GSI018 – Sistemas Operacionais

Revisão de Arquitetura e Organização de Computadores

Universidade Federal de Uberlândia
Faculdade de Computação
Prof. Dr. rer. nat. Daniel D. Abdala

Na Aula Anterior...

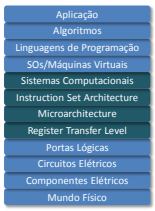
- Estrutura dos SOs;
- Serviços dos SOs;
- Interface dos SOs;
- Chamadas do Sistema;
- Filosofia dos SOs;
- Taxonomia dos SOs;
- Interfaces dos SO;

Nesta Aula

- Abstração de Computadores e a Arquitetura von Neumann;
- Sistemas Computacionais (PC);
- ISA – Arquitetura do Conjunto de Instruções;
- Assembly;
- Organização de Computadores – Caminho de Dados;
- Processadores Mono vs Multiciclo;
- Desempenho de UCPs;
- Pipelining;
- Hierarquia de Memórias;
- Memórias Cache.

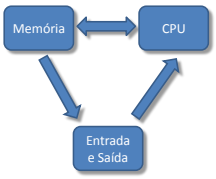
Abstração de Computadores

- O Computador é uma máquina complexa;
- Impossível de lidar com toda a complexidade de uma só vez. Muita informação;
- Solução: Abstrair níveis de complexidade.

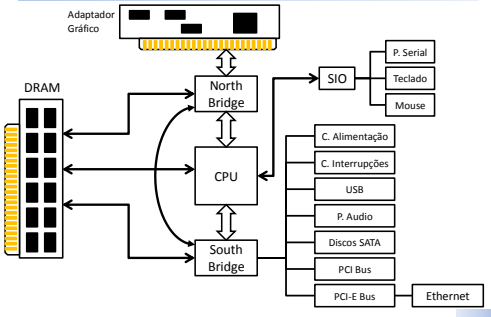


Arq. von Neumann e o Programa Armazenado em Memória

- Organização fundamental de sistemas computacionais;
- Usada até hoje!

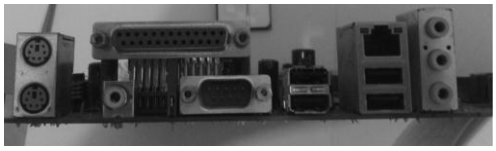


Anatomia de Um Sistema Computacional (PC)



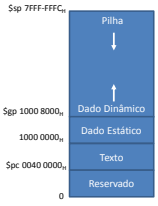
Placa Mãe (Legada – 2004)

Conectores Típicos



Abstração de Memória

Possível Organização

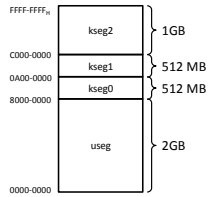


- Do ponto de vista do programador:
- 2^{32} endereços distintos
- 4.294.967.296 bytes
- 2.147.483.648 half words
- 1.073.741.824 words
- 34.359.738.368 bits
- 4 bytes por instrução
- Array de dados;
- Endereço identifica uma célula de memória individual;
- O conteúdo da célula contém efetivamente o dado/instrução;

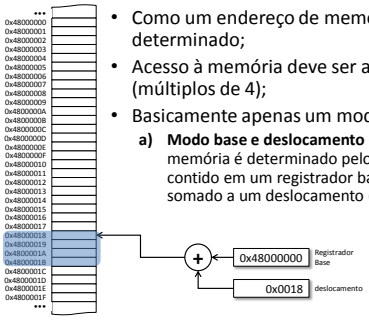
Abstração Memória:

Arquitetura de Memória

- Logicamente, a memória pode ser vista como um arranjo unidimencional de bytes individualmente endereçáveis;
- Organizacionalmente a memória é dividida em duas grandes regiões (kseg, e useg) que podem ser divididas em outras subregiões;



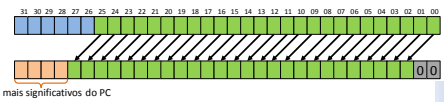
Modos de Endereçamento



- Como um endereço de memória é determinado;
- Acesso à memória deve ser alinhado (múltiplos de 4);
- Basicamente apenas um modo!
  - a) **Modo base e deslocamento** → endereço de memória é determinado pelo endereço contido em um registrador base (32 bits) somado a um deslocamento (16 bits)

