Introdução à Arquitetura de Computadores

Aula22

µArquitetura MIPS Single-cycle: I

Introdução

MicroArquiteturas: Implementações

Arquitetura dum CPU MIPS

Datapath e Controlo

Fases de projeto dum CPU

Datapath

Subconjunto de Instruções

Execução de Instruções:

Acesso à Memória de Dados (lw e sw),

Tipo-R (*add, sub*) e

Branch (beq)

µArq (1) - Introdução

- MicroArquitetura
  - Como implementar a arquitetura dum Processador em *hardware*
- Processador
  - Datapath*: Blocos Funcionais

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

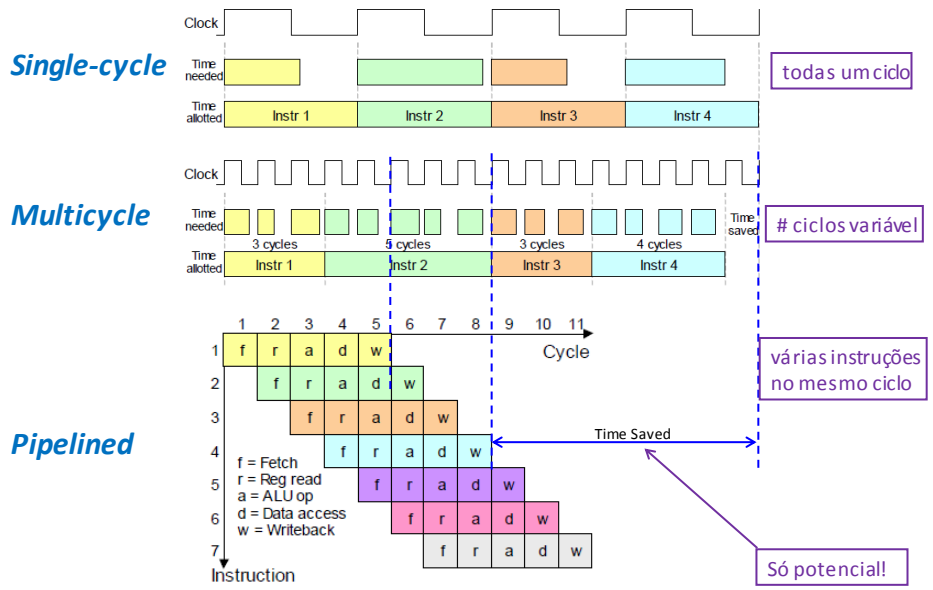
    Determina ('diz') como a instrução deve ser executada no *Datapath*, gerando sinais de seleção de multi-plexers, de *enable* de registos, de *write* de memórias, etc.

*Datapath - Caminho de Dados. Os Blocos e a Unidade de Controlo estão ligados entre si por Buses.

µArq (2) - Implementações Múltiplas

- *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. As instruções requerem 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*.

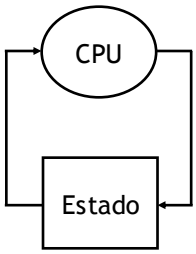
µArq (3) - Single-cycle vs Multicycle vs Pipelined



CPU MIPS (1) - Máquina Síncrona

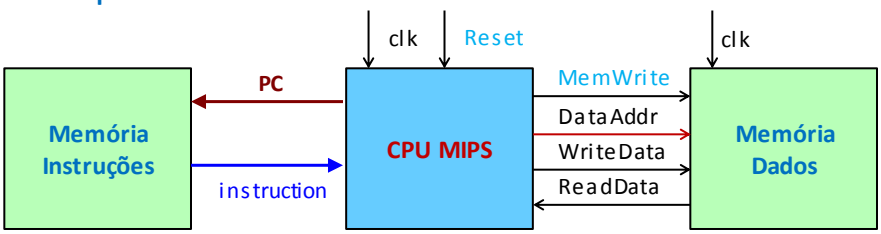
É uma máquina de estados síncrona.

- As Memórias, os Registos e outros autómatos definem o estado.
- O CPU ao executar as instruções dum programa altera o estado.



