Revisão:

PARTE 1: MÓDULO 13,14 E 15

1.Uniciclo/monociclo:

- Utilizava apenas 1 ciclo de clock para todas as instruções.
- Toda as instruções levam o mesmo tempo para serem executada, ou seja, o tempo da instrução mais demorada que corresponde ao tempo de ciclo do clock
- Início: transição positiva do sinal de clock
- Fim: transição positiva do sinal de clock

2. Multiciclo:

- Cada instrução demora um período de clock (fetch, decode, execute.), ou seja, demora o seu tempo para executar e não o da instrução mais demorada como no monociclo
- As Unidades funcionais podem ser utilizadas mais de uma vez por instrução, uma vez que trabalham em ciclos de clock diferentes

Estágios do Multiciclo:

1. **BUSCA:**
   Vai na pc, busca a instrução e devolve para o registrador de instrução, após isso, incrementa o pc em 4 e devolve novamente para o pc
2. **DECODIFICAÇÃO E BUSCA DOS REGISTRADORES:**
   Lê os registrados rs e rt e se for um desvio calcula o endereço se for branch
3. **EXECUÇÃO, CÁLCULO DE ENDEREÇO OU CONCLUSÃO DO DESVIO**
   a. Execução: instruções do tipo R
   b. Calculo do endereço
   c. Conclusão do desvio incialmente calculado na etapa anterior
4. **INSTRUÇÕES TIPO R OU ACESSO À MEMÓRIA**
   a. Tipo R: conclusão da execução da etapa anterior
   b. Acesso à memória: load word ou store word
5. **CONCLUSÃO DA LEITURA DA MEMÓRIA**
   Load word

PARTE 2: MÓDULO 16 E 17

1.PIPELINE

Por que existem

Processadores mais velozes:

1.Tamanho dos barramentos aumentaram

2.Aumentou a frequência de clock

Pipeling: é um técnica utilizada para realizar a sobreposição de instruções, diminuindo, assim, a quantidade de clocks utilizada, pois as intruções são executadas em paralelo.

Estágios:

2 estágios: busca + execução

4 estágios: busca+ decodificação + busca dos operandos + execução

5 estágios: busca + decodificação + cálculo do endereço dos operandos + busca dos operandos + execução

Dois empecilhos no pipeline:

1.hazards de controle: esse acontece quando está sendo feita uma decodificação de uma estrutura condicional ou incondicional (Branch e jump) e então deve-se bloquear as instruções posteriores a fim de que o endereço de memória seja sabido por essas instruções, isso cria stall(bolhas) de pipeline

2.hazards de dados: acontece quando uma estrutura depende explicitamente do resultado da anterior, por exemplo:

Add **$s0**, $t1, $t2

Sub $s1, **$s0**, $t3

Solução:

Realizar um bypassing ou forwarding:

