Sistemas Digitais

Palavra de Controle (*Control Word*)

Referência Bibliográfica:

Logic and Computer Design Fundamentals – Mano & Kime

Adaptações: josé artur quilici-gonzalez

Sumário

- ***Datapaths* e *Control Word***
  - Introdução
  - Exemplo de *Datapath*
  - *Arithmetic Logic Unit (ALU)*
  - Deslocador ou *Shifter*
  - Representação de *Datapath*
  - *Control Word*

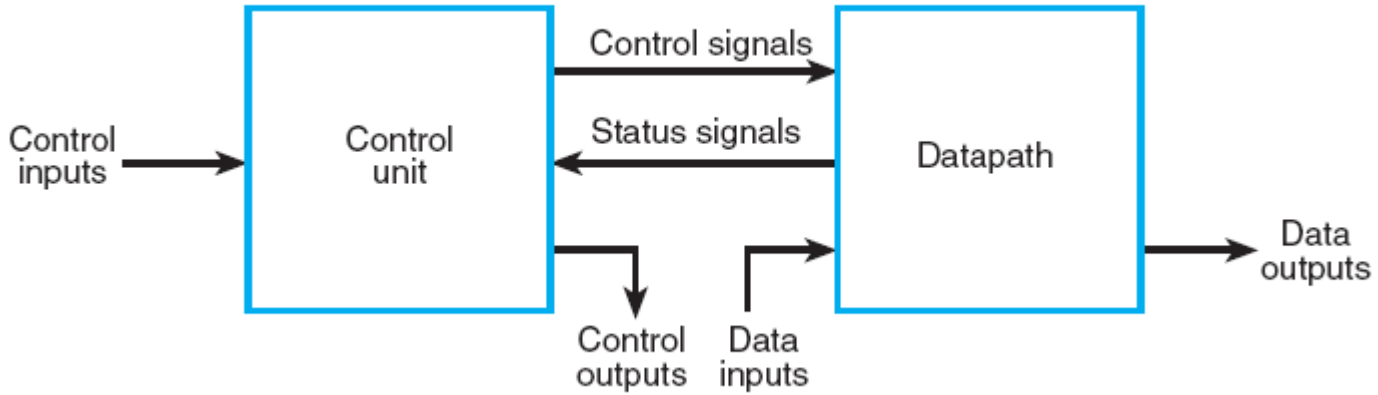
Introdução

- Especificação de um Computador
  - ***Instruction Set Architecture (ISA)*** - a especificação de um computador para um programador em seu nível mais baixo
  - ***Computer Architecture*** – uma descrição em alto nível do *hardware* que implementa o computador a partir da *ISA*
  - Na realidade, a **arquitetura** normalmente inclui especificações adicionais tais como velocidade, custo e confiabilidade.

Introdução (continuação)

- A arquitetura de um computador simples é decomposta em:
  - **Caminho de Dados** (*Datapath*) para realizar operações de transferência e processamento de dados
  - **Unidade de Controle** para autorizar e determinar a sequência de operações do *Datapath*
- Um ***Datapath*** é especificado por:
  - Um conjunto de registradores
  - Uma lógica digital (*Bus, MUX, Decoder, ALU* etc.) que implementa as micro-operações nos dados armazenados nos registradores
  - Uma interface de controle que supervisiona a sequência de operações no sistema

Interação entre *Datapath* e Unidade de Controle



•**Sinais de Controle** são sinais binários que ativam as operações de processamento de dados

•A **Unidade de Controle**, por sua vez, recebe **Sinais de Estado** do ***Datapath*** (úteis para a **Unidade de Controle** determinar a sequência específica de operações)

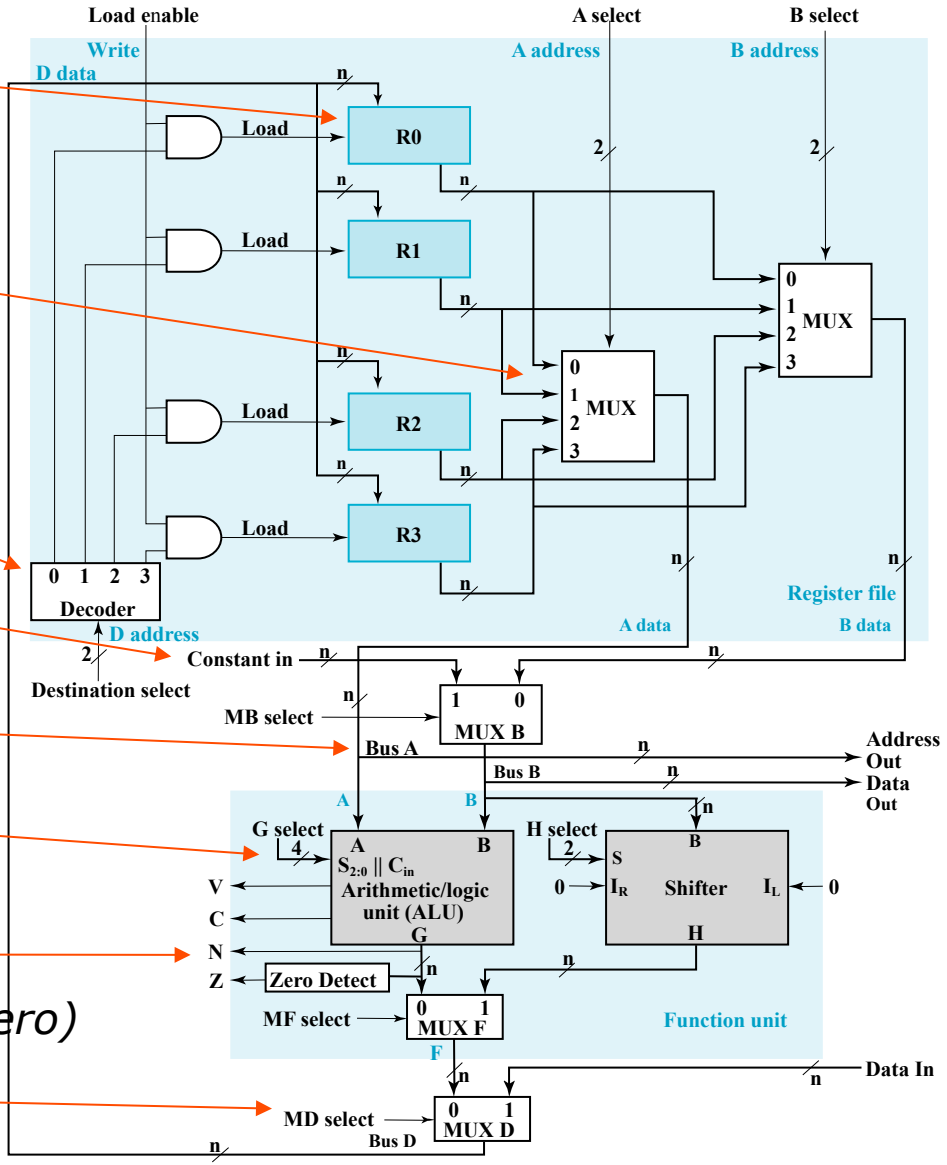
•Tanto a **Unidade de Controle** quanto o ***Datapath*** podem interagir com outras partes do **Sistema Digital** (como Memória, lógica de *Input-Output*) através das Entradas e Saídas de Dados e Controles etc.

Datapaths

- Princípios para *datapaths* básicos:
  - O conjunto de registradores
    - Coleção de registradores individuais
    - Um conjunto de registradores com recursos de acesso comum chamado *register file* (para armazenar resultados parciais)
    - Uma combinação dos itens acima
  - Implementação de micro-operação
    - Um ou mais recursos compartilhados para implementar as micro-operações
    - *Buses* – vias de transferências compartilhadas
    - *Arithmetic-Logic Unit (ALU)* - recursos compartilhados para implementar micro-operações aritméticas e lógicas
    - *Shifter* - recurso compartilhado para implementar micro-operações de deslocamento

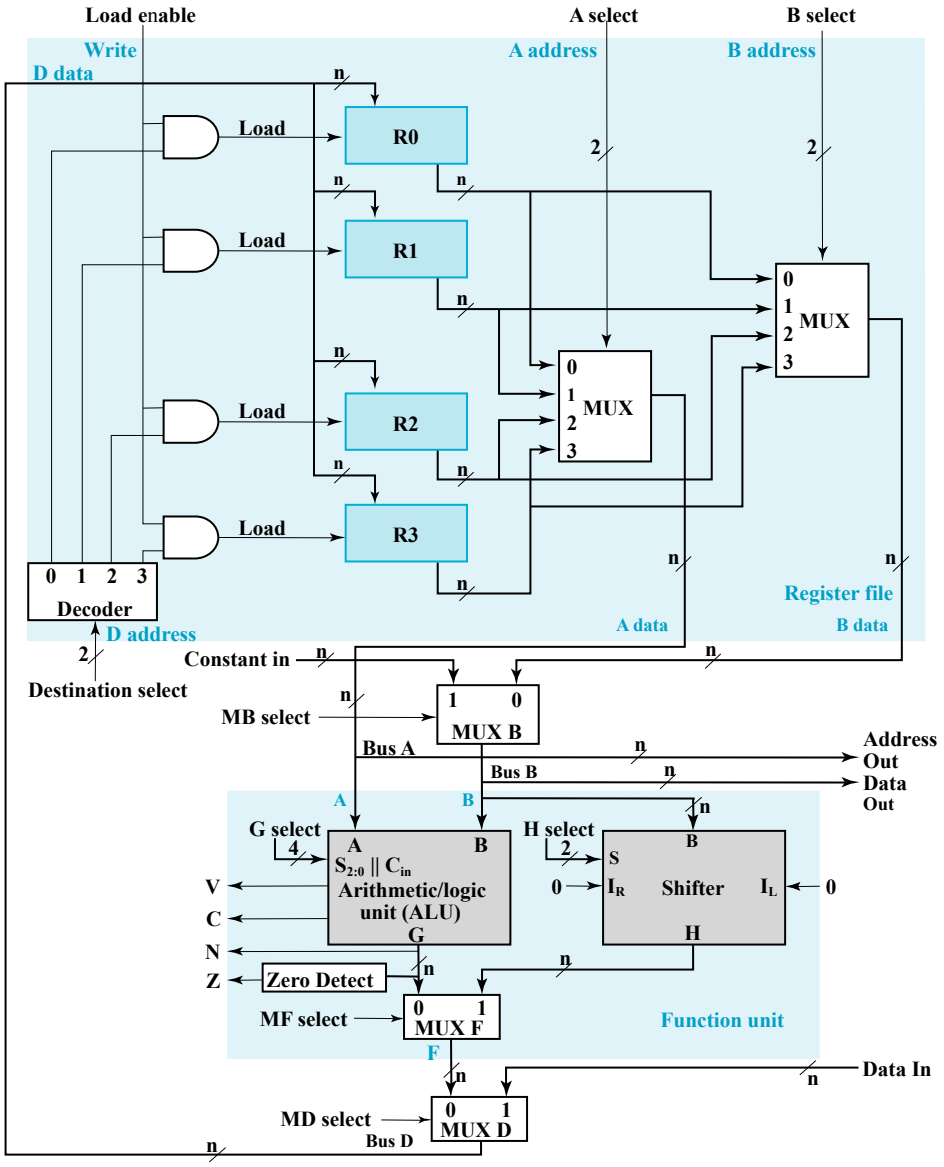
Exemplo de *Datapath*

- Quatro registradores com carga paralela
- Dois selecionadores de registradores baseados em *mux*
- Um decodificador de registrador de destino
- *Mux B* para entrada de uma constante externa
- *Buses A* e *B* com endereço externo e saídas de dados
- *ALU* e *Shifter* com *Mux F* para seleção de saída
- Lógica para geração de *status bits* V, C, N, Z *(Overflow, Carry, Negative, Zero)*
- *Mux D* para entrada de dado externo