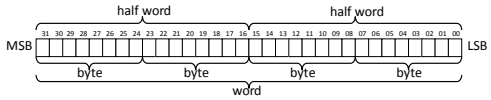
Modos de Endereçamento

- Do ponto de vista funcional, podemos pensar em outros modos de endereçamento, se considerarmos também como referenciar registradores:
  - b) **Modo Registrador** → Apenas registradores são usados na instrução:
    - 5 bits são usados para diferenciar entre os 32 registradores de propósito geral (RPG);
    - Modo mais comum usado na arquitetura MIPS;
  - c) **Modo relativo ao PC** → usado em instruções de salto condicional:
    - Endereço destino é a soma do PC + deslocamento de 16 bits (sinalizado);
    - Deslocamento dado em palavras e deve ser multiplicado por 4;
    - Possível endereçar [-32.768, 32.767] palavras a partir do PC;
  - d) **Modo Imediato** → usado em instruções lógicas e aritméticas apenas:
    - A constante é um int16 [-32.768, 32.767];
  - e) **Modo absoluto** → Usado em desvios incondicionais:



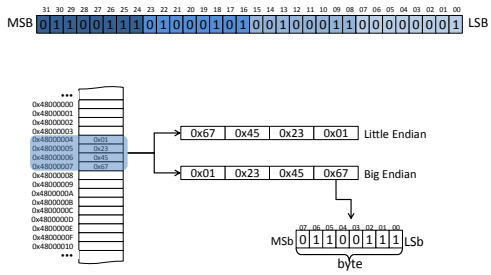
Ordenação de Bytes

- A únidade mínima endereçável é o byte;
- Uma palavra (word) possui 32 bits, ou seja, é composta por 4 bytes;
- Ordenação de bytes diz respeito a posição dentro de uma word dos bytes;
- Dois modelos principais:
  - Big endian
  - Little Endian



13

Ordenação de Bytes



14

ISA – Arquitetura do Conjunto de Instruções

- ISA – Instruction Set Architecture
  - Idealização de como o µprocessador deve funcionar;
  - Visão funcional do µprocessador;
  - µprocessador do ponto de vista do programador;
  - Tipos de Dados;
  - Formatos de Instruções;
  - Instruções suportadas;
  - Registradores;
  - Modos de Endereçamento;
  - Arquitetura de Memória;
  - Interrupções, Traps e Exceções;
  - Modos de Operação.

15

Tipos de Dados

- µProcessadores são máquinas lógicas e aritméticas;
- Atuam diretamente sobre representações numéricas;
- Circuitos elétricos para processar números reais são essencialmente distintos daqueles que processam números naturais;
- Espaço representacional limitado 16, 32, 64 bits;

16

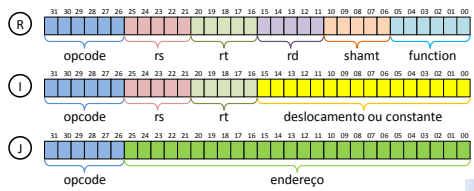
Tipos de Dados

- Na ISA do MIPS3000 (MIPS1)
  - Naturais ($\mathbb{N}$) – Inteiros não sinalizados
    - Double Word – 64 bits (resultados de mults)
    - Word – 32 bits
    - Half – 16 bits
    - Byte – 08 bits (e.g. usado para caracteres ASCII)
  - Inteiros ($\mathbb{Z}$) – Inteiros sinalizados
    - 16 bits, complemento de dois
    - 32 bits, complemento de dois
  - Reais ($\mathbb{R}$) – ponto flutuante (IEEE754)
    - 32 bits – registradores especiais
    - 64 bits – registradores especiais
- Como outros conjuntos reais são suportados?
  - Racionais ($\mathbb{Q}$), Complexos ($\mathbb{C}$), etc...