Antecipar o dado para que a outra instrução seja executada

IF→ID→EX→MEM→WB

IF→ID→EX→MEM→

PARTE 3: MÓDULO 18 E 19

1.Hierarquia de memória: DADOS DE 2012

SRAM – mais rápida e menor capacidade

velocidade: 0.75 a 2.5ns

valor: $500 a $1000 por Gb

DRAM – menos rápida e mais capacidade que a SRAM

Velocidade: 50 a 70ns

Valor: $10 a $20 por GB

FLASH – lenta e maior capacidade que SRAM E DRAM

Velocidade: 5.000 a 50.000ns

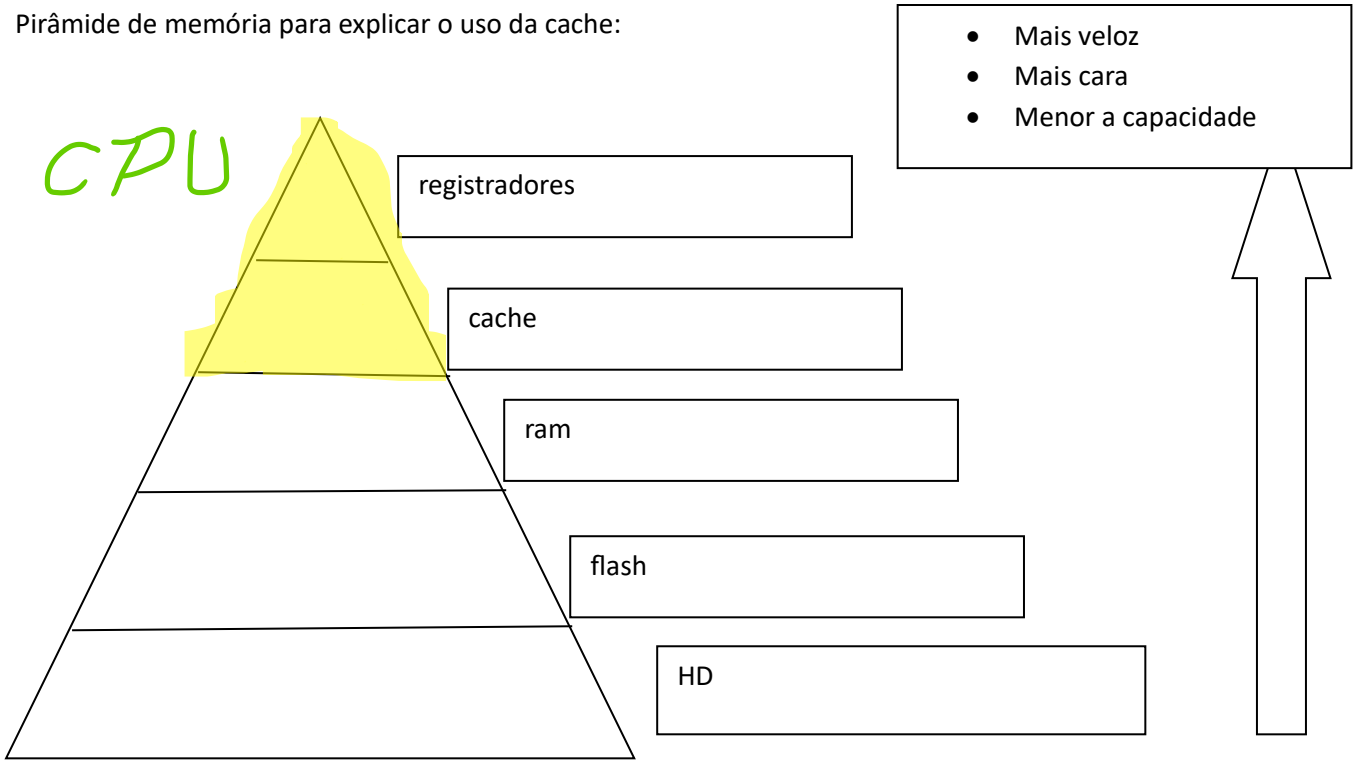
Valor: $0.75 a $1.00 por GB

HD – Mais lenta e maior capacidade

Velocidade: 5M a 20Mns

Valor: $0.05 a $0.1 por GB

Pirâmide de memória para explicar o uso da cache:



Note que a Memória do processador (CPU) é muito mais rápida que a memória Ram que está fora da CPU, essa discrepância de velocidade é dita como GARGALO DE VON NEUMMAN:

Solução: Utilizar a memória cache que fica na CPU e importar para ele os dados mais importantes, acessando-os mais rapidamente.

Como eu defino os dados mais importantes para colocar na Cache:

Princípios de Localidade:

1.Localidade Espacial:

“Se eu busquei uma instrução, a probabilidade de buscar a seguinte é grande, logo eu pego o bloco de instruções”

2.Localidade Temporal:

“Se eu busquei por uma instrução, a probabilidade de buscar essa mesma instrução novamente é grande, logo eu pego o bloco de instruções”

Pronto, matamos 2 coelhos numa cajadada só!

