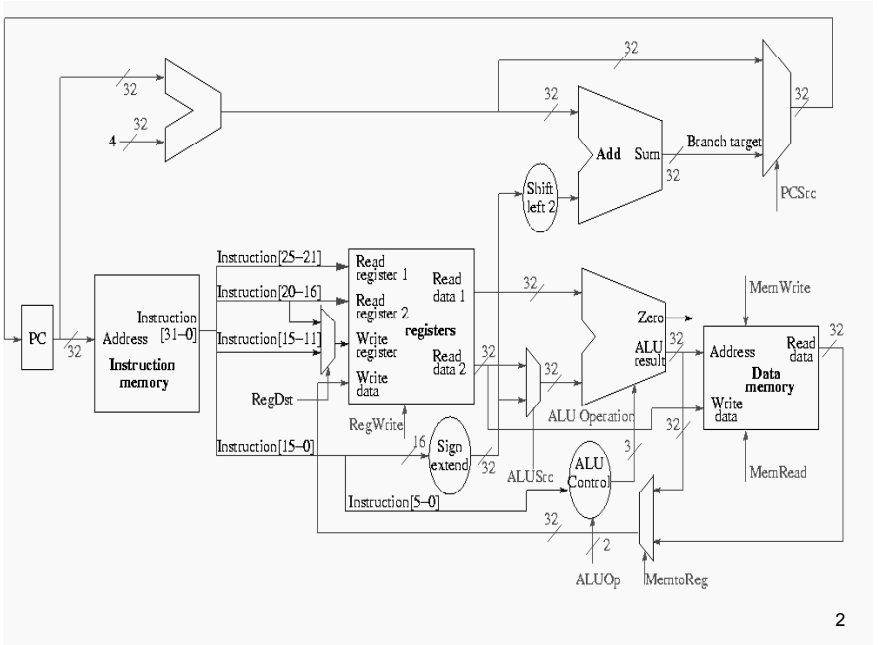
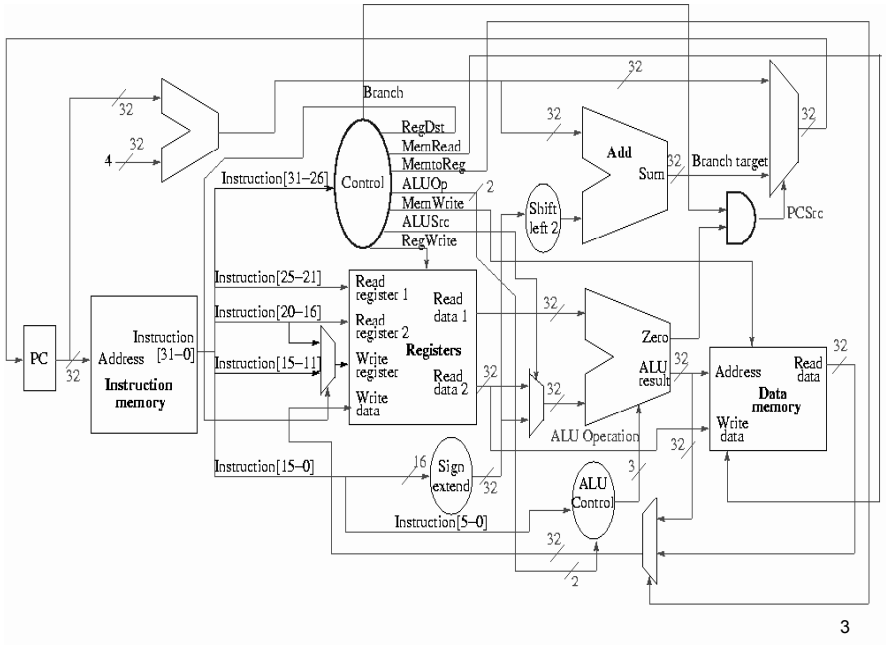
# Caminho de dados – Datapath










## Introdução

- O desempenho de uma máquina pode ser determinado por três fatores:
  - número de instruções executadas
  - período do clock (ou freqüência)
  - Número de ciclos por instrução (CPI)
- O compilador e a ISA (Instruction Set Arquitecture) determinam a quantidade de instruções a serem executadas por certo programa

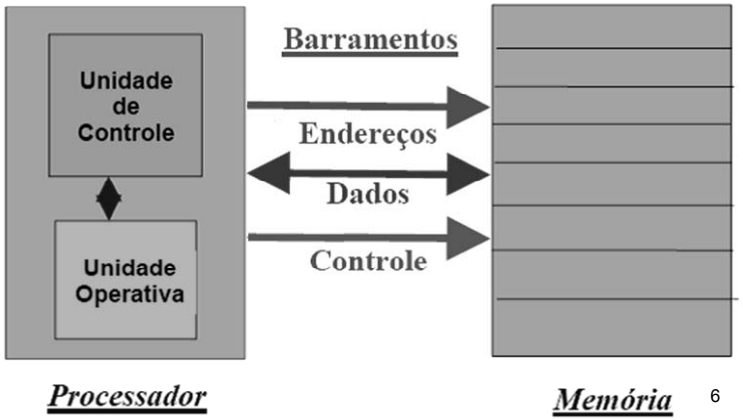
# Introdução

- Este capitulo aborda a implementação de um subconjunto das instruções do MIPS
- O interrelacionamento entre ISA e a implementação é exemplificado em dois projetos alternativos da parte operativa do processador
  - PO uniciclo
  - PO multiciclo



# Arquitetura Von Neumann

- Estrutura básica de um processador






# Unidade Operativa

- Também chamada "Parte Operativa", "Via de Dados" ou, em inglês, *"Datapath"*
- É construída a partir dos seguintes componentes:
  - elementos de armazenamento (registradores, ffs)
  - operadores lógico-aritméticos
  - recursos de interconexão (barramentos, mux)

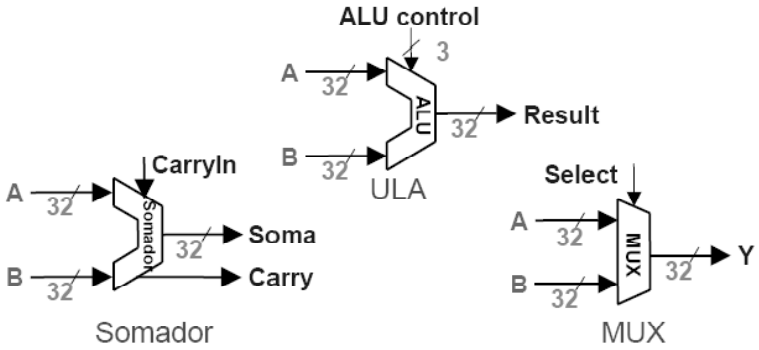


# Unidade Operativa MIPS

- Será projetada para implementar o seguinte subconjunto de instruções do MIPS:
  - Instruções de referência a memória:
    - *load word* (lw) e *store word* (sw)
  - Instruções Aritméticas e lógicas:
    - *add*, *sub*, *and*, *or* e *slt*
  - Instruções de desvio de fluxo:
    - *equal* (beq), *jump* (j)

# Componentes Combinacionais

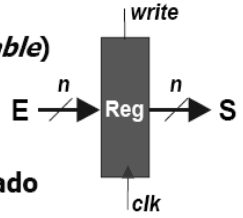
- Componentes combinacionais definem o valor de suas saídas apenas em função dos valores presentes nas suas entradas





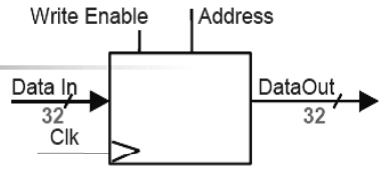

# Componentes Seqüenciais

- Componentes seqüenciais tem um *estado*, que define seu valor em um dado instante de tempo
- Registrador:
  - **Um conjunto de flip-flops tipo D (Registrador)**
    - **Com *n* bits de entrada e saída**
    - **entrada de habilitação de escrita (*write enable*)**
  - **Habilitação de escrita (*write enable*):**
    - **falso (0): O dado de saída não muda**
    - **verdade (1): o dado de entrada será carregado (saída = entrada)**






# Memória



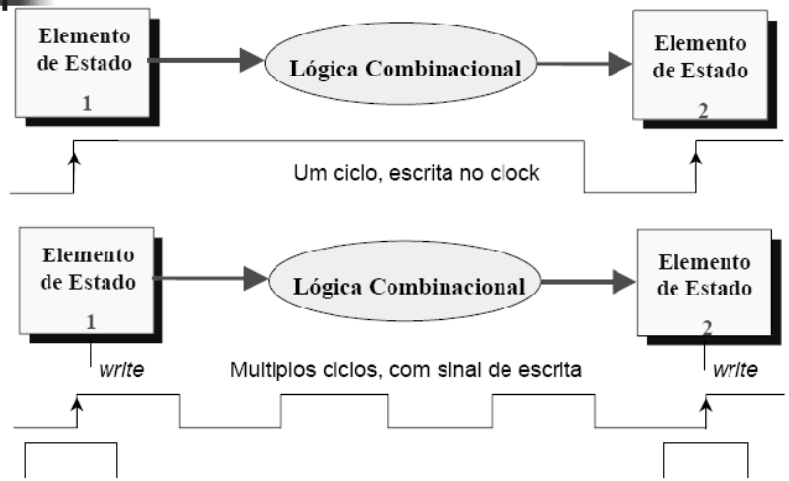

- Memória (idealizada)
  - Um barramento de entrada: *Data In*
  - Um barramento de saída: *Data Out*
- Uma palavra é selecionada por:
  - Um endereço seleciona a palavra para ser colocada na saída (*Data out*)
  - Write Enable = 1: Permite que a palavra selecionada seja escrita com a informação na entrada (*Data in*)
- Entrada de Clock (*CLK*)
  - Sincroniza os processos de acesso à memória
  - ***Geralmente, os processos são sincronizados pela borda de subida ou de descida do clock***



# Estratégia de Temporização

- Uma metodologia de temporização define quando os sinais podem ser lidos e quando eles podem ser escritos
- É necessário evitar situações de conflito, por exemplo, querer ler uma palavra e escrevê-la simultaneamente
- Será adotada uma metodologia de temporização sensível às transições do sinal do clock
- Nesta metodologia, qualquer valor armazenado nos elementos de estado só pode ser atualizado durante a transição do sinal de relógio (clk)

# Estratégia de Temporização



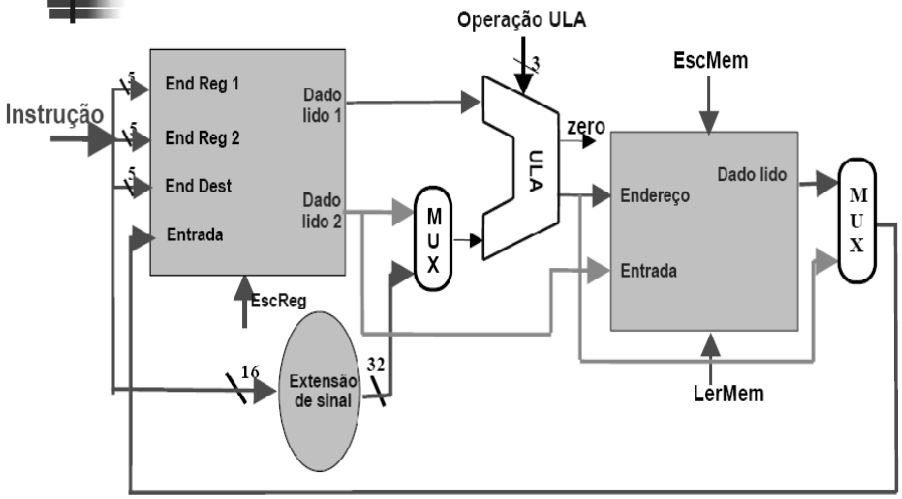



# Criando a Unidade Operativa

- **Uma maneira de se começar o projeto de uma unidade operativa (*data path*) é examinar cada um dos componentes necessários para a execução de cada uma das classes de instruções do MIPS**
- Elementos necessários:
  - um lugar para armazenar as instruções (*memória de instruções*)
  - Um registrador para armazenar o endereço de instrução (*Program Counter – PC*)
  - Um contador para incrementar o PC, para compor o endereço da próxima instrução (soma 4 para endereçar próxima instrução)



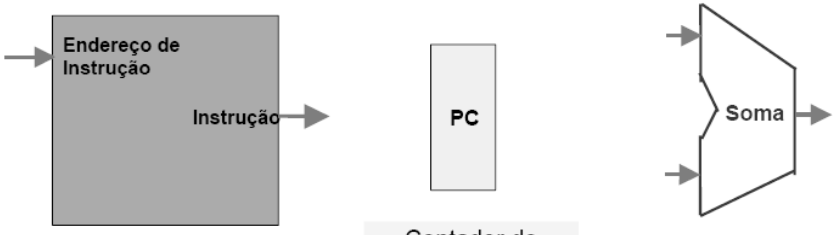
# Criando a Unidade Operativa ...