17

Formatos de Instruções

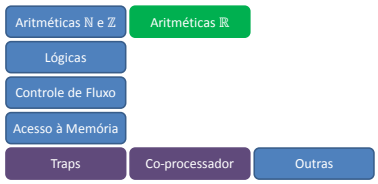
- **Formato de Instrução →** Como codificar instruções em uma sequência de bits;
- **No MIPS1 →** apenas 3 formas possíveis (tipos de formatos);



18

Conjunto de Instruções

- Operações que o µprocessador é capaz de processar;
- Subdivididas para melhor entendimento:



19

Instruções Aritméticas (ℕ e ℤ)

OP	Descrição	Exemplo
add	Adição com trap no overflow (não suportado pelo C)	add $rd, $rs, $rt
addi	Adição com trap no overflow – segundo operando cte.	addi $rt, $rs, cte
addiu	Adição sem trap no overflow – segundo operando cte.	addi $rt, $rs, cte
addu	Adição sem trap no overflow	addu $rd, $rs, $rt
aui	Adiciona a cte. Aos bits mais significativos	aui $rt, $rs, cte
clo	Conta # de zeros até achar um 1	clo $rd, $rs
clz	Conta # de uns até achar um 0	clz $rd, $rs
div	Divisão com overflow (LO ← quociente | HI ← resto)	div $rs, $rt
divu	Divisão Natural sem overflow	divu $rs, $rt
madd	Multiplica e adiciona	madd $rs, $rt
maddu	Multiplica e adiciona (Naturais)	maddu $rs, $rt

20

Instruções Aritméticas (ℕ e ℤ)

OP	Descrição	Exemplo
mfhi	Move o dado contido no registrador HI	mfhi $rs
mflo	Move o dado contido no registrador LO	mflo $rs
msub	Multiplica e subtrai	msub $rs, $rt
msubu	Multiplica e subtrai (Naturais)	msubu $rs, $rt
mtlo	Move o dado contido em $rs para o registrador LO	mtlo $rs
mthi	Move o dado contido em rS para o registrador HI	mthi $rs
mul	Multiplicação s/ overflow	mul $rd, $rs, $rt
mult	Multiplicação (hi ← $32bits_{MSB}$ | lo ← $32bits_{LSB}$)	mult $rs, $rt
multu	Multiplicação Natural (hi ← $32bits_{MSB}$ | lo ← $32bits_{LSB}$)	multu $rs, $rt
sub	Subtração com overflow	sub $rd, $rs, $rt
subu	Subtração sem overflow	sub u $rd, $rs, $rt

21

Instruções Lógicas

OP	Descrição	Exemplo
and	E bit a bit	and $rd, $rs, $rt
andi	E bit a bit – segundo operando cte	andi $rt, $rs, cte
nor	NÃO OU bit a bit	nor $rd, $rs, $rt
or	OU bit a bit	or $rd, $rs, $rt
ori	OU bit a bit – segundo operando cte	ori $rt, $rs, cte
sll	Deslocamento para a esquerda (multiplicação base 2)	sll $rt, $rs, [0:31]
slr	Deslocamento para a direita (divisão base 2)	slr $rt, $rs, [0:31]
sllv	Deslocamento para a esquerda (multiplicação base 2)	sllv $rd, $rs, $rt
slrv	Deslocamento para a direita (divisão base 2)	slrv $rd, $rs, $rt
slt	Seta se menor que (sinalizado)	slt $rd, $rs, $rt

22

Instruções de Controle de Fluxo

OP	Descrição	Exemplo
beq	(=) Salta se igual	beq $rs, $rt, offset
bgez	(≥0)Salta se maior ou igual a zero	bgez $rs, $rt, offset
bgezal	(≥0)Salta se maior ou igual a zero (usado em subrotinas) ($ra ←PC+4)	bgezal $rs, $rt, offset
bgtz	(>0)Salta se maior que zero	bgtz $rs, $rt, offset
blez	(≤0)Salta se menor ou igual a zero	blez $rs, $rt, offset
bltz	(<0) Salta se menor que zero	bltz $rs, $rt, offset
bltzal	(<0) Salta se menor que zero ($ra ←PC+4)	bltzal $rs, $rt, offset
bne	(≠) salta se diferente	bne $rs, $rt, offset
j	Salto incondicional	j offset
jal	Salto incondicional e liga ($ra ←PC+4)	jal offset
jalr	Salto incondicional (registrador) Salva em $ra o endereço da instrução de retorno	jalr $rs
jr	Salta para endereço em registrador	jr $rs

