instruções ocupam o mesmo intervalo de tempo.
- ***Multicycle***: A execução de cada instrução é dividida numa série de passos mais simples; cada um deles ocupa um ciclo de relógio (de maior frequência). As instruções possuem tempos de execução diferentes (e.g., lw =5 ciclos e beq =3 ciclos).
- ***Pipelined***: A execução de cada instrução é dividida numa série de passos mais simples; o processador executa múltiplas instruções em simultaneo (em paralelo), aumentando, deste modo, a *performance*.

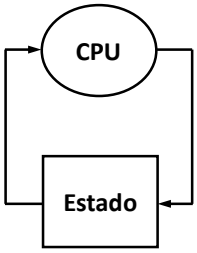
µArquitetura (3) - Single-cycle vs Multicycle vs Pipelined



CPU MIPS (1) - Máquina Síncrona

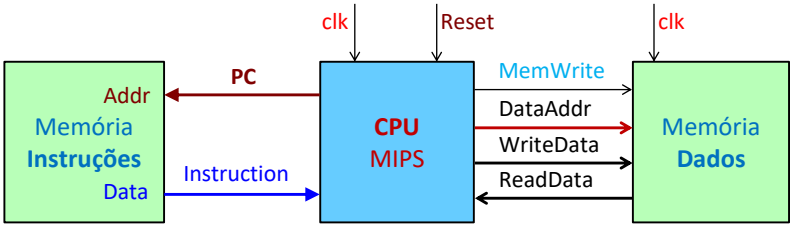
O CPU é uma máquina de estados síncrona.

- As Memórias, os Registos e outros autómatos definem o estado.
- A execução das instruções dum programa altera o seu estado.



CPU MIPS (2) - µArquitetura Harvard*

A µArquitetura dum CPU Single-cycle

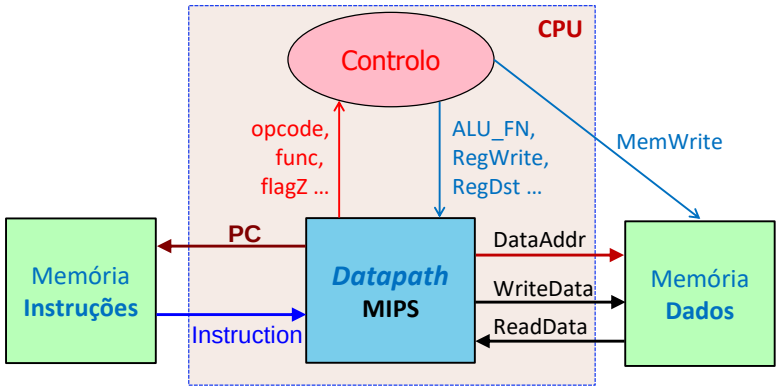


- O CPU MIPS interage com duas memórias.
- Após o *Reset*, o registo PC (Program Counter) é carregado com o endereço da Memória de Instruções, para ler a primeira Instrução a ser executada pelo CPU.
- O CPU gera os sinais de Controlo necessários à execução da Instrução, a qual pode ou não envolver a Memória de Dados.

**Harvard* = A implementação *Single-cycle* (Ciclo-único) usa memórias separadas (externas ao CPU) para permitir o acesso a ambas, em simultaneo (i.e., durante o mesmo ciclo) pelo CPU.

CPU MIPS (3) - Datapath e Unidade de Controlo

• *Datapath*: Componentes que armazenam ou processam dados
  - Registos, ALU, multiplexers, extensão-sinal, etc.



• Controlo: Componentes que 'dizem' ao *Datapath* o que fazer.
  - Lógica combinatória e/ou sequencial (FSMs).

CPU MIPS (4) - Fases de Projeto

1. Análise do Conjunto de Instruções (*ISA*)
   - A execução de cada instrução requere transferências entre registos e/ou memórias;
   - O *Datapath* deve incluir o *hardware* necessário para suportar essas transferências.