CPU MIPS (2) - Arquitetura Harvard*

A arquitetura dum CPU

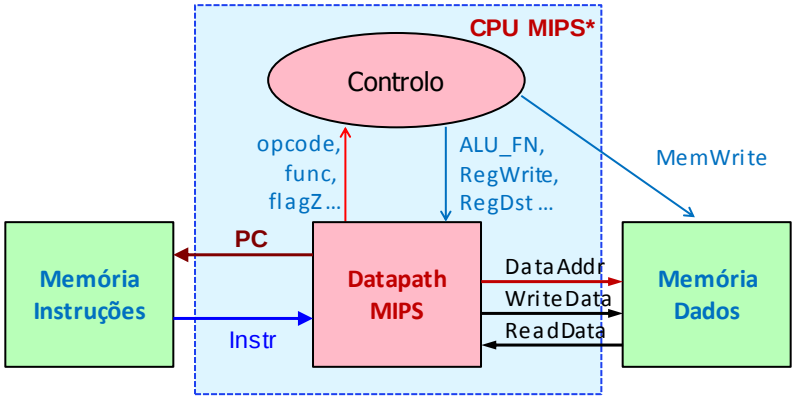


- O CPU MIPS interage com duas memórias.
- Após o *Reset*, o PC é carregado com o endereço da Memória de Instruções, para ler a primeira instrução a ser executada pelo CPU.
- O CPU gera os sinais de controlo necessários à execução, a qual pode ou não envolver a Memória de Dados.

**Harvard* = A implementação *Single-cycle* (Ciclo-único) usa memórias separadas para permitir o acesso, a ambas, em simultaneo (ie, no mesmo ciclo) pelo CPU.

CPU MIPS (3) - Datapath + Controlo

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



- Controlo: Componentes que 'dizem' ao Datapath o que fazer e quando
  - Lógica de controlo combinatória e/ou sequencial (Máq. de Estado - FSMs)

*Consideramos *as* memórias externas ao CPU.

CPU MIPS (4) - Fases de Projeto

- Análise do Conjunto de Instruções
  - A execução de cada instrução requere transferências entre registos e/ou memórias
  - O *Datapath* deve incluir o *hardware* necessário para suportar essas transferências
- Seleção dos Componentes para o Datapath
  - Memórias, Registos, ALU, Multiplexers, etc
- Implementação da Lógica de Controlo
  - Lógica combinatória ou FSMs

SC Datapath (1) - Instruções a Implementar

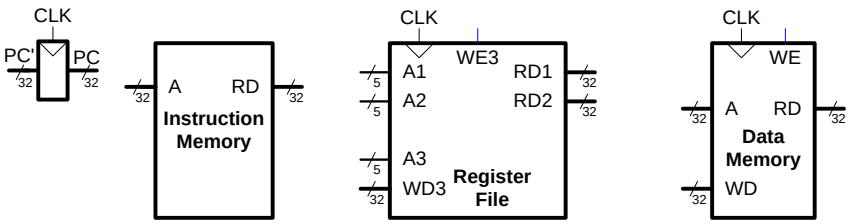
Um número limitado

- Instruções de Acesso à Memória
  `lw`, **`sw`**
- Instruções de Tipo-R
  `and`, **`or`**, **`add`**, **`sub`**, **`slt`**
- Instruções de *Branch*
  `beq`
- Instrução Adicional (prox. aula)
  `addi`
- Instrução Adicional (prox. aula)
  `j`

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

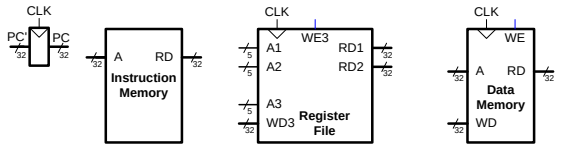
O CPU é construído com base nos seguintes elementos de estado*:

- PC - *Program Counter*
- Banco de 32 Registos (*Register File*)
- Memória(s)



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

SC Datapath (3) - Elementos de Estado (2)



- PC: É um registo normal de 32-bits (com entrada de *Load*).
  A saída (PC) indica o endereço da instrução corrente.
  A entrada (PC') indica o endereço da instrução seguinte.
- Memória de Instruções: Tem um único porto de leitura.
- Banco de Registos (32):
  Tem dois portos de leitura (A1/RD1 e A2/RD2) e um porto de escrita (A3/WD3). (A1, A2 e A3 = 5-bits).
- Memória de Dados: Tem um único porto de leitura (A/RD) ou de escrita (A/WD) e um sinal de *WriteEnable* (WE).

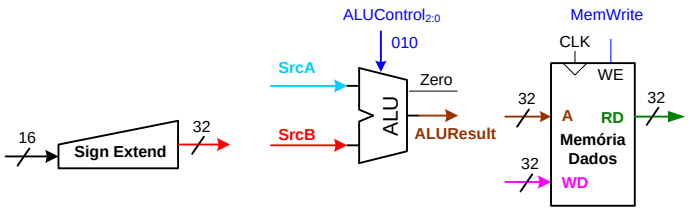
Exceptuando a Memória de Instruções, todos os elementos têm entrada de CLK.

