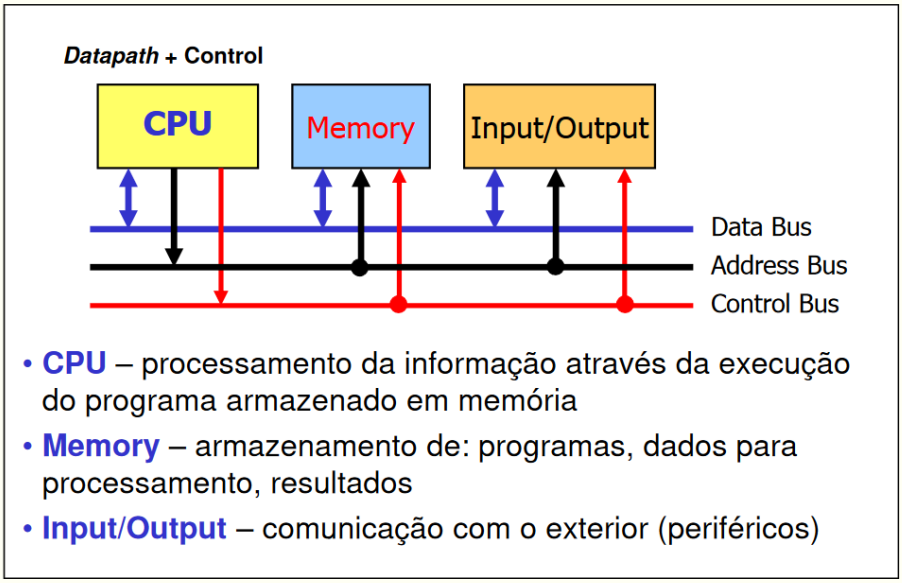
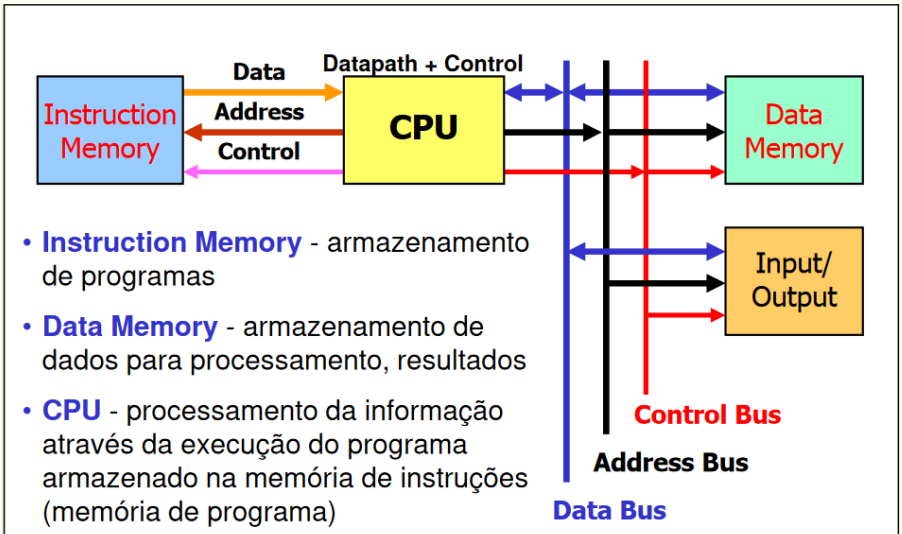
Modelo de von Neumann



- **CPU** – processamento da informação através da execução do programa armazenado em memória
- **Memory** – armazenamento de: programas, dados para processamento, resultados
- **Input/Output** – comunicação com o exterior (periféricos)

Modelo de Harvard



- **Instruction 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 instruções (memória de programa)

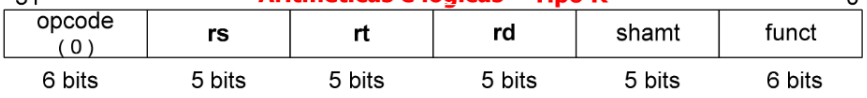
von Neumann *versus* Harvard – resumo

- **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

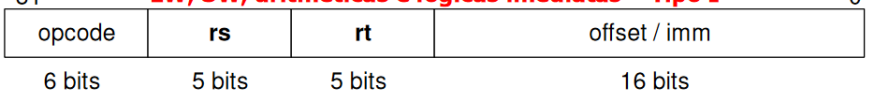
Implementação de um *Datapath* – juntando tudo

