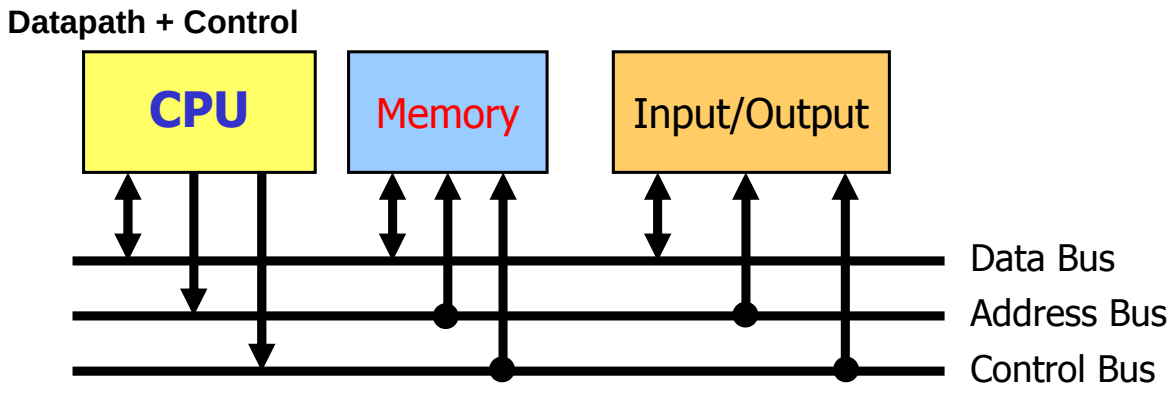
Aulas 16 & 17

- Modelos de Harvard e Von Neumann
- Pressupostos para a construção de um *Datapath* genérico para uma arquitectura tipo MIPS
- Análise dos blocos constituintes necessários à execução de um subconjunto de intruções de cada classe de instruções:
  - Aritméticas e lógicas (add, addi, sub, and, or, slt, slti)
  - Acesso à memória (lw, sw)
  - Controlo de fluxo de execução (beq, bne, j)
- Montagem de um *datapath* completo para execução de instruções num único ciclo de relógio (*single cycle*)

Bernardo Cunha, José Luís Azevedo, Arnaldo Oliveira, Tomás Oliveira e Silva

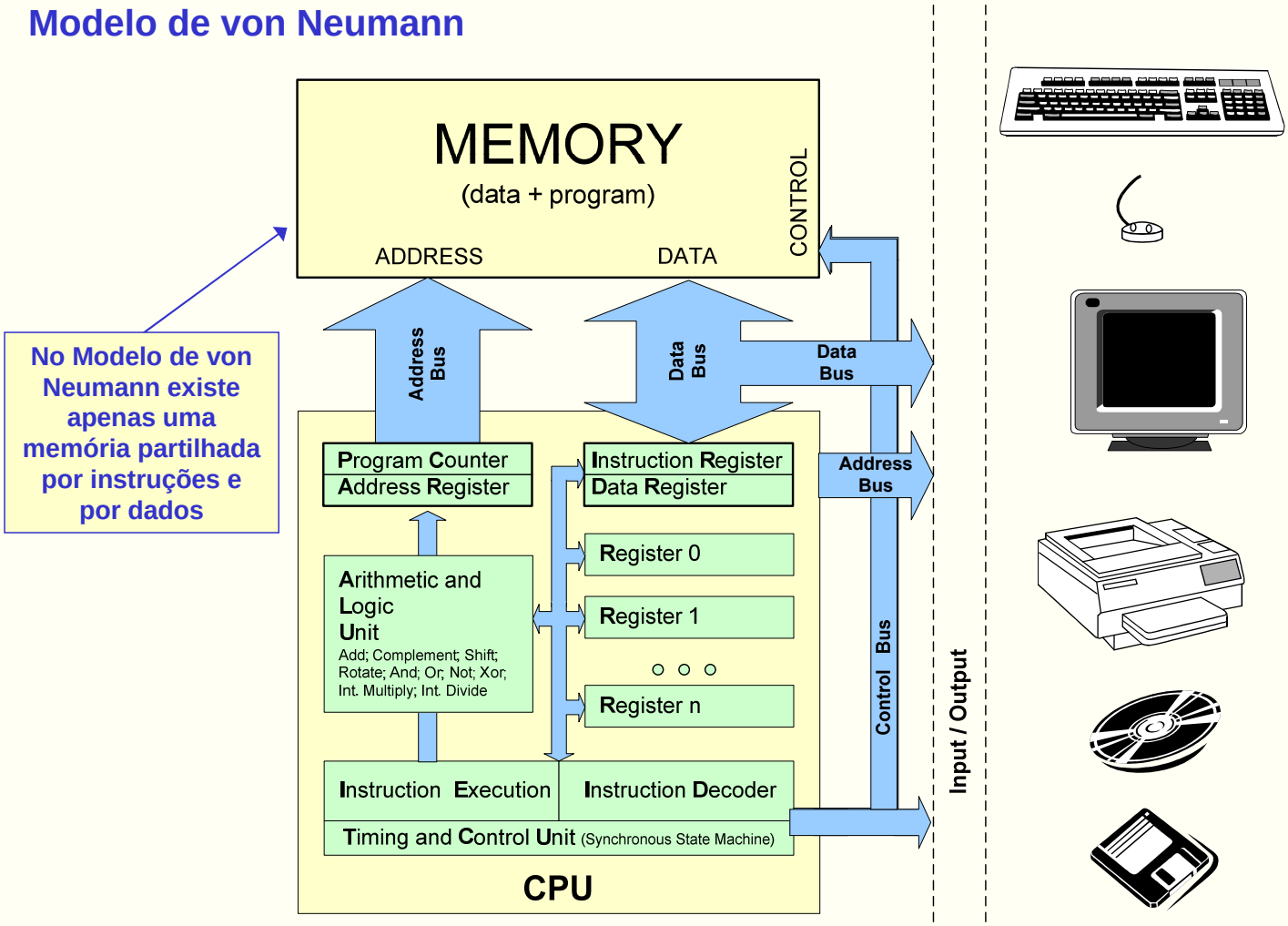
Modelo de **von Neumann**



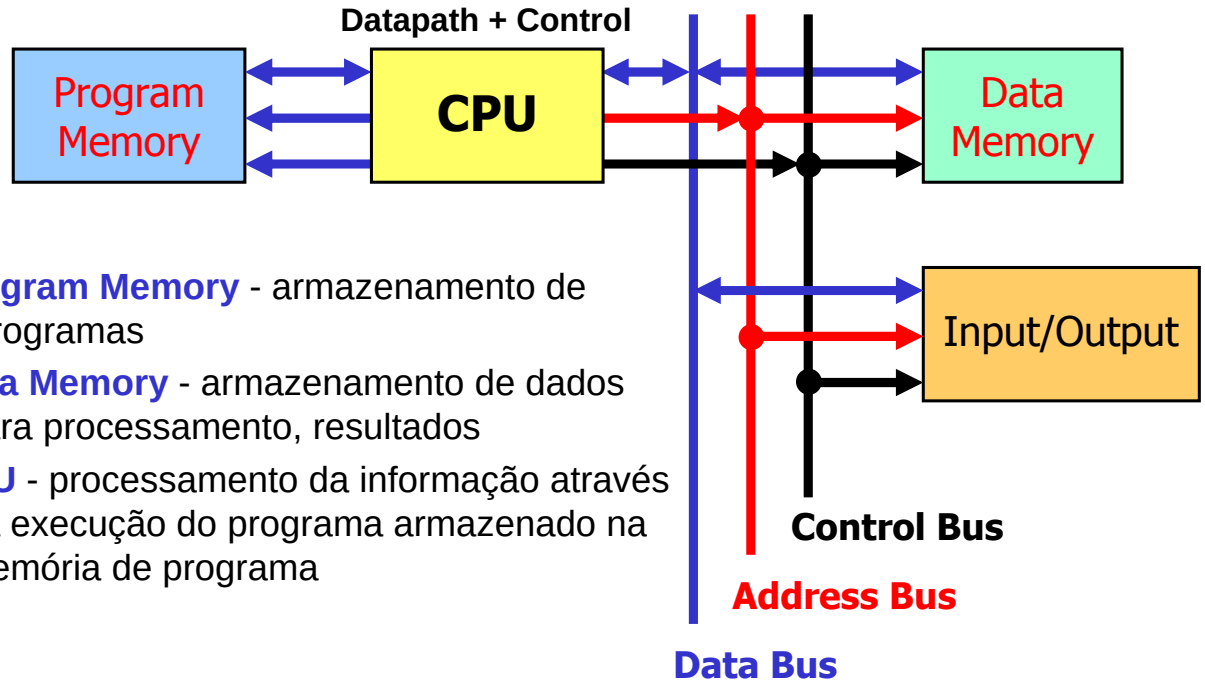
Memory – armazenamento de: programas, dados para processamento, resultados

CPU – processamento da informação através da execução do programa armazenado em memória