23

Instruções de Acesso a Memória

OP	Descrição	Exemplo
lb	Carrega byte da memória (rt ←MEM[rs+offset])	lb $rt, offset($rs)
lbu	Carrega byte da memória (rt ←MEM[rs+offset])	lbu $rt, offset($rs)
lh	Carrega half word da memória (rt ←MEM[rs+offset])	lh $rt, offset($rs)
lhu	Carrega half word da memória (rt ←MEM[rs+offset])	lhu $rt, offset($rs)
lui	Carrega a cte nos 16 bits mais significativos do registrador	Lui $rt, cte
lw	Carrega word da memória (rt ←MEM[rs+offset])	lw $rt, offset($rs)
sb	Salva byte na memória (MEM[rs+offset] ← rt)	sb $rt, offset($rs)
sh	Salva half word na memória (MEM[rs+offset] ← rt)	sh $rt, offset($rs)
sw	Salva wordna memória (MEM[rs+offset] ← rt)	sw $rt, offset($rs)

24

Outras Instruções

OP	Descrição	Exemplo
nop	Nao executada nada	nop
syscall	Gera uma exceção de chamada ao sistema operacional	syscall

25

Registradores

- Apenas 32 registradores?
  - Poucos e rápidos!
- E os registradores $at (1) e $k0,$k1 (26,27)?
  - Montador e SO

Nome	# do registrador	Uso	Preservado na Chamada?
$zero	0	Constante 0	-
$k0, $k1	0-1	Sistema Operacional	Não
$v0,$v1	2,3	Retorno de funções	Não
$a0-$a3	4-7	Argumentos	Não
$t0-$t7	8-15	Temporários	Não
$s0-$s7	16-23	Salvos	Sim
$t8,$t9	24,25	Mais temporários	Não
$gp	28	Ponteiro global	Sim
$sp	29	Ponteiro de pilha	Sim
$fp	30	Ponteiro de quadro	Sim
$ra	31	Endereço de retorno	Sim

26

Código de Máquina

- Processadores entendem apenas 0s e 1s;
- As instruções são especificadas utilizando um código binário;
- Programas e dados são representados desta forma;

Código de Máquina

```
00100000000010000000000000000001   20100001H
00100000000011110000000000001011   200f000bH
00010010000011110000000000000101   120f0005H
00000010000100000010000000100000   02104020H
01110001000100000100000000000010   71104002H
00000001000100001000100000100010   01108822H
00100010000100000000000000000001   22100001H
00001000000100000000000000000010   08100002H
```

27

Assembly – Linguagem de Montagem

- Código de Máquina é virtualmente ilegível para seres humanos;
- A solução é utilizar um código intermediário, ao qual chamamos de código de montagem;
- Correspondência de 1-1 para o código de máquina – conversão entre código de montagem e código de máquina é trivial;
- No entanto, código de montagem é mais inteligível aos seres humanos.

28

Linguagem de Máquina

- Na esquerda código de montagem;
- Na direita código de máquina;

Qual é mais inteligível a nós meros humanos?

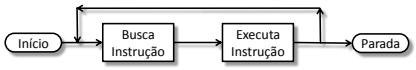
```
1  .text
2    addi  $s0, $zero, 1
3    addi  $t7, $zero, 11
4  FOR:  beq $s0, $t7, SAIFOR
5    add $t0, $s0,$s0
6    mul $t0, $t0,$s0
7    sub $s1, $t0,$s0
8    addi  $s0, $s0,1
9    j FOR
10 SAIFOR:
```

Código de Máquina

```
00100000000010000000000000000001
00100000000011110000000000001011
00010010000011110000000000000101
00000010000100000010000000100000
01110001000100000100000000000010
00000001000100001000100000100010
00100010000100000000000000000001
00001000000100000000000000000010
```

29

Ciclo Básico de Execução de Instruções



30

Ciclo de instrução

Busca de Instrução

Busca dos Operadores

Execução da Instrução

Armazenamento dos Resultados

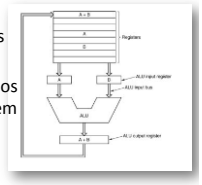
Checagem de Interrupções

Qual a prox. Instrução? Qual a prox. Instrução?

