Tipos de miss (3 C's)

Compulsory (cold-start): A primeira vez que é acedido um endereço pertencente a um bloco, de não está em cache. Relacionados com o tamanho dos blocos.

Capacidade: Misses devidos ao bloco ter sido retirado da cache por a cache não ter capacidade para todos os blocos usados pelo programa. Relacionados com o tamanho da cache.

Conflito (ou colisão): misses devidos ao bloco ter sido retirado da cache por estar a ocupar a posição onde deverá passar a estar outro bloco, e que não ocorreriam se a cache fosse fully associative. Relacionados com a associatividade da cache

Escrita na memória

bloco está na cache (hit)	bloco não está em cache (miss)
atualizar só a cache: write-back	. ler o bloco para a cache, passando a hit: write-Allocate
atualizar cache e memória: write-through	. não ler o bloco para a cache; é atualizada só a memória: write-Allocate

write-through: escritas são imediatamente propagadas para o nível abaixo da memória com ou sem write-allocate. Pode escrever na cache antes de o bloco estar presente.

write-back: escritas só se refletem no nível abaixo da memória quando o bloco é substituído usa em geral write-allocate. substituição de um bloco modificado (dirty) só depois de copiado para o nível inferior (ou para um write-back buffer). Pode ser usado um write buffer para guardar as alterações a efetuar.

Acesso à memória e t_{CPU}

ignorando o acesso à memória: 1 Hit time = 1 ciclo

contando com os acessos à memória

T_{CPU} = nº ciclos de execução × duração 1 ciclo

t_{CPU} = (nº ciclos de execução + nº ciclos memory stall) × duração 1 ciclo

- nº ciclos memory stall = nº ciclos read-stall + nº ciclos write-stall
- nº ciclos read stall = nº reads × read miss-rate × read miss penalty
- nº ciclos write stall = nº writes × write miss rate × write miss penalty + write buffer stalls

(latência de acesso = miss penalty)

Average memory access time

AMAT = hit time + miss rate × miss penalty

caches com vários níveis

cada nível apresenta uma miss rate local

miss rate L_2 = nº miss L_2 / nº acesso à cache L_2

miss rate global: percent. de acessos que obrigam a acesso à memória principal

Speedup = tempo(1) / tempo(L_1, L_2)

miss rate global = miss rate L_1 × miss rate L_2

caches 2 níveis: miss rate L_1 = nº miss L_1 / nº acesso à cache L_1

memória virtual: memória organizada em páginas (blocos entre 4kB e 16kB)

As páginas da memória virtual são mapeadas das páginas da M. Física

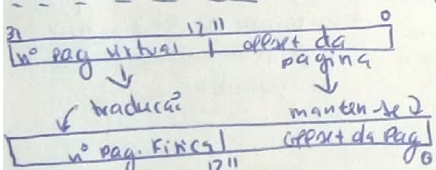
uma página de M. virtual pode residir em qualquer das páginas da memória física.

Miss = Hit time + miss penalty ciclos

CPI global = % inst Hit × CPI Hit + % inst miss × CPI miss

Tradução de endereços: para efetuar um acesso à memória, é necessário traduzir o endereço virtual para o endereço físico correspondente.

End. virtual de 32 bits



- a tabela de páginas mantem a correspondência entre as páginas virtuais de um programa e as páginas físicas
- cada programa em execução constitui um processo e tem a sua tabela de páginas, residente no espaço de memória do sistema operativo.
- o page table register do processador contém o endereço da tabela de páginas do programa que está a ser executado

endereço físico: 30 bits página: 2^{12} bytes = 4kB

implementação da tabela de página:

① **Tradicional (completa)**: uma entrada por cada página virtual identificando a página física correspondente.

Prós: acesso direto contras: para os espaços de endereçamento recentes, necessita de demasiada memória

② **crescente**: uma entrada por cada página virtual cresce à medida que são acedidas mais páginas

Prós: acesso direto contras: grande parte dos programas aloca a pag. no início e no fim do espaço de endereçamento

③ **crescente em 2 sentidos**: uma entrada por cada pag. virtual. Dividida em 2 partes, uma para as páginas de endereços baixos outra para as pag. de endereços altos. cresce em 2 sentidos à medida que são acedidas mais páginas. MIPS. Pros: acesso imediato contras: não se adapta a programas que usam o espaço de endereçamento de modo descontínuo

invertida: uma entrada por cada pag. física. Acesso através de uma função (hash) aplicada ao nº da pag. virtual

Pros: Adaptada ao espaço end. Físico contras: acesso complexo

Multi-nível: partes do nº de pag. virtual são usadas para indexar as tabelas dos vários níveis.