# Criando a Unidade Operativa ...

- Elementos Básicos

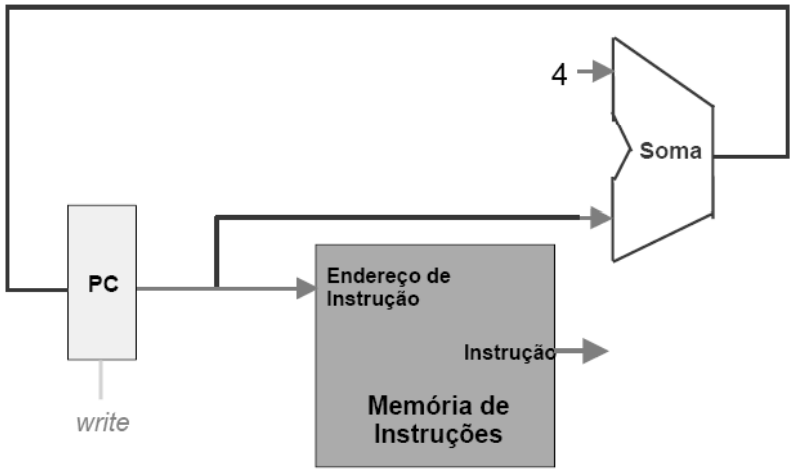



Memória de Instruções

Contador do programa (*program couter*)

Somador

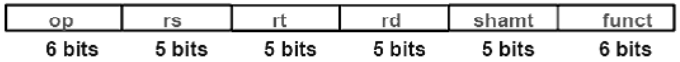
# Busca de Instruções






# Instruções Lógico-Aritméticas

- Formato de uma instrução tipo R no MIPS:

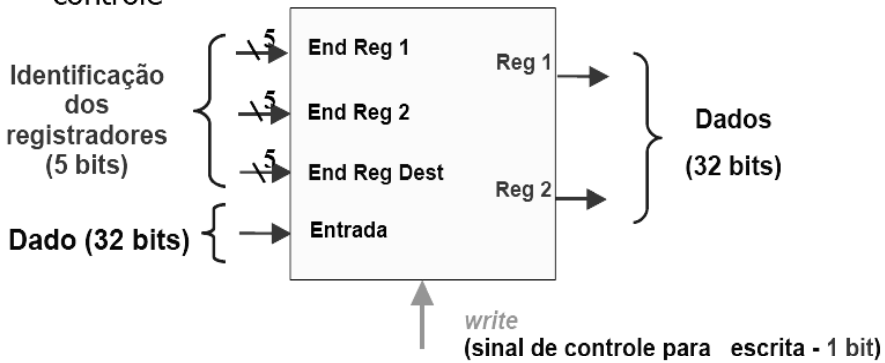



- Semântica:
  - $rd <- op($rs, $rt)
- Estrutura de suporte: banco de registradores



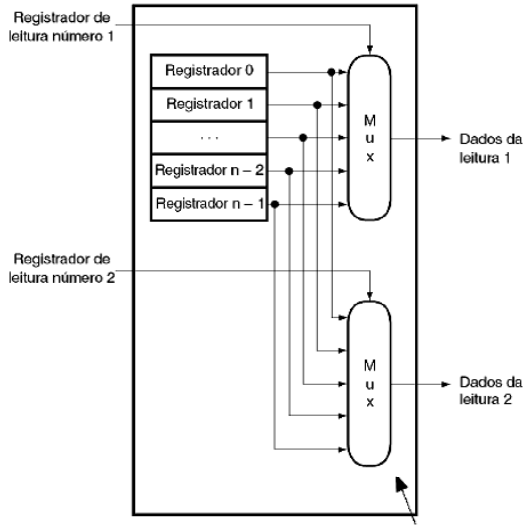
# Banco de Registradores

- Dupla porta: leitura de dois registradores ao mesmo tempo
- Sinal de controle para escrita - leitura não necessita controle

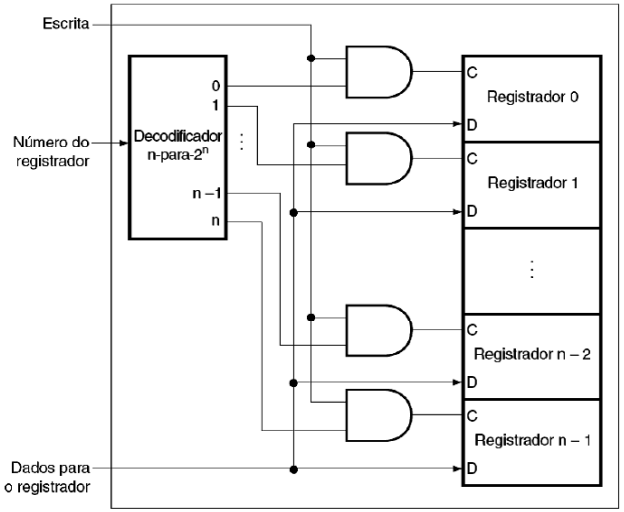





# Banco de Registradores



# Banco de Registradores

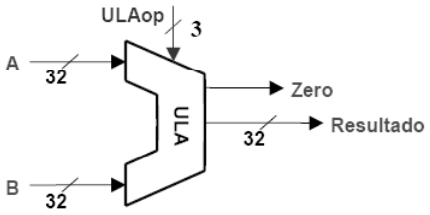





# Projeto da ULA

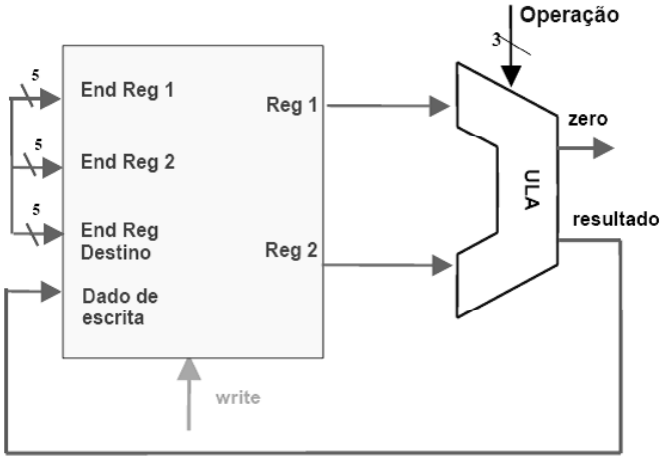
- A ULA foi desenvolvida no capítulo anterior
- 3 bits de controle indicam a operação a ser realizada

| ULAop | Função |
|---|---|
| 000 | and |
| 001 | or |
| 010 | add |
| 110 | sub |
| 111 | slt |






# Instruções Tipo R – unid. operativa






# Instruções de Acesso a Memória

- Formato da instrução tipo I (lw/sw):

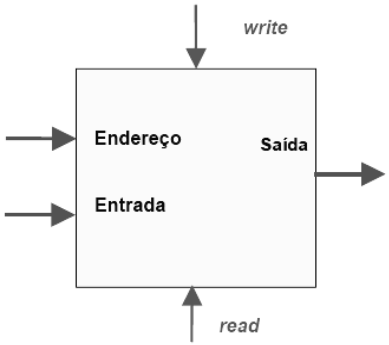
| op | rs | rt | endereço |
|---|---|---|---|
| 6 bits | 5 bits | 5 bits | 16 bits |

- Ex:
  - lw $8, 32($19)
  - end = $19 + 32

| 35 | 19 | 8 | 32 |
|---|---|---|---|

# Memória

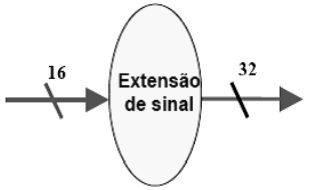
- Memória com um barramento de entrada independente do de saída
- Controle de escrita (*write*) e leitura (*read*)
- Barramento de endereços
- Um acesso de cada vez

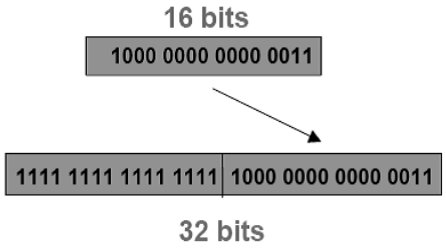





# Extensão de Sinal do Deslocamento

- Deslocamento na instrução deve ser estendido de 16 para 32 bits, mantendo-se o sinal
  - se for negativo, 16 bits superiores = 1
  - se for positivo, 16 bits superiores = 0



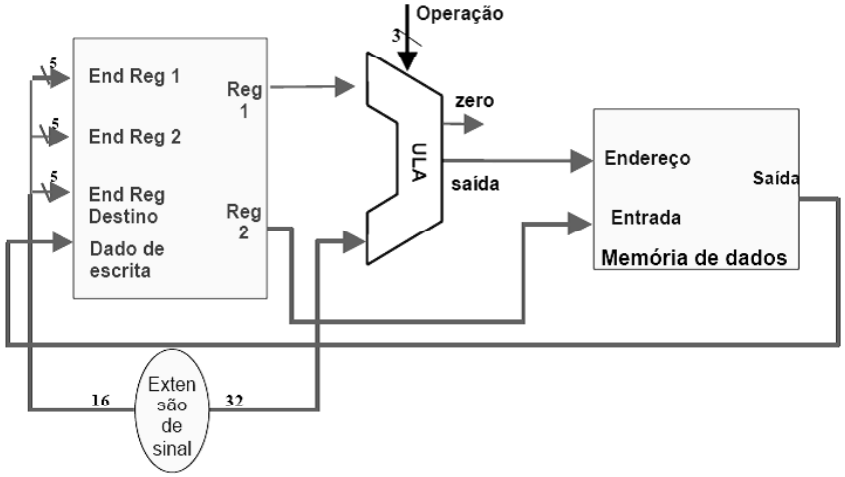



# Acesso a Memória

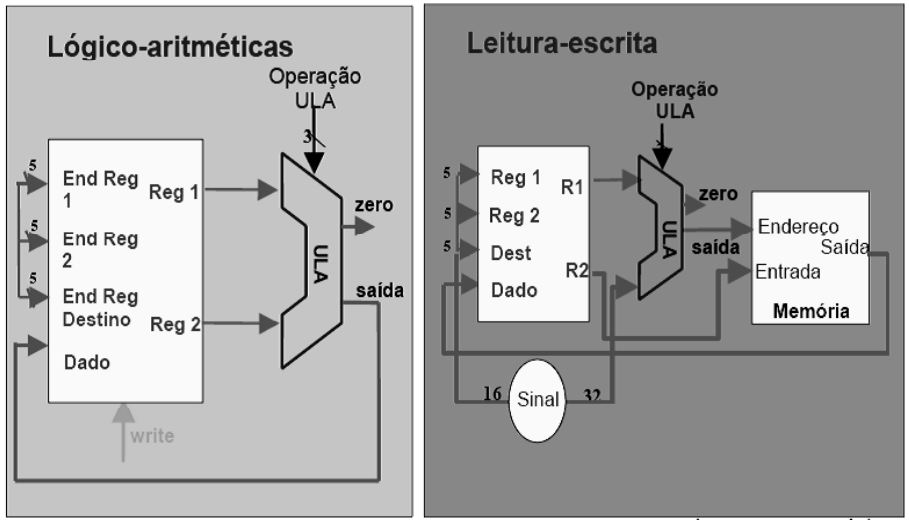
- lw lê um dado da memória e escreve em um registrador
  - conexão entre a saída da memória e a entrada do banco de registradores
- sw lê um dado de um registrador e escreve na memória
  - ligação entre saída do banco de registradores e entrada de dados da memória
- endereço calculado através da ULA
  - saída da ULA ligada ao barramento de endereços da memória



# Unidade Operativa lw/sw

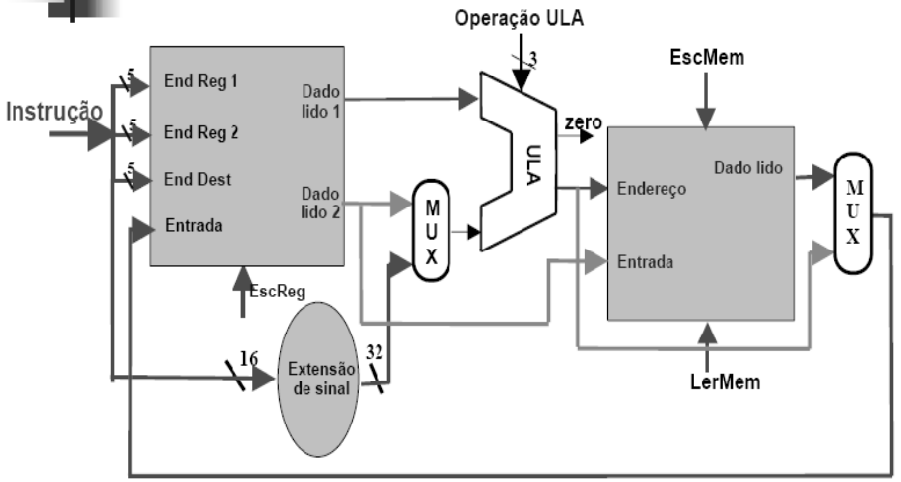


# Instruções Lóg/Arit e lw/sw



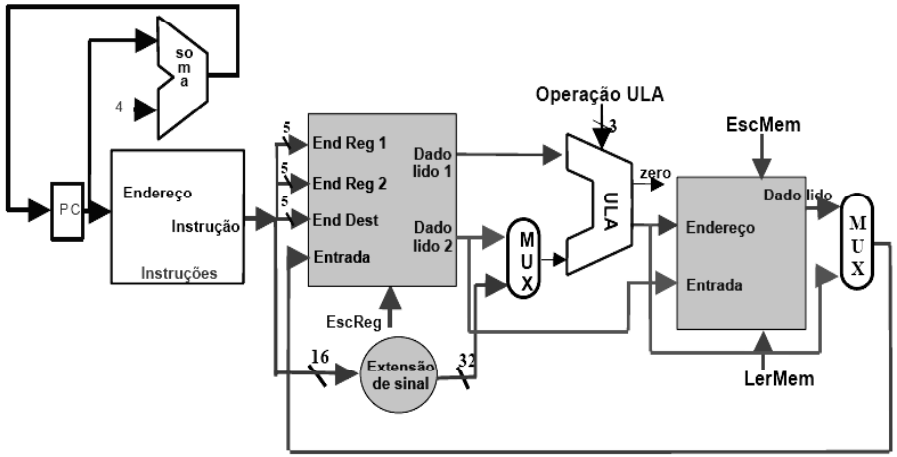



# Combinando as Unidades






# Acrescentando a Busca






# Instruções de Desvio

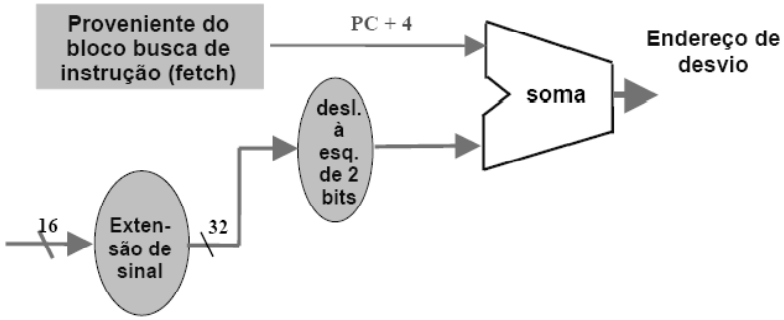
- beq $1, $2, desloc
  - compara dois registradores
  - soma desloc a PC+4 se $1 = $2
    - PC + 4 e o endereço da próxima instrução
  - no montador utiliza-se uma versão simplificada, com o rótulo do destino
    - beq $1, $2, Rótulo
    - neste caso, o montador calcula o deslocamento
  - desloc é um deslocamento de palavras, ou seja, cada unidade de desloc corresponde a 4 bytes