Modelo de von Neumann



Modelo de **Harvard**

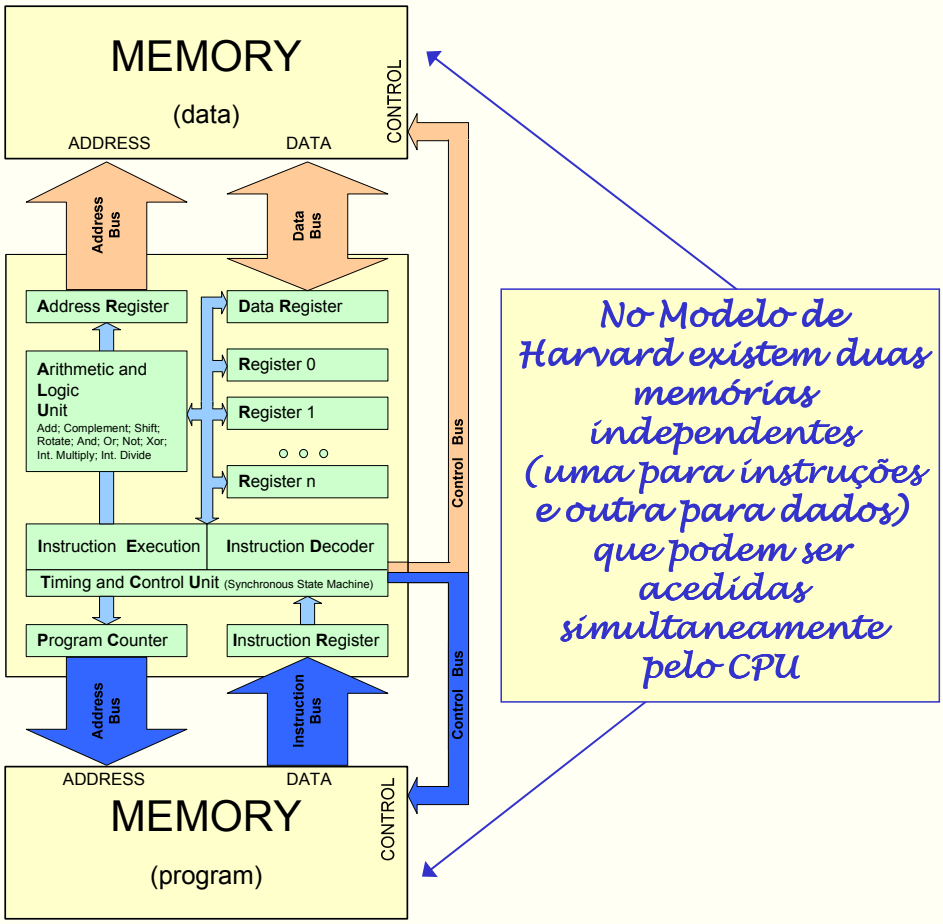


Program Memory - armazenamento de programas

Data Memory - armazenamento de dados para processamento, resultados

CPU - processamento da informação através da execução do programa armazenado na memória de programa

Modelo de Harvard



- **Modelo de Von Neumann**
  - um único espaço de endereçamento para instruções e dados (i.e. uma única memória)
  - acesso a instruções e dados é feito em ciclos de relógio distintos
- **Modelo de Harvard**
  - dois espaços de endereçamento separados: um para dados e outro para instruções (i.e. duas memórias independentes)
  - possibilidade de acesso, no mesmo ciclo de relógio, a dados e instruções (i.e. CPU pode fazer o *fetch* da instrução e ler os dados que a instrução vai manipular no mesmo ciclo de relógio)
  - memórias de dados e instruções podem ter comprimentos de palavra diferentes

Implementação de um *Datapath*

- O CPU consiste, fundamentalmente, em duas secções:
  - ***Datapath*** - elementos operativos e respectiva interligação:
    - Registos internos
    - Unidade Aritmética e Lógica (ALU)
    - Elementos de encaminhamento (*multiplexers*)
  - **Unidade de controlo**: responsável pela coordenação dos elementos do *datapath*, durante a execução de uma instrução

Implementação de um *Datapath*

- As unidades funcionais que constituem o *datapath* são de dois tipos:
  - **Elementos combinatórios** (por exemplo a ALU)
  - **Elementos de estado**, isto é, que têm capacidade de armazenamento (por exemplo os registos internos, a memória*)
- Um elemento de estado possui, pelo menos, duas entradas:
  - Os **dados** a serem armazenados
  - O **relógio**, que determina o instante em que os dados são armazenados (interface síncrona)
- Um elemento de estado pode ser lido em qualquer momento
- A saída de um elemento de estado disponibiliza a informação armazenada na última transição activa do relógio

(*) Na abordagem que se faz a seguir considera-se a memória externa ao CPU como um elemento operativo integrante do *datapath*

Implementação de um *Datapath*

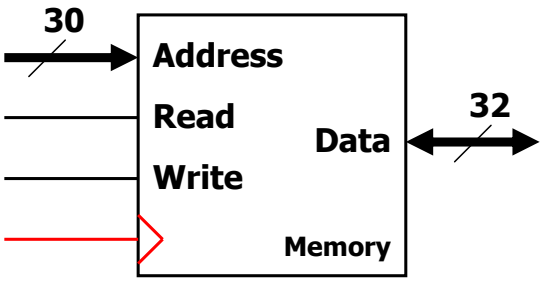
- Para além do sinal de relógio, um elemento de estado pode ainda ter sinais de controlo adicionais:
  - **Um sinal de leitura (*read*)**, que permite (quando activo) que a informação armazenada seja disponibilizada na saída
  - **Um sinal de escrita (*write*)**, que autoriza (quando activo) a escrita de informação na próxima transição activa do relógio
- **Se algum destes dois sinais não estiver explicitamente representado, isso significa que a operação respectiva é sempre realizada**. No caso da operação de escrita ela é realizada uma vez por ciclo, e coincide com a transição activa do sinal de relógio

Havendo um sinal de relógio comum, e por uma questão de simplificação dos diagramas, o sinal de relógio pode não ser explicitamente representado

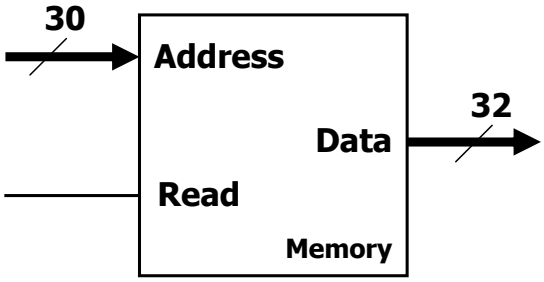
Implementação de um *Datapath*

Exemplos de representação gráfica de blocos funcionais correspondentes a elementos de estado

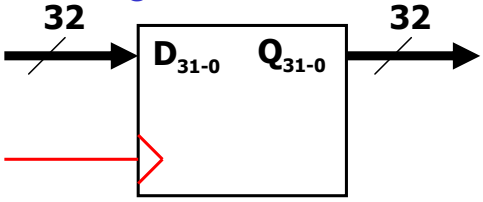
Memória para escrita e leitura (2^{30} words de 32 bits)



Memória apenas para leitura (2^{30} words de 32 bits)



Registo de 32 bits



O sinal "Read" pode não existir. Nesse caso a informação de saída estará sempre disponível e corresponderá ao conteúdo da posição de memória especificada na entrada "address"

Implementação de um *Datapath*

- Nestes slides faz-se uma abordagem à implementação de um *datapath* capaz de interpretar e executar o seguinte subconjunto de instruções do MIPS:
  - As instruções aritméticas e lógicas (**`add`**, **`addi`**, **`sub`**, **`and`**, **`or`**, **`slt`**, **`slti`**)
  - Instruções de acesso à memória: *load word* (**`lw`**) e *store word* (**`sw`**)
  - As instruções de salto condicional (**`beq`**, **`bne`**) e salto incondicional (**`j`**)
- Como iremos ver, independentemente da quantidade e tipo de instruções suportadas por uma dada arquitectura, **uma parte importante do trabalho realizado pelo CPU e da infra-estrutura necessária para executar essas instruções é comum a praticamente todas elas**

Implementação de um *Datapath*

- No caso particular do MIPS, para qualquer instrução que compõe o *set* de instruções, **as duas primeiras operações necessárias à sua realização são sempre as mesmas**:
  1. Usar o conteúdo do registo *Program Counter* (PC) para indicar o endereço da memória do qual vai ser lida a próxima instrução e efectuar essa leitura
  2. Ler um ou mais registos internos, usando para isso os índices obtidos nos respectivos campos da instrução (rs e rt):
     a) Nas instruções de transferência memória→registo (“lw”) e nas instruções que operam com constantes (imediatos) apenas o conteúdo de um registo é necessário
     b) Em todas as outras é sempre necessário o conteúdo de dois registos (excepto na instrução “jump”)
- **Depois destas operações genéricas, realizam-se as acções específicas para completar a execução da instrução em causa**

Implementação de um *Datapath*

- As acções específicas necessárias para executar as instruções de cada uma das três classes de instruções descritas anteriormente são, em grande parte, semelhantes, independentemente da instrução exacta em causa
- Por exemplo, **todas as classes de instruções** (à excepção do salto incondicional) **utilizam a ALU depois da leitura dos registos**:
  - as instruções aritméticas e lógicas para a execução da instrução
  - as instruções de acesso à memória usam a ALU para calcular o endereço de memória
  - a instrução de *branch* para efectuar a subtracção que permite determinar se os operandos são iguais ou diferentes
- A execução da instrução de salto incondicional ("**j**") resume-se à alteração incondicional do registo Program Counter (PC) – o novo valor é obtido a partir dos 26 LSB do código máquina da instrução (ver aula 5)

Implementação de um *Datapath*

- Depois de utilizar a ALU, as acções que completam as várias classes de instruções diferem:
  - as instruções **aritméticas e lógicas** armazenam o resultado da ALU no registo destino especificado na instrução
  - a instrução ***sw*** acede à memória para escrita do valor do registo lido anteriormente
  - a instrução ***lw*** acede à memória para leitura; o valor lido da memória é, de seguida, escrito no registo destino especificado na instrução
  - a instrução de ***branch*** pode ter que alterar o conteúdo do registo Program Counter (i.e. o endereço onde se encontra a próxima instrução a ser executada)

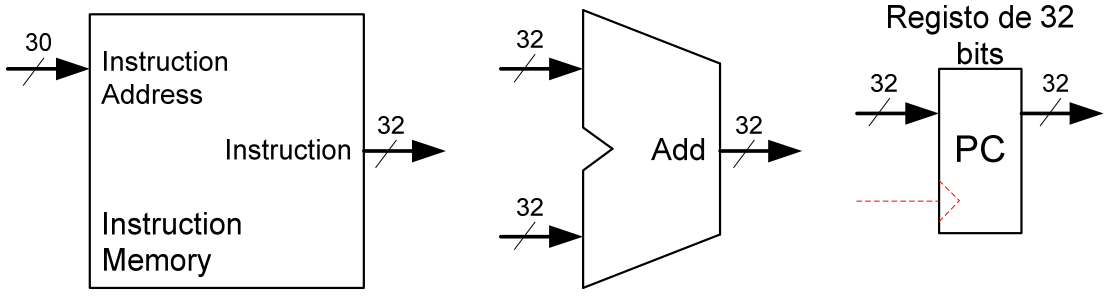
Implementação de um *Datapath* - *Instruction Fetch*