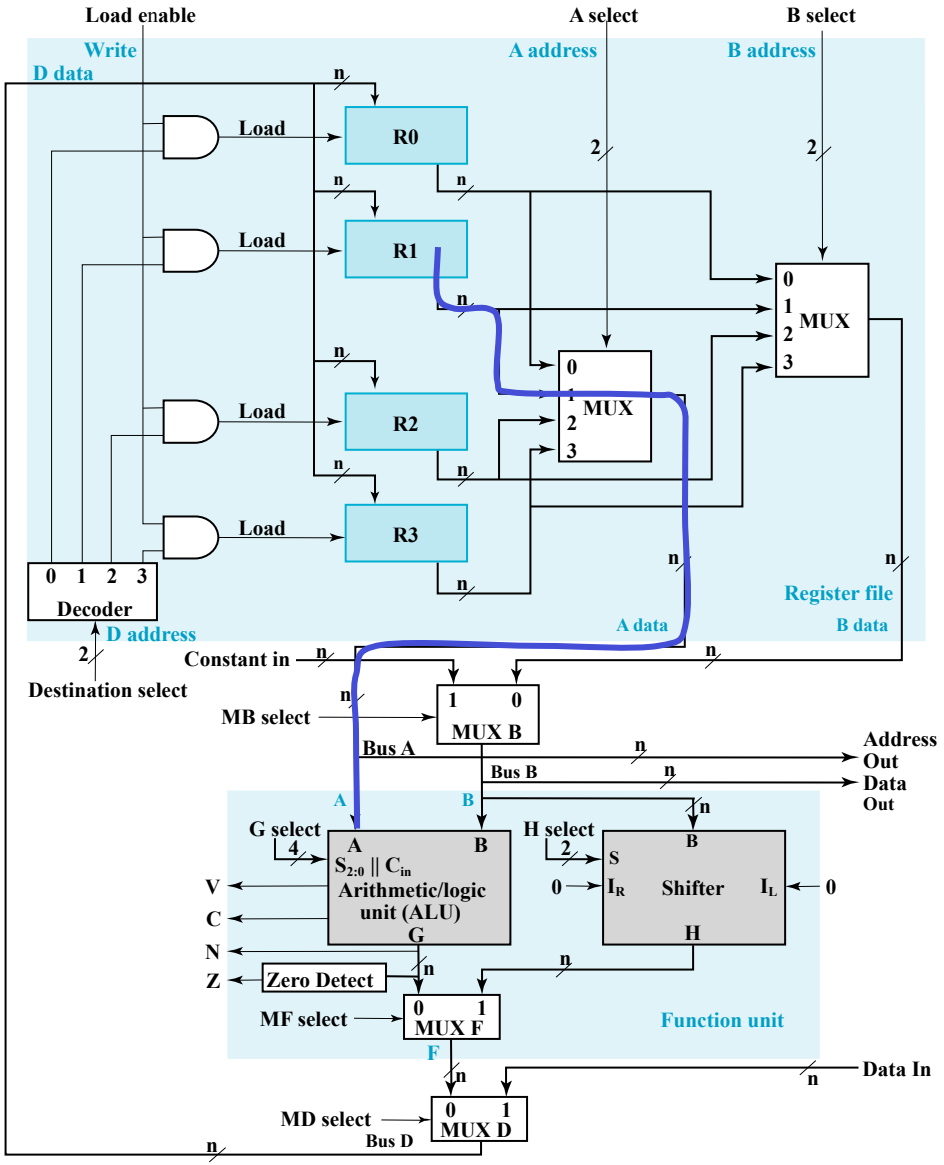
Exemplo de uma Micro-operação no *Datapath*

- **Micro-operação: R0 ← R1 + R2**



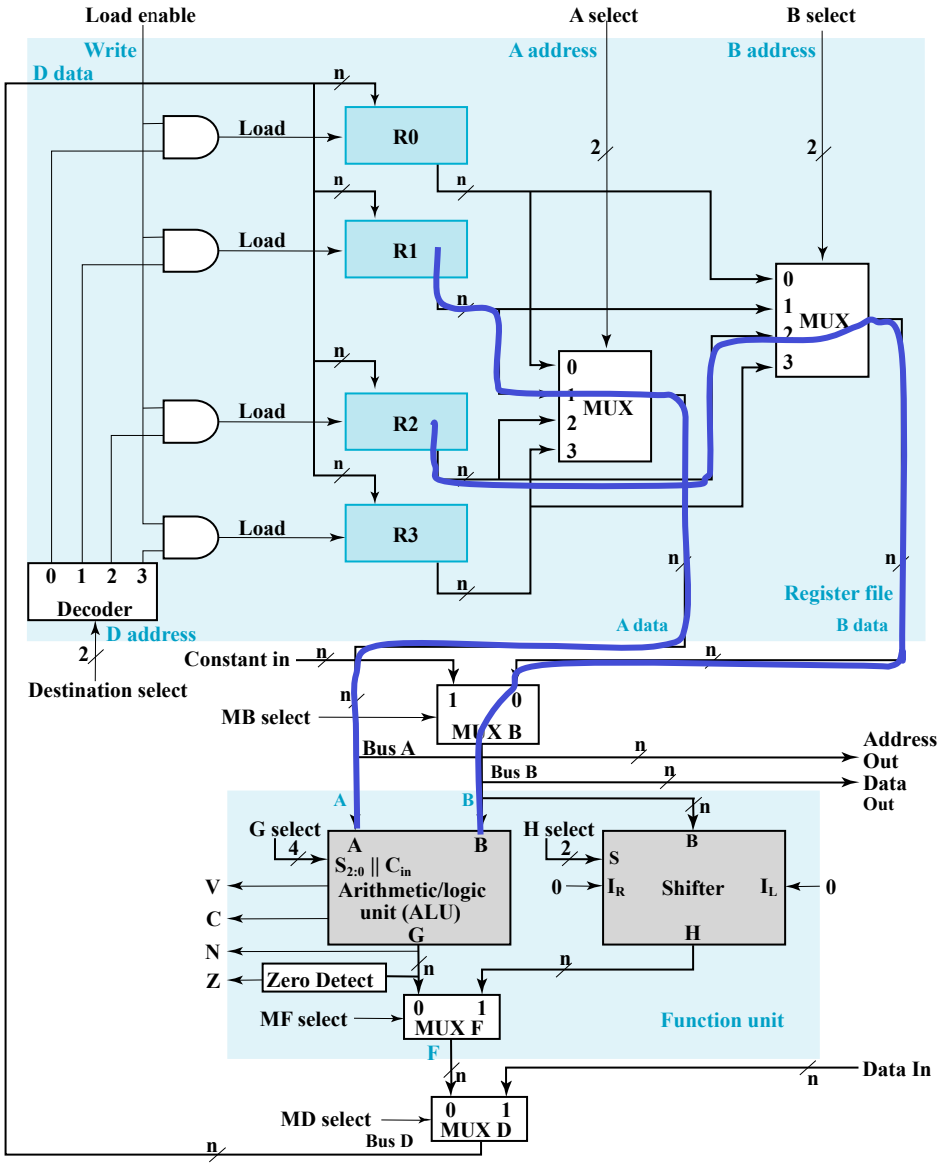
Exemplo de uma Micro-operação no *Datapath*

- **Micro-operação: R0 ← R1 + R2**
  - Aplique *01* em *A select* para carregar o conteúdo de *R1* no *Bus A*



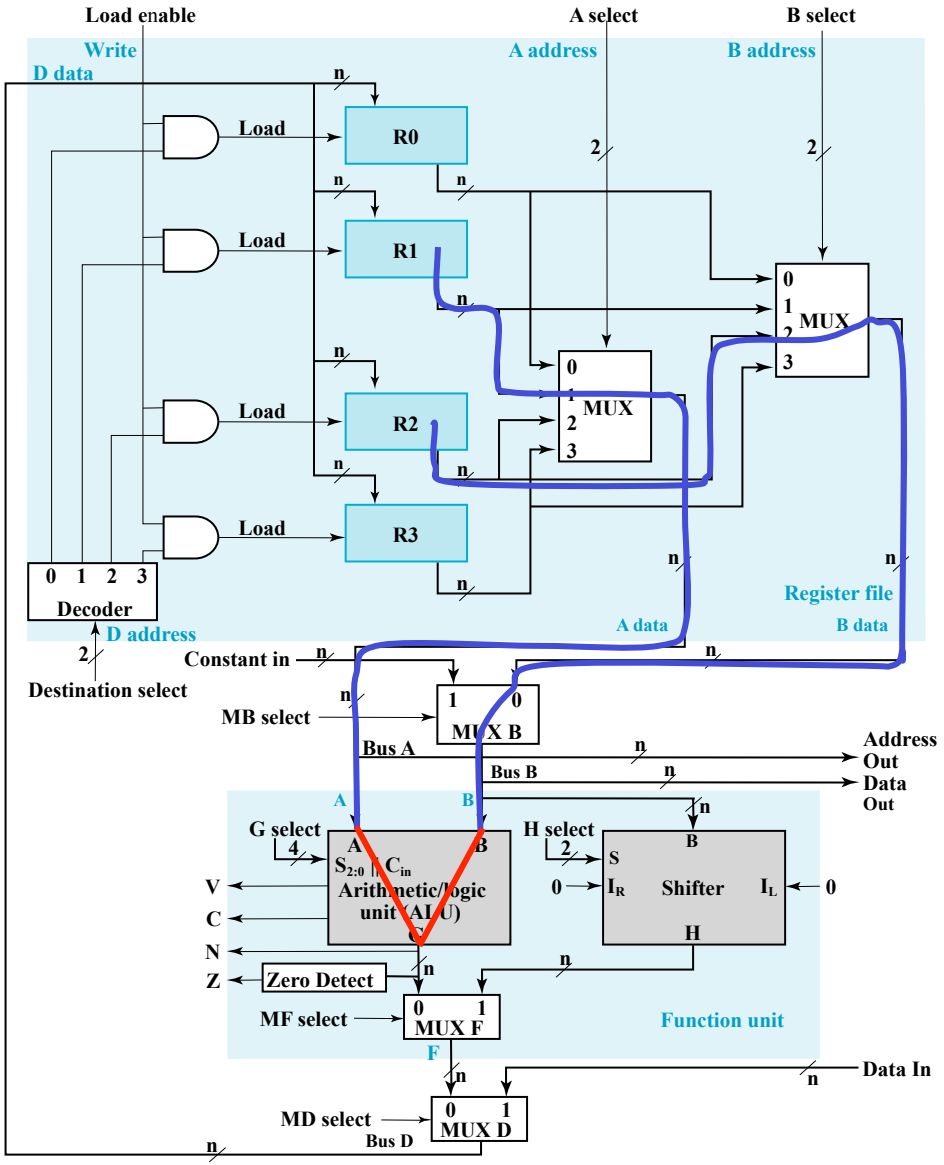
Exemplo de uma Micro-operação no *Datapath*