# Instruções de Desvio

- A comparação entre os registradores é feita subtraindo-os na ULA e verificando se o resultado é zero
- O PC deve ser carregado com PC + 4 ou PC + 4 + desloc*4, de acordo com o resultado do teste
- A multiplicação de *desloc* por 4 é feita deslocando-se de 2 bits o seu valor

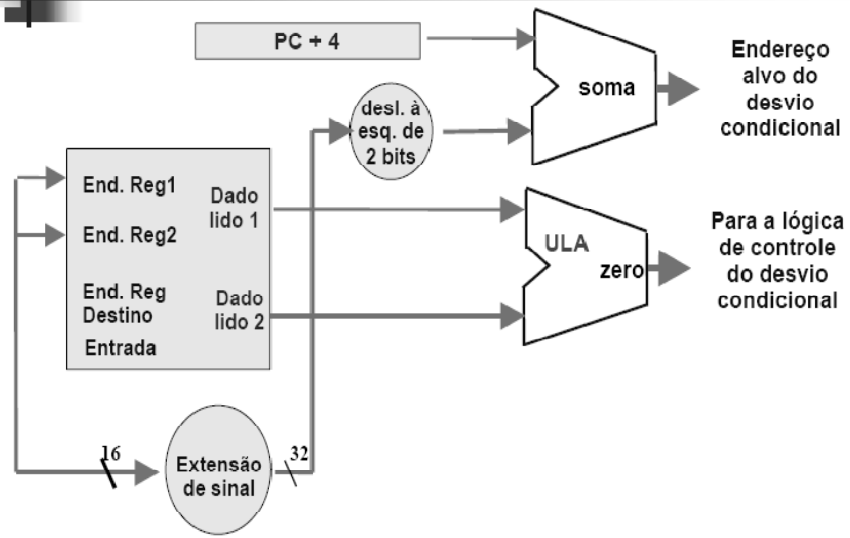


# Cálculo do Endereço de Desvio






# Circuito Cálculo Endereço Desvio






# Observações

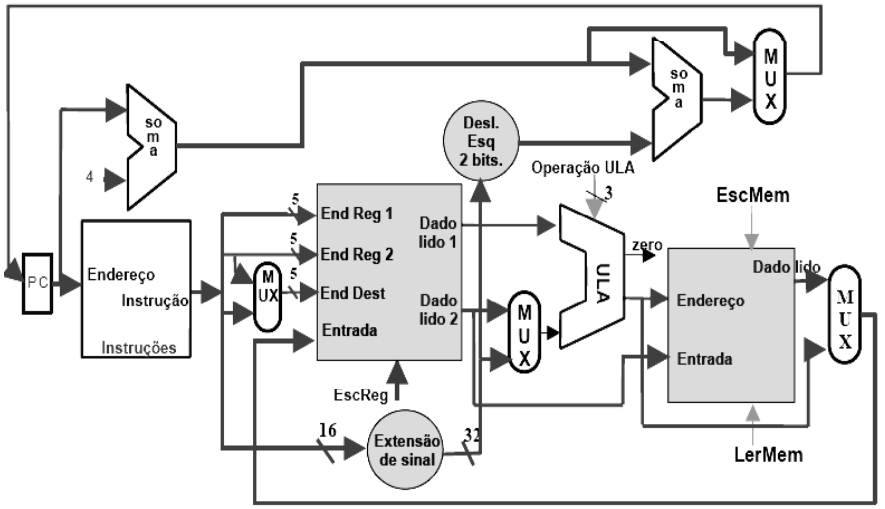
- Para realizar a comparação dos dois operandos precisamos usar o banco de registradores (os dois operandos estão lá)
- O cálculo do endereço de desvio foi incluído no circuito (já estudado)
- Na instrução não é preciso escrever no banco de registradores
- A comparação será feita pela ULA, subtraindo-se os registradores e utilizando a saída *zero para verificar a igualdade*

# MIPS Uniciclo

- Foi desenvolvido, na verdade, três tipos de unidades operativas:
  - **uma para instruções no formato R (add, sub, etc.)**
  - uma para instruções de no formato I ( *load* e *store)*
  - uma para *instruções condicionais (formato I)*
- Na fase de projeto as vezes precisamos replicar recursos
- A via de dados mais simples deve-se propor executar as instruções num único período do clock
- Isto quer dizer que nenhum dos recursos pode ser usado mais de uma vez por instrução

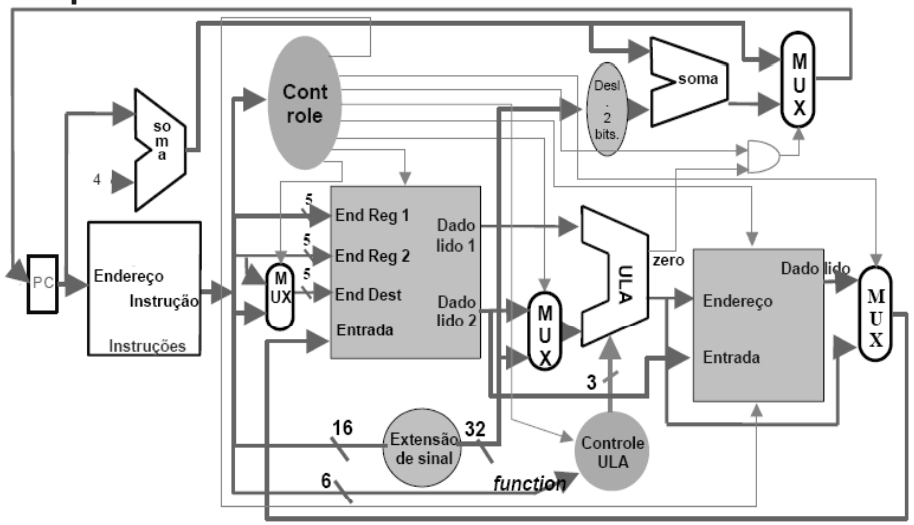


# Final






# MIPS Uniciclo



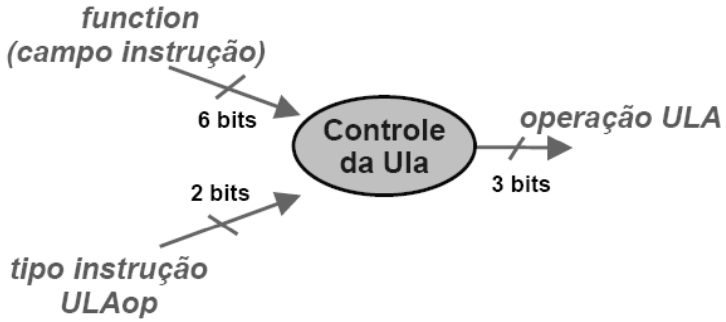



# Controle do MIPS

- Cada operação requer a utilização adequada dos recursos do MIPS
- Ex: *add $t2, $s1, $s2*
  - é necessário que os endereços dos operandos $s1 e $s2 sejam enviados ao banco de registradores
  - Da mesma maneira, o endereço do registrador destino ($t2) deverá ser informando ao banco de registradores
  - Uma vez que o banco de registradores disponibilize os valores de $s1 e $s2, estes deverão ser encaminhados à ULA
  - Posteriormente, o resultado deverá ser escrito no banco de registradores (registrador $t1)

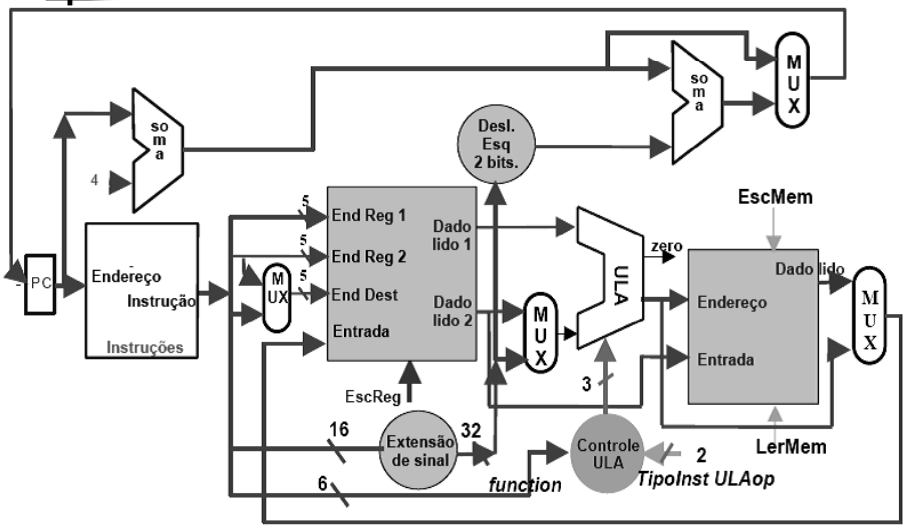
# Controle da ULA

- Lógica da unidade de controle encarregada de gerar os bits que determinam a operação da ULA






# Lógica de Controle da ULA






**Classes de instruções - tipo-R, load, store e branch)**

| Field | 0 | rs | rt | rd | shamt | funct |
|---|---|---|---|---|---|---|
| Bit positions | 31–26 | 25–21 | 20–16 | 15–11 | 10–6 | 5–0 |

a. R-type instruction

| Field | 35 or 43 | rs | rt | address |
|---|---|---|---|---|
| Bit positions | 31–26 | 25–21 | 20–16 | 15–0 |

b. Load or store instruction

| Field | 4 | rs | rt | address |
|---|---|---|---|---|
| Bit positions | 31–26 | 25–21 | 20–16 | 15–0 |

c. Branch instruction



**Classes de instruções - tipo-R, load, store e branch)**

| Field | 0 | rs | rt | rd | shamt | funct |
|---|---|---|---|---|---|---|
| Bit positions | 31–26 | 25–21 | 20–16 | 15–11 | 10–6 | 5–0 |

a. R-type instruction

| Field | 35 or 43 | rs | rt | address |
|---|---|---|---|---|
| Bit positions | 31–26 | 25–21 | 20–16 | 15–0 |

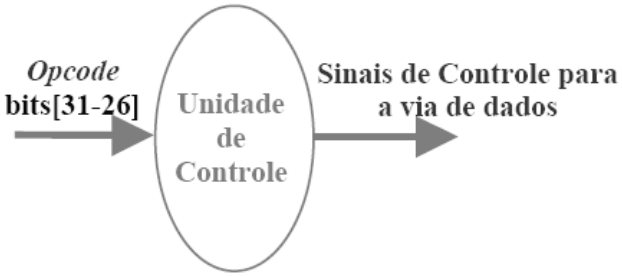
b. Load or store instruction

| Field | 4 | rs | rt | address |
|---|---|---|---|---|
| Bit positions | 31–26 | 25–21 | 20–16 | 15–0 |

c. Branch instruction

# Unidade de Controle Uniciclo

- A unidade de controle deve, a partir do código da instrução, fornecer os sinais que realizam as instruções na unidade operativa





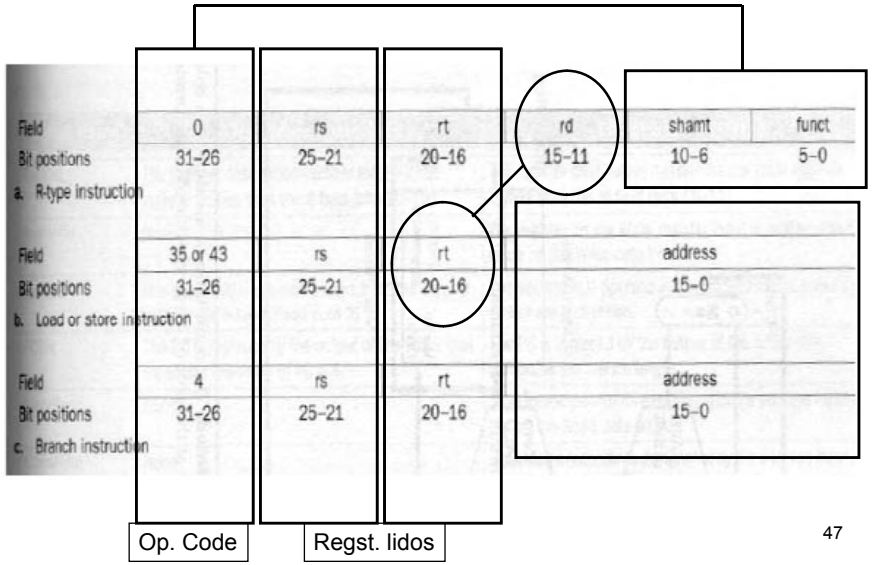

# Controle do Add

- Por exemplo, a execução da instrução
  - *add $t1, $s0, $s2*

  requer as seguintes tarefas:
  - encaminhar para a ULA o conteúdo dos registradores $s0 e $s2
  - indicar para a ULA que vai ser realizada uma operação da adição
  - Encaminhar o resultado para o registrado $t1



**Classes de instruções - tipo-R, load, store e branch)**






# Entrada da Unidade de Controle

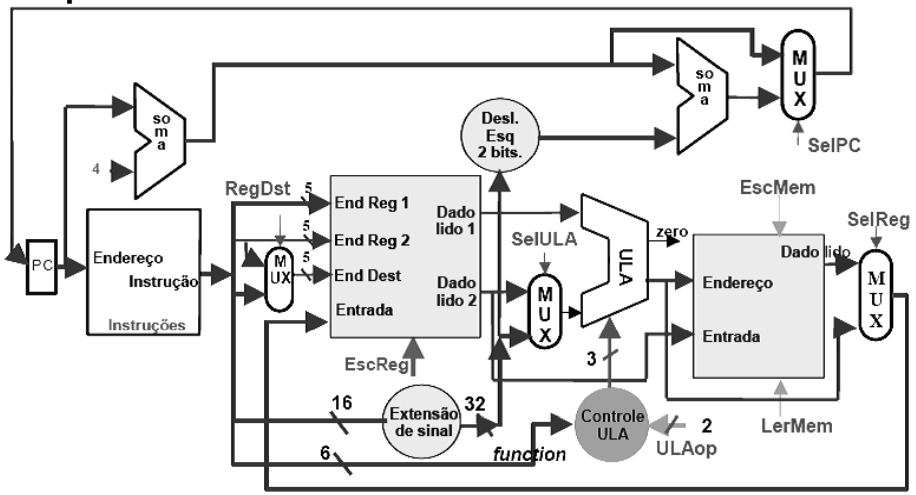
- as informações necessárias a execução de uma instrução são retiradas da própria instrução
  - O *opcode* (código de operação) sempre está nos bits [31-26]
  - Os 2 registradores a serem lidos (rs e rt) sempre estão nas posições [25-21] e [20-16] (para todos os formatos!!!)
  - O registrador base (rs) para as instruções lw e sw sempre está especificado nas posições [25-21]
  - Os 16 bits de deslocamento para as instruções beq, lw e sw estão sempre nas posições [15-0]
  - O registrador destino está em uma das duas posições
  - [20-16] para lw (registrador rt)
  - [15-11] para instruções aritméticas/lógicas (registrador rd)