- O processo de acesso à memória para leitura da próxima instrução é genericamente designado por ***Instruction Fetch***
- Por uma questão de simplificar a organização da informação, as instruções que compõem um programa são armazenadas sequencialmente na memória:
  - se a instrução ***n*** se encontra armazenada no endereço ***k***, então a instrução ***n+1*** encontra-se armazenada no endereço ***k+x***, em que ***x*** é a dimensão da instrução ***n,*** medida em bytes
- No MIPS, a dimensão das instruções é fixa e igual a 4 bytes; o endereço ***k*** é sempre um múltiplo de 4
- **O processo de *Instruction Fetch* deverá, uma vez concluído, deixar o conteúdo do PC pronto para endereçar a próxima instrução**
  - No caso do MIPS, tal corresponde a adicionar a constante 4 ao valor actual do PC

Implementação de um *Datapath* - *Instruction Fetch*

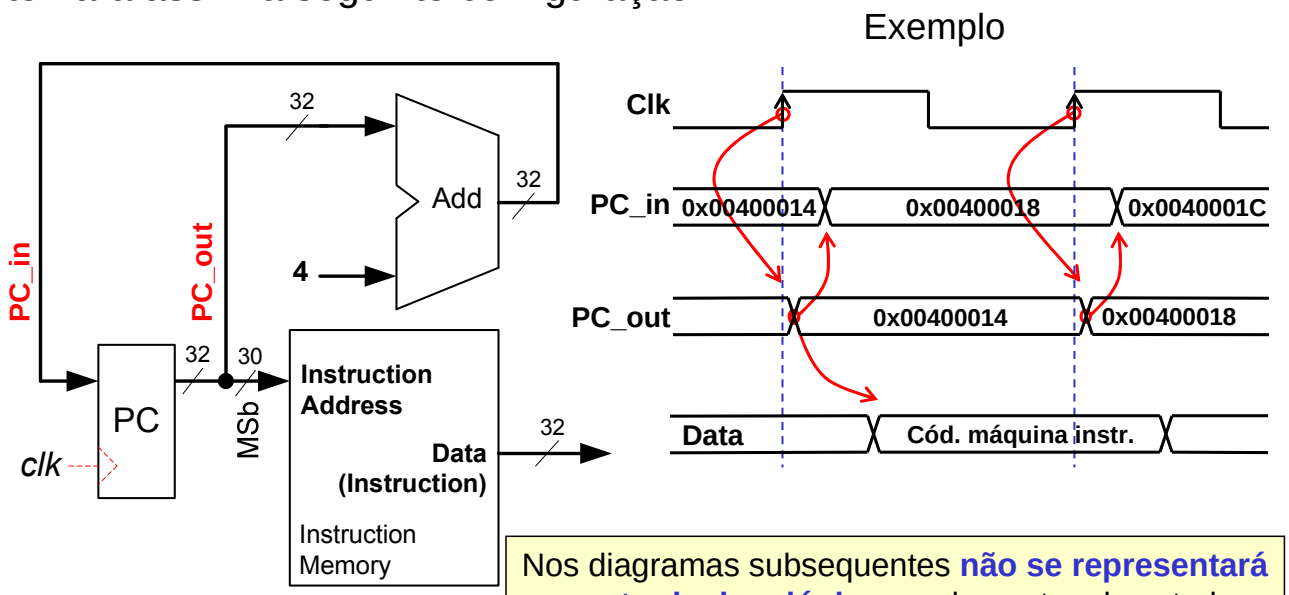
Os elementos operativos (combinatórios e/ou de memória) necessários à implementação de uma operação de *Instruction Fetch* serão portanto:

- A **memória** (de código)
- O ***Program Counter*** (um registo de 32 bits dos quais só são aproveitados os 30 mais significativos)
- Um **somador**



Implementação de um *Datapath* - *Instruction Fetch*

A parte do *Datapath* necessária à execução de um *Instruction Fetch* tomará assim a seguinte configuração:



Nos diagramas subsequentes **não se representará a entrada de relógio** nos elementos de estado

Implementação de um *Datapath*

Que outros elementos operativos básicos serão necessários para suportar a execução das várias classes de instruções que estamos a considerar?

- Instruções aritméticas e lógicas
  - Tipo R: **add**, **sub**, **and**, **or**, **slt**
  - Tipo I: **addi**, **slti**
- Instruções de leitura e escrita da memória (Tipo I: **lw**, **sw**)
- Instruções de salto condicional (Tipo I: **beq**, **bne**)

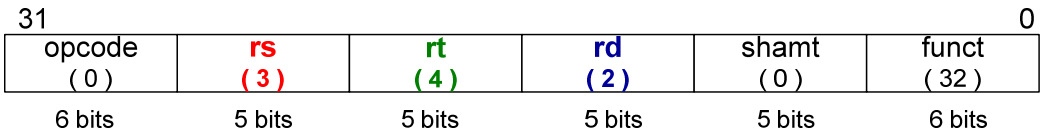
Na análise que se segue, não se explicita a Unidade de Controlo. Esta unidade é responsável pela geração dos sinais de controlo que asseguram a coordenação dos elementos do *datapath* durante a execução de uma instrução

Implementação de um *Datapath* *(Instruções tipo R)*

Operações realizadas na execução de uma instrução do tipo R:

- *Instruction Fetch* (leitura da instrução, cálculo de PC+4)
- Leitura dos registos operandos (registos especificados nos campos "rs" e "rt" da instrução)
- Realização da operação na ALU (especificada no campo "funct")
- Escrita do resultado no registo destino (especificado no campo "rd")

Exemplo: **add $2, $3, $4**

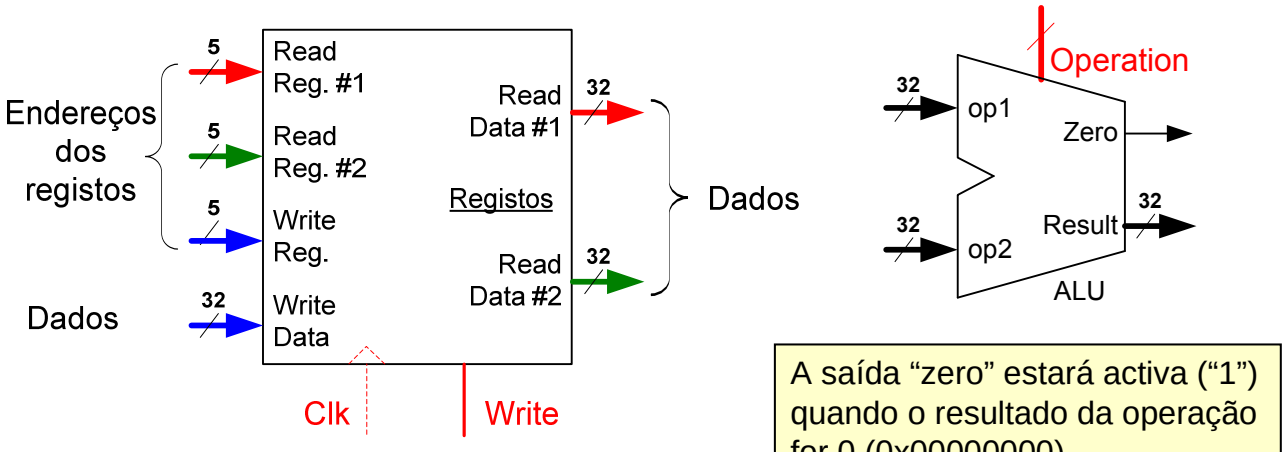


Código máquina: 0x00641020

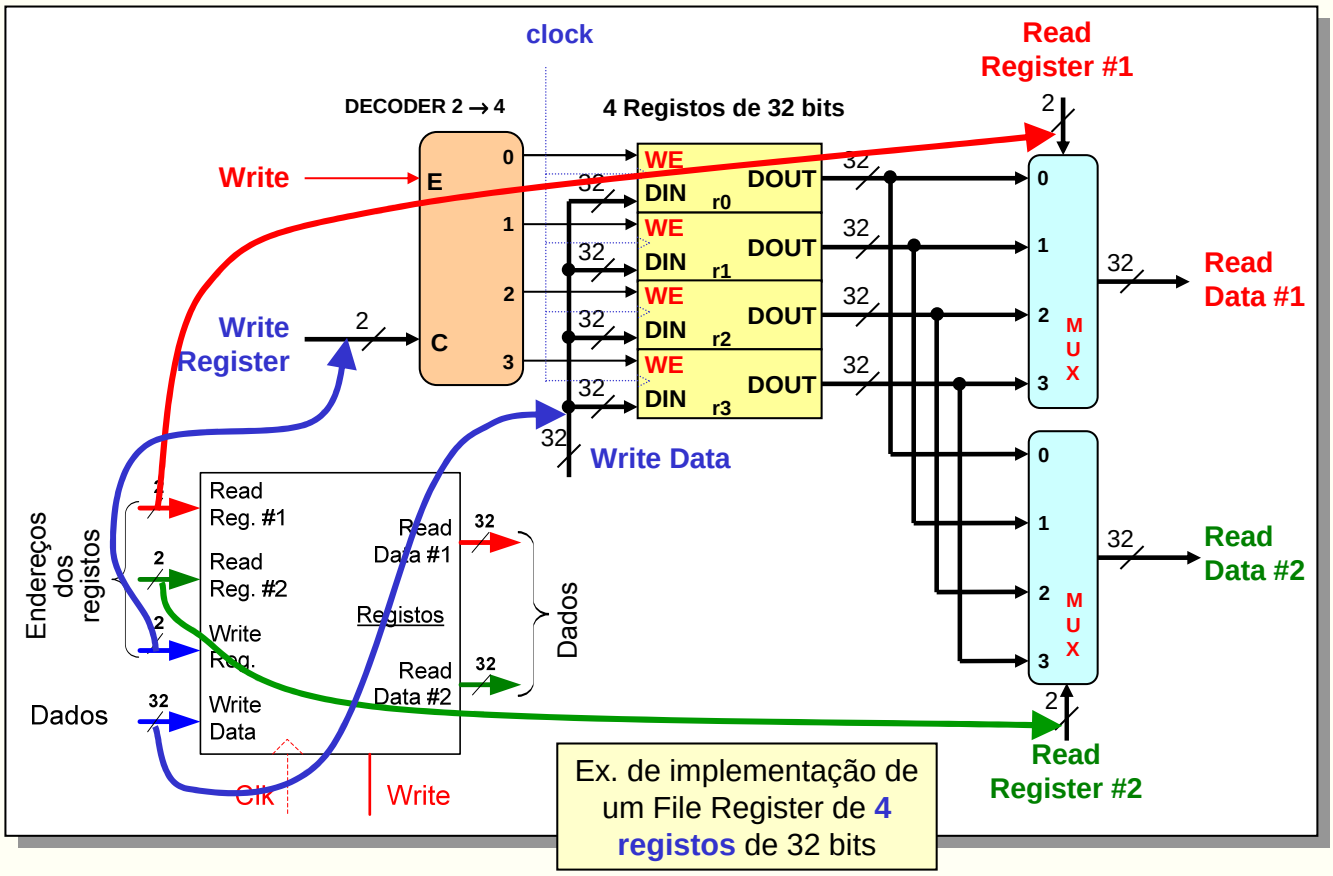
Implementação de um *Datapath* *(Instruções tipo R)*