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

Operações

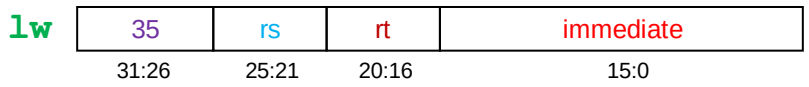
`lw rt, imm(rs)` | `sw rt, imm(rs)`

- Usam os registos rs e rt como operandos
- Calculam (ALU) o endereço efetivo usando rs e o imm
- *Load*: Lê da memória o novo valor do registo rt.
- *Store*: Escreve na memória o valor do registo rt.

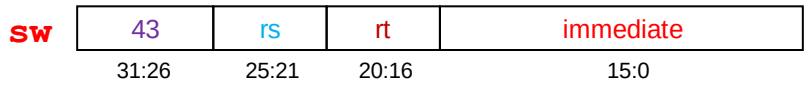


SC Datapath (5) - Instruções Load/Store (2)

Formato da Instrução - Tipo-I



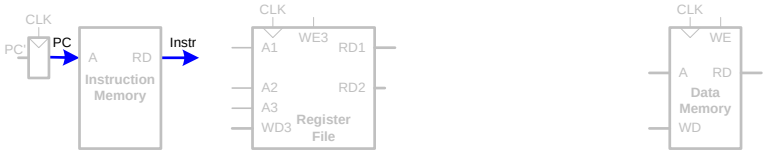

```
lw rt, imm(rs)
```




```
sw rt, imm(rs)
```

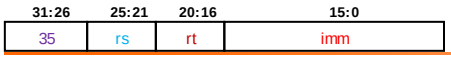

SC Datapath (6) - lw Fetch

Passo 1: Leitura da Instrução (*Fetch*)



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

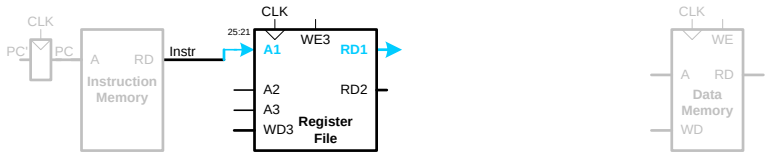
A instrução lida (RD) vai, em seguida, ser descodificada (ie, os campos de bits, eg, Instr25:21, vão ser interpretados).