## Sinais de Controle

- A unidade de controle de prover:
  - sinais para os multiplexadores
  - sinais de leitura e escrita para as memórias
  - seleção da operação da ULA
  - controle do novo endereço a ser carregado no PC, para instruções de salto



## Sinais de controle






## Instruções de Desvio

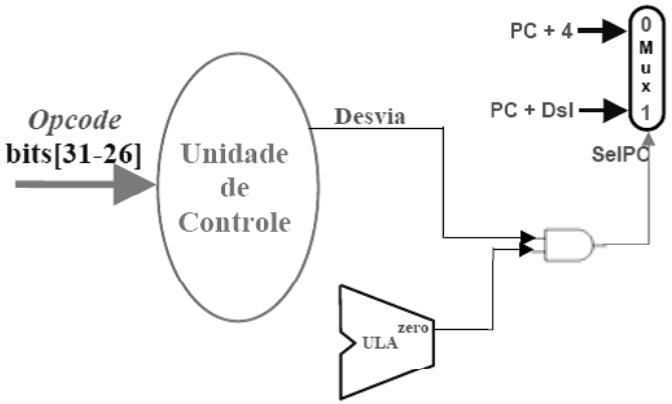
- A instrução de desvio condicional coloca:
  - PC+Desl ou (salto)
  - PC + 4 (instrução seguinte)

no contador de programa PC

- É necessário selecionar SelPC em função:
  - do código da instrução (beq, bne)
  - do resultado da comparação (sinal zero da ULA)



## Controle de Desvio

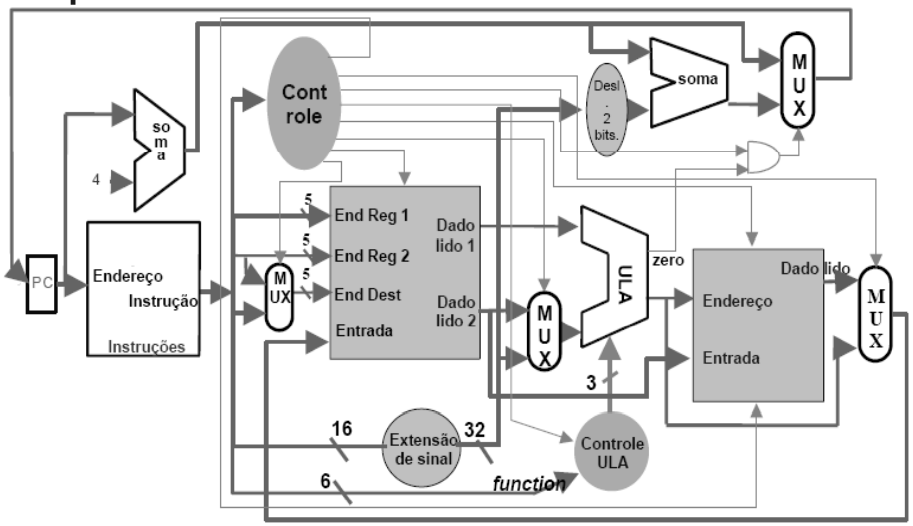


# Sinais de Controle

- LerMEM: dado da memória no endereço especificado é lido e colocado na saída
- EscMEM: conteúdo da memória endereçado recebe o dado
- SelULA:
  - 0 – operando é a segunda saída do banco de registradores
  - 1 – operando é o deslocamento extendido
- RegDST:
  - 0 – índice do registrador destino é o campo rt da instrução
  - 1 – índice do registrador destino é o campo rd da intrução
- EscREG: escreve dado no registrador indexado
- SelPC:
  - 0 – PC recebe próximo endereço (PC+4)
  - 1 – PC recebe endereço de desvio
- SelREG:
  - 0 – registrador recebe saída da ULA
  - 1 – registrador recebe saída da memória



# MIPS Uniciclo






# Exercício

- Estender a organização do MIPS para dar suporte a execução de JUMP, desvio incondicional
- O endereço de desvio é obtido por:
  - PC[31 – 28] # Instrução[25 – 0] # 00
  - onde # indica concatenação de bits



# Problemas com MIPS Uniciclo

- Período do relógio determinado pelo caminho mais longo
  - instrução lw:
    - leitura da instrução
    - leitura do registrador de base, extensão de sinal
    - cálculo do endereço
    - leitura do dado da memória
    - escrita em registrador
- TODAS as instruções levam o mesmo tempo para executar

# Exemplo

- Supondo os seguintes tempos de execução das unidades do MIPS:
  - Acesso a memória: 10 ns
  - ULA e somadores: 10 ns
  - Acesso ao banco de registradores: 5 ns
  - outros: 0 ns
  - **Quais os tempos de execução das instruções supondo uma implementação uniciclo e outra com ciclo variável, ou seja, duração do ciclo igual a duração da instrução?**




- Acesso a memória: 10 ns
- ULA e somadores: 10 ns
- Acesso ao banco de registradores: 5 ns
- outros: 0 ns
- **Quais os tempos de execução das instruções supondo uma implementação uniciclo e outra com ciclo variável, ou seja, duração do ciclo igual a duração da instrução?**


Determinar o tempo de execução de cada instrução

lw =

sw =

tipo-R =

beq =

j =




- Acesso a memória: 10 ns
- ULA e somadores: 10 ns
- Acesso ao banco de registradores: 5 ns
- outros: 0 ns
- **Quais os tempos de execução das instruções supondo uma implementação uniciclo e outra com ciclo variável, ou seja, duração do ciclo igual a duração da instrução?**


Determinar o tempo de execução de cada instrução

lw =
10 + 5 + 10 + 10 + 5 = 40

sw =
10 + 5 + 10 + 10 = 35

tipo-R =
10 + 5 +10 + 5 = 30

beq =
10 + 5 + 10 = 25

j =
10



# Exemplo ...

- Considerando a distribuição de instruções do benchmark gcc, a diferença de velocidade entre as implementações seria:
  - GCC: 22% lw, 11% sw, 49% tipo-R, 16% beq, 2% jump
  - Período uniciclo: 40 ns
  - Período ciclo variável:

    40*0.22 + 35*0.11 + 30*0.49 + 25*0.16 + 10*0.2 = 31.6 ns
  - Ganho: 40/31.6 = 1,27

## MIPS Multiciclo

- Idéia: cada fase de execução de uma instrução dura um ciclo de relógio
- Instruções podem ser executadas em um número diferente de ciclos
  - lw leva 5 ciclos
  - jump leva 1 ciclo
  - add leva 4 ciclos

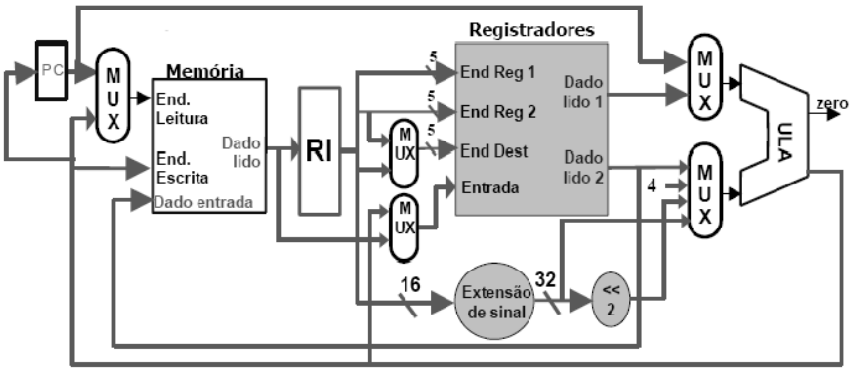


## MIPS Multiciclo

- Ciclo dimensionado de acordo com a fase mais demorada
- Unidades funcionais podem ser utilizadas para realizar mais de uma operação durante a execução de uma instrução
- A organização da parte operativa pode ser re-estruturada em função destas características



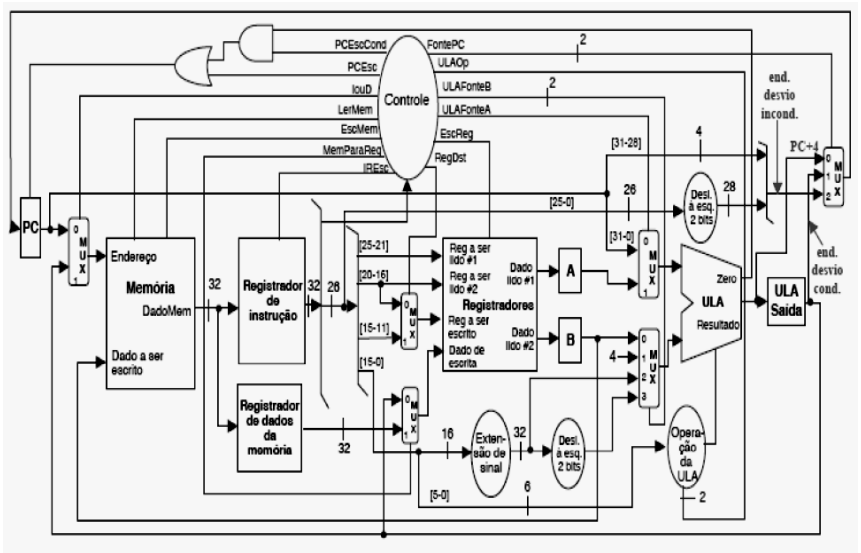
## Multiciclo



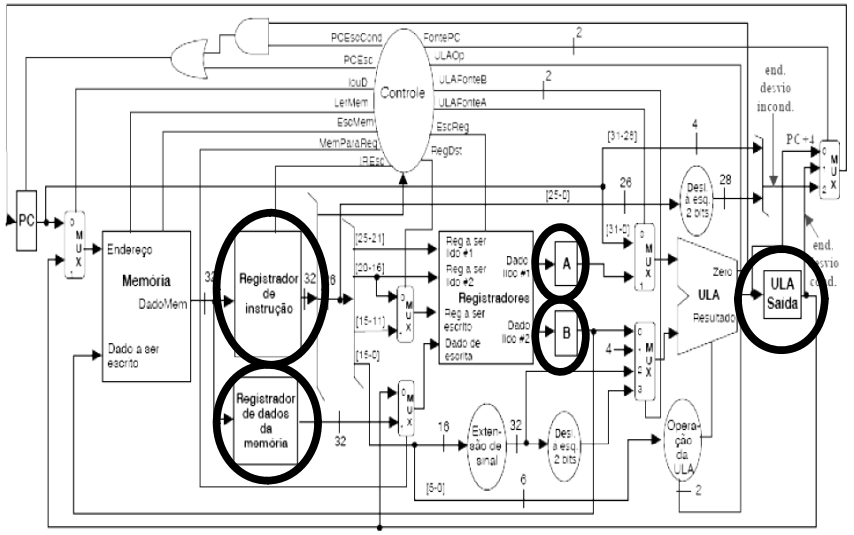



## PO Multiciclo x PO Uniciclo

- Apenas uma memória, com instruções e dados
- Incremento do PC, cálculo do endereço de desvio, operações lógico-aritméticas realizadas todas pela mesma unidade funcional
- Introdução e ampliação dos multiplexadores nas entradas da ULA
- Introdução do registrador de Instruções (RI), para armazenar a instrução lida




65




66

# Controle Multiciclo

1. Busca da Instrução
   1. RI = Memoria[PC]
   2. PC = PC + 4
2. Decodificação e busca de operandos
   - 2.1 A = Reg[RI[25-21]]
   - 2.2 B = Reg[RI[20-16]]
   - 2.3 Rdesvio = PC + (ext-sinal(IR[15-0])) << 2
   - Rdesvio armazena o endereço de desvio pré-calculado de forma a liberar a ULA para outras operações
   - a utilização destes valores depende do tipo de instrução

67

# Controle Multiciclo ...

3. Execução
   - 3.1 sULA = A + ext-sinal(IR[15-0]) (acesso a memória)
   - 3.2 sULA = A op B (tipo R)
   - 3.3 if (A == B) PC = Rdesvio (desvio condicional)
4. Acesso à memória ou escrita de registador
   - 4.1 Acesso à memória:
     - saídaMem = Memória[saiULA] ou (load)
     - Memória[saiULA] = B (store)
   - 4.2 Reg[IR[15-11]] = saiULA; (R-type)

68

# Controle Multiciclo ...

5.Escrita da Memória

Reg[IR[20-16]] = saidaMemória;

Atualização do PC:

- saída da ULA: endereçamento sequencial
- Rdesvio: qdo um salto condicional é realizado
- Jump: concatenação do PC com bits do RI

→ multiplexador na entrada do PC para selecionar uma destas entradas

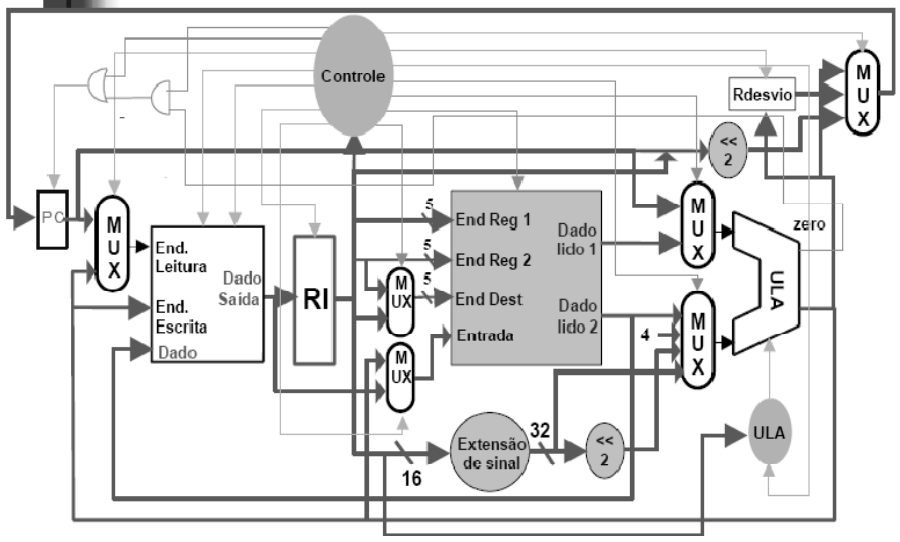


- Resumo da execução

| Nome do passo | Instrução tipo R | Instrução lw | Instrução sw | Instrução de desvio condicional | Instrução de desvio incondicional |
|---|---|---|---|---|---|
| Busca da instrução | RI = Mem[PC]<br>PC = PC + 4 | | | | |
| Decodificação da instrução & leitura dos registradores Rs e Rt & cálculo do endereço de desvio (cond.) | A = Reg [RI[25-21]]<br>B = Reg [RI[20-16]]<br>ULASaída = PC + (extensão de sinal(RI[15-0]) <<2) | | | | |
| Execução, cálculo do endereço de acesso à memória, término de uma instrução branch/jump | ULAOp = A op B | ULASaída = A + extensão de sinal (RI[15-0]) | | Se (A == B) então PC = ULASaída | PC = PC[31-28] \|\| (RI[25-0] <<2 ) |
| Término de uma instrução store word ou de tipo R | Reg [RI[15-11]] = ULASaída | RDM = Mem [ULASaída] | Mem [ULASaída] = B | | |
| Término de uma instrução load word | | Reg[RI[20-16]] = RDM | | | |
| **Número de passos** | **4** | **5** | **4** | **3** | **3** |