Os elementos necessários à execução das instruções aritméticas e lógicas (tipo R) são:

- Uma ALU de 32 bits
- Um conjunto de registos internos (***File Register*** com 32 registos)



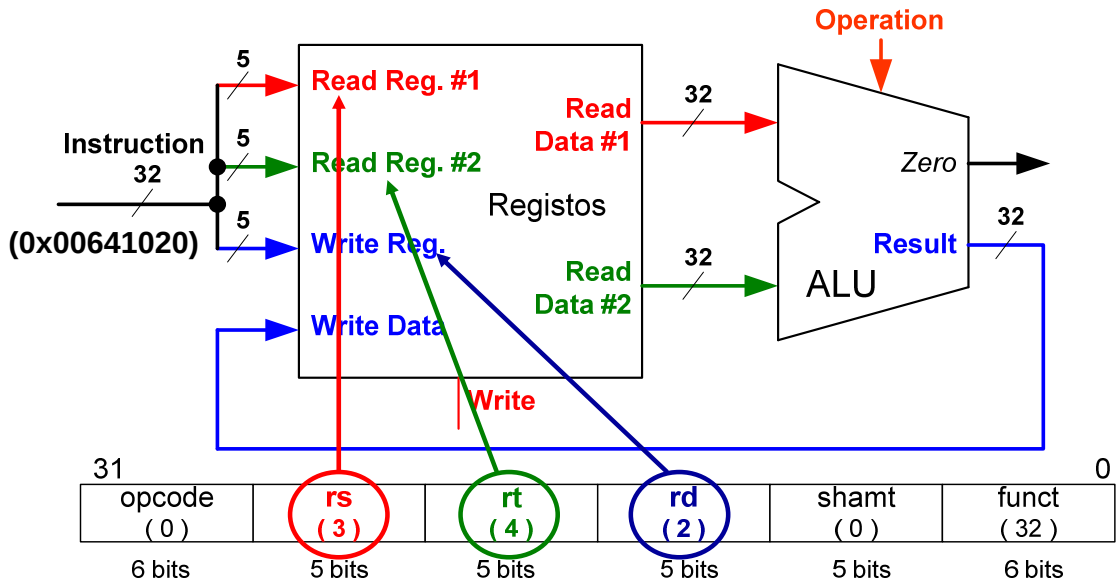
A saída "zero" estará activa ("1") quando o resultado da operação for 0 (0x00000000)



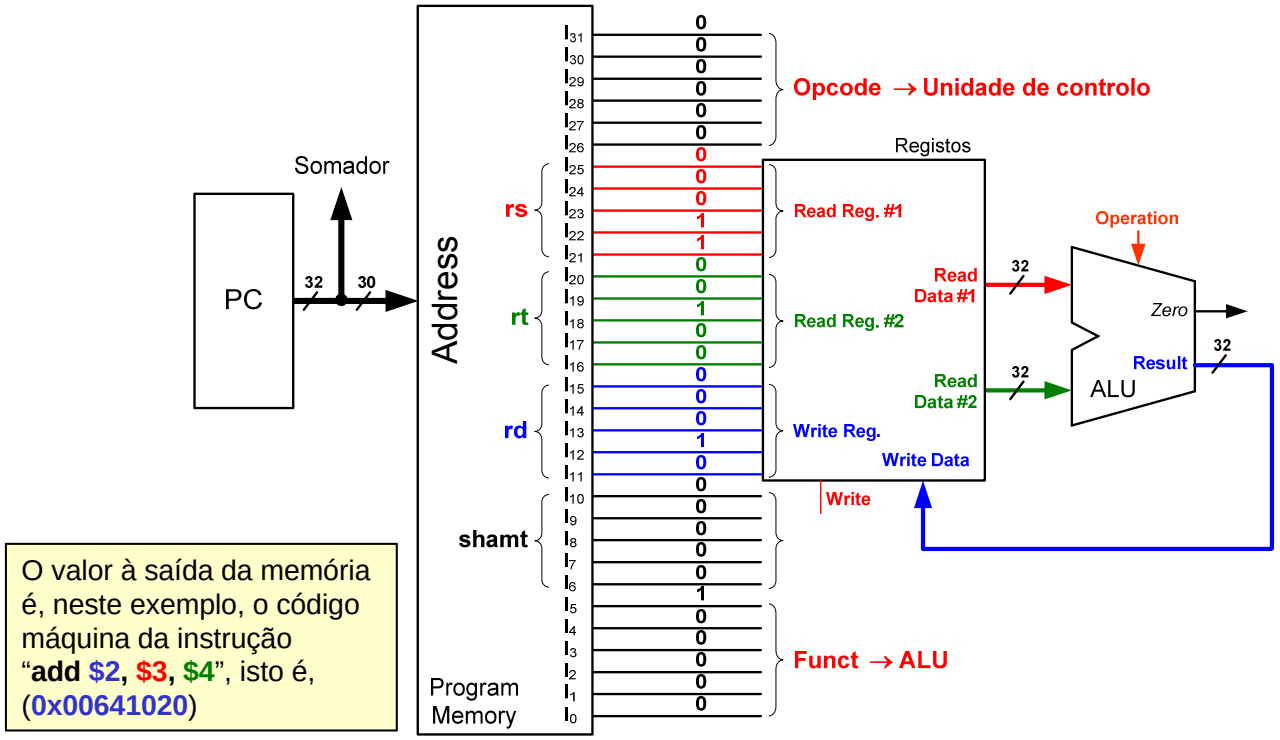
Implementação de um *Datapath* *(Instruções tipo R)*

A interligação dos elementos operativos será:

Exemplo: **add $2, $3, $4**



Ligação entre a memória de código e o File Register *(Instruções tipo R)*



O valor à saída da memória é, neste exemplo, o código máquina da instrução "**add $2, $3, $4**", isto é, (**0x00641020**)

Implementação de um *Datapath* *(Instrução SW)*

Operações realizadas na execução de uma instrução "**sw**":

- *Instruction Fetch* (leitura da instrução, cálculo de PC+4)
- Leitura dos registos que contêm o **endereço-base** e o **valor a transferir** (reg. especificados nos campos "**rs**" e "**rt**"da instrução)
- Cálculo, na ALU, do endereço de acesso (soma algébrica entre o conteúdo do registo "**rs**" e o ***offset*** especificado na instrução)
- Escrita na memória

Endereço da memória

Exemplo: **sw $2, 0x24($4)**

opcode (43)	rs (4)	rt (2)	offset (0x24)

Implementação de um *Datapath* *(Instrução LW)*

Operações realizadas na execução de uma instrução "**lw**"

- *Instruction Fetch* (leitura da instrução, cálculo de PC+4)
- Leitura do registo que contém o endereço base (registo especificado no campo "rs" da instrução)
- Cálculo, na ALU, do endereço de acesso (soma algébrica entre o conteúdo do registo "**rs**" e o ***offset*** especificado na instrução)
- Leitura da memória
- Escrita do valor lido da memória no registo destino (especificado no campo "**rt**" da instrução)

Endereço da memória

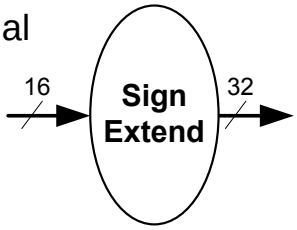
Exemplo: **lw $4, 0x2F($15)**

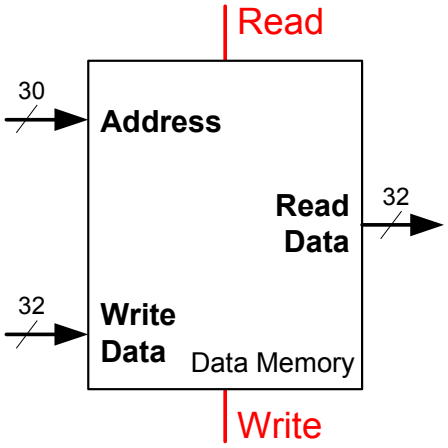
opcode (35)	rs (15)	rt (4)	offset (0x2F)

Implementação de um *Datapath* *(Instruções lw e sw)*

Os elementos necessários à execução das instruções de transferência de informação entre registos e memória (*load* e *store*) são, para além da ALU e do *File Register*:

- A memória externa (de dados)
- Um extensor de sinal





O extensor de sinal cria uma constante de 32 bits em complemento para 2, a partir dos 16 bits menos significativos da instrução (o bit 15 é replicado nos 16 mais significativos da constante de saída)

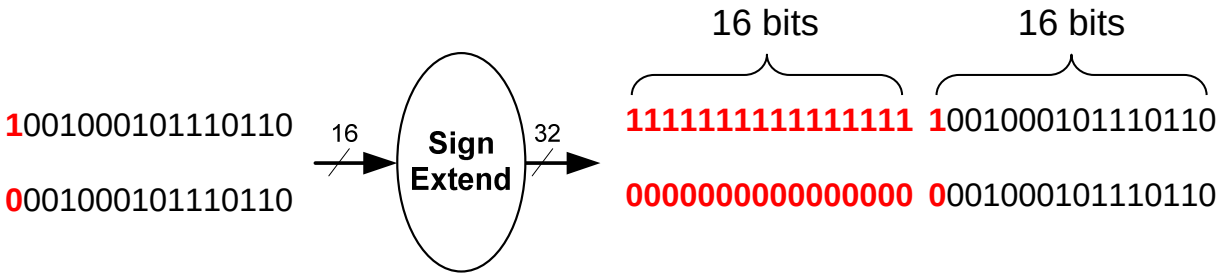
Por uma questão de conveniência de desenho dos diagramas, o barramento de dados da memória (bidireccional) está separado em entrada e saída