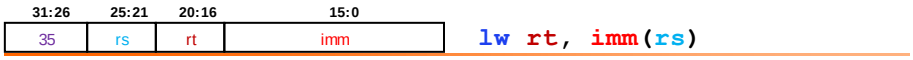

```
lw rt, imm(rs)
```

SC Datapath (7) - lw Leitura do Operando

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

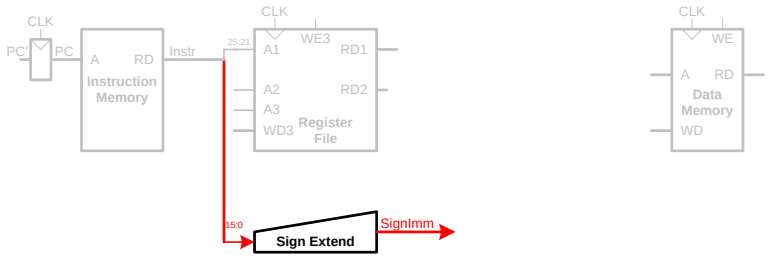


2. Usando os bits Instr25:21, ligados ao porto A1, obtemos em RD1 (ReadData1) o conteúdo do registo rs.

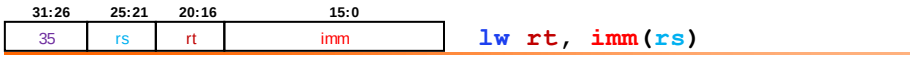


SC Datapath (8) - lw Obtenção do Offset

Passo 3: Extensão de sinal do valor Imediato

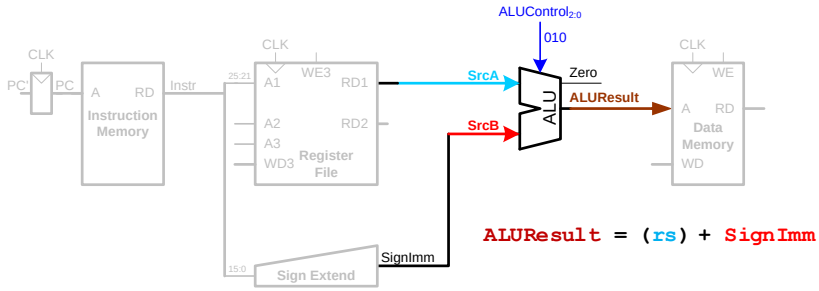


3. Converte-se o valor Imm_{16}, Instr15:0, em $SignImm_{32}$



SC Datapath (9) - lw Cálculo do Endereço

Passo 4: Cálculo do endereço (efetivo) de memória

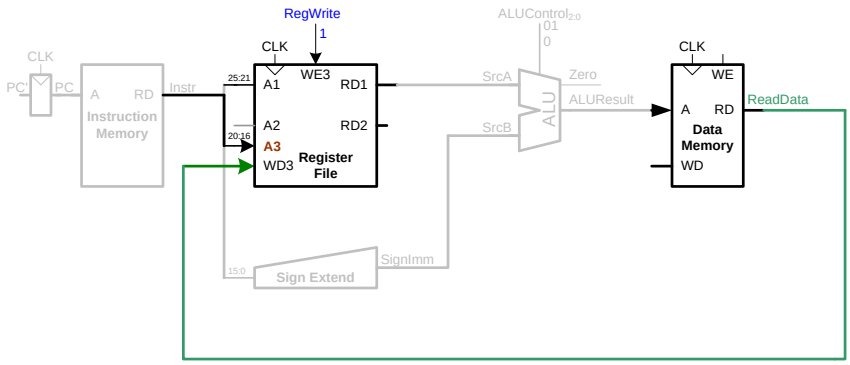


4. Na ALU soma-se o endereço base SrcA ao SrcB, para obter o endereço de memória efetivo em ALUResult.

31:26	25:21	20:16	15:0	
35	rs	rt	imm	lw rt, imm(rs)

SC Datapath (10) - lw Valor da Memória no RF

Passo 5: Escrita do valor da Memória (Dados) no RF

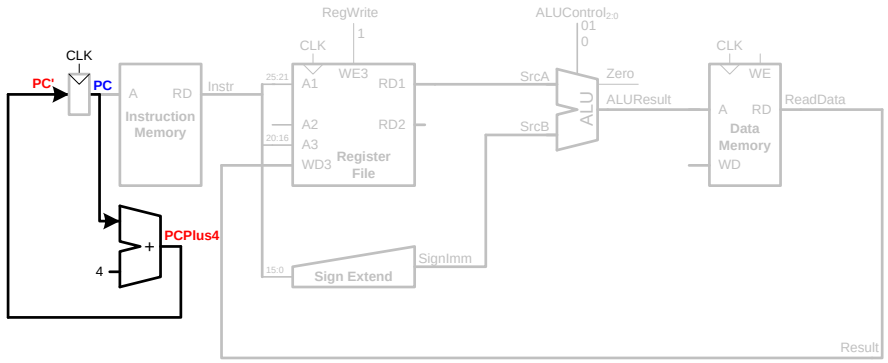


5. O valor lido da memória (ReadData) é escrito (RegWrite=1) no registo rt (Instr20:16) usando o porto A3/WD3 do *Register File*.

31:26	25:21	20:16	15:0	
35	rs	rt	imm	lw rt, imm(rs)

SC Datapath (11) - lw Incrementar o PC

Passo 6: Calcular PC'



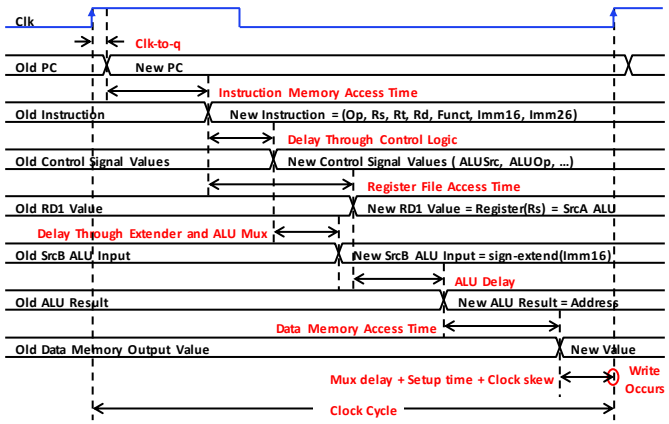
6. Ao valor atual do *Program Counter*, PC, soma-se 4 para obter PC', ie, o endereço da instrução seguinte a executar.