2. Seleção dos Componentes para o *Datapath*
   - Memórias, Registos, ALU, Multiplexers, etc

3. Implementação da Lógica de Controlo
   - Lógica combinatória ou FSMs

ISA = *Instruction Set Architecture*; **FSM** = *Finite State Machine*.

SC Datapath (1) - Instruções (ISA) a Implementar

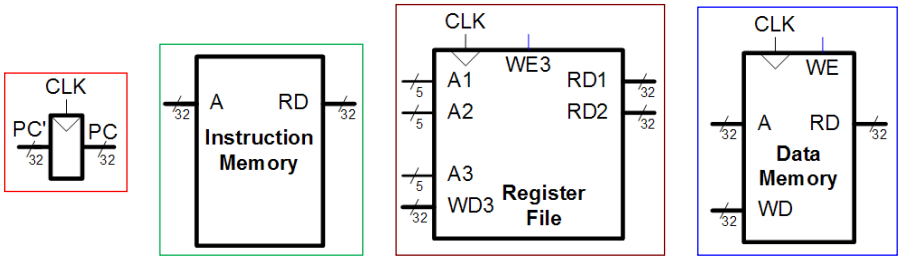
ISA: Começamos com um número limitado

- Acesso à Memória
  `lw` e **`sw`**
- Tipo-R
  `and`, **`or`**, **`add`**, **`sub`** e **`slt`**
- *Branch*
  `beq`
- Instruções adicionais (próx. aula)
  `addi`
  `j`

SC Datapath (2) - Elementos de Estado (1)

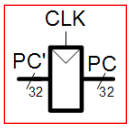
O *Datapath** dum CPU é baseado nos seguintes elementos de estado:

- *Program Counter* (PC)
- Memória de Instruções
- Banco de 32 Registos
- Memória de Dados



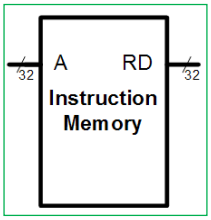
*O projeto dum sistema complexo começa com o *hardware* dos elementos de estado.

SC Datapath (3) - Elementos de Estado (2): PC e Memórias

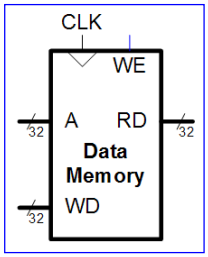


- PC: É um registo de 32-bits normal;
  - A saída (PC) é o endereço da instrução corrente;
  - A entrada (PC') é o endereço da instrução seguinte.

- Memória de Instruções:
  Tem um único porto de leitura (A/RD).

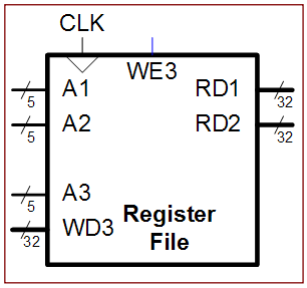


- Memória de Dados:
  Tem um único porto de leitura (A/RD) ou de escrita (A/WD) e um sinal de *WriteEnable* (WE).



SC Datapath (3) - Elementos de Estado (3) - Banco de Reg.

- Banco de 32 Registos:
  Tem dois portos de leitura (A1/RD1 e A2/RD2) e um porto de escrita (A3/WD3).



Os três *buses* A1, A2 e A3 possuem 5-bits, visto que $2^5 = 32$ registos.

Ex: As instruções do tipo-R lêm os operandos dos portos (RD1 e RD2) e escrevem o resultado no porto (WD3), e.g., add $t2, $t1, $t0.