- **Micro-operação: R0 ← R1 + R2**
  - Aplique *01* em *A select* para carregar o conteúdo de *R1* no *Bus A*
  - Aplique *10* em *B select* para carregar o conteúdo de *R2* em *B data* e aplique *0* em *MB select* para carregar *B data* no *Bus B*



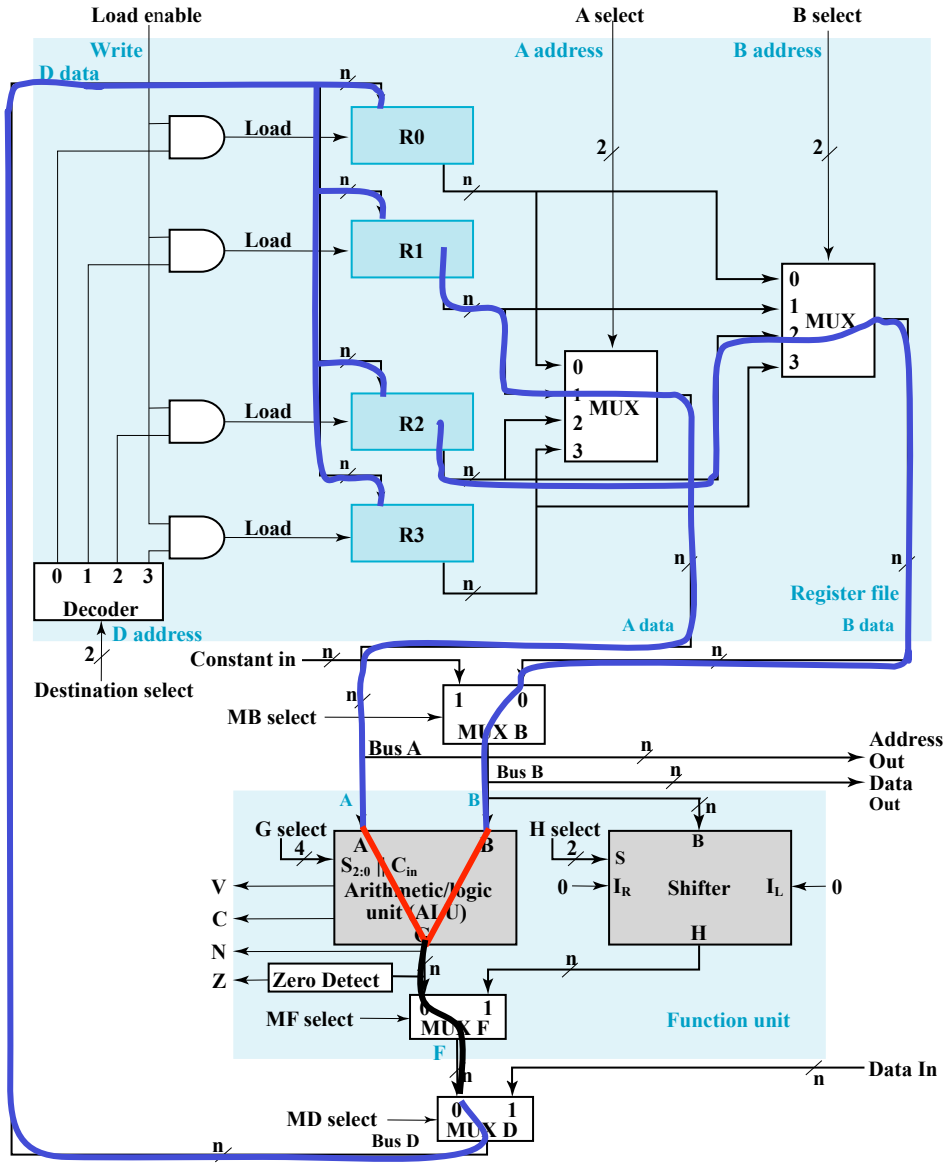
Exemplo de uma Micro-operação no *Datapath*

- **Micro-operação: R0 ← R1 + R2**
  - Aplique *01* em *A select* para carregar o conteúdo de *R1* no *Bus A*
  - Aplique *10* em *B select* para carregar o conteúdo de *R2* em *B data* e aplique *0* em *MB select* para carregar *B data* no *Bus B*
  - Aplique *0010* em *G select* para realizar a adição *G = Bus A + Bus B*



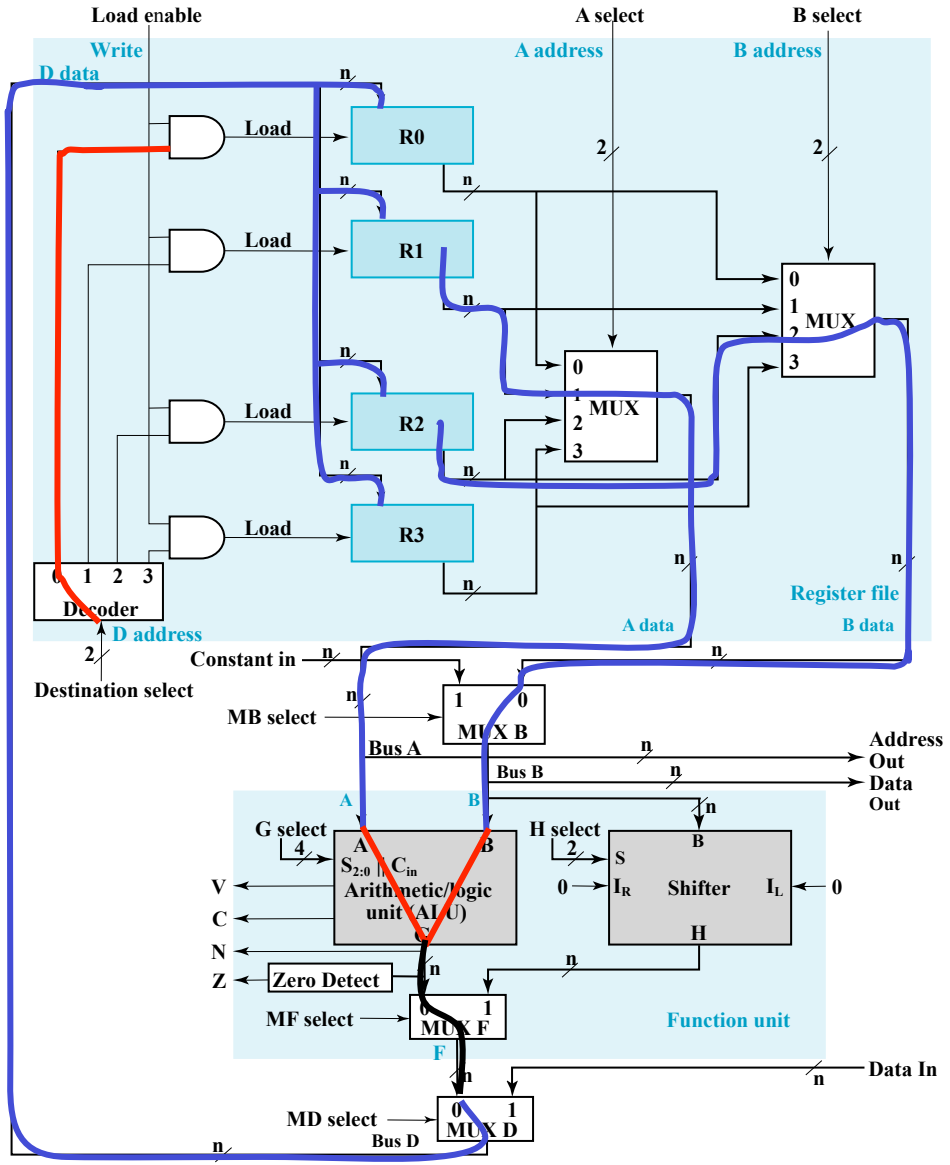
Exemplo de uma Micro-operação no *Datapath*

- **Micro-operação: R0 ← R1 + R2**
  - Aplique *01* em *A select* para carregar o conteúdo de *R1* no *Bus A*
  - Aplique *10* em *B select* para carregar o conteúdo de *R2* em *B data* e aplique *0* em *MB select* para carregar *B data* no *Bus B*
  - Aplique *0010* em *G select* para realizar a adição *G = Bus A + Bus B*
  - Aplique *0* tanto em *MF select* quanto em *MD select* para carregar *G* no *Bus D*