Tabelas do último nível: identificam as pag. físicas. Só existem as correspondentes às pag. usadas

Pros: adapta-se à parte utilizada do espaço de endereçamento. contras: acesso em 2 ou + passos

as páginas virtuais que não cabem na memória física são guardadas em memória secundária. se a pag. virtual acedida não está em memória física ocorre Page Fault. Uma pag. virtual tem de ser carregada na pag. física

a memoria física comporta-se como uma cache da memoria virtual.

TLB: Translation-lookaside buffer é uma cache da tabela de pag que contem traduções de paginas virtuais em físicas.
a tabela só é consultada se a tradução nº Pag. virtual → nº Pag. Física não estiver presente no TLB.

acesso ao TLB → TLB hit o nº da pag física contido no TLB é concatenado com o offset na pag. para obter o endereço físico

TLB miss:
é gerada a exceção tlb miss.

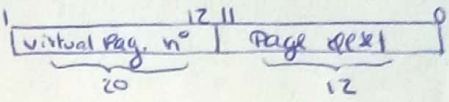
Passos de um acesso à memoria: 1º Tradução do endereço virtual para físico → procurar no TLB:
TLB miss, tlb hit
→ acesso à tabela pag
• está em memoria
• Page Fault: carrega pag. virtual para memoria física
→ atualizar TLB

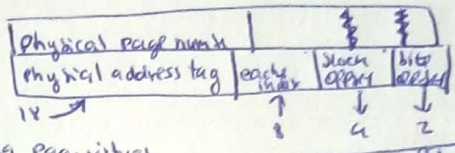
2º Acesso à memoria
→ Procura na cache:
• hit
• miss: acesso à memoria física

acesso a um disco magnetico [redacted]

velocidade de rotação: velocidade a que o disco gira. mede-se em rotações por minuto (rpm)
seek Time: tempo de colocação, tempo que demora a colocar a cabeça de leitura/escrita na pista a aceder
latencia rotacional: tempo que é necessário esperar, em media, até que o sector pretendido esteja sob a cabeça de leitura/escrita. depende da velocidade de rotação e corresponde ao tempo de meia rotação.
taxa de transferencia: velocidade a que é possivel ler (ou escrever) informação do (ou no) disco. mede-se em MB/s
controlador: circuito que controla o funcionamento do disco induzalgum overhead nos acessos.

- Fully associative
- LRU
- write back: a cada Pag. virtual está associado um dirty bit que indica se o conteudo da Pag foi alterado e se é necessario copiá-lo para memoria secundária quando a pag for substituida na memoria física.





giga = 10^9		kbyte = 2^{10} = 1024	
mega = 10^6		mbyte = 2^{20}	
Tera = 10^{12}		tbyte = 2^{30}	
peta = 10^{15}		pbyte = 2^{40}	
exa = 10^{18}		ebyte = 2^{50}	
zeta = 10^{21}			

Tempo de acesso = seek time + latência rotacional + tempo de transferência + overhead do controlador

$$\text{latencia rotacional} = \frac{1}{2} \times \frac{60}{\text{velocidade de rotação}}$$

exemplo: Qual o tempo necessario para ler um bloco de 512 bytes de um disco com:
seek time medio = 6,0 ms
velocidade de rotação = 15 000 rpm
taxa de transferencia = 100 MB/s
overhead do controlador = 0,2 ms
e o periodo do relogio do processador de 0,5 ns, quantos ciclos de relogio leva a leitura?

L. rotacional = $\frac{1}{2} \times \frac{60}{15000}$ = 2,0 ms

$t_{transf} = \frac{\text{bytes a transf}}{\text{taxa de transf}} = \frac{512}{100 \times 10^6} = 0,00512$ ms

$t_{acesso} = 6,0 + 2,0 + 0,00512 + 0,2 = 8,20512 \approx 8,2$ ms

overhead = bytes a trf / taxa. transf do controlador ms/s

$$\text{ciclos}_{acesso} = \frac{t_{acesso}}{T} \approx \frac{8,2 \times 10^{-3}}{0,5 \times 10^{-9}} = 17,4 \times 10^6$$

situação ideal:

Tempo depois da paralelização = tempo antes da paralelização / nº de processadores

Speed up = tempo antes da paralelização / tempo depois da paralelização = nº de processadores

desafio do Paralelismo (lei de Amdahl)

$$\text{Speed up} = \frac{\text{tempo antes}}{\text{tempo antes} - \text{tempo não afetado} + \frac{\text{tempo não afetado}}{\text{nº de processadores}}}$$

$$\text{tempo depois paralelização} = \frac{\text{Tempo antes paralelização} - \text{Tempo não afetado paralelização}}{\text{nº de processadores}} + \text{tempo não afetado paralelização}$$

$$\text{Speed up} = \frac{1}{1 - \%\text{ tempo não afetado} + \frac{\%\text{ tempo não afet.}}{\text{nº processadores}}}$$

escalabilidade: diz respeito ao modo como evolui o speedup de um programa com o aumento do nº de CPU's

escalabilidade forte: evolução do speedup com o nº de processadores
escalabilidade fraca: evolução do speedup quando a dimensão do problema aumenta com o nº de processadores

sistemas de memoria partilhada: uniform memory access (UMA) non-uniform ... (NUMA)