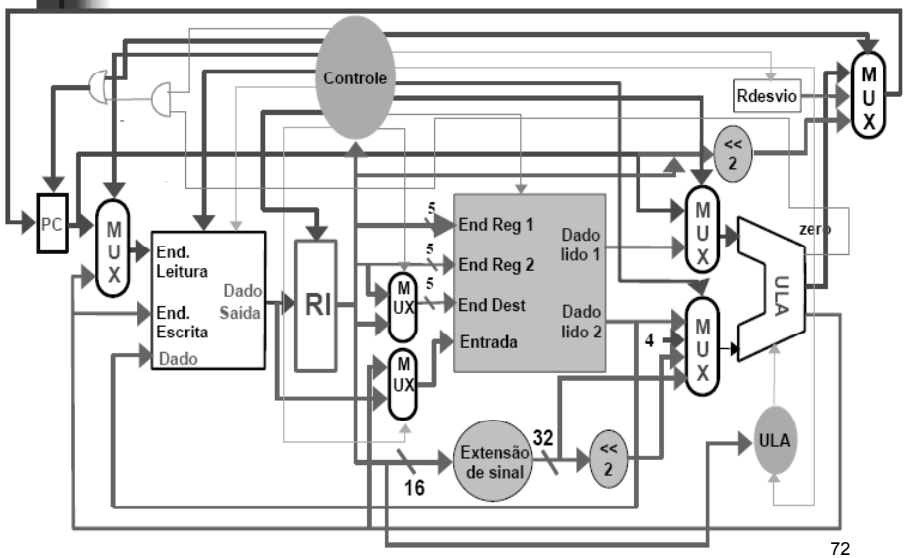


# MIPS Multiciclo

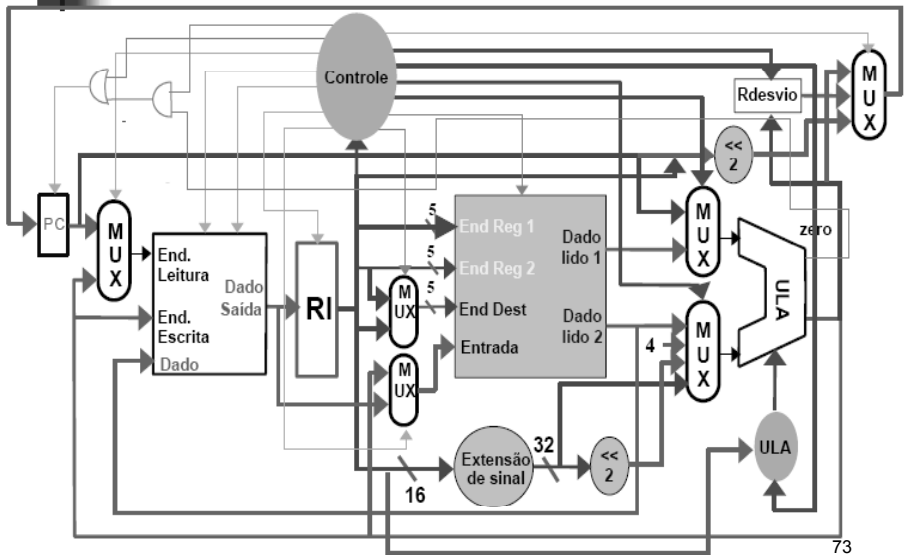





# Fase 1: Busca

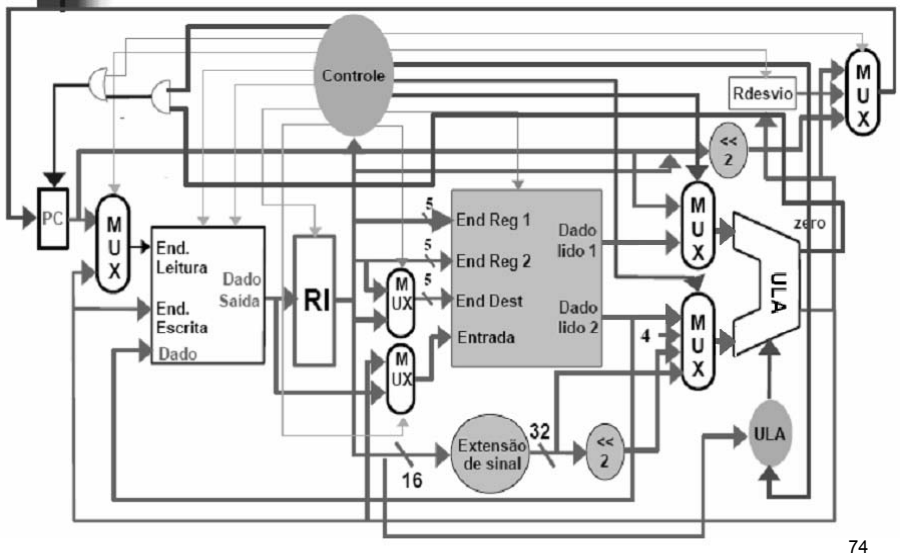


# Fase 2: decodificação



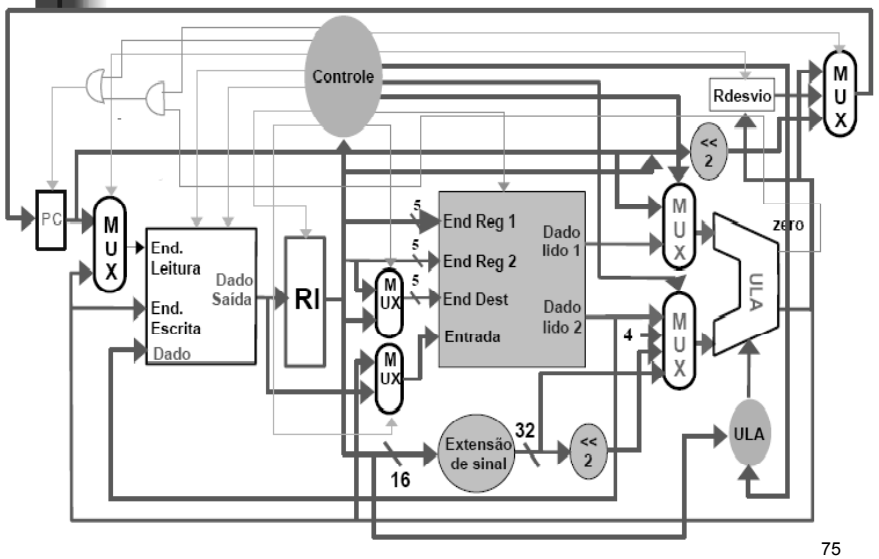



# Fase 3: Execução Desvio



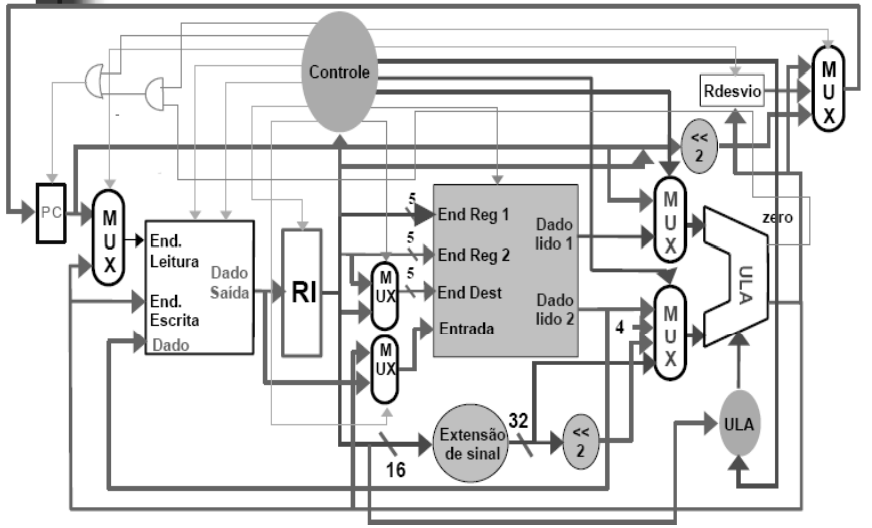



# Fase 3: Execução Log-Aritmética

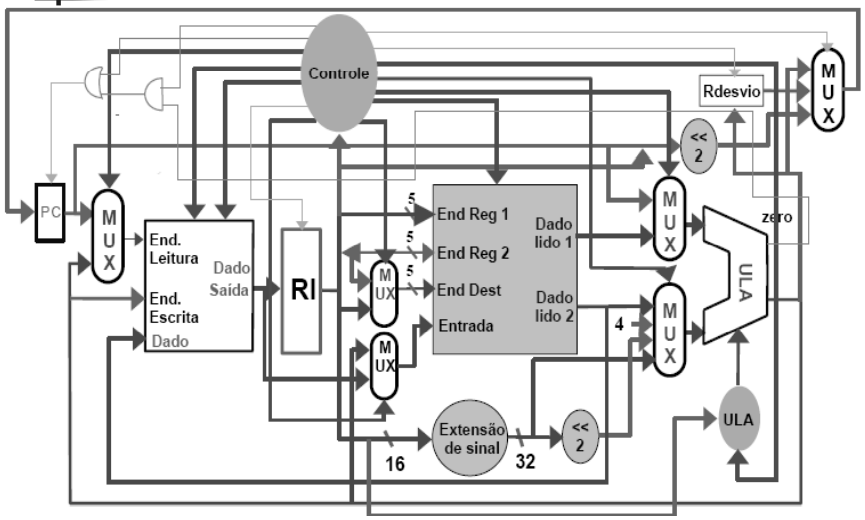





# Fase 3: Execução Acesso Memória

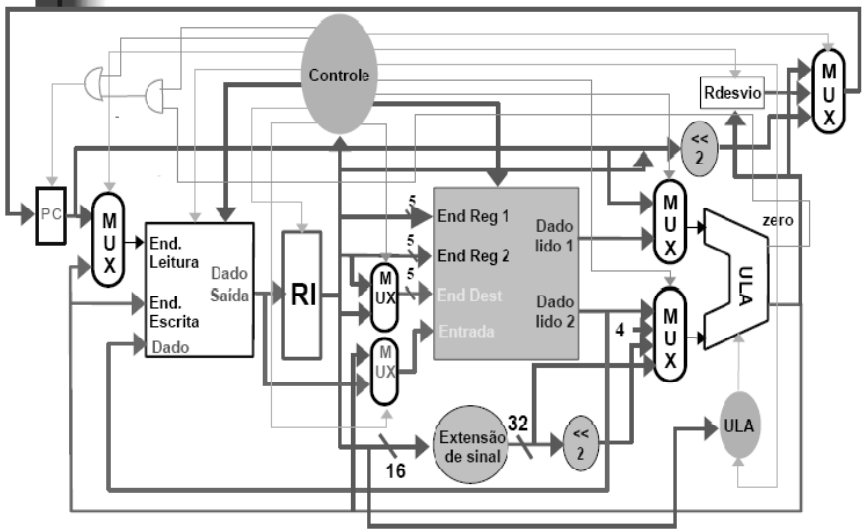


# Fase 4: Acesso Memória/Registrador





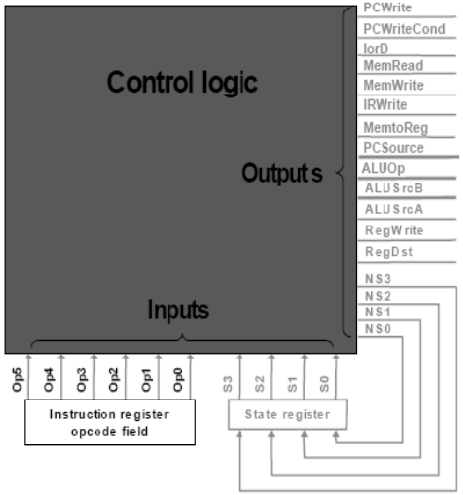

# Fase 5: Escrita Registrador (lw)






# Controle com MEF

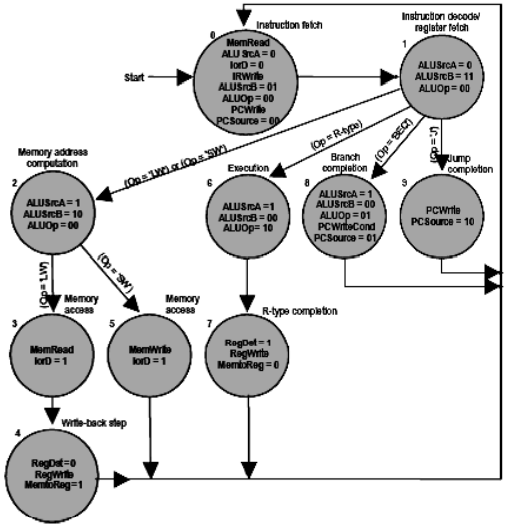
- estrutura típica de uma máquina de estados:
  - lógica de saída
  - lógica de transição
  - registrador de estado
  - entradas externas (código da instrução)