Acabou a execução da instrução **lw** !

SC Datapath (12) - Passo a Passo?

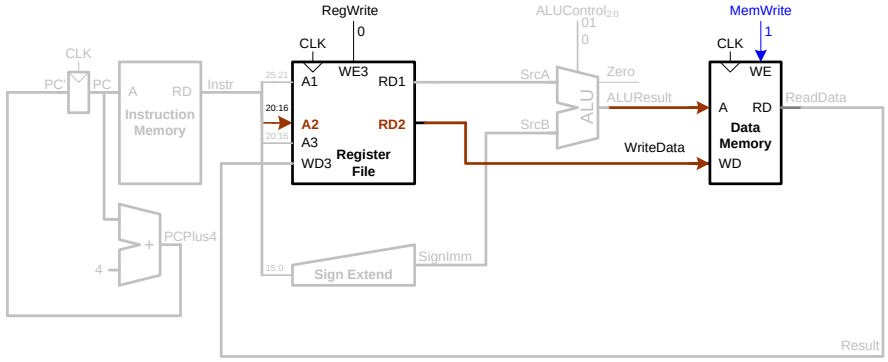
Embora os slides digam 'Passo'

… todas estas operações são executadas no mesmo ciclo de relógio (*clock*)!



SC Datapath (13) - sw Escrever na Mem. de Dados

Passo 5: Escrita do valor do registo rt na Memória



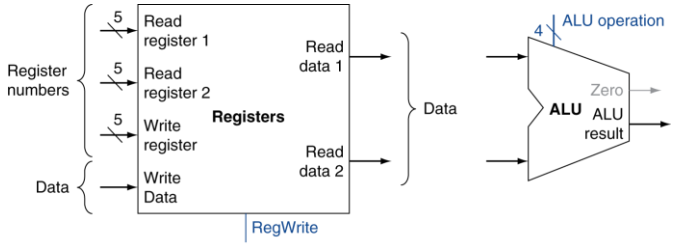
5. O valor do registo rt, RD2 (ReadData2), é escrito (MemWrite = 1) na memória (WriteData). Ao contrário de **lw**, nada é escrito no RF.

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

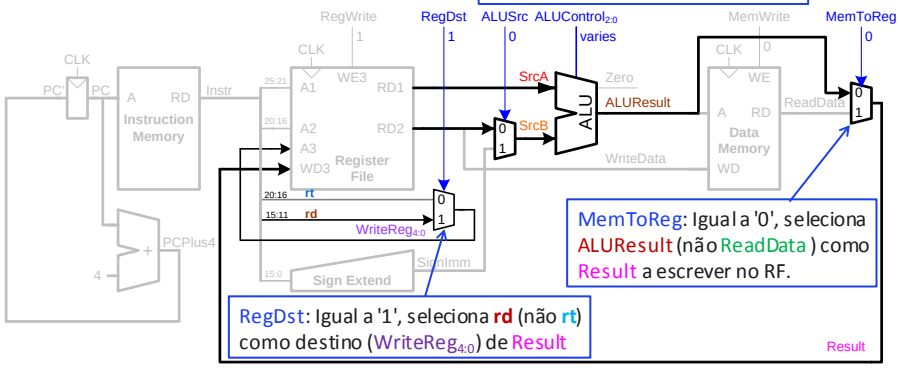
0	rs	rt	rd	0	32
31:26	25:21	20:16	15:11	10:6	5:0

add rd, rs, rt

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** (e não **SignImm**) como SrcB da ALU.



MemToReg: Igual a '0', seleciona ALUResult (não ReadData) como Result a escrever no RF.

RegDst: Igual a '1', seleciona **rd** (não rt) como destino (WriteReg4:0) 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