Podemos usar com 1 cache ou 3 caches (L1, L2 e L3)

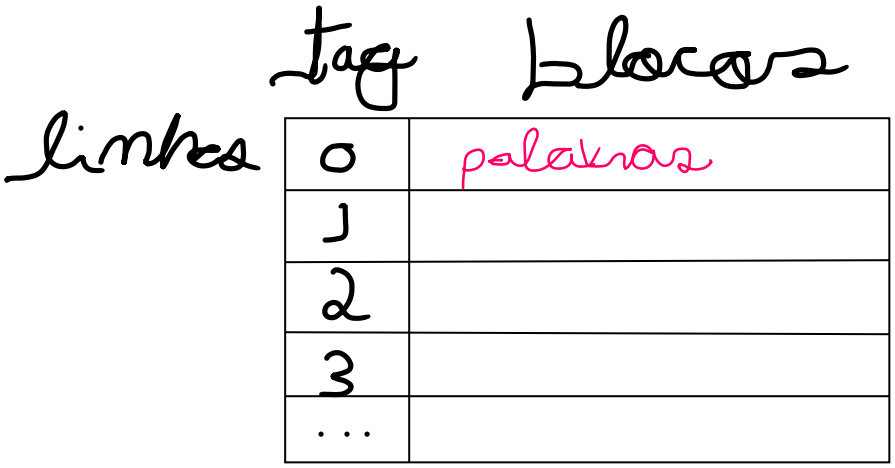
Mas como funciona a questão de eu encontrar ou não a palavra ou instrução na cache?