31

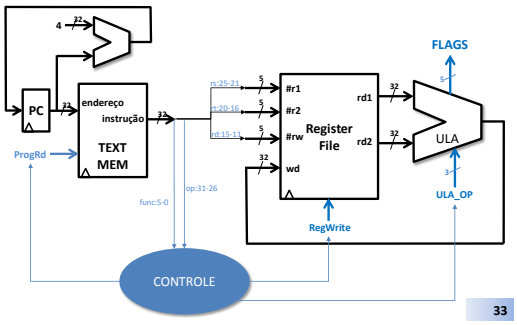
Organização de Computadores Caminho de Dados

- Em computação de propósito geral, os sistemas digitais que compoem o processador devem ser reconfigurados para cada instrução a ser executada;
- O caminho de dados delineia os diversos possíveis caminhos que os dados podem atravessar durante a execução de uma instrução;



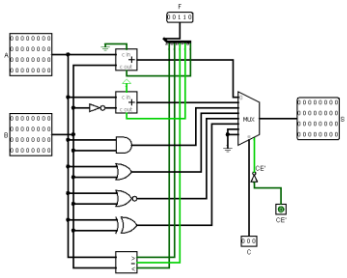
32

Caminho de Dados – Apenas Tipo R



33

ULA – Diagrama Esquemático



34

µarquitetura (Organização)

- Forma de construir a ISA de um processador;
- A mesma ISA pode possuir diversas µarquiteturas distintas;
  - Diferentes formas de se implementar os circuitos aritméticos;
  - Otimizado para baixo consumo de energia;
  - Com ou sem pipeline;
  - Uma instrução por ciclo de clock ou vários ciclos por instrução;
-

35

Problemas da implementação Monociclo

- Intuitivamente tendemos a pensar que a implementação MIPS32-Monociclo é mais rápida que a MIPS-32-Multiciclo;
  - Processador que executa uma instrução por ciclo de clock deve ser mais rápido que um processador que executa uma instrução a cada vários ciclos de clock;
- Ciclo de Clock deve ser suficientemente longo para acomodar a instrução mais lenta!
  - Geralmente a que utiliza mais "partes" do datapath;
  - Acesso a memória (LW/SW);
- Ponto chave
  - Qual a duração dos ciclos de clock comparados?

36

Exemplo

opcode	BI	DI	ULA	MEM	ER	TOTAL
j	40	20	–	–	–	60ηs
beq	40	20	20	–	–	80ηs
add	40	20	40	–	20	120ηs
lw	40	20	40	40	20	160ηs

```
addi $s0, $zero, 42
Lw   $s1, 0($s7)
beq  $s0, $s1, Label1
j    Label2
```

- Os tempos que cada subsistema demora são devidos a:
  - Tempo de chaveamento dos transistores;
  - Distância a ser percorrida pelo sinal (comprimento das conecções);
- Dentre as 4 instruções acima, o lw é a mais lenta (160ηs);
- O Clock necessita ser de 1/160ηs → F = 6,25MHz;
- Processador fica algum tempo ocioso para todas as instruções exceto a mais lenta (lw);
- Como minimizar este efeito?