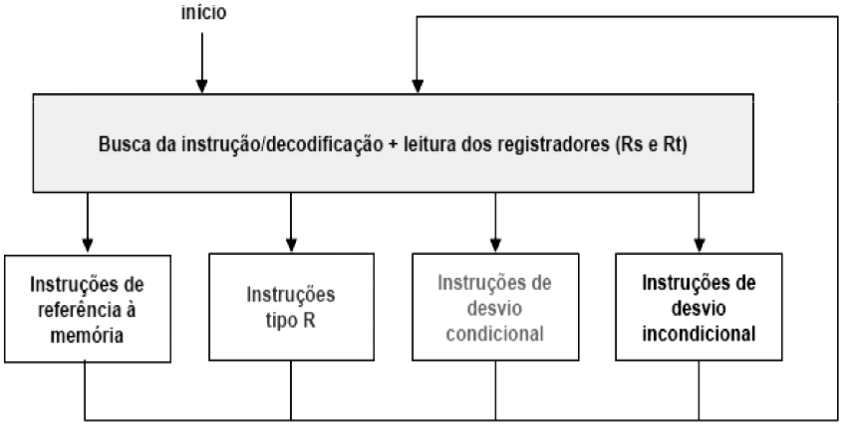



# MEF: Diagrama de Estados

- Cada nó do diagrama representa um estado
- A transição entre estados é indicada por arcos
- As condições de disparo de uma transição são associadas aos arcos
- Cada estado corresponde a um ciclo de relógio



- Controle com máquina de estados:

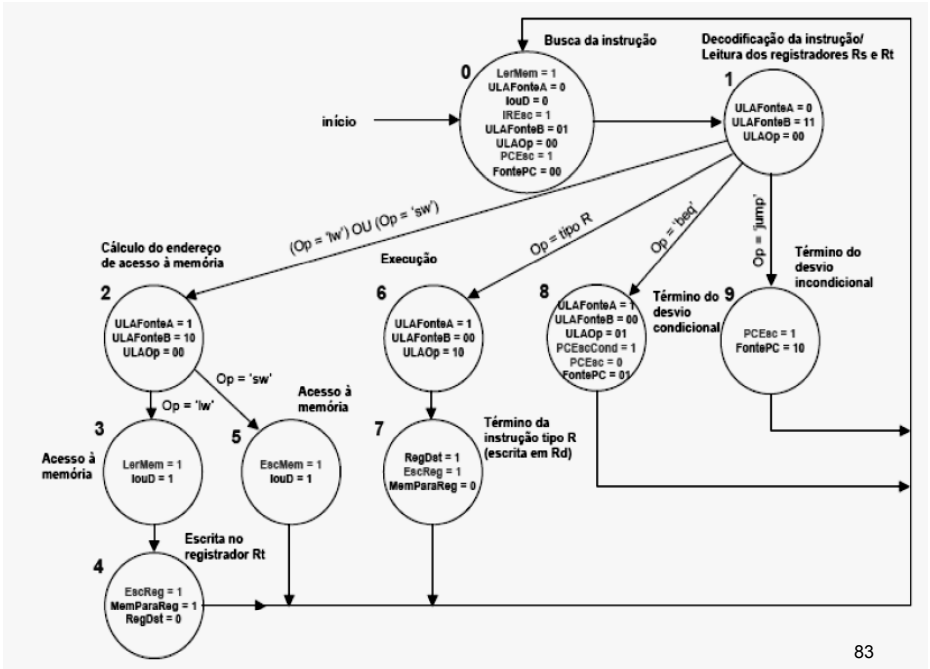





- Resumo da Execução

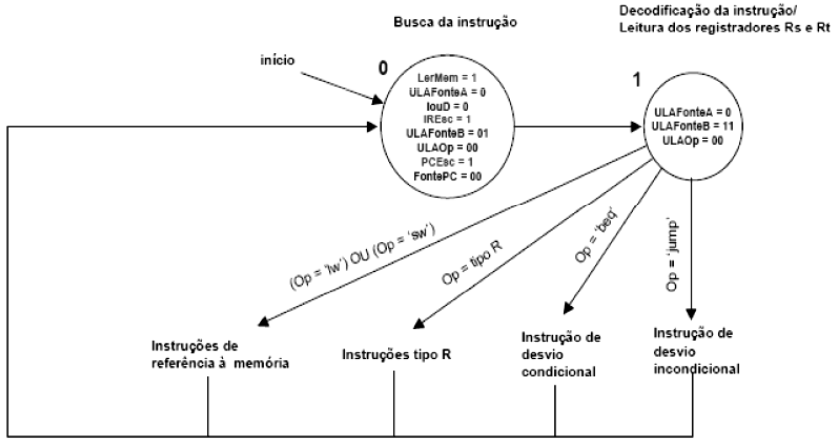
| Nome do passo | Instrução tipo R | Instrução lw | Instrução sw | Instrução beq | Instrução j |
|---|---|---|---|---|---|
| Busca da instrução | 0 RI = Mem[PC]<br>PC = PC + 4 | | | | |
| Decodificação da instrução & leitura dos registradores Rs e Rt & cálculo do endereço de desvio (cond.) | 1 A = Reg [RI[25-21]]<br>B = Reg [RI[20-16]]<br>ULASaída = PC + (extensão de sinal(RI[15-0]) <<2) | | | | |
| Execução, cálculo do endereço de acesso à memória, término de uma instrução branch/jump | ULAOp = A op 6 | ULASaída = A + extensão de (RI[15-0]) 2 | | Se (A == B) 8 PC = ULASaída | PC = PC[ 9 \|\| (RI[25-0] << 2 ) |
| Término de uma instrução store word ou de tipo R | Reg [RI[15-11] ULASaída 7 | RDM = Mem 3 [ULASaída] | Mem [ULASa 5 = B | | |
| Término de uma instrução load word | | Reg[RI[20- 4 = RDM | | | |
| Número de passos | 4 | 5 | 4 | 3 | 3 |



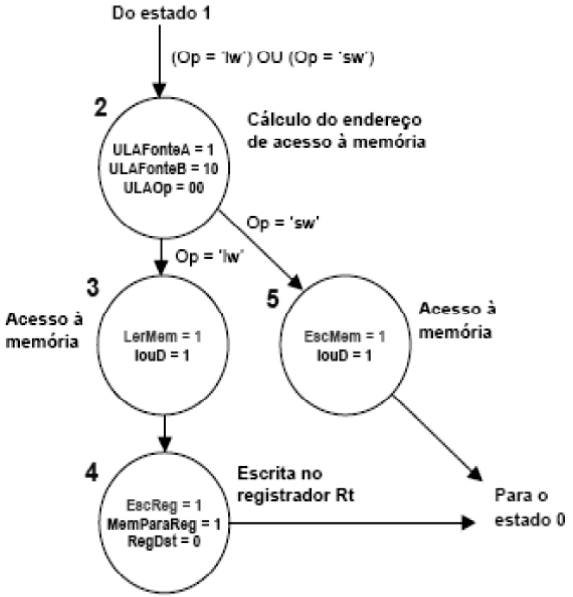





- Busca e decodificação

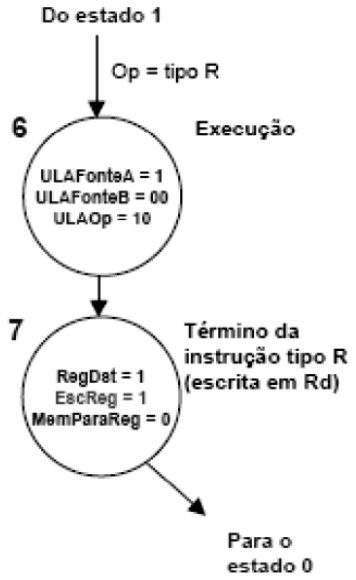


- Instruções *lw* e *sw*



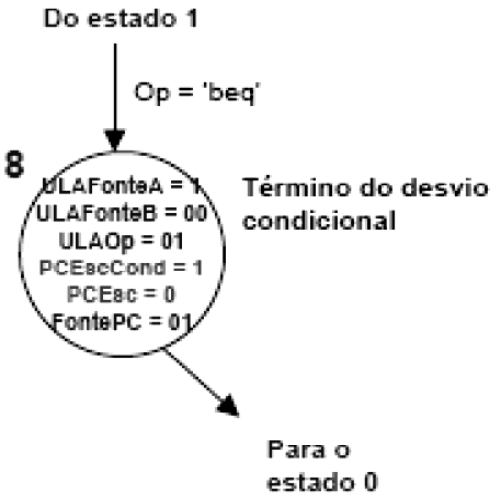



- Instruções tipo R



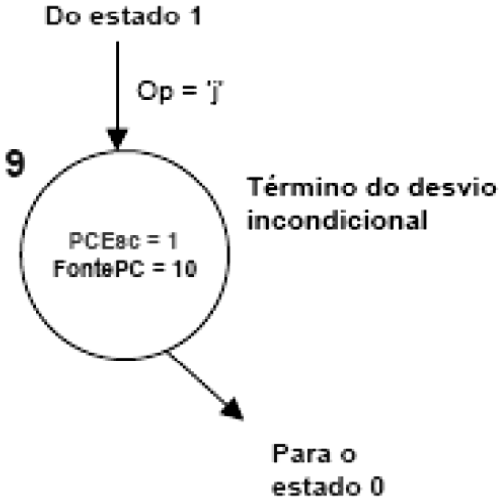



- Instrução *beq*






- Instrução *j*



- Resumo da Execução

| Nome do passo | Instrução tipo R | Instrução lw | Instrução sw | Instrução beq | Instrução j |
|---|---|---|---|---|---|
| Busca da instrução | 0 RI = Mem[PC]<br>PC = PC + 4 | | | | |
| Decodificação da instrução & leitura dos registradores Rs e Rt & cálculo do endereço de desvio (cond.) | 1 A = Reg [RI[25-21]]<br>B = Reg [RI[20-16]]<br>ULASaída = PC + (extensão de sinal(RI[15-0]) <<2) | | | | |
| Execução, cálculo do endereço de acesso à memória, término de uma instrução branch/jump | 6 ULAOp = A op | ULASaída = A + extensão de sinal (RI[15-0]) 2 | | 8 Se (A == B) PC = ULASaída | 9 PC = PC[ ] \|\| (RI[25-0] << 2 ) |
| Término de uma instrução store word ou de tipo R | 7 Reg [RI[15-11] ULASaída | 3 RDM = Mem [ULASaída] | 5 Mem [ULASaída] = B | | |
| Término de uma instrução load word | | 4 Reg[RI[20-16]] = RDM | | | |
| Número de passos | 4 | 5 | 4 | 3 | 3 |

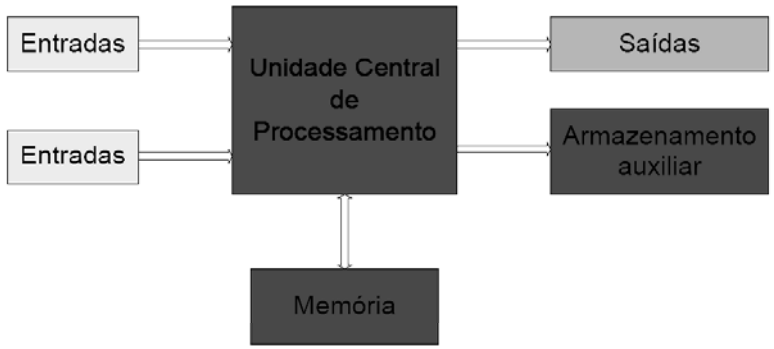


# Revisão rápida sobre Microcontroladores

(PIC visto no laboratório)