- Relembremos o formato de codificação dos três tipos de instruções:

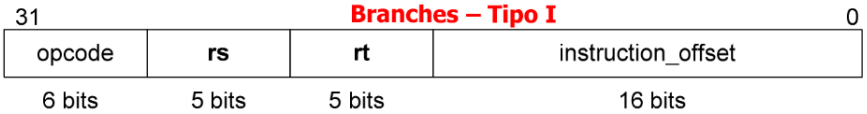
Aritméticas e lógicas – Tipo R



LW, SW, aritméticas e lógicas imediatas – Tipo I



Branches – Tipo I

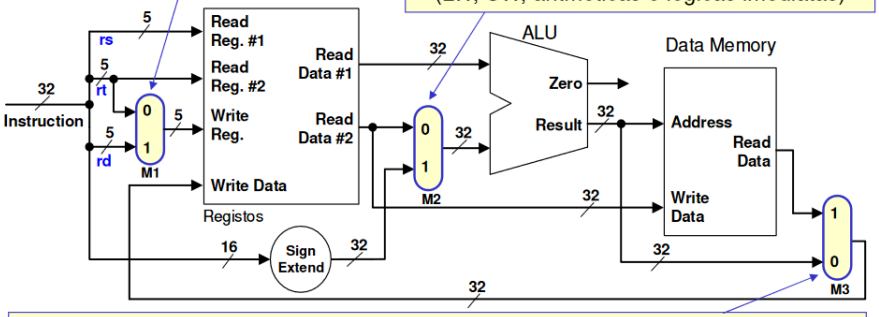


Implementação de um *Datapath* – juntando tudo

- **1º passo**: combinação das instruções de acesso à memória com as instruções aritméticas e lógicas do tipo R e do tipo I:

Seleção do registo destino: **rd** (instruções tipo R), **rt** (LW e nas aritméticas e lógicas imediatas)

Seleção do 2º operando da ALU: **conteúdo de um registo** (instruções tipo R e branches); **offset estendido para 32 bits** (LW, SW, aritméticas e lógicas imediatas)

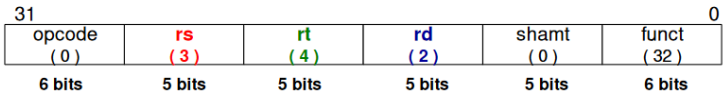


Seleção do valor a escrever no banco de registos: **valor lido da memória** (LW), **valor calculado na ALU** (instruções tipo R, aritméticas e lógicas imediatas)

Implementação de um *Datapath* – instruções tipo R

- Operações realizadas no decurso da execução de uma instrução tipo R:
  - ***Instruction Fetch*** (leitura da instrução, cálculo de PC+4)
  - **Leitura dos registos** operando (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`")

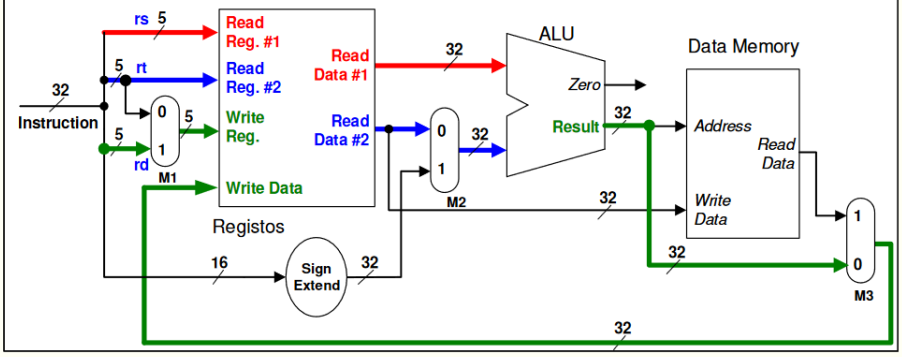
Exemplo: **add $2, $3, $4**



Código máquina: 0x00641020

Implementação de um *Datapath* – juntando tudo

- Fluxo da informação na execução de uma instrução do tipo R. Exemplo: `add $2,$3,$4`



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** (registos especificados nos campos "**`rs`**" e "**`rt`**"da instrução, respetivamente)
  - Cálculo, na ALU, do endereço de acesso (soma algébrica entre o conteúdo do registo "**`rs`**" e o **`offset`** especificado na instrução)
  - Escrita na memória