Implementação de um *Datapath* *(Instruções lw e sw)*

Os elementos necessários à execução das instruções de transferência de informação entre registos e memória (*load* e *store*) são, para além da ALU e do *File Register*:

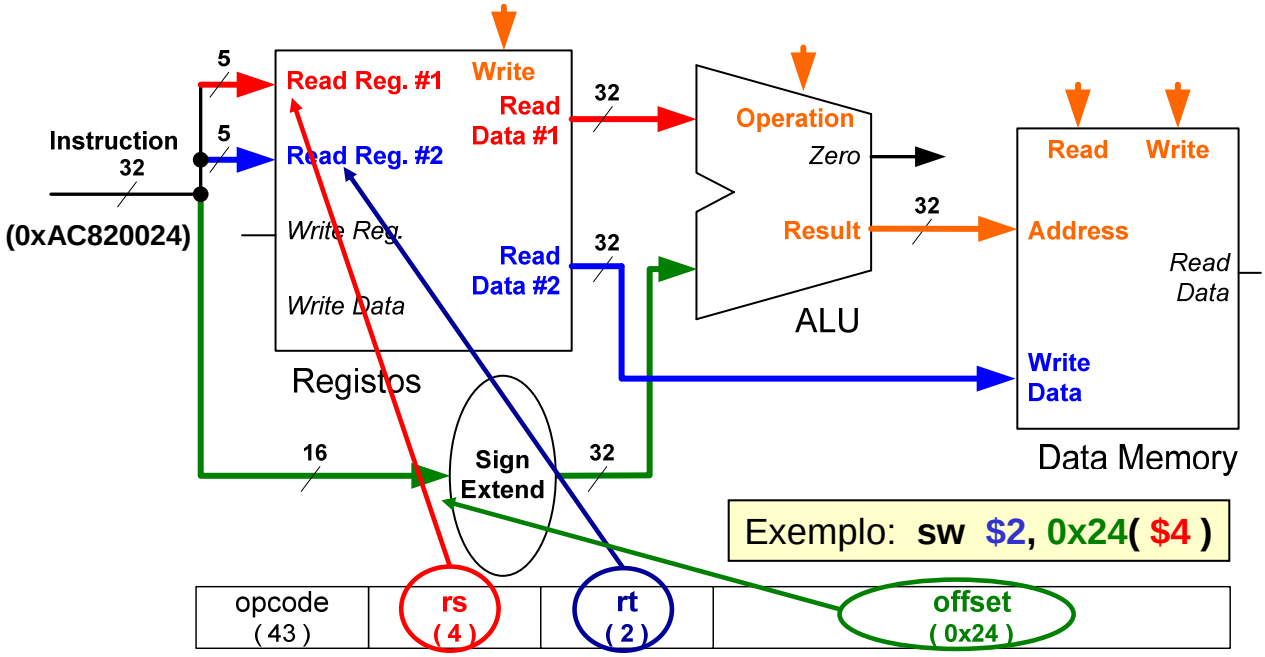
- A memória externa (de dados)
- **Um extensor de sinal**

Exemplo:



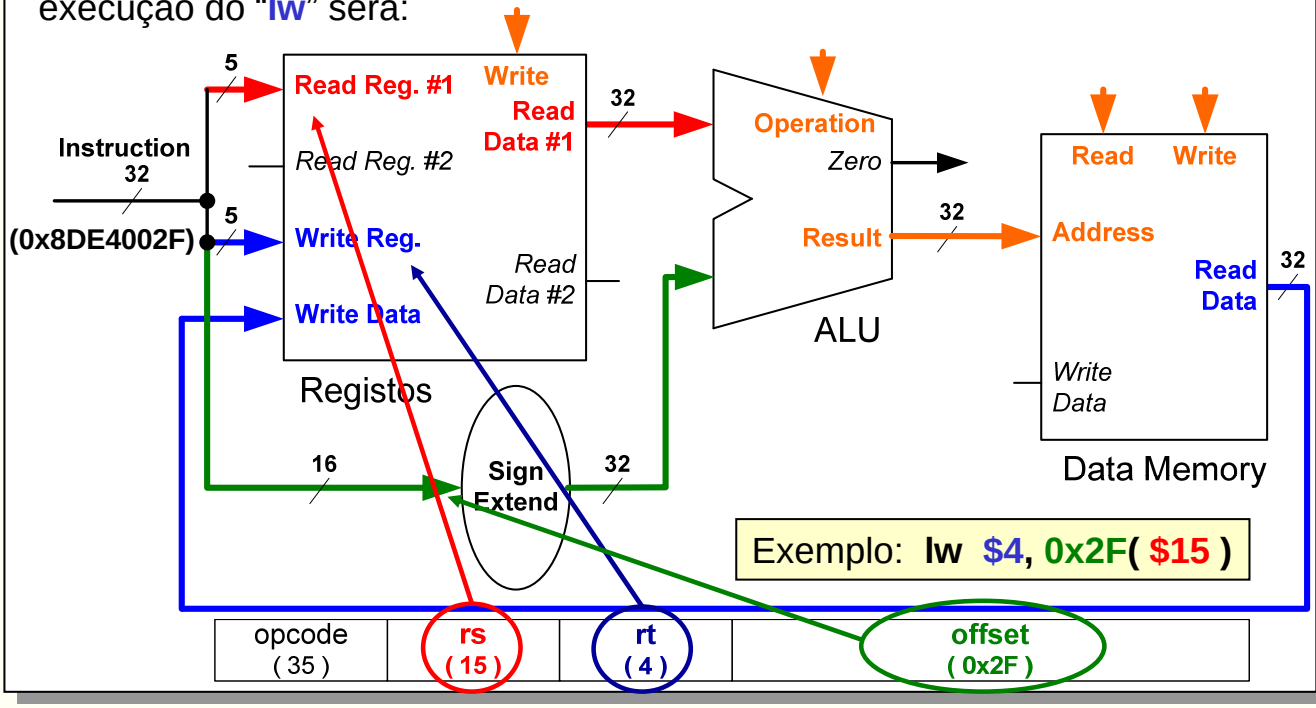
Implementação de um *Datapath* *(Instruções lw e sw)*

A interligação dos elementos operativos necessários à execução do "**sw**" será:



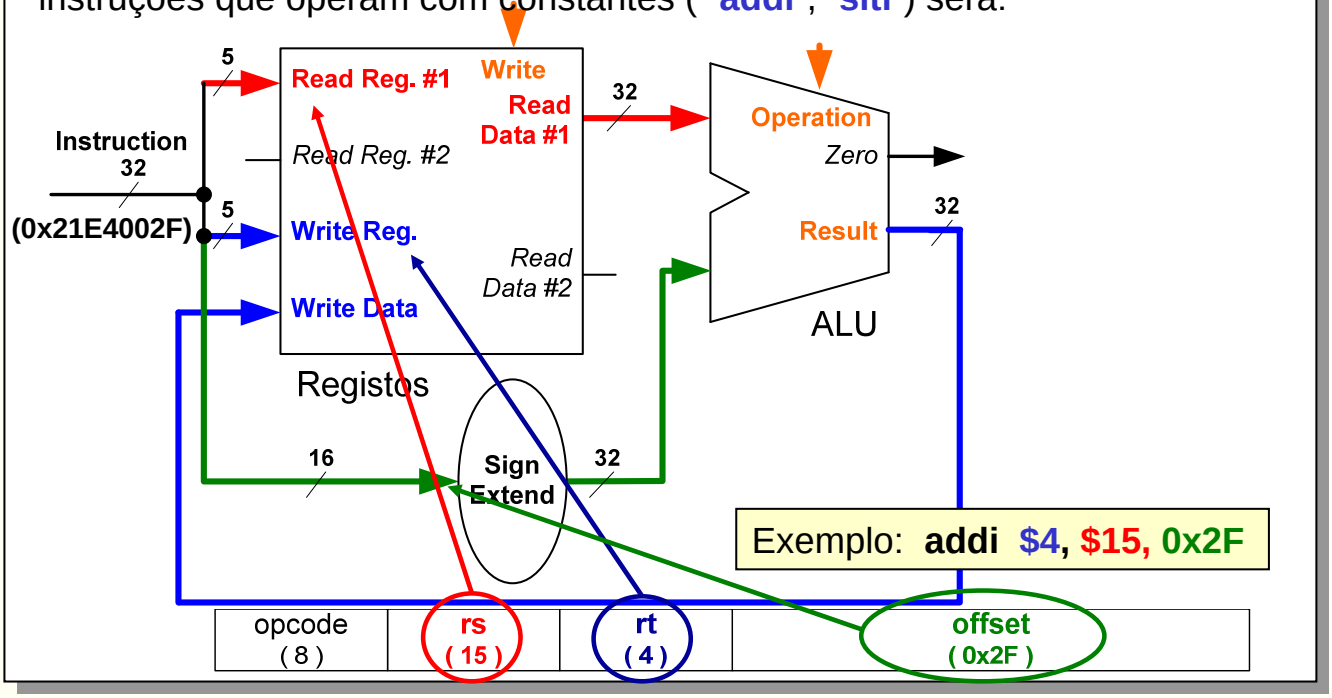
Implementação de um *Datapath* *(Instruções lw e sw)*

A interligação dos elementos operativos necessários à execução do "**lw**" será:



Implementação de um *Datapath* *(Instruções "imediatas")*

A interligação dos elementos operativos necessários à execução de instruções que operam com constantes ("**addi**", "**slti**") será:



Implementação de um *Datapath* *(Instruções de branch)*

Operações realizadas na execução de uma instrução de *branch*:

- *Instruction Fetch* (leitura da instrução, cálculo de PC+4)
- Leitura dos registos a comparar
- Comparação dos valores dos registos (realização de uma operação de subtracção na ALU)
- Cálculo do endereço-alvo da instrução de *branch*
  (*Branch Target Address* - BTA) - ver aula 5

  BTA = (PC + 4) + (*instruction_offset* * 4)
- Alteração do valor do registo PC:
  - se a condição testada pelo *branch* for verdadeira PC = BTA
  - se a condição testada pelo *branch* for falsa PC = PC + 4