Solução

- Uma possível solução para o problema descrito acerca da organização Monociclo:
  - Subdividir o datapath em subsistemas funcionais;
  - Estimar o tempo necessário para cada subsistema;
  - Definir como período de clock o subsistema mais lento;
  - Executar cada instrução utilizando apenas o número exato de clocks necessários;

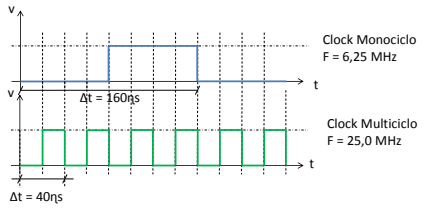
Revisitando o exemplo...

opcode	BI	DI	ULA	MEM	ER	TOTAL
j	40	20	–	–	–	60ηs
beq	40	20	20*	–	–	80ηs
add	40	20	40	–	20†	120ηs
lw	40	20	40	40	20†	160ηs

```
addi $s0, $zero, 42
Lw   $s1, 0($s7)
beq  $s0, $s1, Label1
j    Label2
```

- Cinco subsistemas:
  - BI – Busca de Instrução (B);
  - DI – Decodificação da Instrução (D);
  - ULA – Operação Lógica ou Aritmética (U);
  - MEM – Escrita/Leitura de Dado da Memória (M);
  - ER – Escrita do Resultado no Registrador (E).
- BI, ULA e MEM são os subsistemas mais lentos. Todos levam 40 ηs;
- Período de clock = 40 ηs → F = 25MHz.

Relação entre Clocks

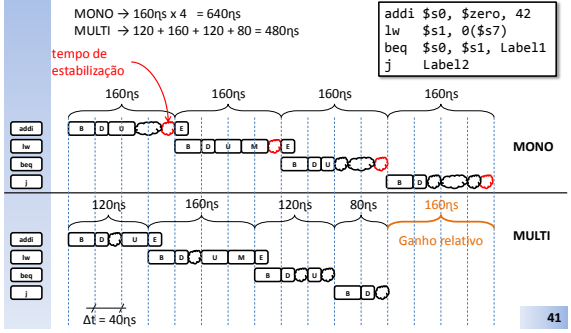


- Questão: Duplicar ou até mesmo quadruplicar a velocidade do clock duplica ou quadruplica a velocidade do Processador?

Comparação Mono vs Multi

MONO → 160ηs x 4 = 640ηs
MULTI → 120 + 160 + 120 + 80 = 480ηs

```
addi $s0, $zero, 42
lw   $s1, 0($s7)
beq  $s0, $s1, Label1
j    Label2
```

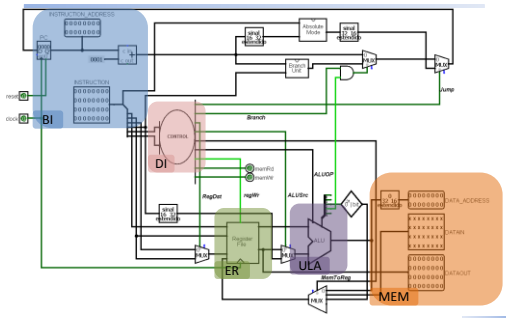


O Custo da Implementação Multiciclo

"There is no free lunch!"

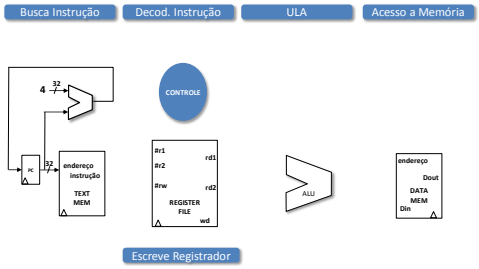
- Execução das instruções agora toma mais de um ciclo e ainda um número variável de ciclos;
- Como coordenar a execução de uma instrução por vários ciclos de clock?
  - R: Gerar sinais de controle específicos para cada ciclo de clock em cada passo da execução da instrução;
- Requisitos:
  - O subsistema de controle não pode ser mais um circuito combinacional, pois o estado anterior do subsistema importa;
  - Máquinas de estados finitos!
  - Registradores adicionais para armazenar:
    - Instrução sendo executada;
    - Dado lido/escrito da memória;
  - Novos sinais de controle.