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

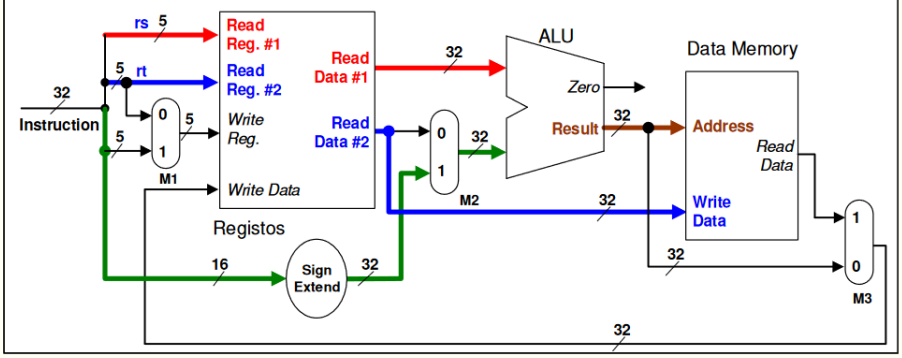
Endereço inicial da memória onde vai ser escrita a word de 32 bits armazenada no registo $2

opcode (0x2B)	rs (4)	rt (2)	offset (0x24)

Implementação de um *Datapath* – juntando tudo

- Fluxo da informação na execução de uma instrução SW (*store word*). 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)

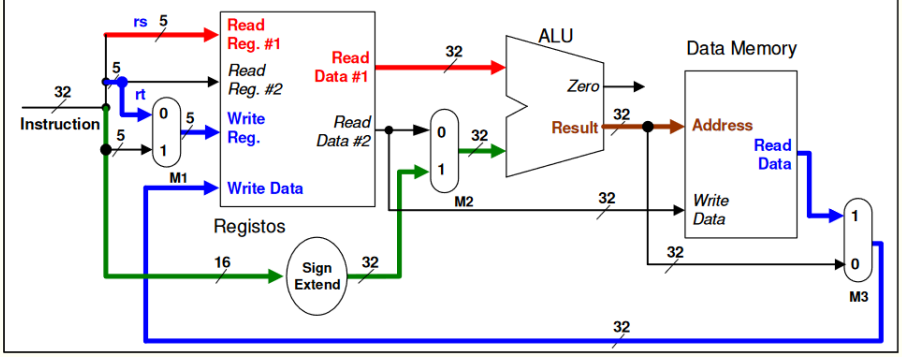
Exemplo: **lw** **$4,** **0x2F($15)** ← Endereço inicial da memória para leitura de uma word de 32 bits (vai ser escrita no registo $4)

opcode (0x23)	rs (15)	rt (4)	offset (0x2F)

Implementação de um *Datapath* – juntando tudo

- Fluxo da informação na execução de uma instrução LW (*load word*). Exemplo: **`lw $4,0x2F($15)`**

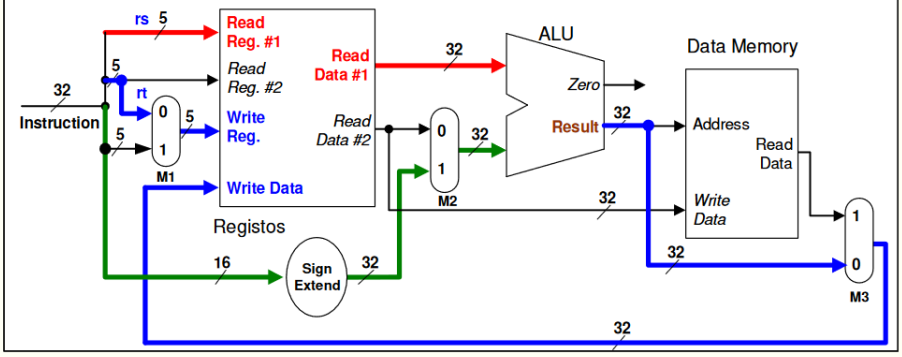
opcode (35)	rs (15)	rt (4)	offset (0x2F)