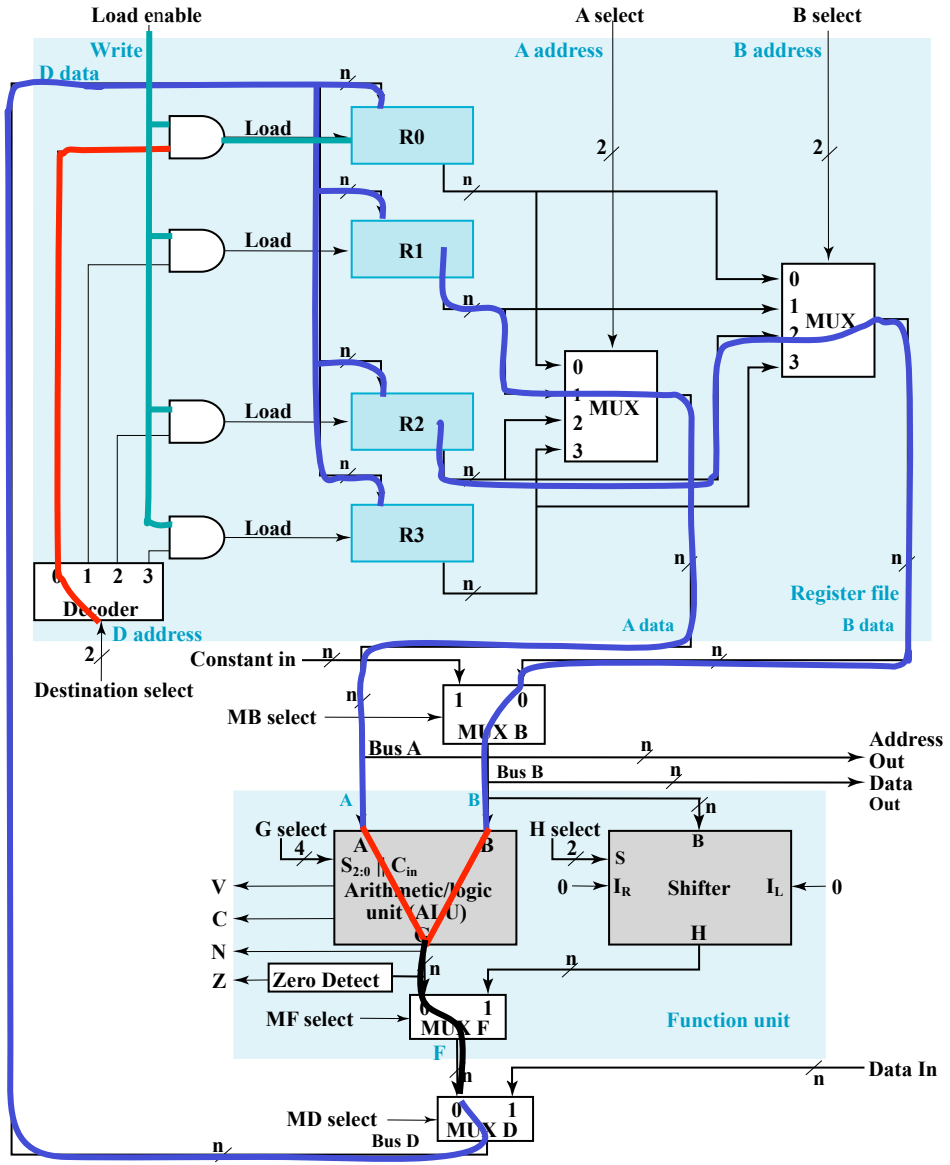
Exemplo de uma Micro-operação no *Datapath*

- **Micro-operação: R0 ← R1 + R2**
  - Aplique *01* em *A select* para carregar o conteúdo de *R1* no *Bus A*
  - Aplique *10* em *B select* para carregar o conteúdo de *R2* em *B data* e aplique *0* em *MB select* para carregar *B data* no *Bus B*
  - Aplique *0010* em *G select* para realizar a adição *G = Bus A + Bus B*
  - Aplique *0* tanto em *MF select* quanto em *MD select* para carregar *G* no *Bus D*
  - Aplique *00* em *Destination select* para habilitar a entrada *Load* em *R0*



Exemplo de uma Micro-operação no *Datapath*

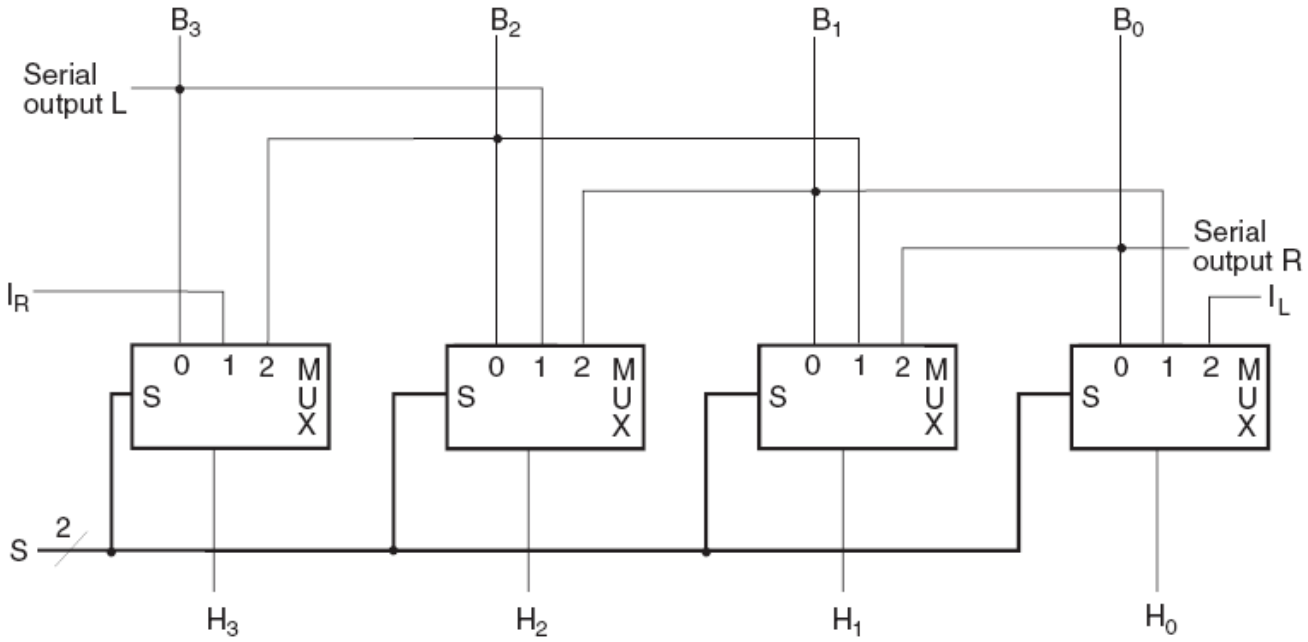
- **Micro-operação: R0 ← R1 + R2**
  - Aplique *01* em *A select* para carregar o conteúdo de *R1* no *Bus A*
  - Aplique *10* em *B select* para carregar o conteúdo de *R2* em *B data* e aplique *0* em *MB select* para carregar *B data* no *Bus B*
  - Aplique *0010* em *G select* para realizar a adição *G = Bus A + Bus B*
  - Aplique *0* tanto em *MF select* quanto em *MD select* para carregar *G* no *Bus D*
  - Aplique *00* em *Destination select* para habilitar a entrada *Load* em *R0*
  - Aplique *1* em *Load enable* para forçar a entrada *Load* de *R0* em *1* (para que *R0* seja carregado no próximo *clock* – não mostrado)
  - A micro-operação completa requer 1 ciclo de *clock*



Deslocador Combinacional

O deslocamento de dados pode ser feito com um **Registrador de Deslocamento**, porém três pulsos de *clock* são necessários para completar a operação (um pulso para carregar o dado, outro para fazer o deslocamento e um terceiro para enviar o resultado ao registrador de destino)

Em vez disso, a transferência de um registrador de origem para um registrador de destino pode ser feita num único pulso de *clock*, utilizando-se um **Circuito Combinacional**, construído com MUXes



Com a **Variável de Seleção** ***S = 00***, ***B*** passa pelo **Deslocador Combinacional** sem mudanças

Com ***S = 01***, ocorre um deslocamento para a direita

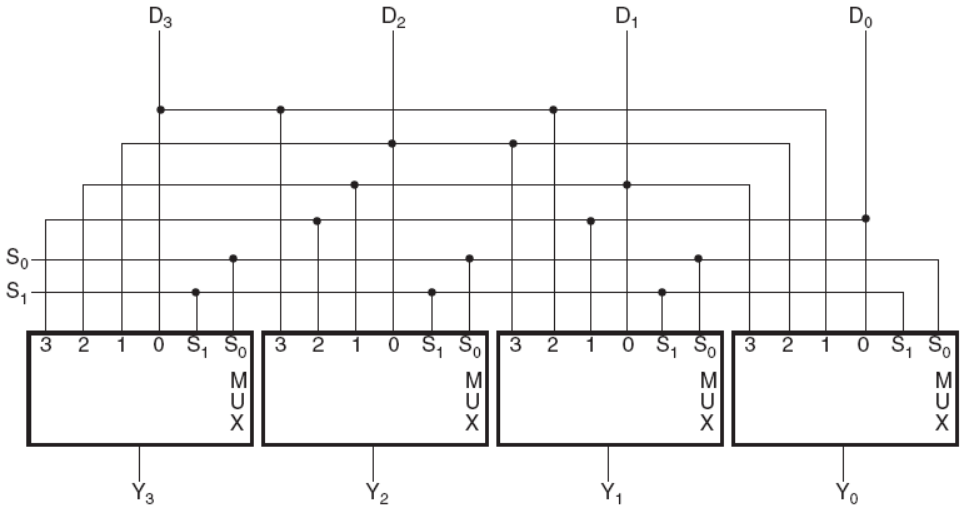
Com ***S = 10***, ocorre um deslocamento para a esquerda

Deslocador Circular (*Barrel Shifter*)

Em algumas aplicações, o dado precisa ser deslocado mais de uma posição num único pulso de *clock*

O **Deslocador Circular** é um circuito combinacional que desloca ou "roda" um número de posições especificado por um valor binário

No tipo de deslocador aqui considerado, os *bits* provenientes da parte mais significativa (*MSB*) são enviados para a posição menos significativa (*LSB*), realizando um deslocamento de *n* posições para a esquerda ou direita (note que o deslocamento de três posições para a esquerda corresponde ao deslocamento de uma posição para a direita!)



Select		Output				
S_1	S_0	Y_3	Y_2	Y_1	Y_0	Operation
0	0	D_3	D_2	D_1	D_0	No rotation
0	1	D_2	D_1	D_0	D_3	Rotate one position
1	0	D_1	D_0	D_3	D_2	Rotate two positions
1	1	D_0	D_3	D_2	D_1	Rotate three positions

Com as **Variáveis de Seleção** $S_1S_0 = 00$, D passa pelo **Deslocador Circular** sem mudanças

Com $S_1S_0 = 01$, ocorre o deslocamento de uma posição para a esquerda (D_0 vai para Y_1, D_1 para Y_2 etc.)