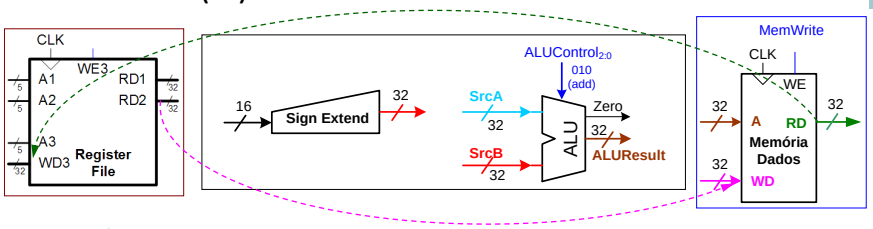
Exceptuando a Memória de Instruções, todos os elementos de estado têm entrada de CLK.

SC Datapath (4) - Instruções Load/Store (1)

Operações

`lw rt, imm(rs)` | **`sw rt, imm(rs)`**

- Usam os registos rs e rt como operandos
- Calculam (na ALU) o endereço de memória efetivo usando (rs) e o imm.



- *Load*: Lê da Memória (para o RF) o novo valor do registo rt.
- *Store*: Escreve o valor do registo rt (do RF) na Memória.

SC Datapath (5) - Instruções Load/Store (2)

Formato da Instrução - tipo-I

`lw`

35	rs	rt	immediate
31:26	25:21	20:16	15:0

`lw rt, imm(rs)`

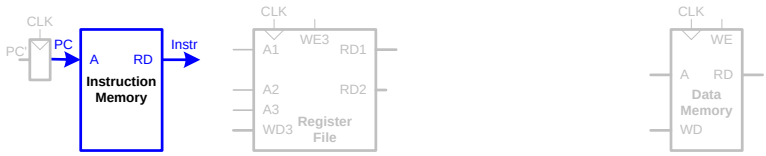
`sw`

43	rs	rt	immediate
31:26	25:21	20:16	15:0

`sw rt, imm(rs)`

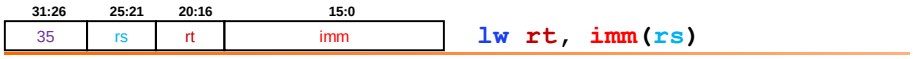
SC Datapath (6) - lw Fetch

Passo 1: Leitura da Instrução (*Fetch*)



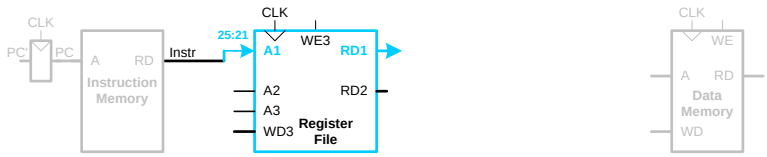
1. O PC gera o endereço (A) para a Memória de Instruções.

A Instrução lida (RD) vai, em seguida, ser descodificada (e.g., o campo de bits $Instr_{31:26}$ vai ser interpretado).

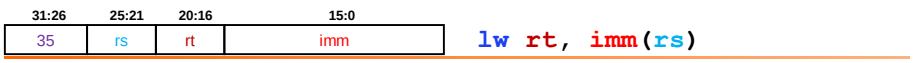


SC Datapath (7) - lw Leitura do Operando

Passo 2: Leitura do operando (rs) do *Reg File* (RF)

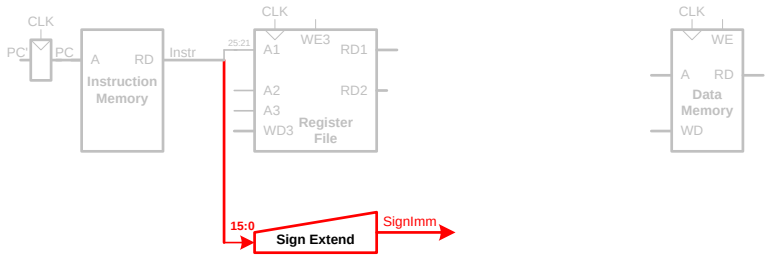


2. Ligando os bits $Instr_{25:21}$,rs, ao porto A1 do RF, obtemos na saída RD1 (ReadData1) o conteúdo desse registo, (rs).