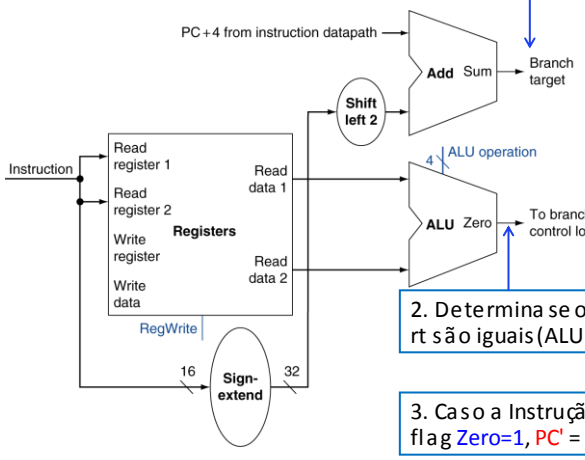
Ex: beq rs,rt,imm

- Usa os registos rs e rt como operandos
- Compara o valor dos operandos
  - Usa a ALU, subtrai-os e gera a saída Zero
- Calcula o endereço-alvo (*Branch Target Address*)
  - Obtem o $SignImm_{32}$ a partir do Imm_{16}
  - Multiplica $SignImm_{32}$ por 4 (ou <<2)
    (para obter o endereço de *byte*)
  - Soma esse valor a 'PC + 4'
  - Caso a Instrução seja beq e a saída Zero=1, PC' = BTA

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

Operações (cont)

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



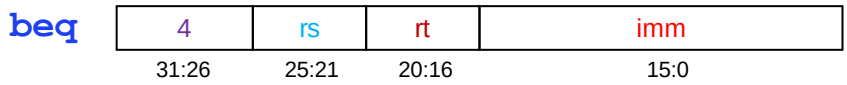
2. Determina se os valores de rs e rt são iguais (ALU Zero Flag)

3. Caso a Instrução seja beq e a flag Zero=1, PC' = BTA.

Os símbolos do *sign-extend* e *shift-left 2* são diferentes mas também se usam noutros livros.

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

Formato da Instrução - Tipo-I



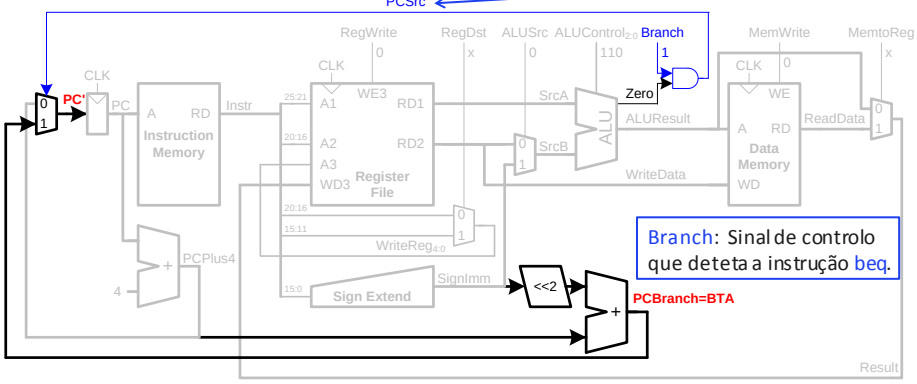
beq rs, rt, imm

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

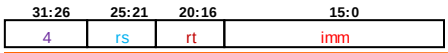
- Determina se o valor dos registos rs e rt é igual (Zero)
- Calcula o endereço-alvo:

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

PCSrc: Activo seleciona BTA (não PC+4) como o novo PC'

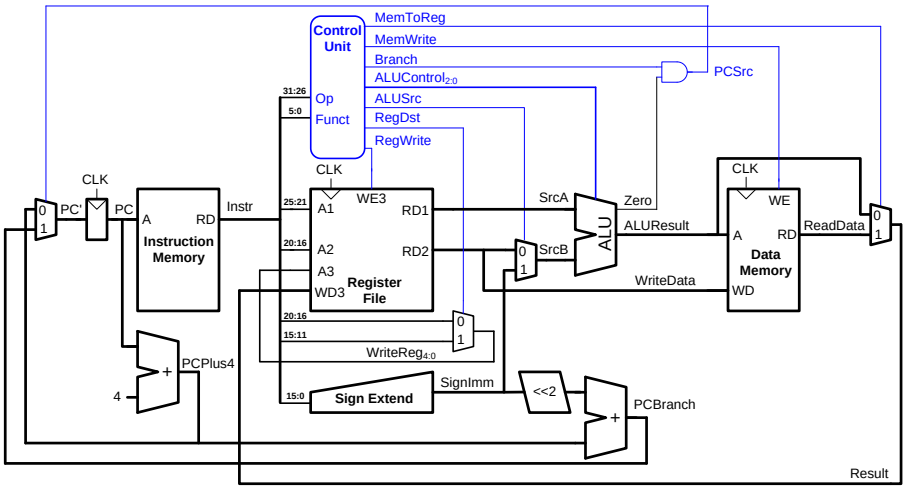


Branch: Sinal de controlo que deteta a instrução beq.



beq rs,rt,imm

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



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