Com $S_1S_0 = 10$, ocorre um deslocamento de duas posições para a esquerda, e com $S_1S_0 = 11$, três posições para a esquerda (D_0 vai para $Y3$, D_1 para Y_0, D_2 para Y_1, e D_3 para Y_2)

Datapath = *Register File* + Unidade Funcional

Com uma **estrutura hierárquica de representação**, é possível reduzir a complexidade aparente do ***Datapath*** agrupando os **Registradores** e a **Lógica de Seleção dos Registradores** num ***Register File*** **(Banco de Registradores)**
Numa **Unidade Funcional** são agrupados o **Deslocador**, a ***ALU*** e os três ***MUXes***

Dessa forma, a implementação de um módulo pode ser substituída por outra, sem prejuízo da funcionalidade geral

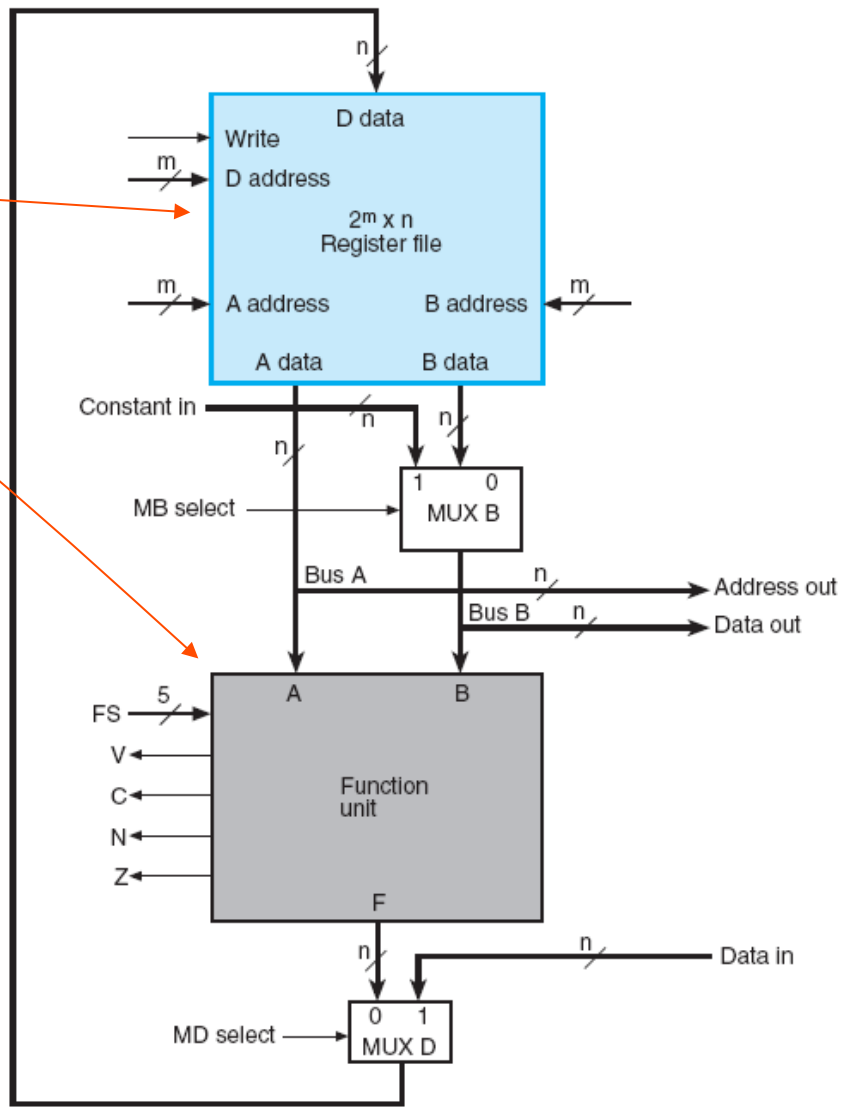
Na realidade, um ***Register File*** é implementado normalmente como um tipo especial de **Memória Rápida**, que permite a leitura ou escrita de mais de uma palavra simultaneamente

Assim, as entradas originais *A select*, *B select* e *Destination select* se tornam três endereços, como mostrados nesta figura

A address acessa uma palavra a ser lida em ***A data***, ***B address*** acessa uma segunda palavra a ser lida em ***B data***, e ***D address*** acessa uma palavra proveniente de ***D data*** a ser escrita (todos estes acessos ocorrem no mesmo ciclo de *clock*)

A entrada ***Write*** é equivalente ao ***Load Enable*** anterior Quando ***Write = 0***, o conteúdo dos registradores não se altera

Na ***Function Unit***, a entrada ***FS*** incorpora ***G***, ***H*** e ***MF select***



Logic and Computer Design Fundamentals – Mano e Kime

Register File Original

Suponha um ***Register File*** com *16* registradores de *32 bits*. Para selecionar estes registradores, seria necessário um *MUX 16x1* de *32 bits*. Na prática, este *MUX* seria muito ineficiente: haveria a necessidade de *16x32=512* conexões para ligar os registradores ao *MUX* ! (Problema de **Congestionamento**)
Além disso, as entradas de dados para todos os *16* registradores criaria um problema de ***fanout***, com correntes muito pequenas para controlar os transistores! Sem mencionar que conexões com baixas correntes tornam os dispositivos lentos, gerando problemas de **atrasos**!

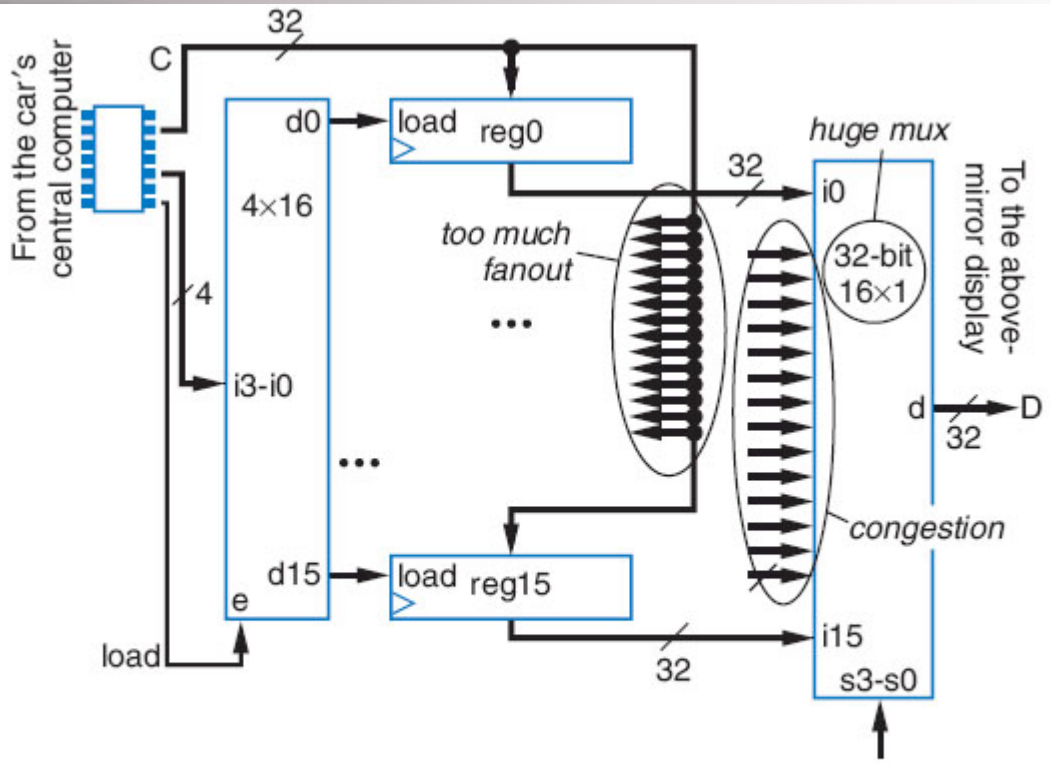


Figure 4.79 Above-mirror display design, assuming sixteen 32-bit registers. The mux has too many input wires, resulting in congestion. Also, the data lines C are fanned out to too many registers, resulting in weak current.

Os problemas de congestionamento, *fanout* e atrasos descritos acima podem ser contornados, considerando-se que os registradores não são todos escritos ou lidos simultaneamente.

Register File Aperfeiçoado

Um conjunto de registradores pode ser implementados de forma semelhante a uma memória, com a mesma rapidez, e acesso aos dados através de endereçamento

As conexões internas entre os componentes são cuidadosamente planejadas de modo a contornar os problemas de **congestionamento** e ***fanout***

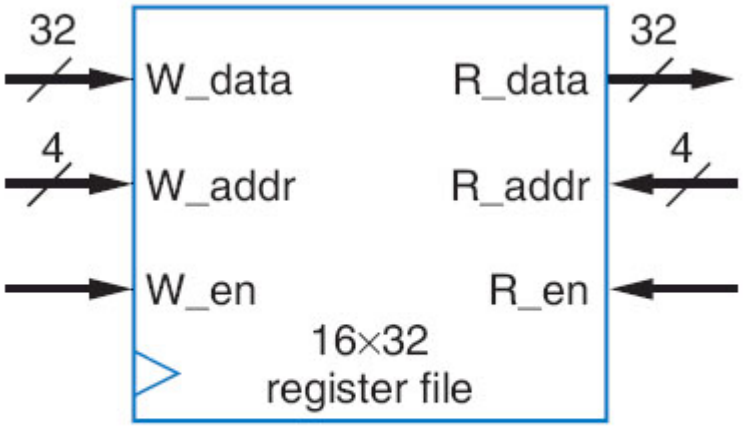


Figure 4.80 16x32 register file block symbol.