SC Datapath (8) - lw Obtenção do Offset

Passo 3: Extensão de sinal do valor Imediato



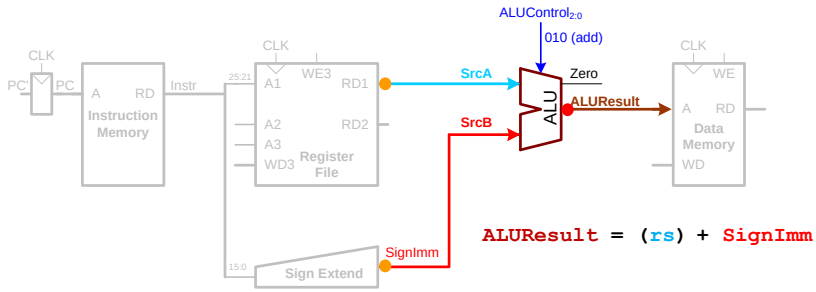
3. Ligando os bits $Instr_{15:0}$ ao extensor de sinal, converte-se o valor $imm_{15:0}$ em $SignImm_{31:0}$ (16 => 32 bits).

31:26	25:21	20:16	15:0
35	rs	rt	imm

`lw rt, imm(rs)`

SC Datapath (9) - lw Cálculo do Endereço

Passo 4: Cálculo do endereço (efetivo) de memória



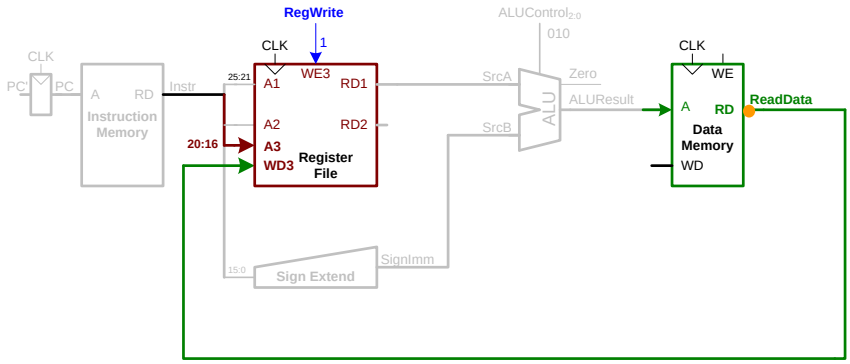
4. Na ALU soma-se o endereço base SrcA ao SrcB, para obter o endereço de memória efetivo em ALUResult.

31:26	25:21	20:16	15:0
35	rs	rt	imm

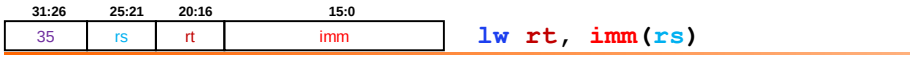
`lw rt, imm(rs)`

SC Datapath (10) - `lw` Leitura da Memória de Dados

Passo 5: Leitura do valor da Memória e escrita no RF



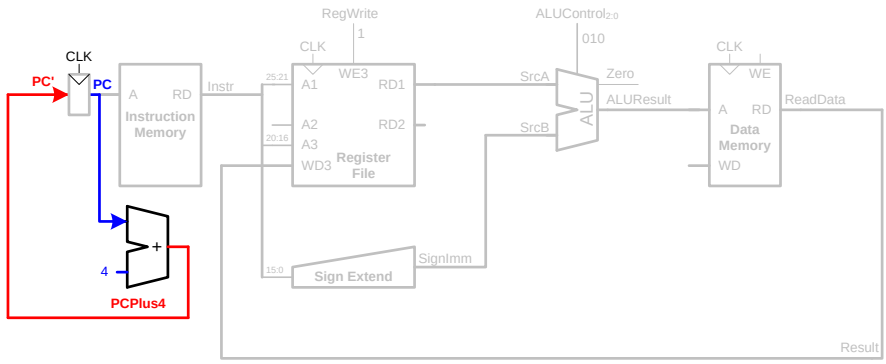
5. O valor lido da memória (ReadData) é escrito (RegWrite=1) no registo rt ($Instr_{20:16}$) usando o porto A3/WD3 do *Register File*.




```
lw rt, imm(rs)
```