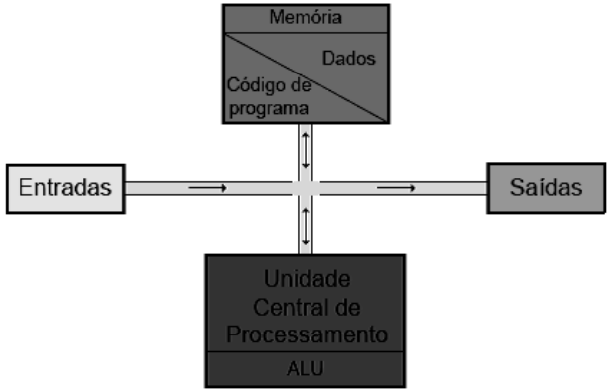


**Arquitetura básica de um Microprocessador ou Microcontrolador**

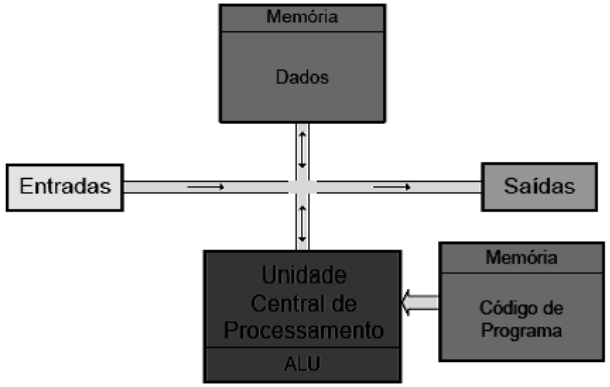




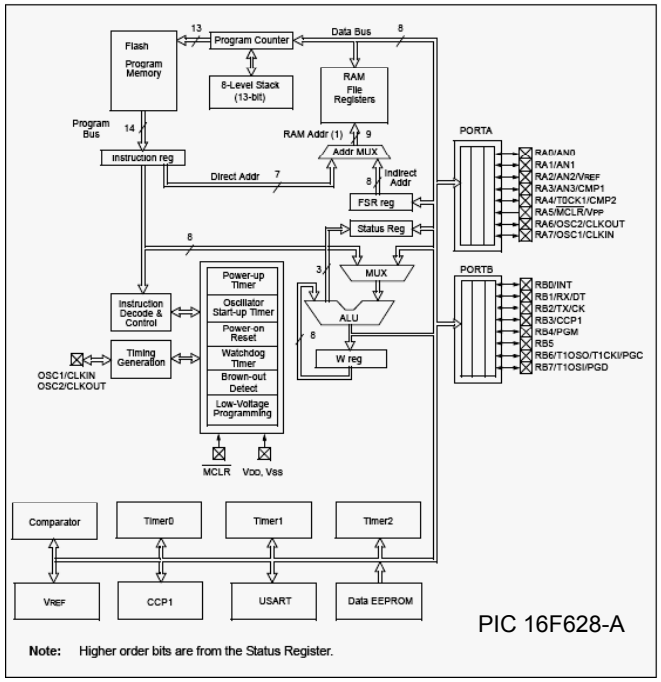

**Arquitetura Von Neumann**



**Arquitetura Harvard**

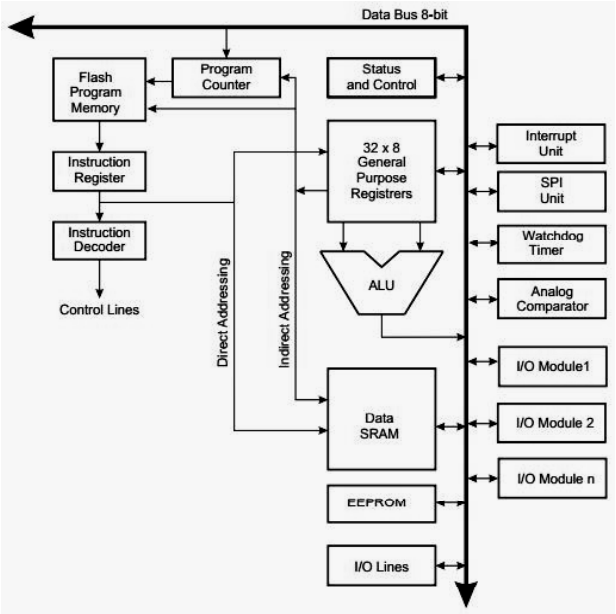





PIC 16F628-A






**AVR**

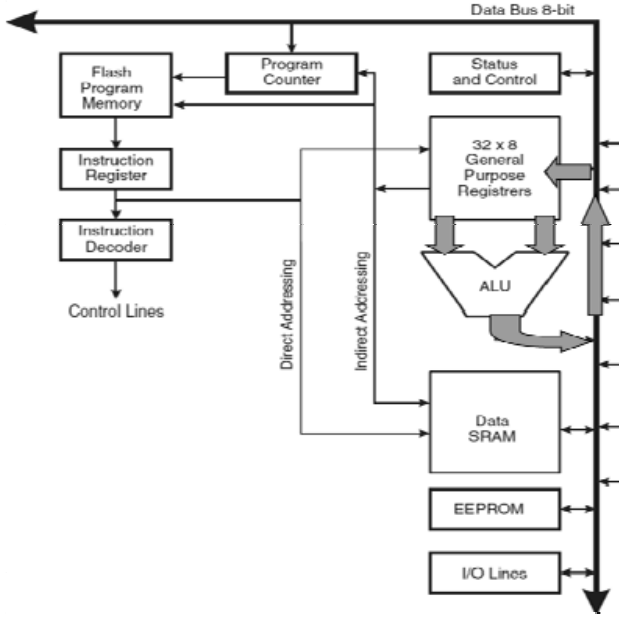
2 estudantes em:
Norwegian Institute of Technology (NTH)

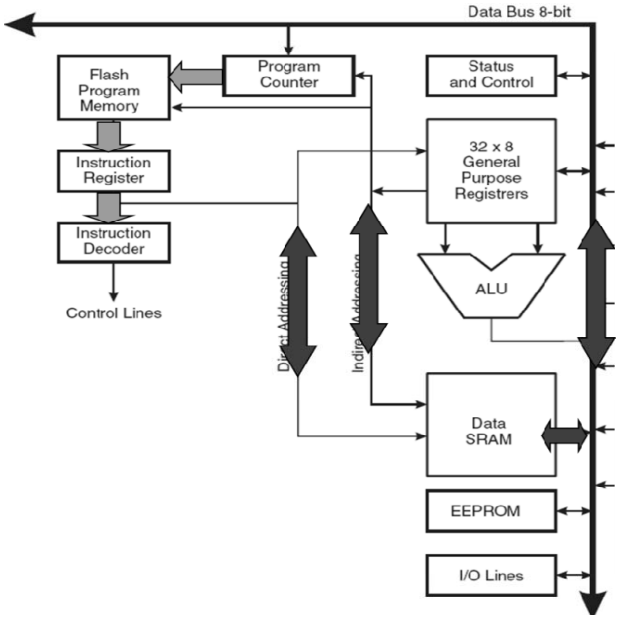
Alf-Egil Bogen e Vegard Wollan.

O acrônimo de AVR "Advanced Virtual RISC"

mas também pode ser:
Alf and Vegard's RISC.










A Microchip divide os PICs em famílias, às quais chama *Cores*.
- Core de 12 bits
- Core de 14 bits
- Core de 16 bits
- Core de 16 bit avançado

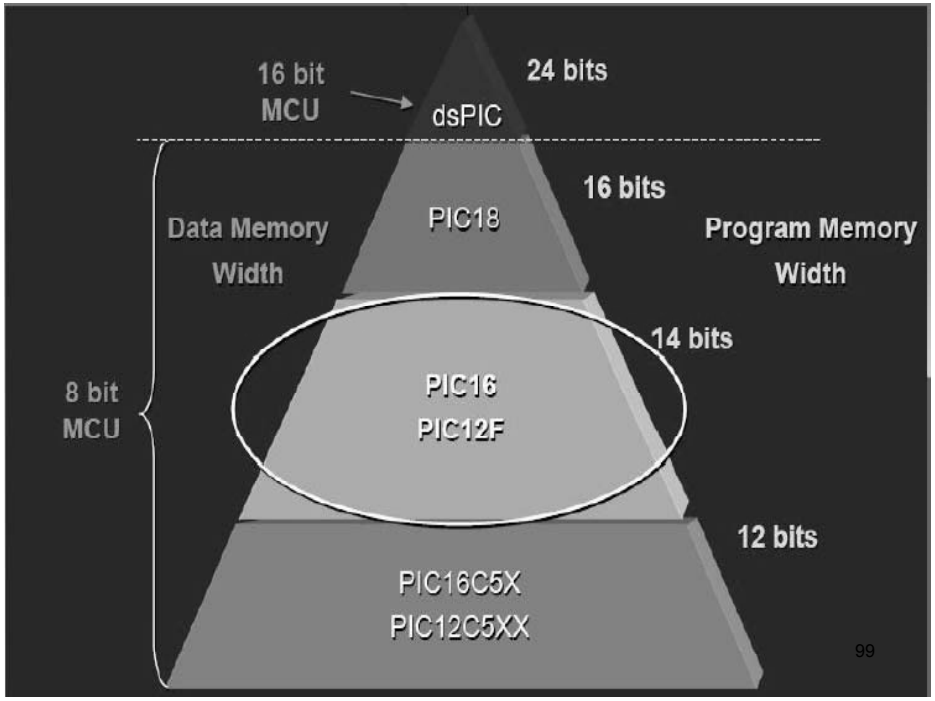
O número de bits não se refere ao barramento.

O barramento é sempre de 8 bits, que é o número de bits que uma posição de memória pode armazenar.

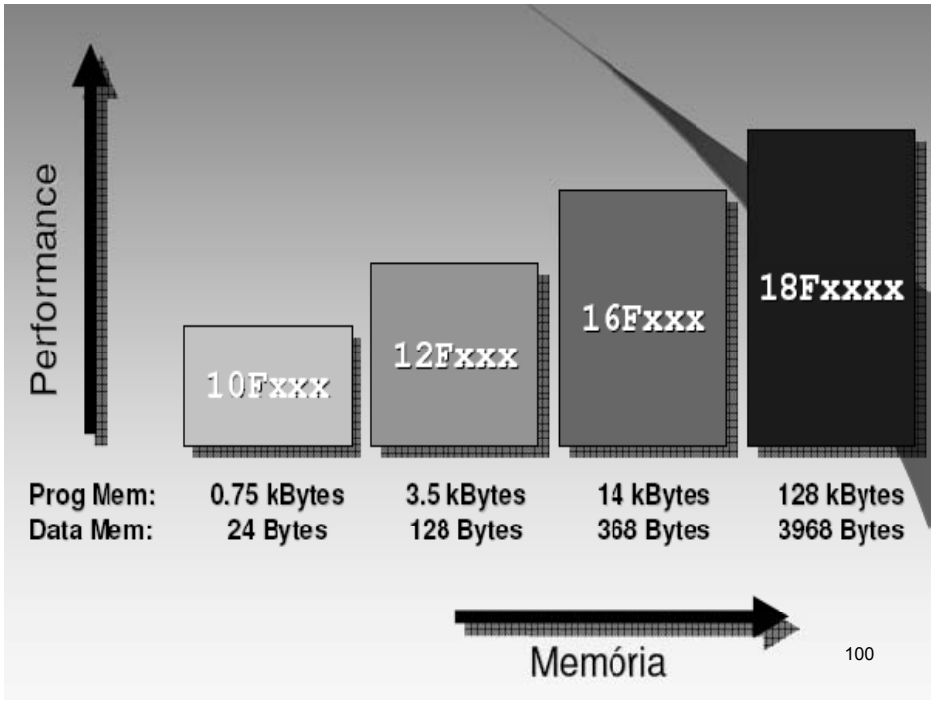
Refere-se, sim, ao tamanho da memória de programa, onde cada instrução pode ter 14 ou 16 bits de tamanho.

- Core de 12 bits - PIC12x
- Core de 14 bits - PIC16x
- Core de 16 bits - PIC17x
- Core de 16 bit avançado - PIC18x

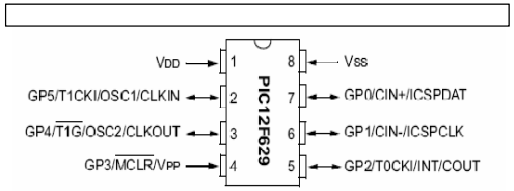






## Microcontrolador da família 12Fxxx com *core* de 12 bits



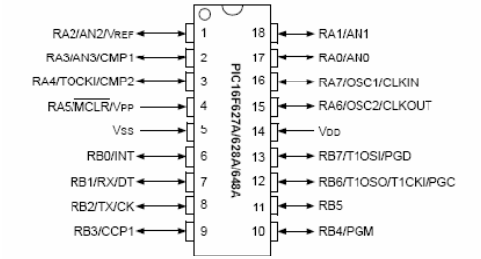

Podem endereçar 1Kbyte de memória de programa, e disponibilizam um máximo de 128 bytes de RAM. Este tipo de arquitetura suporta 35 instruções de programa, além de, também, estarem disponíveis em encapsulamentos de 8 pinos.

A maioria das famílias de 12 bits foi apresentada com memória OTP e EPROM apagável por UV.

OTP - One Time Programable
EPROM - Erasable Programable Read Only Memory
UV - Ultra Violeta
Flash Erasable Programmable Read-Only Memory - EPROM Flash



## Microcontrolador da família 16F6xxA com *core* de 14 bits

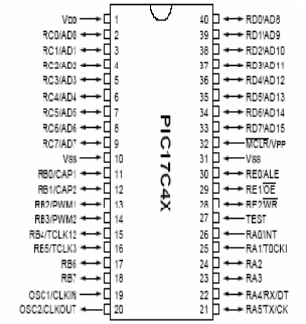



Podem endereçar 8Kbyte de memória de programa, e disponibilizam até 368 bytes de RAM. Suportam as mesmas instruções que a família de 12 bits, mas algumas instruções podem endereçar mais memória.

Estão disponíveis em encapsulamentos de 14, 18, 28 e 44 pinos, bem como em diversas versões de montagem em superfície.



## Microcontrolador da família 17C4xA com *core* de 16 bits



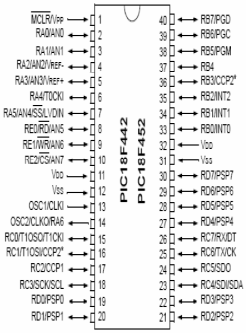

A quase inexistência de suporte, os preços elevados, e o aparecimento de novas famílias mais poderosas, fez com que esta família dos PIC17 se tornasse uma família pouco popular e cada vez mais em desuso.

Podem endereçar 32Kbyte de memória de programa, e disponibilizam até 902 bytes de RAM. A lista de instruções sofre alguns aumentos, e é partilhada com as famílias anteriores.

Só existem encapsulamentos disponíveis acima de 40 pinos, e apenas 8 chips diferentes nesta família.



## Microcontrolador da família 18Fxx2 com *core* de 16 bits melhorado




A pequena quantidade de memória de cada PIC, e a curta *Stack*, fez com que o uso de compiladores *C* se tornasse difícil.
As famílias de 12 e 14 bits têm uma *Stack* com 8 níveis, enquanto que as de 16 bits já possuem *Stack* de 16 níveis.
A falta de instruções para manipular a Pilha de memória *Stack*, leva o utilizador a implementar a *Stack* em software, aliada à falta de RAM, que tornava o processo muito difícil, e a performance diminuída.

A família de 16 bits avançada, implementa uma arquitetura pouco diferente, focada no uso de compiladores, podendo suportar 64Kbytes de memória de programa e 3Kbytes de RAM. O número de instruções já ascende a um total de 75, e estão disponíveis em diversos encapsulamentos acima dos 18 pinos.

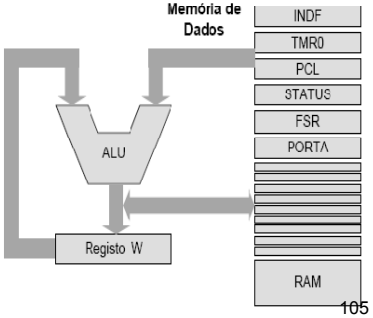
A disponibilidade de compiladores, a melhor lista de instruções e o seu baixo custo, fez com que esta família se popularizasse depressa.

A unidade central de processamento dispõe de duas memórias distintas, uma contendo o código de programa, carregado previamente no microcontrolador, e uma segunda memória de uso específico onde estão contidos os registos internos, necessários ao funcionamento do microcontrolador, e também uma área reservada ao uso genérico do programa que está a correr (RAM).

A família PIC16 tem 8bits de tamanho para toda a informação que circula internamente no barramento de dados. O barramento de endereçamento da memória de programa tem um tamanho de 14bits

A família PIC16 tem a memória de programa organizada em páginas e a memória de dados organizada em bancos.

Cada página de memória tem 2k *words* de tamanho. O tamanho de cada banco de memória RAM varia entre dispositivos. Na família PIC16 só é possível endereçar um máximo de 128 bytes, pelo que o uso de bancos, torna possível a utilização de uma quantidade de RAM maior.






Cada instrução de programa tem um determinado código (OPCODE). O microcontrolador utiliza uma WORD para codificar as instruções.