Para escrever um dado num registrador do *Register File*, bastaria colocar o dado em ***W_data***, indicar o endereço de um dos 16 registradores de 16 *bits* em ***W_addr*** e, no ciclo de *clock* desejado, habilitar a escrita com uma autorização em ***W_en***

O conjunto de entradas ***W_data***, ***W_addr***, e ***W_en*** é conhecido como Porta de Escrita do *Register File* (***Write Port***)

A operação de Leitura é similar à de Escrita

Register File Implementado

Quando *W_en=0*, nenhum registrador poderá ser escrito porque as saídas do decodificador permanecem em 0

Quando *W_en=1*, o decodificador decodifica o endereço em *W_addr* e coloca *1* no *load* de um dos registradores apenas. No próximo ciclo de *clock*, este registrador vai ser escrito com o valor de *W_data*

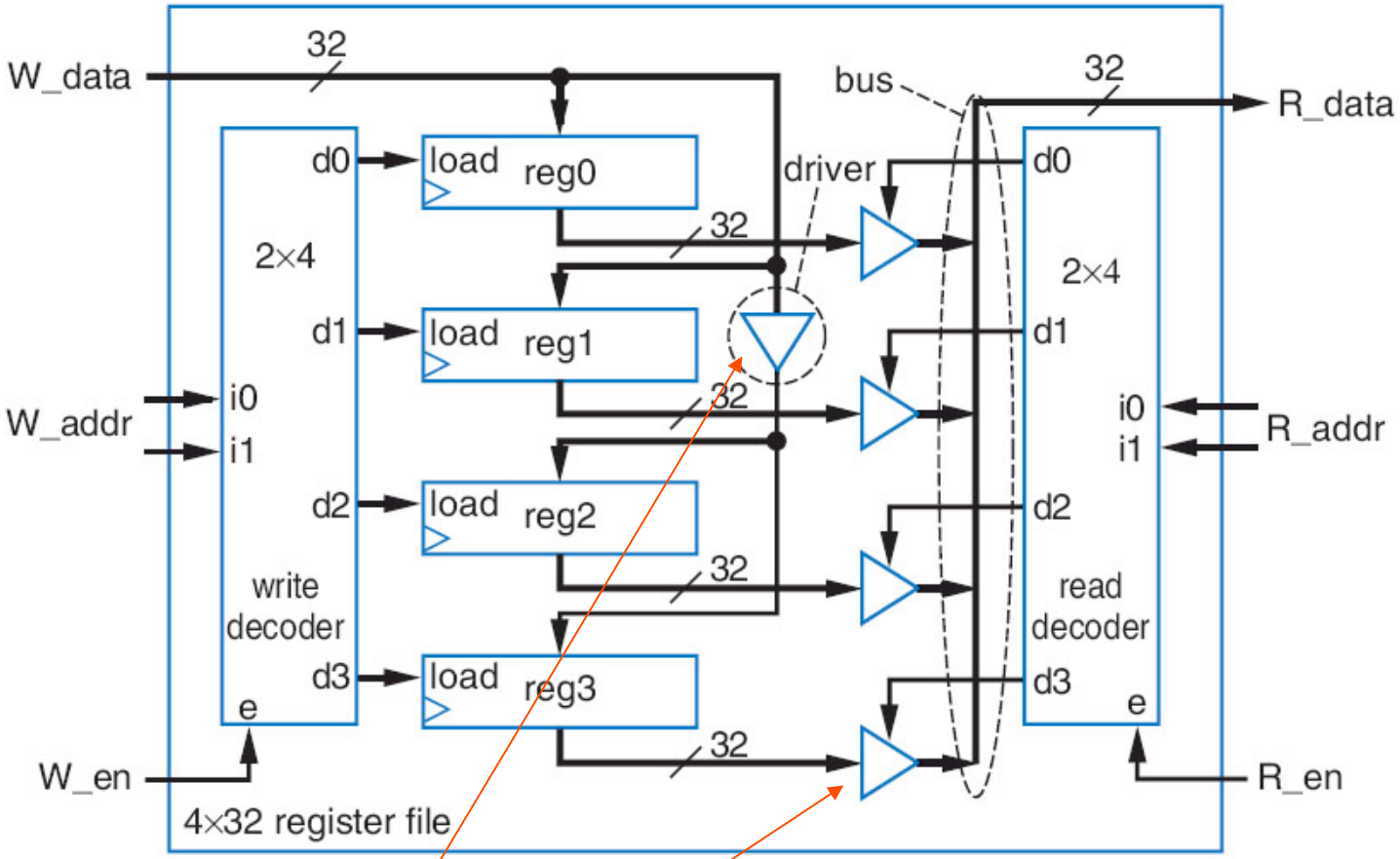


Figure 4.81 One possible internal design of a 4x32 register file.

Para resolver o problema de *fanout* são utilizados *drivers* (ou *buffers*), cuja entrada é semelhante à saída, porém com um sinal de saída mais forte (corrente maior)

Para substituir os MUXes, são usados *three-state drivers* (ou *three-state buffers*)

Códigos *FS* da Unidade Funcional

Para especificar todas as operações da ***Function Unit***, os códigos de ***MF select***, ***G select*** e ***H select*** estão redefinidos nos códigos da entrada ***FS***

Os códigos do ***MF Select*** estão na coluna mais à esquerda de *FS*

Os códigos do ***G Select*** estão nas colunas *2* a *5* de ***FS*** (com MF = 0)

Os códigos do ***H Select*** estão nas colunas *2* e *3* de ***FS*** (com MF = 1)

Se ***MF Select = 0***, então o código de ***G select*** determina a função na saída da ***Function Unit***

Se ***MF Select = 1***, então o código de ***H select*** determina a função na saída da ***Function Unit***

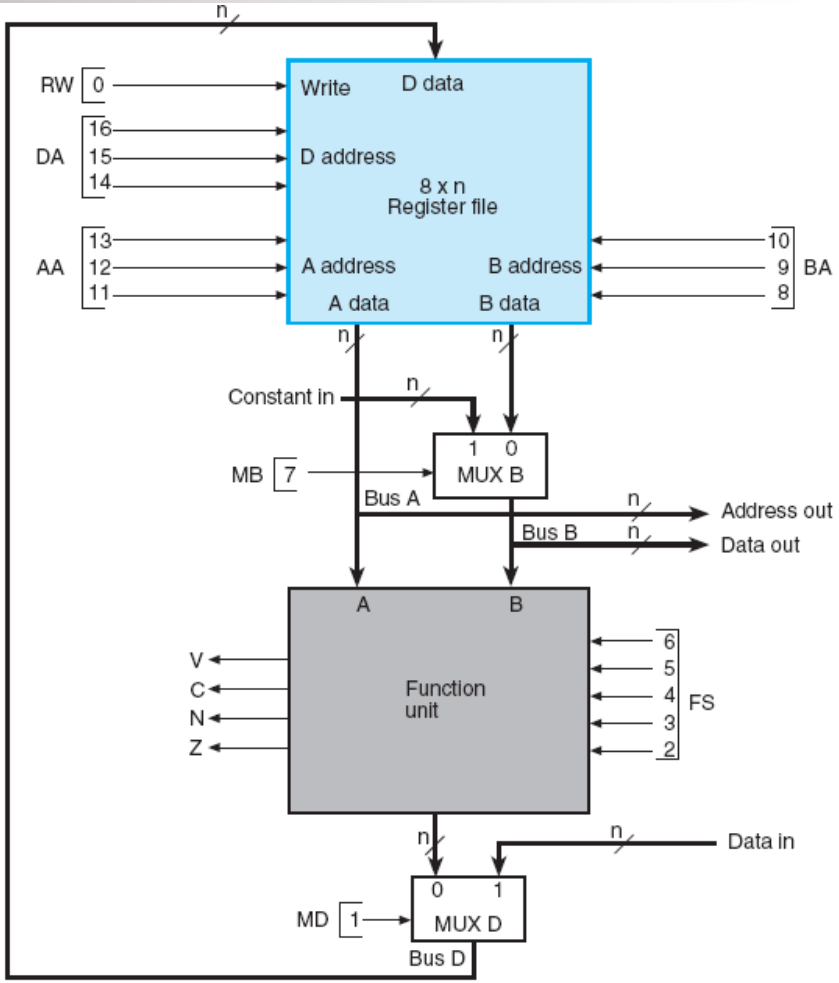
Os códigos ressaltados em azul são os que determinam as saídas da ***Function Unit***

FS	MF Select	G Select	H Select	Microoperation
00000	0	0000	00	$F = A$
00001	0	0001	00	$F = A + 1$
00010	0	0010	00	$F = A + B$
00011	0	0011	00	$F = A + B + 1$
00100	0	0100	01	$F = A + \overline{B}$
00101	0	0101	01	$F = A + B + 1$
00110	0	0110	01	$F = A - 1$
00111	0	0111	01	$F = A$
01000	0	1000	0	$F = A \wedge B$
01010	0	1010	10	$F = A \vee B$
01100	0	1100	10	$F = A \oplus B$
01110	0	1110	10	$F = \overline{A}$
10000	1	0000	00	$F = B$
10100	1	0100	01	$F = \text{sr } B$
11000	1	1000	10	$F = \text{sl } B$

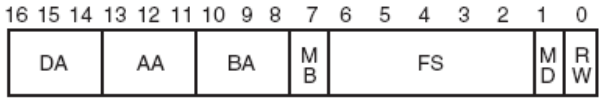
Palavra de Controle *(Control Word)*

As variáveis de seleção do ***datapath*** controlam as micro-operações executadas em cada pulso de *clock*, ou seja, elas controlam o endereço da leitura de dados dos registradores, a função executada pela ***Function Unit***, e os dados carregados no ***Register File***, assim como a seleção dos dados externos

- O ***Register File*** da ilustração ao lado possui oito registradores, ***R0*** a ***R7***
- O ***Datapath*** possui 17 entradas de controle binárias que, combinadas, formam uma **Palavra de Controle**
- Uma **Palavra de Controle** é normalmente constituída por vários ***campos*** (neste caso específico, sete campos), cada um deles identificado por um par de letras
- Os três campos dos registradores possuem três *bits* cada (necessários para selecionar um dos oito registradores utilizados ora como registradores de origem, ora como de destino do resultado das micro-operações)
- Os cinco *bits* do campo ***FS*** controlam as 15 operações da FU
- Esta **Palavra de Controle** de 17 *bits*, quando aplicada às entradas de controle, especifica uma micro-operação particular



(a) Block Diagram



(b) Control word

Codificação da Palavra de Controle

Esta tabela especifica os códigos de controle associados a cada uma das funções do ***Datapath***

Para cada campo são dados um nome simbólico e um código binário, associados a uma função

DA, AA, BA		MB		FS		MD		RW	
Function	**Code**	**Function**	**Code**	**Function**	**Code**	**Function**	**Code**	**Function**	**Code**
$R0$	000	Register	0	$F = A$	00000	Function	0	No write	0
$R1$	001	Constant	1	$F = A + 1$	00001	Data In	1	Write	1
$R2$	010			$F = A + B$	00010				
$R3$	011			$F = A + B + 1$	00011				
$R4$	100			$F = A + \overline{B}$ 1	00100				
$R5$	101			$F = A + \overline{B} + 1$	00101				
$R6$	110			$F = A - 1$	00110				
$R7$	111			$F = A$	00111				
				$F = A \wedge B$	01000				
				$F = A \vee B$	01010				
				$F = A \oplus B$	01100				
				$F = \overline{A}$	01110				
				$F = B$	10000				
				$F = \text{sr } B$	10100				
				$F = \text{sl } B$	11000				

Os campos ***DA, AA*** e ***BA*** possuem códigos binários equivalentes ao valor decimal do registrador escolhido

O campo ***RW*** corresponde à função que autoriza (*Write*) ou previne (*No Write*) a escrita a um dos registradores

Micro-operação com Notação Simbólica e Binária

A **Palavra de Controle** para uma micro-operação pode ser obtida com a especificação do valor de cada uma das variáveis de controle da **Notação Simbólica**

Por ex., a subtração ***R1 ← R2 + $\overline{R3}$ + 1***, especifica ***R2*** para a ***Entrada A*** e ***R3*** para a ***Entrada B***
A operação envolvida é ***F = A + $\overline{B}$ + 1*** e o registrador de destino é ***R1***, implicando ***RW = Write***