SC Datapath (11) - `lw` Incremento do PC

Passo 6: Calcular PC'



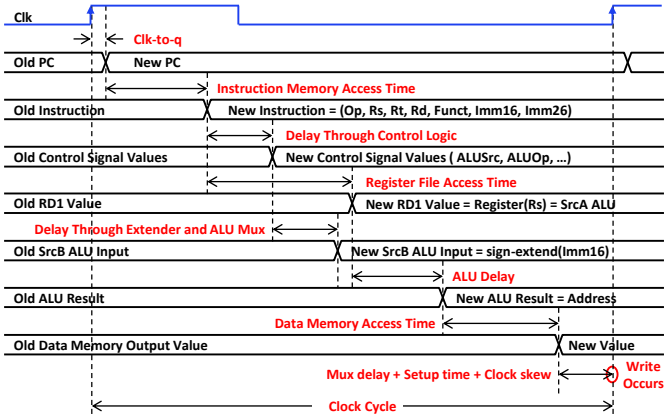
6. Ao valor atual do *Program Counter*, PC, soma-se 4 para obter PC', i.e., o endereço da instrução seguinte a executar.

Acabou a execução da instrução `lw`!

SC Datapath (12) - Passo a Passo?

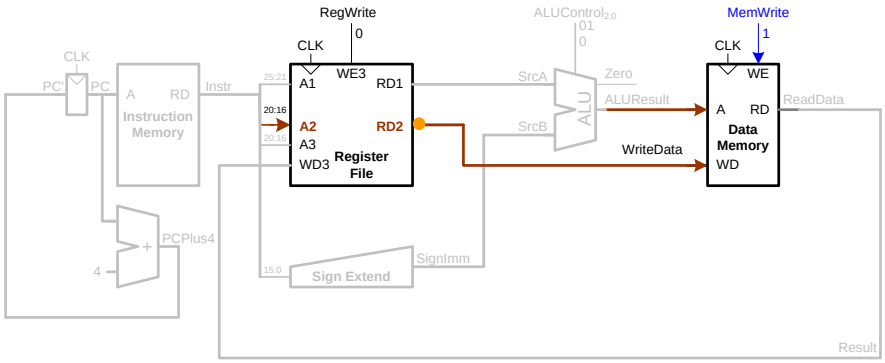
Embora os slides digam 'Passo'

… todas estas operações são executadas no mesmo ciclo de relógio!



SC Datapath (13) - sw Escrever na Memória de Dados

Passo 5: Escrita do valor do registo rt na Memória



5. O valor do registo rt, RD2 (ReadData2), é escrito (MemWrite = 1) na memória (WriteData). Ao contrário de *lw*, nada é escrito no RF.

31:26	25:21	20:16	15:0
43	rs	rt	imm

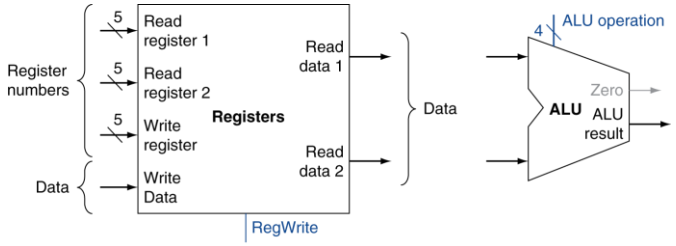
`sw rt, imm(rs)`

SC Datapath (14) - Tipo-R (1) - add

Operações

Ex: add rd, rs, rt

- Usa os registos rs e rt como operandos
- Executa a operação aritmética/lógica na ALU
- Escreve o resultado no registo rd (RF)