GPU: graphics processing unit: Processador especializado para criação de elementos graficos
GPGPU: general programming GPU: multiprocessador que mantem aptidões da GPU's para o processamento de elementos graficos mas que pode ser usado para progr. genericos

n-way set associative: conjunto com n posições da cache
fully associative: colocação em qualquer posição da cache

palavra = endereço / bytes por palavra
bloco = palavra / palavra por bloco = endereço / bytes por bloco

indice / conjunto = Palavra % nº indices ou bloco % nº conj
tag = palavra / nº indices ou bloco / nº conj
endereço físico = nº pag física × tam. Pag + offset
capacidade total cache? [valid bit + tag + 32] × n / 8 bits
tag = (n - offset - indice)

endereço / tamanho Pag

Hit/miss rate = / cache

1GH = 1×10^9 Hz

• aumento de CPU's não significa aumento do desempenho.

tempo processamento paralelo ou programa paralelo quando os processadores utilizam multiplos

speed up = tempo antes / tempo novo (depois)

Quanto menor for o tempo de execução de um programa no computador x, maior será o desempenho desse:

$$desempenho_x = \frac{1}{Tempo\ Execução_x}$$

$desempenho_x > desempenho_y$ = $tempo_{execução_x} < tempo_{execução_y}$

- x é mais rápido do que y
- n>1: comp x é n× mais rápido que o comp y

$$speedup\ x \text{ e } y = \frac{desempenho_x}{desempenho_y} = \frac{tempo\ ex_y}{tempo\ ex_x} = n$$

- n é a melhoria de desempenho que se obtem usando o comp x, em relação a usar-se o y.
- n<1: slowdown

$T = 250\ ps = 250 \times 10^{-12} = 0{,}25 \times 10^{-9}$

$f = 1/T = \frac{1}{0{,}25 \times 10^{-9}} = 4 \times 10^{9}\ Hz = 4\ GHz$

$t_{CPU} = n^o\ ciclos \times T$

$t_{CPU} = \frac{n^o\ ciclos}{f}$

$n^o\ ciclos = n^o\ instruções \times CPI$

$CPI_{global} = \sum_{classe} (\%)_c \times CPI_c$

$t_{CPU} = n^o\ instruções \times CPI \times T$

$t_{CPU} = \frac{n^o\ instruções \times CPI}{f}$

Instrução Tipo-R

opcode	rs	rt	rd	sh	f
31	26	21	16	11	6

Lei de Amdahl's. tempo de execução antes da melhoria = tempo de execução afetado pela melhoria / quantidade de melhoria + tempo de execução não afetado

controlo da operação da ALU

maior MIPS não significa maior rapidez.

MIPS (10^6 = mil milhões)

$MIPS = \frac{n^o\ de\ instruções}{t_{CPU} \times 10^6}$

$MIPS = \frac{f}{CPI \times 10^6}$

Instrução Tipo-I

opcode	rs	rt	imm
(6)	(5)	(5)	(16)

Instrução Tipo-J

opcode	imm
31 (6)	26 (26) 0

Instruction opcode	ALUop	Instru. operation	Funct field	Desired ALU action	ALU control input
lw	00	Load word	xxxxxx	add	0010
sw	00	Store word	xxxxxx	add	0010
beq	01	branch equal	xxxxxx	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	and	100100	and	0000
R-type	10	or	100101	or	0001
R-type	10	set less than	101010	SLT	0111

ALUop		Function Field						operation
op1	op0	F5	F4	F3	F2	F1	F0	
0	0	x	x	x	x	x	x	0010
x	1	x	x	x	x	x	x	0110
1	x	x	x	0	0	0	0	0010
1	x	x	x	0	0	1	0	0110
1	x	x	x	0	1	0	0	0000
1	x	x	x	0	1	0	1	0001
1	x	x	x	1	0	1	0	0111

ALU control	Function
0000	And
0001	OR
0010	Add
0110	Substr
0111	SLT
1100	NOR

unidades não ativas Tipo-R: sign-ext, ALUbranch, shift, memoria

unidades não ativas lw: shift, ALU Branch, memoria

unidade não ativa sw: mux (regdst), mux (memtoreg)