Micro-operation	DA	AA	BA	MB	FS	MD	RW
$R1 \leftarrow R2 + \overline{R3} + 1$	$R1$	$R2$	$R3$	Register	$F = A + \overline{B} + 1$	Function	Write
$R4 \leftarrow$ sl R6	$R4$	—	$R6$	Register	$F =$ sl B	Function	Write
$R7 \leftarrow R7 + 1$	$R7$	$R7$	—	Register	$F = A + 1$	Function	Write
$R1 \leftarrow R0 + 2$	$R1$	$R0$	—	Constant	$F = A + B$	Function	Write
Data out $\leftarrow R3$	—	—	$R3$	Register	—	—	No Write
$R4 \leftarrow$ Data in	$R4$	—	—	—	—	Data in	Write
$R5 \leftarrow 0$	$R5$	$R0$	$R0$	Register	$F = A \oplus B$	Function	Write

Micro-operation	DA	AA	BA	MB	FS	MD	RW
$R1 \leftarrow R2 - R3$	001	010	011	0	00101	0	1
$R4 \leftarrow$ sl R6	100	000	110	0	11000	0	1
$R7 \leftarrow R7 + 1$	111	111	000	0	00001	0	1
$R1 \leftarrow R0 + 2$	001	000	000	1	00010	0	1
Data out $\leftarrow R3$	000	000	011	0	00000	0	0
$R4 \leftarrow$ Data in	100	000	000	0	00000	1	1
$R5 \leftarrow 0$	101	000	000	0	01100	0	1

A **Palavra de Controle** para a micro-operação de subtração deve ser especificada pelos seus sete campos

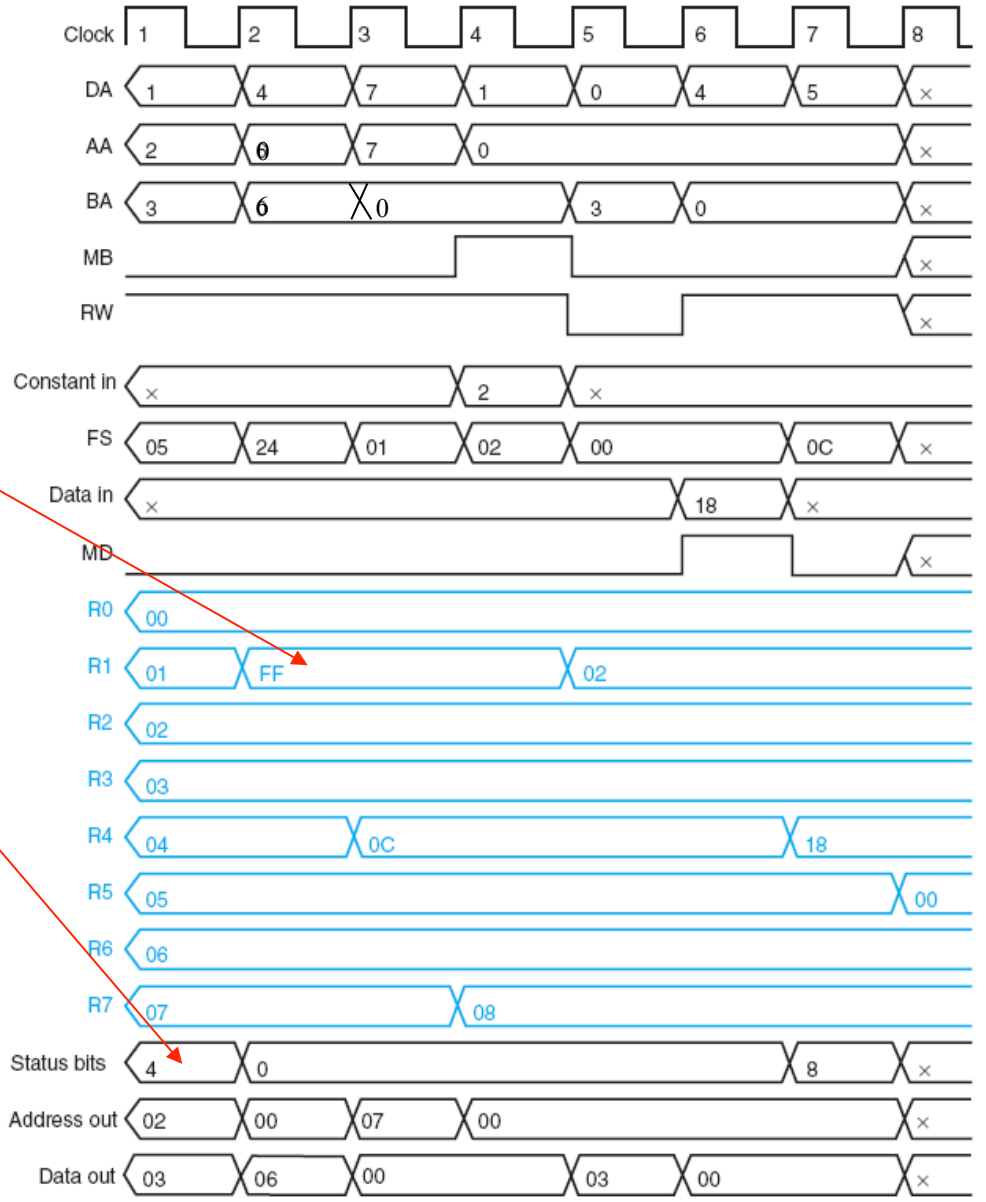
Para esta micro-operação de subtração, a **Palavra de Controle** deve ser 00101001100010101

Simulação de Sequência de Micro-operações

Nesta simulação, que corresponde à sequência de micro-operações da tabela anterior, o conteúdo de cada registrador de 8 *bits* é inicialmente carregado com um valor igual ao seu número de identificação (ex. ***R5*** contém 00000101_2 ou 05_{16})

Assim, no primeiro pulso do *clock* é realizada a operação ***R1*** ← ***R2*** – ***R3*** (= *2* – *3*), com o resultado aparecendo em ***R1*** no próximo pulso de *clock* como FF_{16}, ou seja, 11111111_2 e o *flag* do sinal (terceiro *bit* do ***Status bits*** - *Z, N, C, V*) igual a *1* (0100_2)

Obs. - na notação Complemento de *2*, 11111111_2 corresponde a $(-1)_{10}$



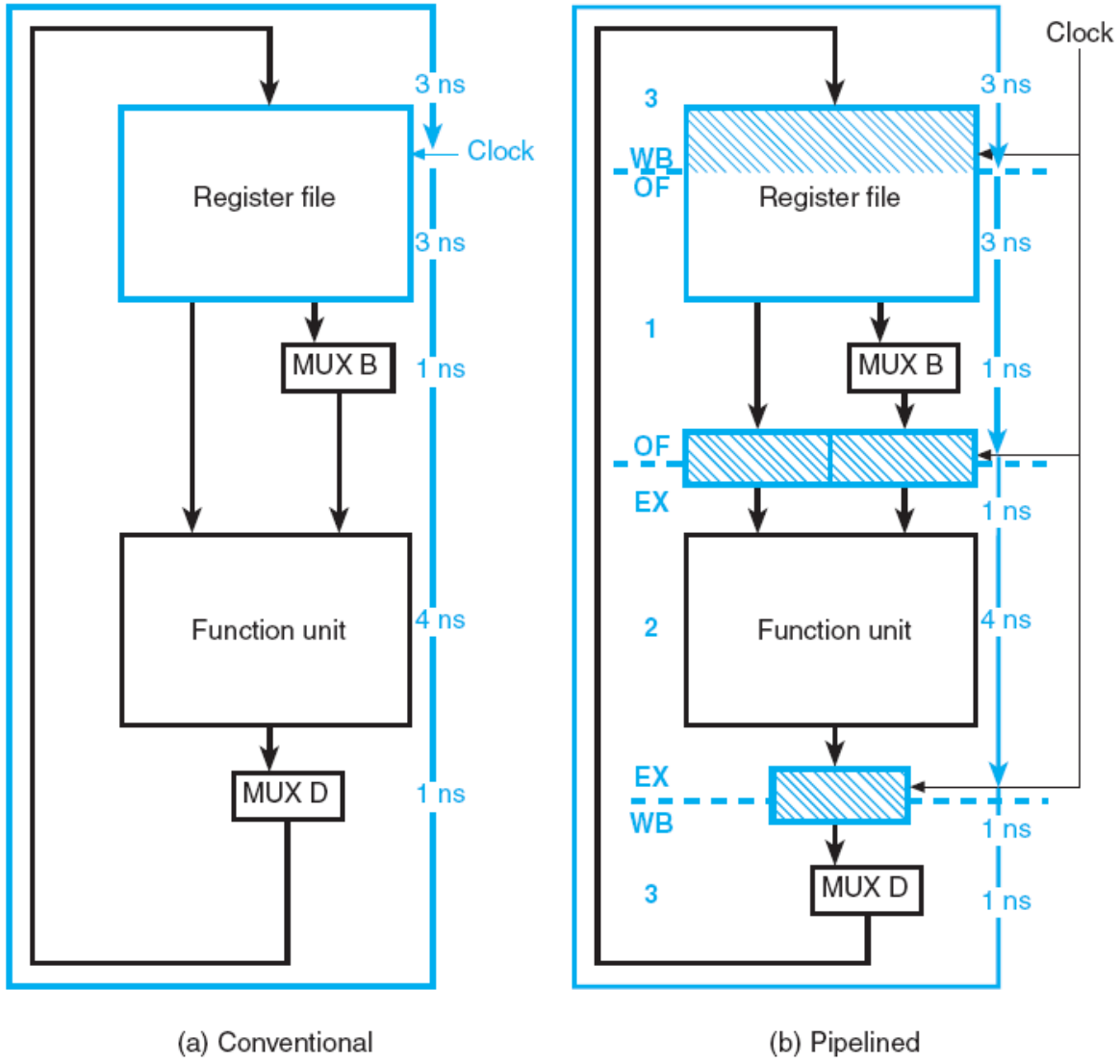
Pipelined Datapath

A taxa de execução de micro-operações no ***Datapath*** pode ser aumentada com uma técnica conhecida como ***Pipeline***

Para a realização de uma **micro-operação** completa são necessários *4ns* (*3+1*) para ler os dois operandos do ***register file***, outros *4ns* para executar a operação na ***function unit*** e, finalmente, mais *4ns* (*1+3*) para escrever o resultado no ***register file*** (incluindo o atraso no *MUX D*)

Portanto, o atraso total (período) para realizar uma micro-operação é de *12ns*, o que significa uma taxa máxima de execução (frequência) de *f = 83.3 MHz* (*1/12ns*)