a. Registers b. ALU

SC Datapath (15) - Tipo-R (2) - add

Formato da Instrução - tipo-R

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

add rd, rs, rt

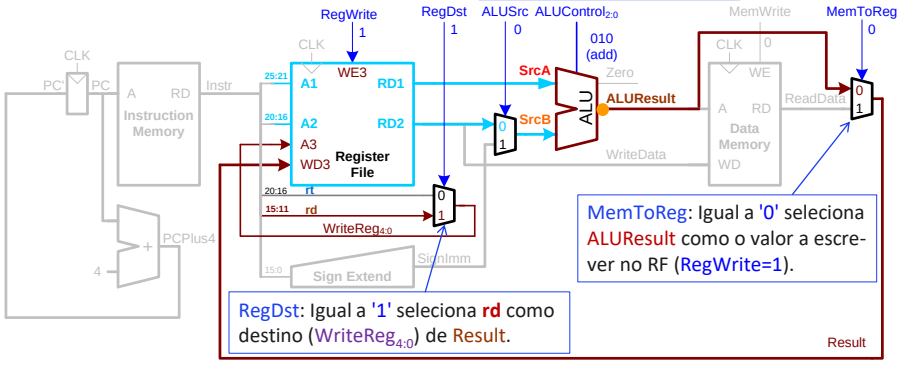
add

0	rs	rt	rd	0	32
31:26	25:21	20:16	15:11	10:6	5:0

SC Datapath (16) - Tipo-R (3) - add

- Lê os valores de rs e rt (add soma-os na ALU)
- Escreve o ALUResult em rd

ALUSrc: Igual a '0' seleciona **RD2** como SrcB da ALU.



MemToReg: Igual a '0' seleciona ALUResult como o valor a escrever no RF (RegWrite=1).

RegDst: Igual a '1' seleciona **rd** como destino ($WriteReg_{4:0}$) de Result.

A Memória de Dados não intervem na execução!

3 Multiplexers!

31:26	25:21	20:16	15:11	10:6	5:0
0	rs	rt	rd	0	32

`add rd, rs, rt`

SC Datapath (17) - Branch (1) - beq

Operações `Ex: beq rs,rt,imm`

3.4 Datapath - Beq

- Calcula o endereço-alvo do *branch* (*BTA*)
  - Obtem o $SignImm_{32}$ a partir do Imm_{16}
  - Multiplica $SignImm_{32}$ por 4
    (para obter o endereço de *byte*)
  - Soma esse valor a 'PC + 4'

    $BTA = (SignImm_{32} << 2) + (PC + 4)$

- Em paralelo, compara os operandos (rs) e (rt)
  - Subtrai-os na ALU e gera a saída Zero;
  - Caso Zero=1 então PC' = BTA;
    caso contrário PC' = PC + 4.

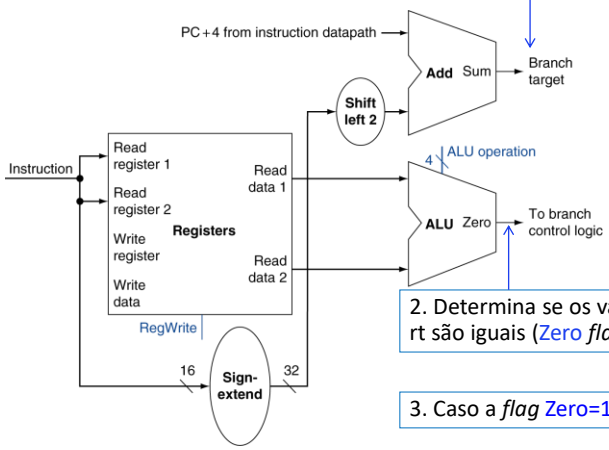
BTA = *Branch Target Address.*

(rs) = conteúdo do registo rs.

SC Datapath (18) - Branch (2) - beq

Operações (cont.)