Dos 14 bits, os 8 bits de menor peso servem para guardar informação de utilização direta (*Literal Instructions*). Os restantes bits guardam o código da instrução a executar.

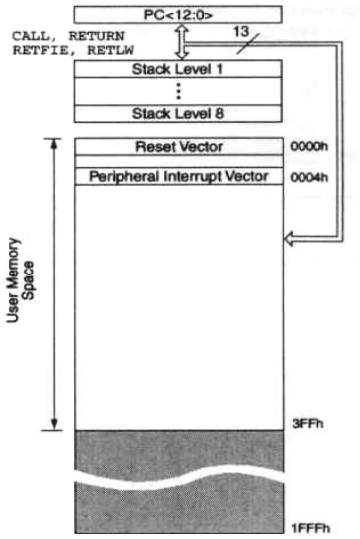



**Alguns periféricos que podem ser encontrados dentro de um PIC16**

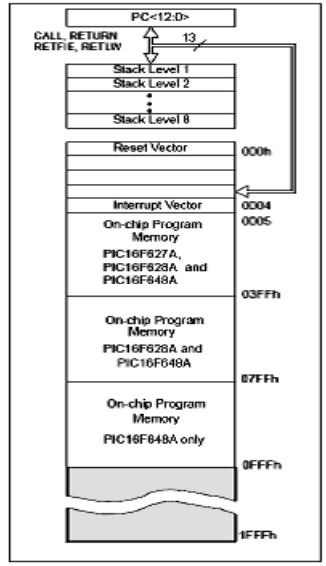
- Temporizadores: 1, 2 ou mais, dependendo do modelo
- RS232 (USART): Porta Série que permite os modos síncronos e assíncronos
- PSP: Porta Paralela de 8bits
- EEPROM: Memória não volátil adicional para registro de dados
- Comparadores para sinais externos
- PWM: Modulação por largura de impulsos
- SPI (*Serial Peripheral Interface*): Porta de dados série
- I2C: Porta de dados série
- ADC: Conversores analógico/digital
- ICSP (*In Circuit Serial Programming*): Programação série direta no circuito
- *Watchdog Timer*: Temporizador de guarda para o microcontrolador
- USB (Universal Serial Bus)
- CAN (Controller Area Network)
- LIN (*Local Interconnect Network*)
- LCD (*Liquid Cristal Display*): Controle de display de cristais líquidos
- rfPIC: Transmissor de ASK e FSK
- Controlador de motores DC
- BOR (*Brown-Out* RESET): Detector de limites inferiores de tensão de alimentação



**MEMORIA DE PROGRAMA**




**16F84A**




**16F(627, 628A, 648A)**

## A PILHA (STACK)

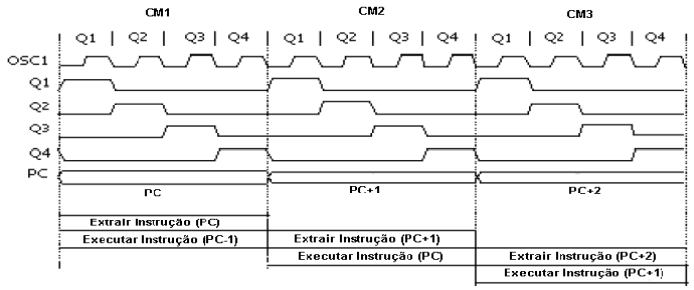
A pilha é um local da RAM onde é guardado o endereço da memória de programa antes de ser executado um pulo ou uma chamada de função localizada em outra posição de memória.

## CICLO DE MÁQUINA

O oscilador externo (geralmente um cristal) ou o interno (circuito RC) é usado para fornecer um sinal de clock ao microcontrolador. O clock é necessário para que o microcontrolador possa executar as instruções de um programa.

Nos microcontroladores PIC, um ciclo de máquina (CM) possui quatro fases de clock que são Q1, Q2, Q3 e Q4. Dessa forma, para um clock externo de 4MHz, temos um ciclo de máquina (CM=4 x 1/F) igual a 1µs (ou 1MHz)

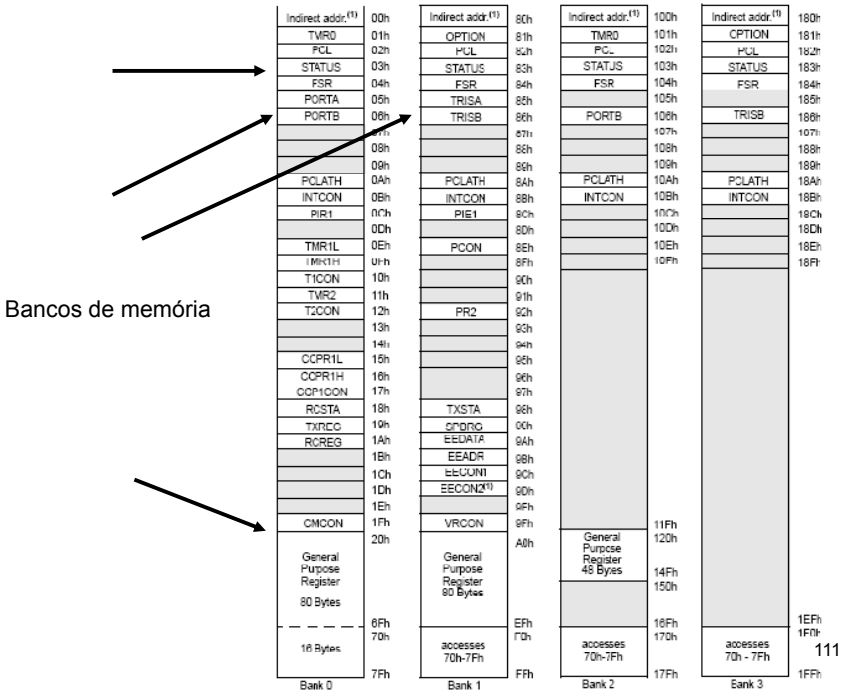





•O Contador de Programa (PC) é incrementado na fase Q1 do ciclo de máquina e a instrução seguinte é resgatada da memória de programa e armazenada no registro de instruções da CPU no ciclo Q4.
Ela é decodificada e executada no próximo ciclo, no intervalo de Q1 e Q4.
•O PIPELINE (sobreposição) permite que quase todas as instruções sejam executadas em apenas um ciclo de máquina, gastando assim 1 µs (para um clock de 4 MHz).
•As únicas exeções referem-se às instruções que geram “saltos” no contador de programa, como chamadas de funções em outro local da memória de programa e os retornos dessas funções.



Bancos de memória






## Registrador STATUS

**O registro STATUS, configura os bancos de registros, flags da ULA e outros.**

**Seu endereço físico é 03h (banco 0) e 83 (banco1).**

| Nº dos bits | Bit 7 | bit 6 | Bit 5 | bit 4 | bit 3 | bit 2 | bit 1 | bit 0 |
|---|---|---|---|---|---|---|---|---|
| Bits | IRP | RP1 | RP0 | /TO | /PD | Z | DC | C |

Bit 6-5: RP1:RP0, Bit de seleção de banco de registradores (usado para endereçamento direto). P/ bits 6-5=00 implica seleção do banco 0 (00 - 7Fh), para bits 6-5=01 implica seleção do banco 1 (80h - FFh), para bits 6-5=10 implica seleção do banco 2 (100 - 17Fh) e, para bits 6-5=11 implica seleção do banco 3 (180h - 1FFh). obs.: Cada banco é de 128 bytes e somente o bit RP0 é usado no PIC16F628 (considerar RP1=0).

Bit 2: Z, bit de Zero. Vai a 1 quando o resultado de uma operação aritmética ou lógica é zero. Vai a 0 quando o resultado de uma operação aritmética ou lógica é diferente de zero.

BSF STATUS, RP0 BCF STATUS, RP0

**TABLE 4-3: SPECIAL REGISTERS SUMMARY BANK0**

| Address | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Value on POR Reset(1) | Details on Page |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Bank 0 | | | | | | | | | | | |
| 00h | INDF | Addressing this location uses contents of FSR to address data memory (not a physical register) | | | | | | | | xxxx xxxx | 28 |
| 01h | TMR0 | Timer0 module's Register | | | | | | | | xxxx xxxx | 45 |
| 02h | PCL | Program Counter's (PC) Least Significant Byte | | | | | | | | 0000 0000 | 28 |
| 03h | STATUS | IRP | RP1 | RP0 | $\overline{TO}$ | $\overline{PD}$ | Z | DC | C | 0001 1xxx | 22 |
| 04h | FSR | Indirect data memory address pointer | | | | | | | | xxxx xxxx | 28 |
| 05h | PORTA | RA7 | RA6 | RA5 | RA4 | RA3 | RA2 | RA1 | RA0 | xxxx 0000 | 31 |
| 06h | PORTB | RB7 | RB6 | RB5 | RB4 | RB3 | RB2 | RB1 | RB0 | xxxx xxxx | 36 |



**TABLE 4-4: SPECIAL FUNCTION REGISTERS SUMMARY BANK1**

| Address | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Value on POR Reset(1) | Details on Page |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Bank 1 | | | | | | | | | | | |
| 80h | INDF | Addressing this location uses contents of FSR to address data memory (not a physical register) | | | | | | | | xxxx xxxx | 28 |
| 81h | OPTION | $\overline{RBPU}$ | INTEDG | T0CS | T0SE | PSA | PS2 | PS1 | PS0 | 1111 1111 | 23 |
| 82h | PCL | Program Counter's (PC) Least Significant Byte | | | | | | | | 0000 0000 | 28 |
| 83h | STATUS | IRP | RP1 | RP0 | $\overline{TO}$ | $\overline{PD}$ | Z | DC | C | 0001 1xxx | 22 |
| 84h | FSR | Indirect data memory address pointer | | | | | | | | xxxx xxxx | 28 |
| 85h | TRISA | TRISA7 | TRISA6 | TRISA5 | TRISA4 | TRISA3 | TRISA2 | TRISA1 | TRISA0 | 1111 1111 | 31 |
| 86h | TRISB | TRISB7 | TRISB6 | TRISB5 | TRISB4 | TRISB3 | TRISB2 | TRISB1 | TRISB0 | 1111 1111 | 36 |



Os registradores TRISA e TRISB servem para configurar como entrada ou saída os pinos das portas PORTA e PORTB, respectivamente. Para configurar como entrada basta escrever 1 no bit correspondente e como saída basta escrever 0 no bit correspondente. Os bits podem ser selecionados como entrada e saída da forma que se desejar. Por exemlo, para o PORTB, se a sequência 10010111 for utilizada para configurar o registrador TRISB, isto significa que as portas RB0, RB1, RB2, RB4 e RB7 foram configuradas como entrada (1) e que RB3, RB5 e RB6 foram configuradas como saída (0). Lembrado que RB0 representa o bit menos significativo (LSB - à direita) e RB7 o bit mais significativo (MSB - à esquerda).

Os registradores PORTA e PORTB são utilizados para modificar diretamente o estado (0 ou 1) das portas. Quando o bit é configurado como entrada, o estado da porta é armazenado diretamente no bit correspondente a porta nestes registradores. Quando o bit é configurado como saída o seu estado pode ser modificado escrevendo-se diretamente no bit correspondente destes registradores. Por exemplo se o bit 3 do PORTA estiver configurado como entrada (bit 3 do TRISA em 1) basta ler o estado do bit 3 do registrador PORTA para conhecer o estado da entrada.



Exemplos de uso:

```
BCF status, rp0

CLRF    portb

BSF status, rp0

CLRF trisb

BCF status, rp0

MOVLW b'11111111'

MOVWF portb
```

```
BANK1
movlw  b'00010111'
movwf  TRISA
movlw  0x00
movwf  TRISB
BANK0

clrf   PORTB
```

```
#INCLUDE <P16F628A.INC>                         ;ARQUIVO PADRÃO MICROCHIP PARA 16F628A
__CONFIG  _BODEN_ON & _CP_OFF & _PWRTE_ON & _WDT_OFF & _LVP_OFF & _MCLRE_ON & _XT_OSC

#DEFINE     BANK0       BCF STATUS,RP0          ;SETA BANK 0 DE MEMÓRIA
#DEFINE     BANK1       BSF STATUS,RP0          ;SETA BANK 1 DE MAMÓRIA
            CBLOCK      0x20            ;ENDEREÇO INICIAL DA MEMÓRIA DE USUÁRIO
            ENDC                                ;FIM DO BLOCO DE MEMÓRIA

#DEFINE     BOTAO       PORTA,1     ;PORTA DO BOTÃO
                                                        ; 0 -> PRESSIONADO
                                                        ; 1 -> LIBERADO

#DEFINE     LED         PORTB,0     ;PORTA DO LED
                                                ; 0 -> APAGADO
                                                ; 1 -> ACESO
            ORG         0x00                    ;ENDEREÇO INICIAL DE PROCESSAMENTO
            GOTO        INICIO
            ORG         0x04                    ;ENDEREÇO INICIAL DA INTERRUPÇÃO
            RETFIE                              ;RETORNA DA INTERRUPÇÃO

INICIO
            CLRF        PORTA                   ;LIMPA O PORTA
            CLRF        PORTB                   ;LIMPA O PORTB
            BANK1                               ;ALTERA PARA O BANCO 1
            MOVLW       B'00000010'
            MOVWF       TRISA                   ;DEFINE RA1 COMO ENTRADA E DEMAIS COMO SAÍDAS
            MOVLW       B'00000000'
            MOVWF       TRISB                   ;DEFINE TODO O PORTB COMO SAÍDA
            MOVLW       B'00000000'
            MOVWF       INTCON                  ;TODAS AS INTERRUPÇÕES DESLIGADAS
            BANK0                               ;RETORNA PARA O BANCO 0
            MOVLW       B'00000111'
            MOVWF       CMCON                   ;DEFINE O MODO DO COMPARADOR ANALÓGICO
```



```
MAIN

            BTFSC       BOTAO           ;O BOTÃO ESTÁ PRESSIONADO?
            GOTO        BOTAO_LIB       ;NÃO, ENTÃO TRATA BOTÃO LIBERADO
            GOTO        BOTAO_PRES      ;SIM, ENTÃO TRATA BOTÃO PRESSIONADO

BOTAO_LIB
            BCF         LED             ;APAGA O LED
            GOTO        MAIN            ;RETORNA AO LOOP PRINCIPAL

BOTAO_PRES
            BSF         LED             ;ACENDE O LED
            GOTO        MAIN            ;RETORNA AO LOOP PRINCIPAL


            END                         ;OBRIGATÓRIO
```