Se o atraso de *12ns* do *Datapath* for dividido em três estágios por registradores adicionais, sempre que cada um destes estágios terminar suas operações, novos dados poderão ser enviados. Estágio 1: **Busca do Operando (*OF*)**, Estágio 2: **Execução (*EX*)** e Estágio 3: **Escrita (*WB*)**. Agora *f = 200 MHz* (*1/5ns*)



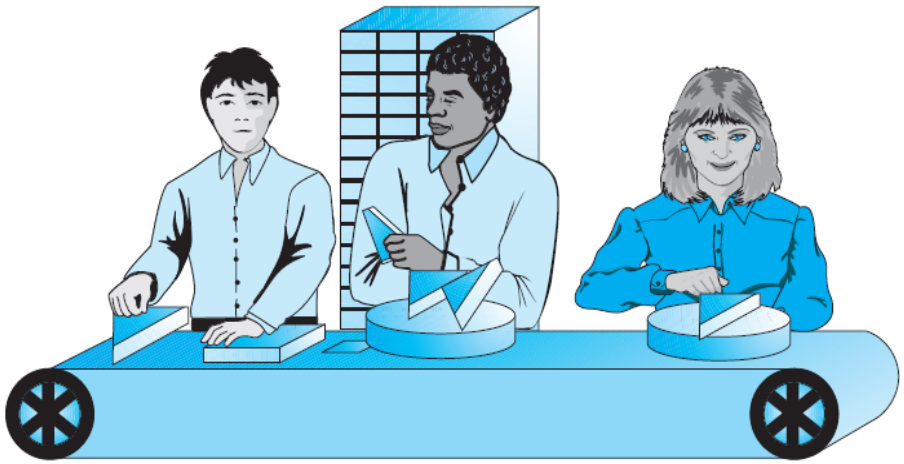
Analogia com uma Linha de Montagem

•Numa linha de montagem, o primeiro operário pega alguns componentes do armário e os coloca numa correia

•O segundo operário monta estes componentes

•Uma terceira funcionária pega os componentes montados numa peça e a coloca numa bandeja para operações subsequentes de montagem

•Cada operário precisa realizar apenas uma tarefa simples



•Tão logo cada um deles termine sua parte, a correia pode ser avançada para que as mesmas operações sejam repetidas em novos itens

•Imagine uma linha de montagem de estágio simples com apenas uma pessoa executando três operações em um minuto (cada uma das operações toma *20s*). Em outras palavras, a cada minuto uma peça é produzida

•Suponha agora uma linha de montagem com três estágios. Como um operação de montagem é completada a cada *20s* no último estágio da linha, a cada minuto três peças são produzidas !

•Embora ambas as linhas de montagem precisem de um minuto para completar uma peça, a linha de montagem com três estágios completa um número de peças três vezes maior ! Há um aumento no ***throughput*** !

Diagrama de Blocos do *Pipelined Datapath*

No *Datapath*, o primeiro estágio ou a primeira micro-operação consiste em **Buscar o Operando – OF** (em inglês, *Operand Fetch*) no *Register File*

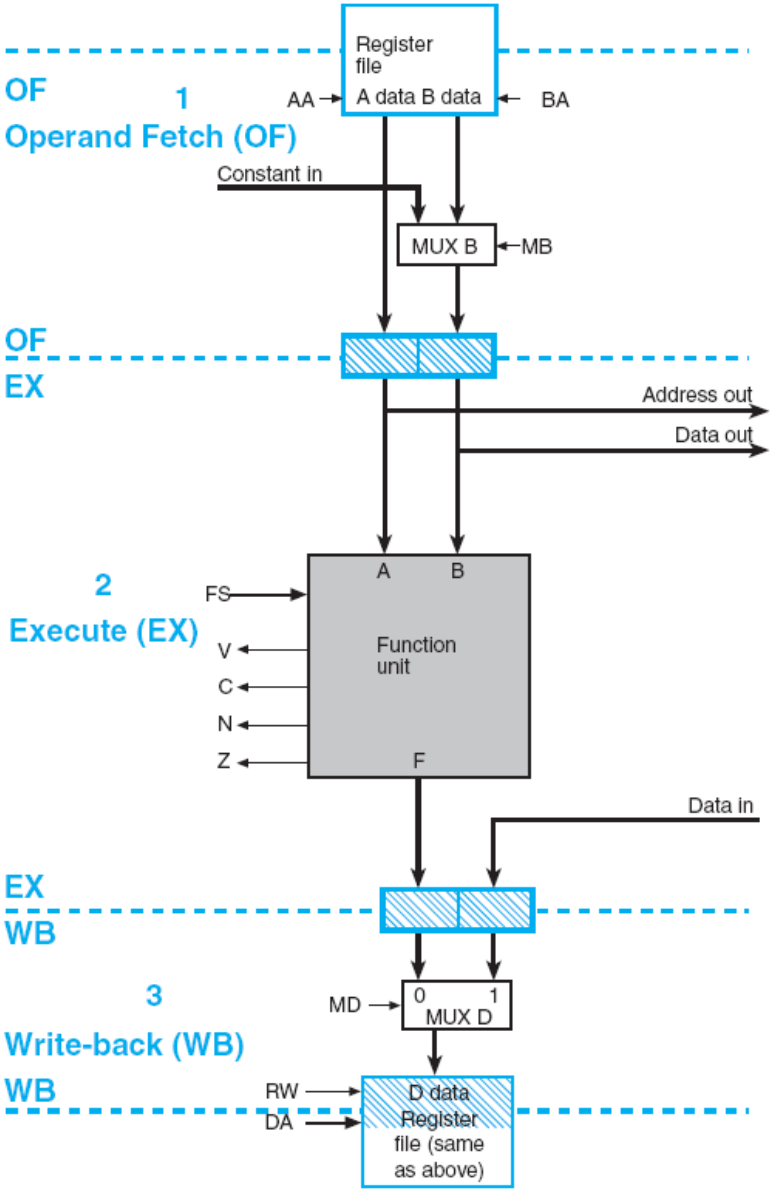
A segunda micro-operação corresponde à **Execução – EX** (em inglês, *Execute*) de uma operação na *Function Unit*

A terceira micro-operação consiste em **Escrever** ou **Gravar – WB** (em inglês, *Write-Back*) o resultado da operação no *Register File*

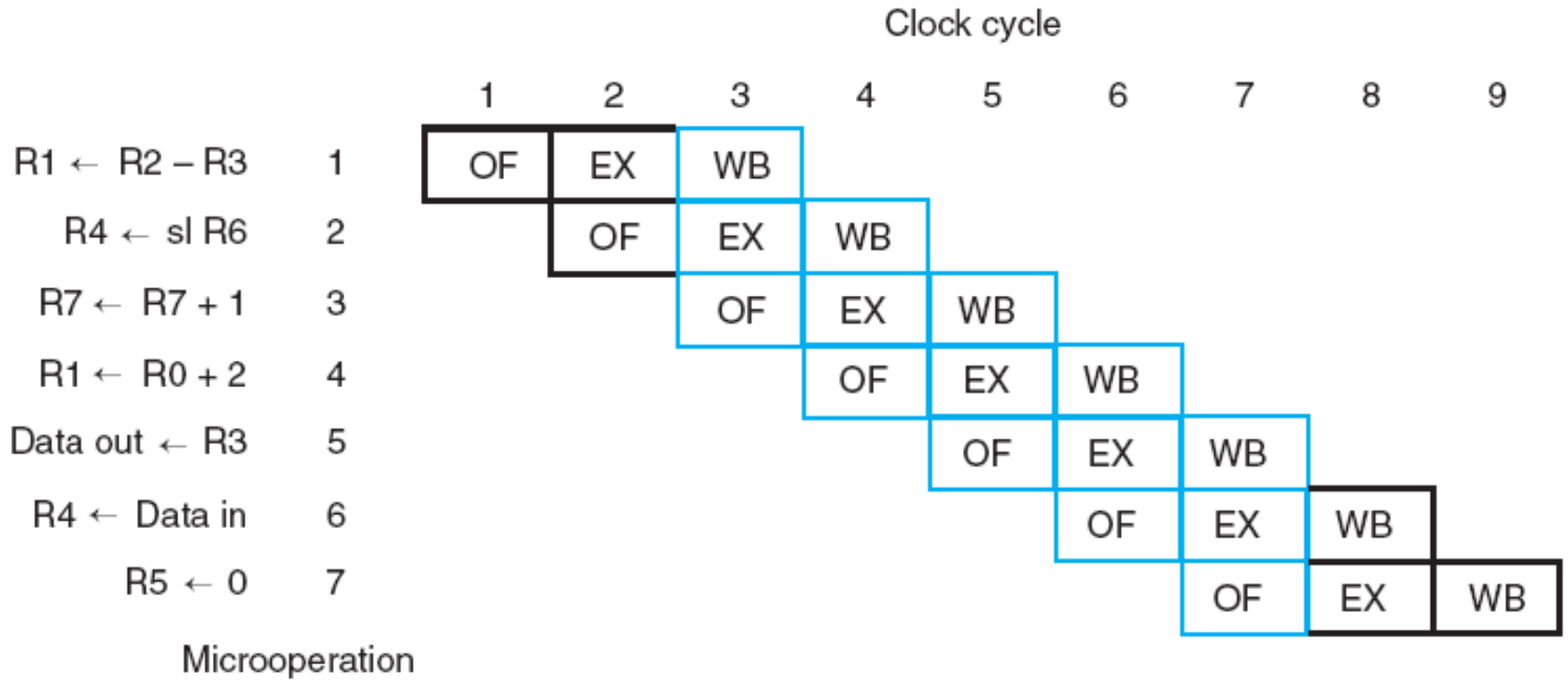
A analogia anterior sugere que se for possível ajustar uma estrutura com três estágios correspondentes aos avanços periódicos da correia, então serão processadas três vezes mais micro-operações no mesmo intervalo de tempo gasto pelo *Datapapth* convencional

A estrutura resultante é chamada *Pipeline* e os registradores entre os estágios são chamados de *Pipeline Platforms*

A vantagem do *Pipelined Datapath* é o *throughput* maior e o custo adicional são os *Pipeline Platforms*



Padrão de Execução do *Pipelined Datapath*



No período de *clock 1*, a micro-operação *1* está no estágio *OF*
No período de *clock 2*, a micro-operação *1* está no estágio *EX*, e a micro-operação *2* está no estágio *OF*
No período de *clock 3*, a micro-operação *1* está no estágio *WB*, a micro-operação *2*, no estágio *EX*, e a micro-operação *3*, no estágio *OF*

Nos dois primeiros períodos, nem todos os estágios do *pipeline* estão ativos, porém nos próximos cinco períodos (em azul), todos os estágios do *pipeline* estão ativos