Revisitando a Organização MONO



43

Visão das Unidades Funcionais - Subsistemas



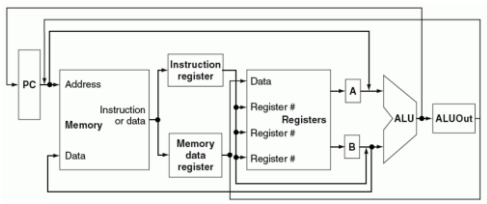
44

Subsistemas Utilizados por Instrução

Instrução	Duração	Subsistemas				
add	120	BI	DI	ULA		ER
sub	120	BI	DI	ULA		ER
and	120	BI	DI	ULA		ER
or	120	BI	DI	ULA		ER
nor	120	BI	DI	ULA		ER
xor	120	BI	DI	ULA		ER
slt	120	BI	DI	ULA		ER
addi	120	BI	DI	ULA		ER
andi	120	BI	DI	ULA		ER
ori	120	BI	DI	ULA		ER
xori	120	BI	DI	ULA		ER
beq	120	BI	DI	ULA		
lw	160	BI	DI	ULA	MEM	ER
sw	160	BI	DI	ULA	MEM	
j	80	BI	DI			

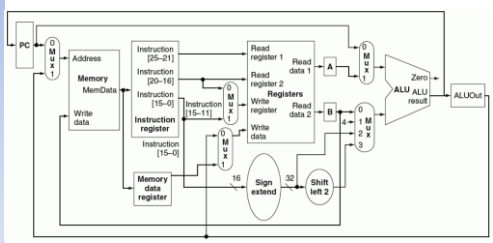
45

Implementação de Alto Nível Multiciclo



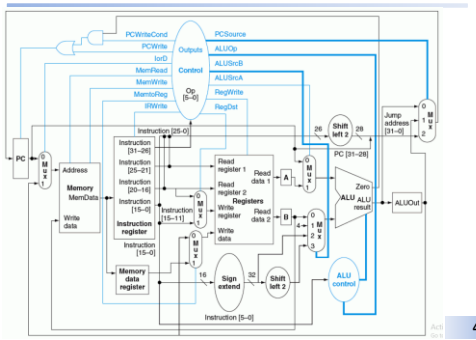
46

Datapath – Instruções Tipo-R



47

Datapath Completo



48

Desempenho de UCPs

- Forma de medir o esforço necessário para o processador executar uma tarefa;
- Conceitos chave:
  - Velocidade do Clock;
  - Desempenho Relativo;
  - Throughput;
  - Tempo de Execução;
  - Tempo de CPU;
  - CPI – Clock Cicles per Instruction;

49

Desempenho em Sistemas Paralelos

- Intuição
  - 1 µProc|1 thread → tempo de execução = x ηs
  - 2 µProc|1 thread → tempo de execução = x/2 ηs
  - 4 µProc|1 thread → tempo de execução = x/4 ηs
- Certo?
  - Infelizmente ... NÃO!!!!!

50

Lei de Amdahl

- Frequentemente usado em computação paralela para prever o máximo speedup teórico usando múltiplos processadores;

$$s(n) = \frac{1}{(1-B) + \frac{B}{n}}$$

- $n = número\ de\ threads$
- $B =$ parcela do algoritmo puramente sequencial [0,1]

51

Pipelining

- A organização do processador em subsistemas funcionais resolve o problema de ociosidade devido a diferença de tempo de execução entre instruções;
- No entanto, deixa claro outro fator → durante a execução de uma instrução, apenas um dos subsistemas é utilizado por vez, gerando outro tipo de ociosidade;
  - Ociosidade entre subsistemas;
- Solução
  - Executar instruções seguindo o mesmo modelo de linhas de produção;

52

Observações sobre Pipelining

- O conceito de pipelining foi inventado originalmente em supercomputadores;
- MIPS foi projetado para tirar o melhor proveito de pipelining;
  - Tamanho fixo de instruções
  - Favorecer o uso de registradores
  - Operações atômicas de acesso a memória
  - ...
- Atualmente, virtualmente todos os processadores comerciais utilizam o conceito de pipelining;

53

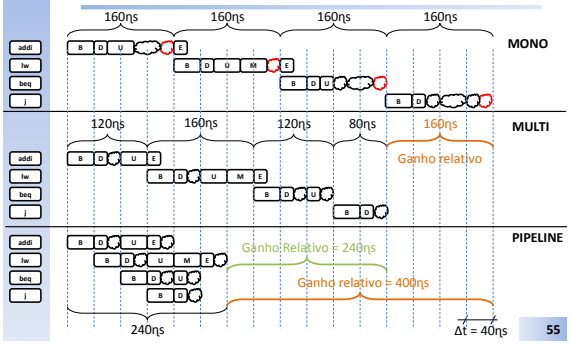
O Custo da Implementação Pipelining

"There is no free lunch!"