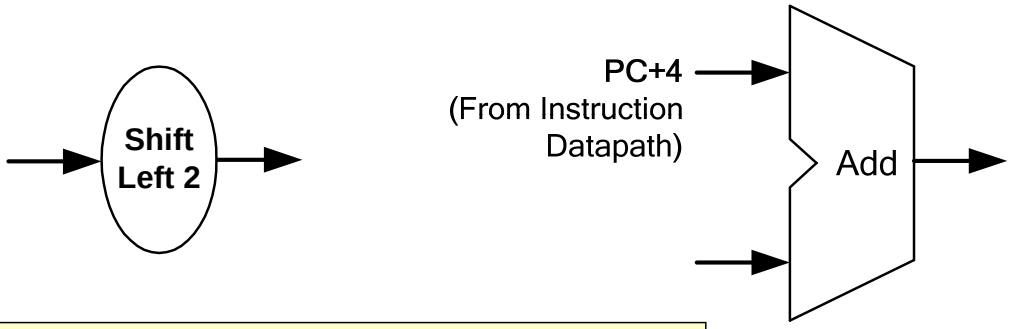
Exemplo: **beq $2, $3, 0x20**

opcode (4)	rs (2)	rt (3)	instruction_offset (0x20)

Implementação de um *Datapath* *(Instruções de branch)*

Finalmente, os elementos necessários à execução das instruções de salto condicional implicam a inclusão dos seguintes elementos:

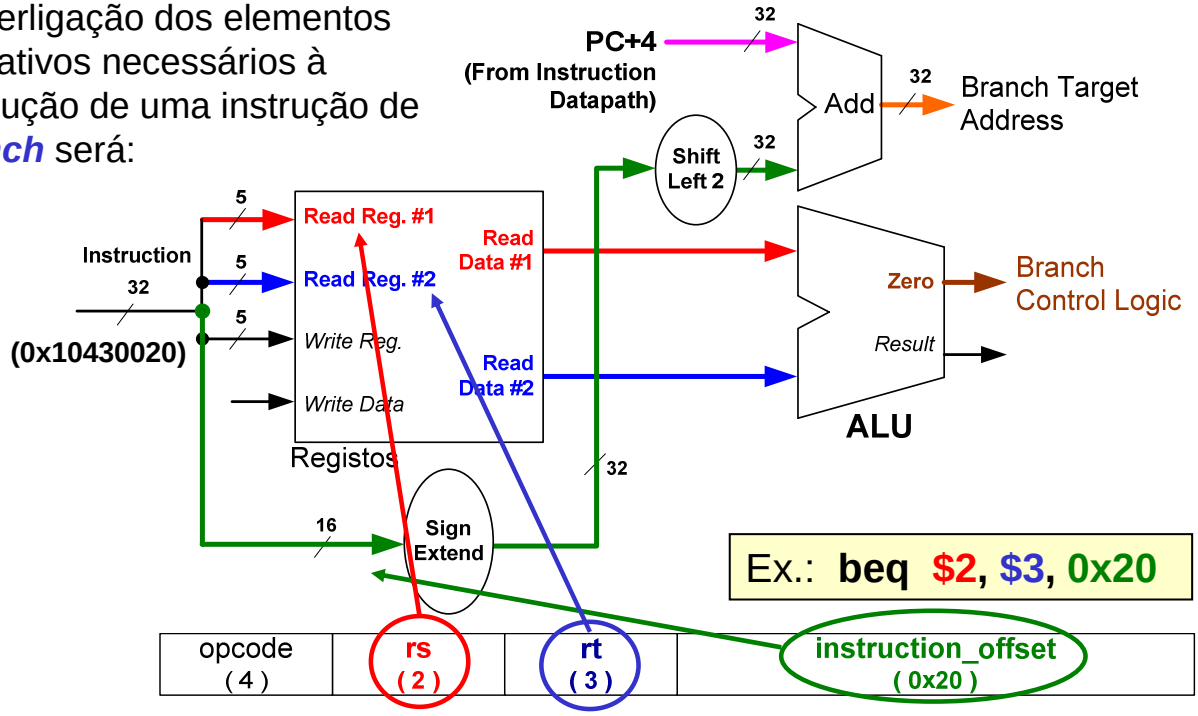
- *left shifter* (2 bits)
- Um somador



O *left shifter* recupera os 2 bits menos significativos do *instruction_offset* que são desprezados no momento da codificação da instrução (ver aula 5)

Implementação de um *Datapath* *(Instruções de branch)*

A interligação dos elementos operativos necessários à execução de uma instrução de ***branch*** será:



Implementação de um *Datapath*

- Tendo identificado, separadamente, os blocos básicos constituintes do *Datapath* necessários à execução dos vários tipos de instruções do MIPS (dos quais se omitiram desta discussão, por enquanto, as instruções do tipo J), coloca-se a seguinte questão:
  - **Como juntar e interligar os diversos blocos, por forma a servir todas as instruções?**
- A resposta a esta pergunta passa pela identificação dos blocos que podem ser partilhados pelos vários tipos de instruções e pelo desenvolvimento de uma estratégia que permita que os mesmos possam ser "configurados" para cada caso.

Implementação de um *Datapath*

Relembremos o formato dos três tipos de instruções!

Aritméticas e lógicas – Tipo R

31					0
opcode (0)	**rs**	**rt**	**rd**	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

LW, SW, aritméticas imediatas – Tipo I

31			0
opcode	**rs**	**rt**	offset / imm
6 bits	5 bits	5 bits	16 bits

Branches – Tipo I

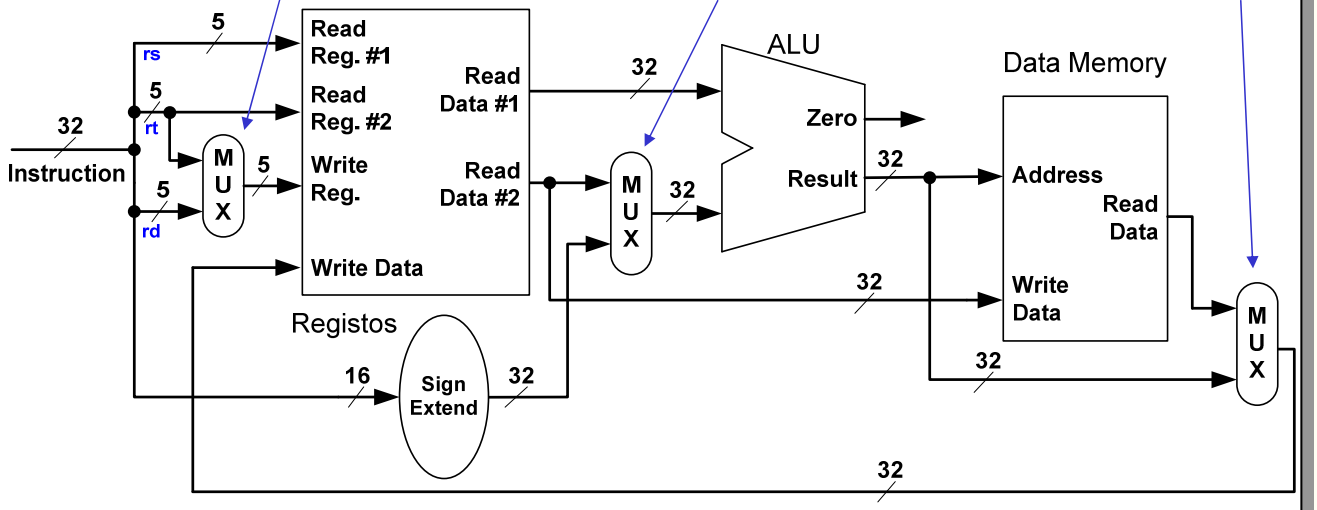
31			0
opcode	**rs**	**rt**	instruction_offset
6 bits	5 bits	5 bits	16 bits

Implementação de um *Datapath* *(1º passo)*

A combinação das instruções de acesso à memória com as instruções aritméticas e lógicas do tipo R e do tipo I pode ser feita do seguinte modo:

Escolha do registo destino (3º campo nas instruções tipo R, 2º na instrução LW e nas aritméticas e lógicas imediatas)

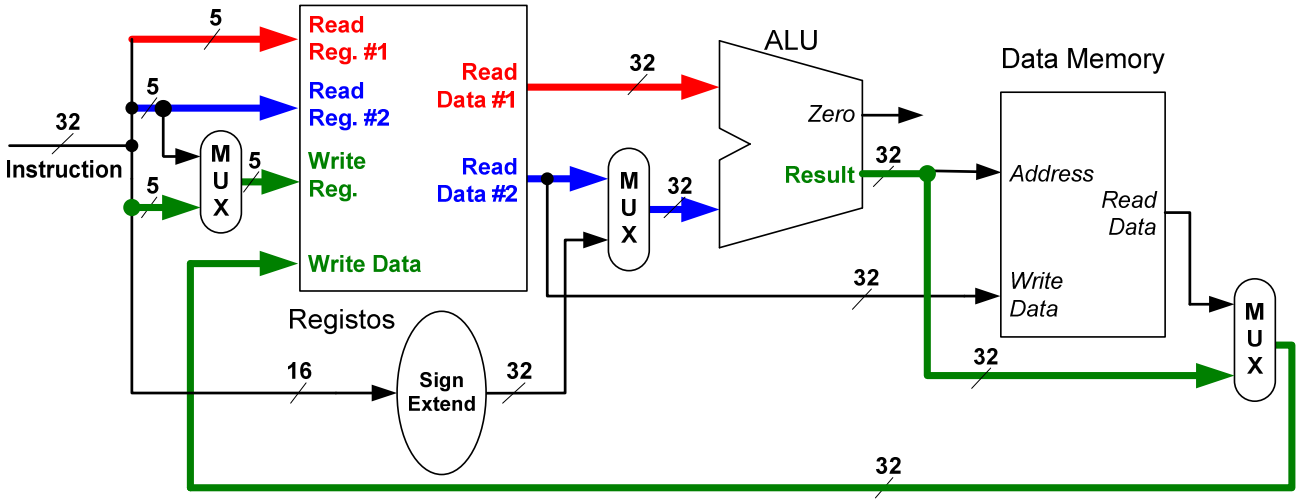
Note-se a utilização de multiplexers para adequar os recursos às necessidades