Acerto(Hit): é quando a palavra buscada é encontrada na cache

Falha(Miss): é quando a palavrada buscada não é encontrada na cache

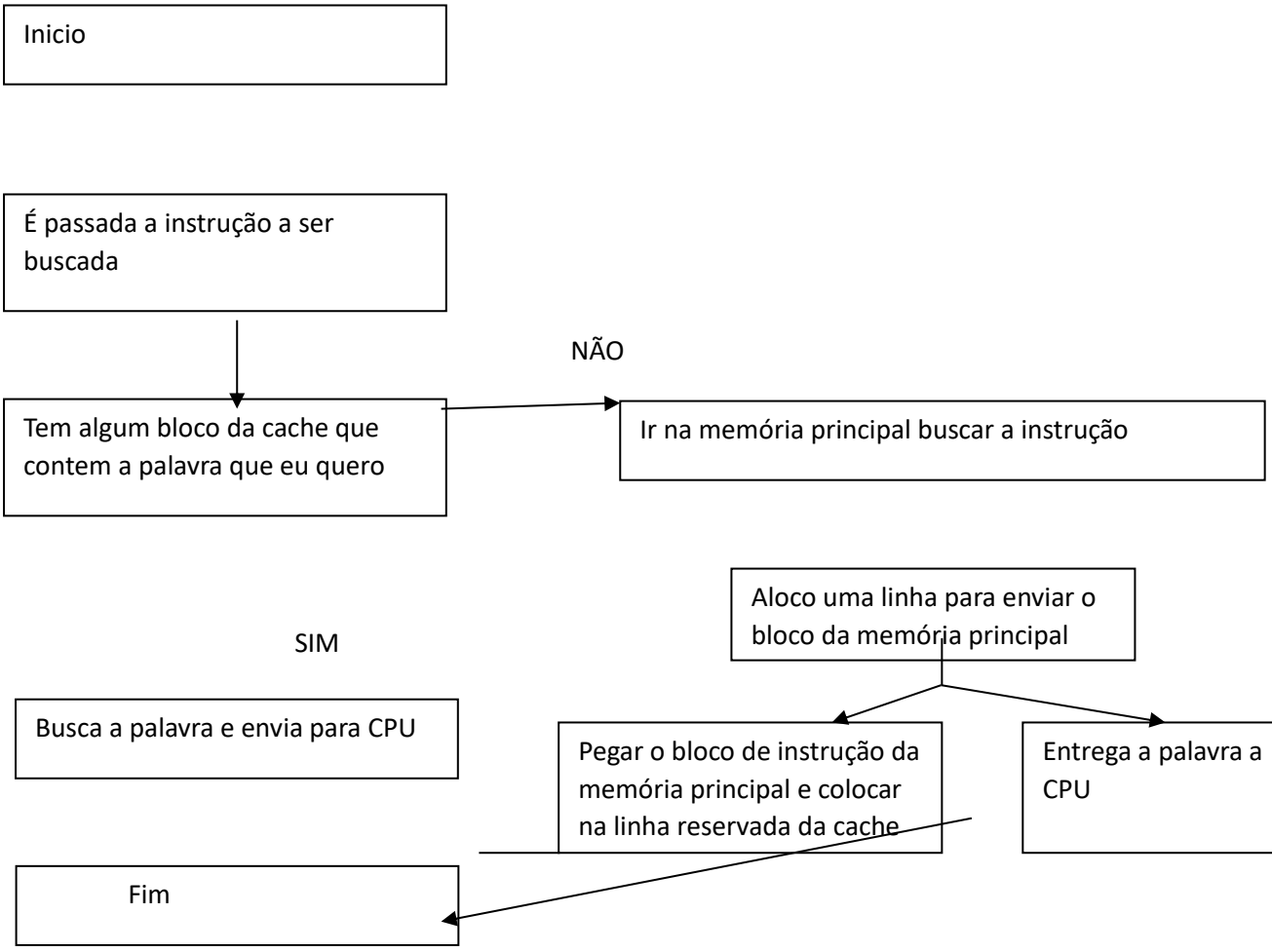
Taxa de acerto: (hits)/ hits + miss

Como é uma cache?



É buscada na memória cache uma palavra que está dentro do bloco referenciado por uma tag que é devolvida ao processador caso seja encontrada

Esquema de funcionamento da cache:



PARTE 3: MÓDULO 20

1.Barramentos:

Barramentos são meios de comunicação entre 2 ou + dispositivos por meio da transmissão compartilhadas

Barramentos de sistema:

1.Dados: bits de dados ou bits de instruções

2.Endereço: bits de endereço

3.Controle: bits de controle

Como funciona cada um?

1.Endereço: designa a fonte ou a origem dos dados enviados pelo barramento de dados

O tamanho máx. de endereço é dado por 2^l em que l é a quantidade de bits

2.Dados: envia os dados ou instruções a serem realizadas

Taxa de transmissão: largura do barramento x velocidade

OBS.: se o barramento tiver uma largura de 8 bits e os dados tiverem 16 bits ele terá que acessar 2 vezes o módulo da memória

3.Controle: controla as linhas de dado e de endereço para não haver sobreposição ou alteração

PARTE 4: ASSEMBLY - MIPS

#Fibonacci esquisito:

```
.data
        brk: .asciiz “\n”
.text
Main:
        #lendo a quantidade de termos
        li $v0, 5
        syscall
        move $t0, $v0
```

```
        #inicia o contador
        move $s0, $zero
        #variável que dará a resposta
        move $t1, $zero
Loop:
        #checa se o contador e a quantidade de elementos são iguais
        beq  $t0, $s0, End
        li $v0,5
        syscall
        add $t1,$t1,$v0
        addi $s0, $s0,1
        J Loop

End:
        li $v0, 1
        move $a0, $t1
        syscall

        li $v0 , 4
        la $a0 , brk
        syscall

        li $v0 ,10
        syscall

#PG
.data
        brk: .asciiz "\n"
.text
        #lendo a quantidade de termos
        li $v0, 5
```

```
        syscall
        move $t0, $v0
        #lendo a razão da PG
        li $v0, 5
        syscall
        move $t1 ,$v0
        #ler o primeiro termo da PG
        li $v0 , 5
        syscall
        move $t2, $v0
        #criando um contador
        move $s0 , $zero
Loop:
        #checando se o contador é igual ao n° de termos
        beq $t0, $s0, end
        #printando o primeiro elemento
        li $v0,1
        move $a0 ,  $t2
        syscall
        #quebra de linha
        li $v0 ,4
        la $a0, brk
        syscall
        #calculando o valor do próximo termo
        mul $t2, $t2, $t1
        #incrementando o contador
        addi $s0,$s0 ,1
        #voltando para o loop
        j Loop
end:
        li $v0 , 10
```

syscall