- Execução das instruções agora toma mais de um ciclo e ainda um número variável de ciclos;
- Como coordenar a execução de uma instrução por vários ciclos de clock?
  - R: Gerar sinais de controle específicos para cada ciclo de clock em cada passo da execução da instrução;
- Requisitos:
  - O subsistema de controle não pode ser mais um circuito combinacional, pois o estado anterior do subsistema importa;
  - Máquinas de estados finitos!
  - Registradores adicionais para armazenar:
    - Instrução sendo executada;
    - Dado lido/escrito da memória;
  - Novos sinais de controle;
  - Unidades para detecção de Hazards.

54

Intuição e Comparativo



Hierarquia de Memórias

- Densidade, Custo, Volatidade e Persistência
  - SRAM, DRAM, ROM, Hard-Disk, CD, DVD, etc
- Idealmente ...

  "Implementar um sistema de memória que custe o mínimo possível e que tenha o maior desempenho alcançavel!"
- Na prática ...

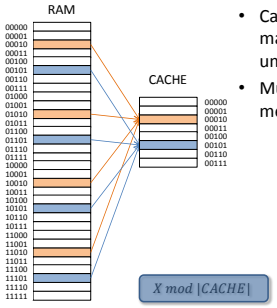
  "Gastar o mínimo possível e espertamente organizar memórias de diferentes velocidades e com características distintas para produzir a **ilusão** de que toda a memória é rápida."

Memórias Cache

Tipo Mem.	T. Acesso
SRAM	0,5 a 2,5 ns
DRAM	50 a 70 ns
HD	5 a 20 ms

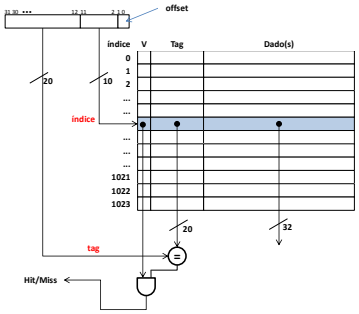
- Memória Cache é uma SRAM;
- Pode ser organizada em vários níveis (L1, L2, L3);
- Copiar dados da memória principal para a Cache de modo a tornar o acesso mais rápido;
- Termos relacionados:
  - Princípio da Localidade Temporal;
  - Princípio da Localidade Espacial;
  - Hit/Miss;
  - Taxa de acertos, taxa de falhas;
  - Tempo de acerto, Tempo de falha (penalidade de falha);

Mapeamento Direto

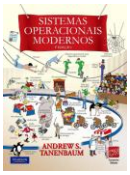


- Cada posição da RAM é mapeada diretamente para uma posição da cache;
- Múltiplas posições da RAM na mesma posição da CACHE;

Acesso a CACHE

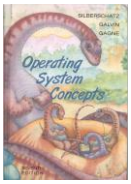


Bibliografia



- Ótimo livro texto;
- Referência básica para a disciplina;
- A disciplina é estruturada com base neste e no livro do Silberschatz;

Bibliografia



- Ótimo livro texto;
- Referência básica para a disciplina;
- A disciplina é estruturada com base neste e no livro do Tanenbaum;

61

Bibliografia

- **PATTERSON, D. A. e HENNESSY, J. L. 2014.** *Organização e Projeto de Computadores – A Interface Hardware/Software.* Elsevier/ Campus 4ª edição.

- **HENNESSY, J. L. e PATTERSON, D. A. 2012.** *Arquitetura de Computadores – Uma Abordagem Quantitativa.* Elsevier/ Campus 5ª edição.

62

Bibliografia

- **STALLINGS, W. 2002.** *Arquitetura e Organização de Computadores.* 2002.

- **TANENBAUM, A. S. 2007.** *Organização Estruturada de Computadores.* 2007.

63

Bibliografia

64