Implementação de um *Datapath* (1º passo)

Uma instrução do tipo R executada sobre um *datapath* misto

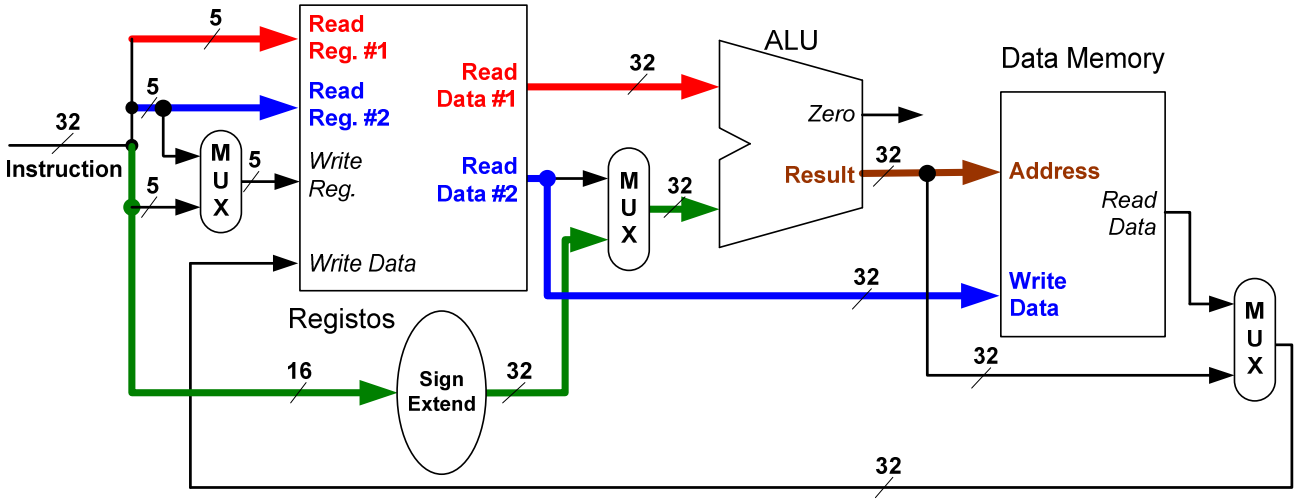
opcode (0)	rs (3)	rt (4)	rd (2)	shamt (0)	funct (32)



Implementação de um *Datapath* (1º passo)

A instrução SW (*store word*) executada sobre um *datapath* misto

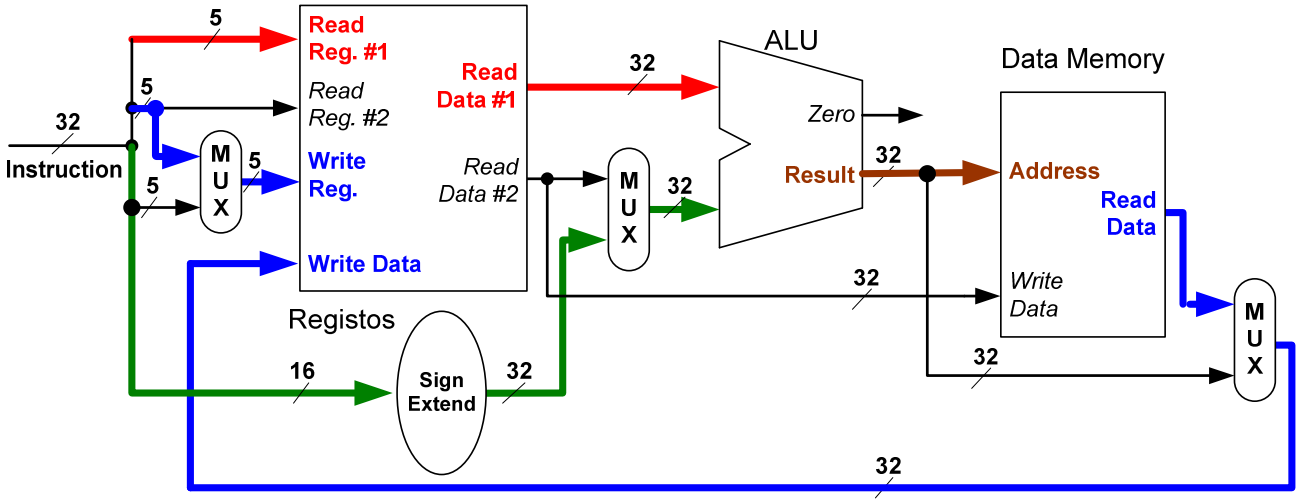
opcode (43)	rs (4)	rt (2)	offset (0x24)



Implementação de um *Datapath* (1º passo)

A instrução LW (*load word*) executada sobre um *datapath* misto

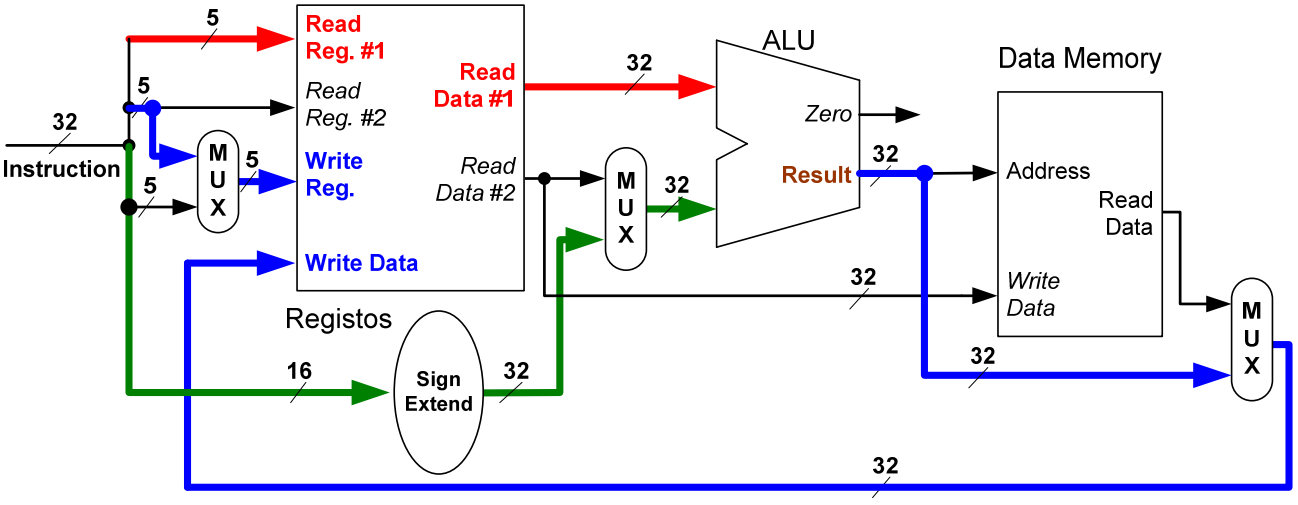
opcode (35)	rs (15)	rt (4)	offset (0x2F)



Implementação de um *Datapath* (1º passo)

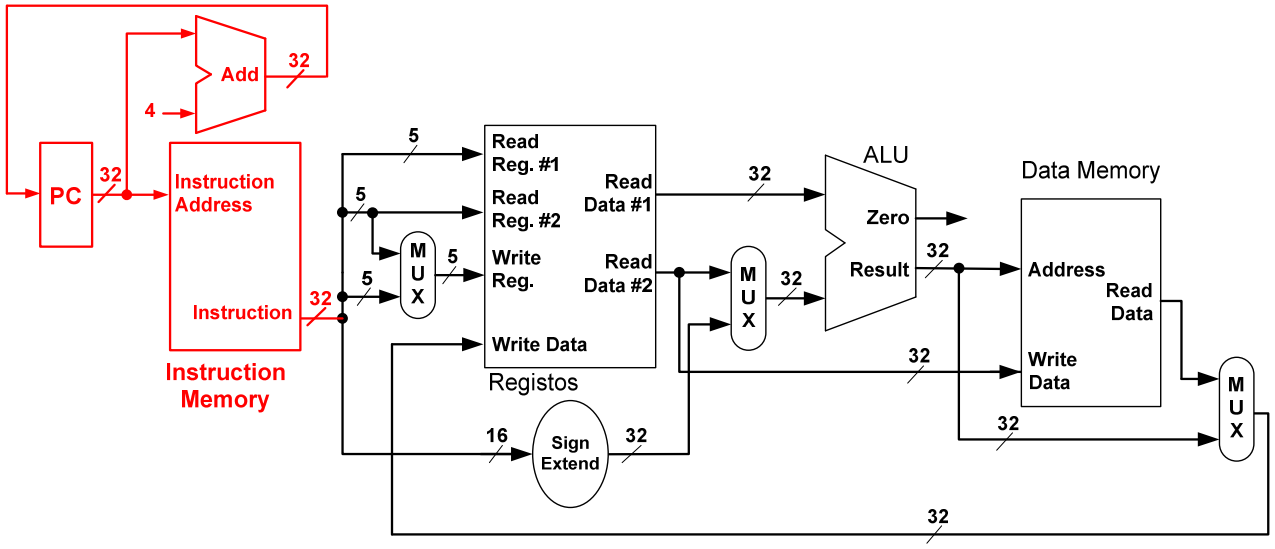
Uma instrução aritmética imediata executada sobre um *datapath* misto (ex. addi $4, $15, 0x2F)

opcode (8)	rs (15)	rt (4)	offset (0x2F)