Implementação de um *Datapath* – juntando tudo

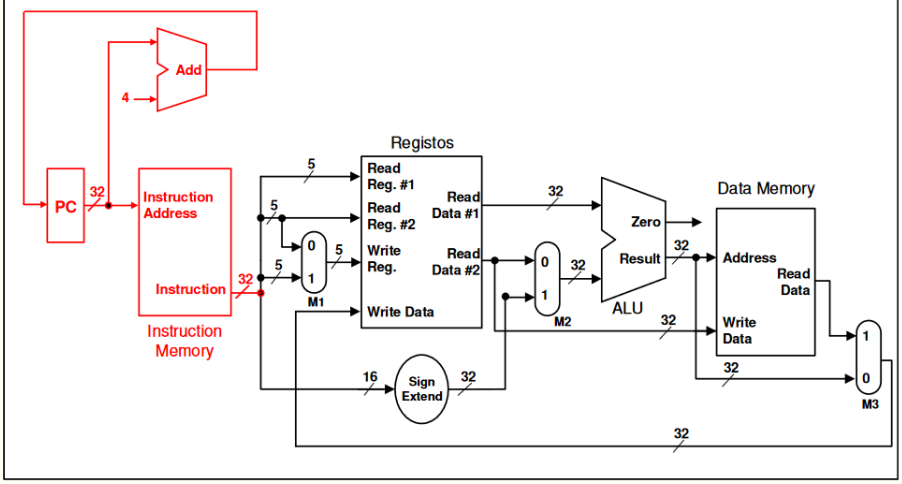
- Fluxo da informação na execução das instruções imediatas.
  Exemplo: **`addi $4,$15,0x2F`**

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



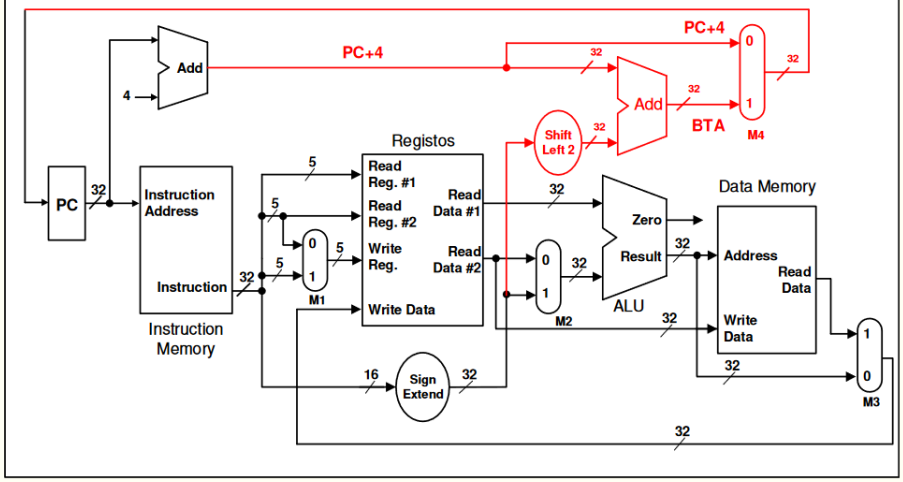
Implementação de um *Datapath* – juntando tudo

- **2º passo**: inclusão do bloco *Instruction Fetch*



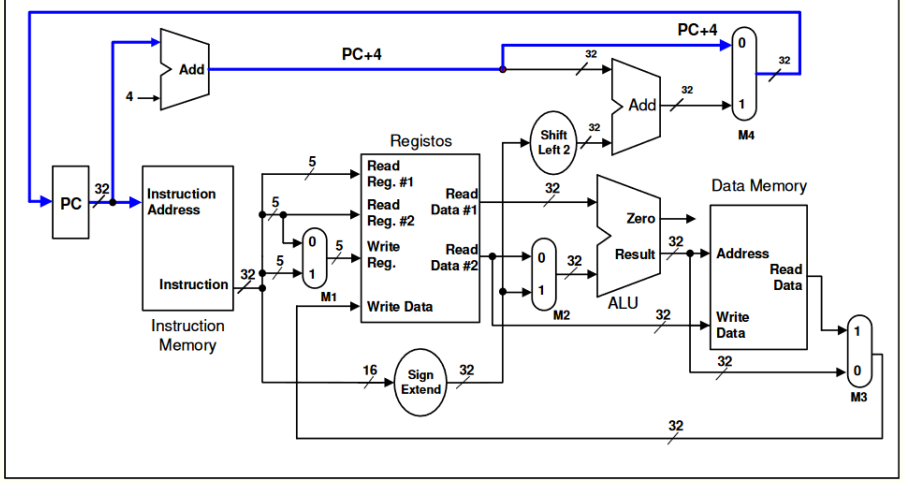
Implementação de um *Datapath* – juntando tudo

- **3º passo**: adição das instruções de salto condicional (*branches*)



Implementação de um *Datapath* – juntando tudo

- Encaminhamento de PC+4 para a entrada do Program Counter



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 de dois registos, do banco de registos
  - Comparação dos conteúdos dos registos (realização de uma operação de subtração na ALU)
  - Cálculo do endereço-alvo da instrução de *branch* (*Branch Target Address* - BTA)