1. Calcula o *Branch Target Address* (BTA):
   BTA = (SignImm$_{32}$ << 2) + (PC + 4)



2. Determina se os valores de rs e rt são iguais (Zero *flag*)

3. Caso a *flag* Zero=1, PC' = BTA

Os símbolos do *sign-extend e shift-left2* são diferentes mas também se usam noutros livros.

SC Datapath (19) - Branch (3) - beq

Formato da Instrução - tipo-I

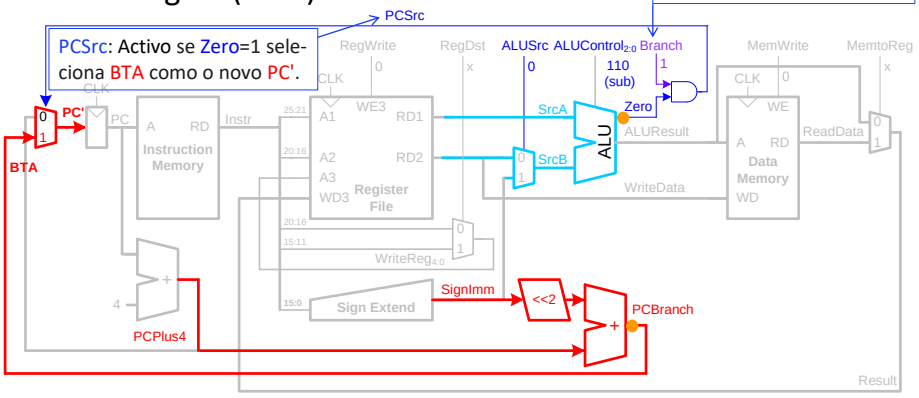
beq	4	rs	rt	imm
	31:26	25:21	20:16	15:0

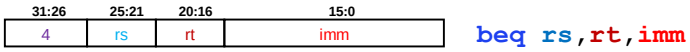
beq rs, rt, imm

SC Datapath (20) - Branch (4) - beq

- Calcula (PCBranch) BTA = (PC+4) + ($SignImm_{31:0}$ << 2)
- Determina (na ALU) se o valor dos registos rs e rt é igual (Zero)

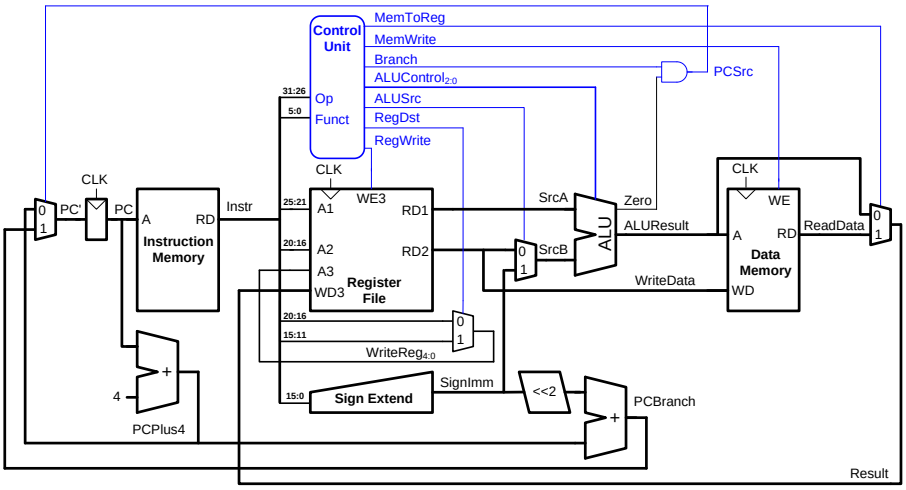
Branch: Sinal de controlo que deteta a instrução beq.






```
beq rs,rt,imm
```


SC Datapath (21) - Completo* (sem Controlo)



*Mas, com um *instruction set* limitado a: lw/sw, tipo-R e beq