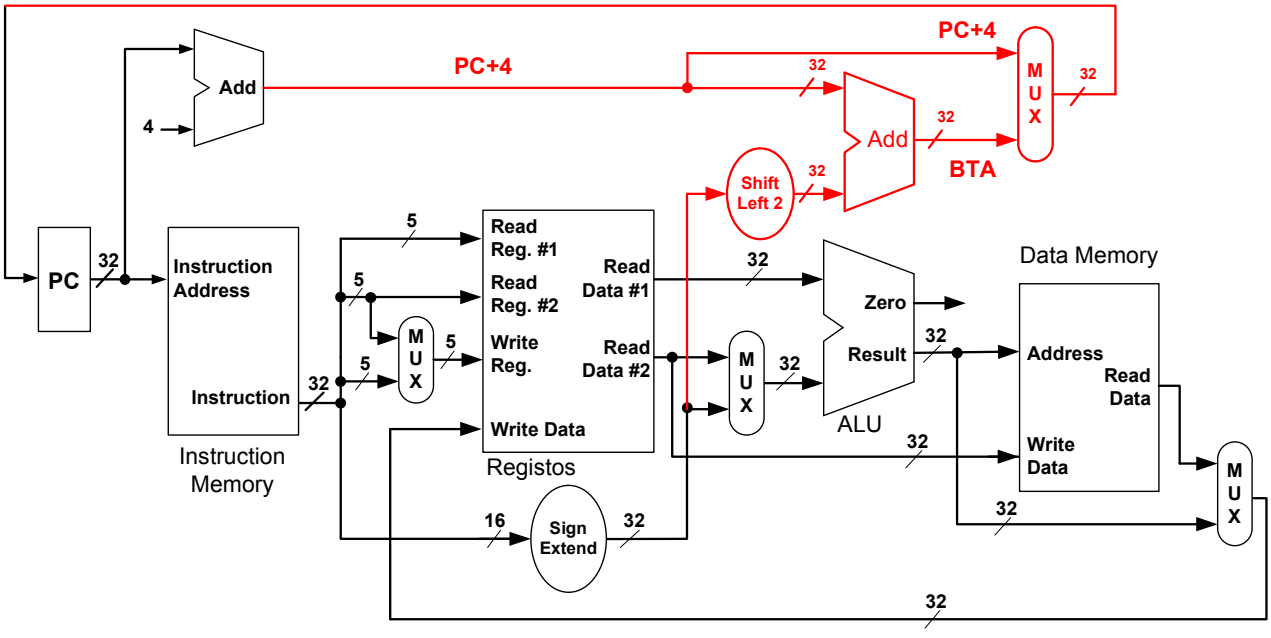
Implementação de um *Datapath* (2º passo)

O bloco de *Instruction Fetch* pode ser acrescentado sem grandes complicações



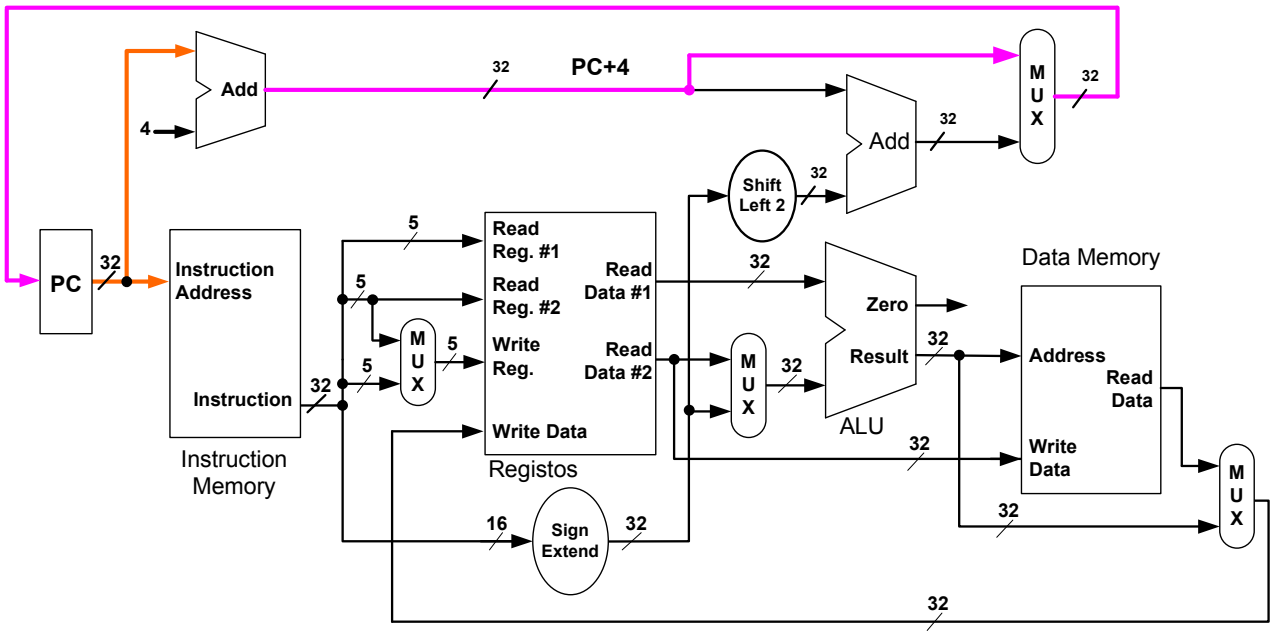
Implementação de um *Datapath* (3º passo)

Finalmente, a adição das instruções de salto condicional, implica uma pequena alteração à imagem de conjunto...



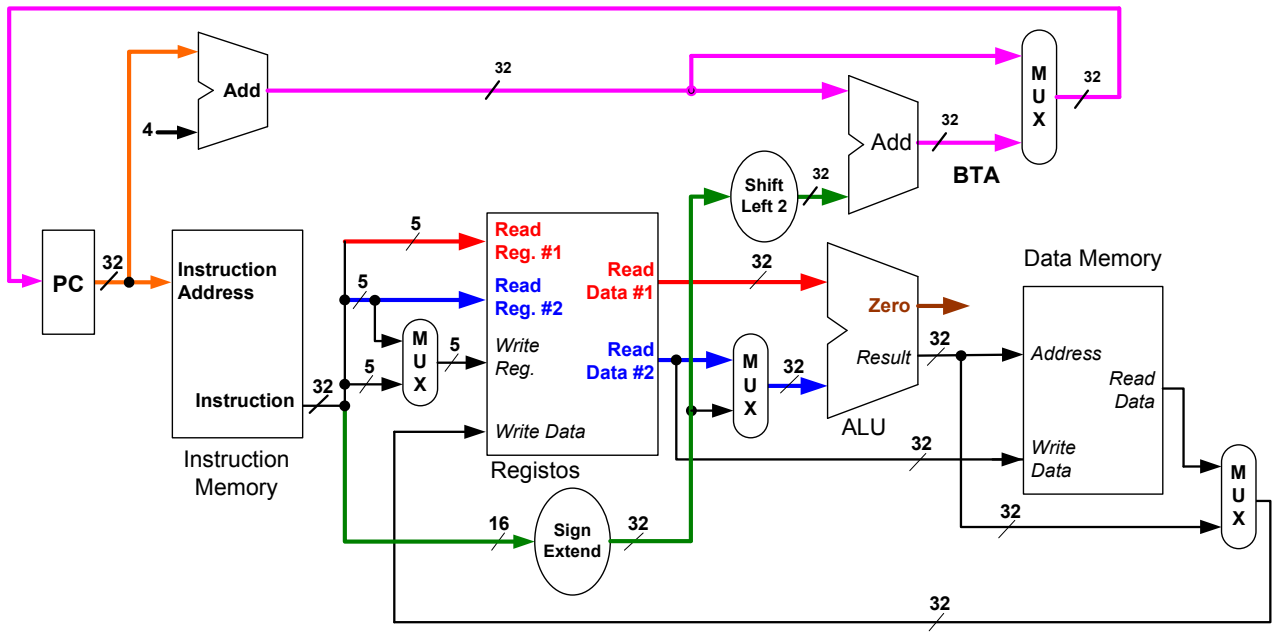
Implementação de um *Datapath* (3º passo)

Datapath misto (*Instruction fetch*)



Implementação de um *Datapath* (3º passo)

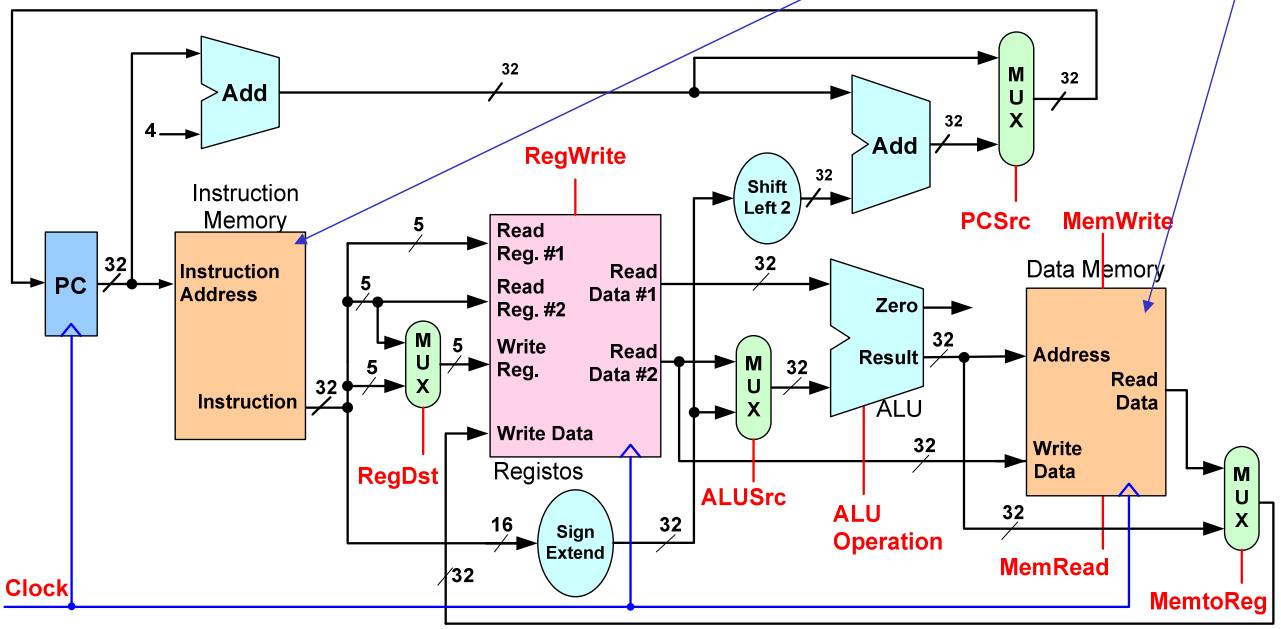
Datapath misto (Instrução *Branch if Equal*)



Implementação de um *Datapath*

Datapath misto com identificação dos sinais de controlo

Esta solução, com memórias distintas, permite a execução de cada instrução num único ciclo de relógio



Que alterações é necessário fazer para suportar também a execução do "bne" ?