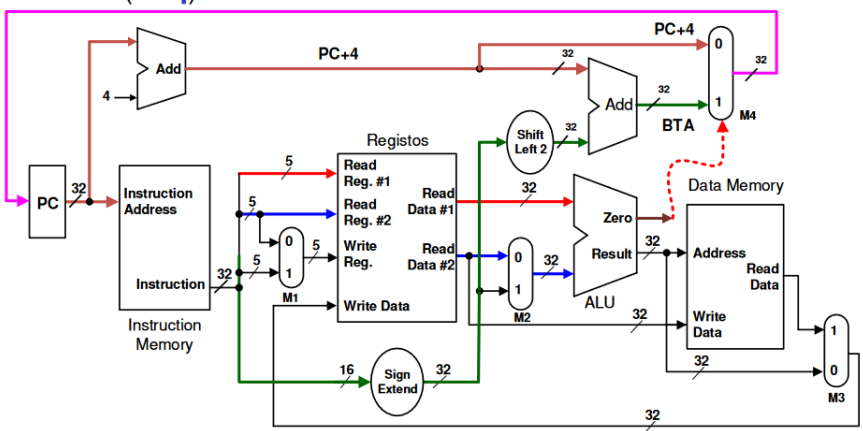
    `BTA = (PC+4) + (instruction_offset << 2)`
  - Alteração do valor do registo PC:
    - se a condição testada pelo *branch* for verdadeira PC = BTA
    - se a condição testada pelo *branch* for falsa PC = PC + 4

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

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

Implementação de um *Datapath* – juntando tudo

- Fluxo da informação na execução de uma instrução de *branch* (**beq**)

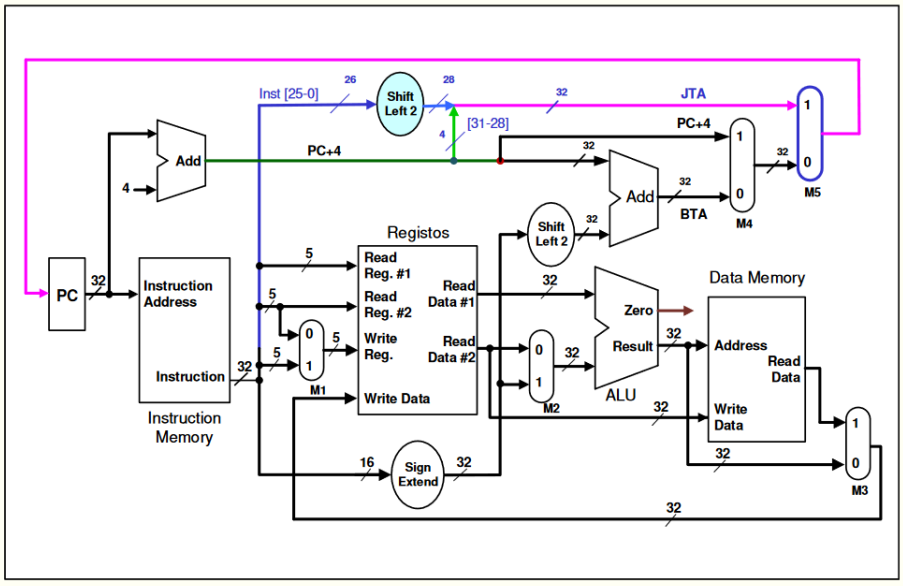


O valor a ser escrito no registo PC, no próximo flanco ativo do relógio, **depende** da saída "zero" da ALU: **"PC+4" se zero=0; BTA se zero=1**

Datapath com suporte para a instrução "j" (jump)

- A instrução “j” é codificada com um caso particular de codificação, o formato J
- No formato J existem apenas dois campos:
  - o campo opcode (**bits 31-26**) e o
  - campo de endereço (**bits 25-0**)
- Na instrução “j”, o endereço alvo (*Jump Target Address* - **JTA**) obtém-se pela **concatenação:**
  - dos bits **31-28** do PC+4 com
  - os bits do campo de endereço da instrução (26 bits) multiplicados por 4 (2 *shifts* à esquerda)
- No próximo flanco ativo do relógio, o valor do PC será **incondicionalmente** alterado com o valor do JTA

Datapath com suporte para a instrução "jump" (j)



Datapath single-cycle completo (com sinais de controlo)