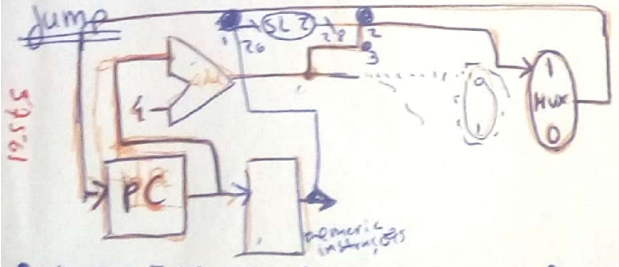
unidade não ativa beq: memória, mux (not true), ALUBranch, shift

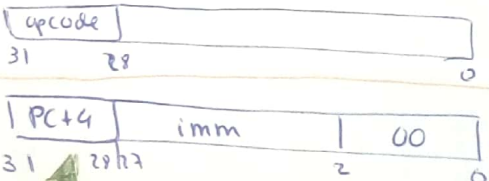
unidade não ativa jump: nada ativa, exceto o mux (esta no desenho)

Sinais de controlo

RegDst	Regwrite	ALUSr	ALUcontrol	MemtoReg	MemRead	MemWrite	Branch	Instruction
1	1	0	Function	0	0	0	0	R-type
0	1	1	Function	1	1	0	0	lw
x	0	1	"	x	0	1	0	sw
x	0	0	"	x	0	0	1	beq

Jump





19,5,6,5

1 instr c [25-0] 2 instruc [31-0] 3 PC+4 [31-28]

lw é mais longo $T \geq 800\ ps$ $f \leq$

no caso do jump não faz sentido falar acerca delay slot dado que no fim de cada ciclo em que uma instrução de salto é executada, é conhecido o endereço da próxima inst. a ser executada [PC+4] destino do salto

latência de um circuito: tempo desde que os valores estão presentes nas entradas do circuito até as suas saídas estarem estabilizadas.

caminho crítico: caminho que compreende a sequência de operações efetuadas durante a execução da instr. cuja duração é a mais longa. aumento caminho aumento duração instr

Pipeline: | IF | ID | EX | MEM | WB |

1. Todas as inst. têm o mesmo tamanho e podem ser lidas no 1º ciclo no pipeline e descodificadas no 2º.
2. Poucos formatos diferentes de instrução e com os registos sempre nas mesmas posições, pelo que é possível iniciar a sua leitura em simultâneo com a descodificação.
3. sc loads e stores lidam com endereços então é necessário usar a ALU para calculo de endereços e para outra operação na mesma instrução
4. valores alinhados em memoria permitem que um unico acesso seja suficiente para os transferir.
5. Instruções só produzem um valor que é escrito no último andar.
6. duração do ciclo de relogio diminui tempo de execução de uma inst. diminui tende a aumentar sobretudo se os andares do pipeline não são perf. equi aumenta o nº ins. executadas por unidade de tempo throughput
7. os registos do pipeline isolam os andares do pipeline e guardam a info necessária sobre a inst. a executar em cada lugar

deca = 10^1 yota = 10^{24}

hecto = 10^2

quilo = 10^3

mega = 10^6

giga = 10^9

tera = 10^{12}

peta = 10^{15}

exa = 10^{18}

zetta = 10^{21}

quilobyte = 10^3 megabyte = 10^6 gigabyte = 10^9 terabyte = 10^{12}

quibibyte = $2^{10} = 1024$

mebibyte = 2^{20}

gibibyte = 2^{30}

tebibyte = 2^{40}

Trata-se do uso de paralelismo na execução das instruções de um programa sequencial num único CPU.

Não se trata da execução de programas paralelos, compostos por várias partes que podem ser executadas em paralelo por múltiplos CPUS.

ILP: Paralelismo ao nível da execução das instruções

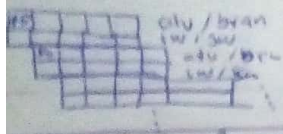
Formas de ILP:

- Pipelining: várias instruções estão a ser executadas em cada ciclo de relógio. Quanto mais profundo é o pipeline, maior é o ILP. Cada instrução está numa fase diferente da execução.
- Multiple issue: várias instruções estão a começar a ser executadas em cada ciclo de relógio. Instruções recorrem a unidades funcionais duplicadas. O CPI pode tornar-se inferior a 1, recorre-se ao IPC. nº médio de instruções executadas por ciclo.
  - issue width do processador é o nº de instr. que podem ser lançadas em simultâneo.
  - as instruções candidatas a lançamento encontram-se nos issue slots do processador.
  - as instruções que são efetivamente lançadas em simultâneo constituem um issue packet.

Multiple issue estático: é o compilador que decide que instr. serão executadas em simultâneo, organizando-as nos issues slots do processador. Cabe ao compilador reduzir ou garantir que não ocorrem conflitos. As instr. nos issues slots (que vão constituir o issue packet) podem ser encaradas como única grande instrução, apelidade de very long inst. word (VLIW)

Mips com VLIW double issue: Pode executar uma instrução de acesso à memória em simultâneo com uma aritmética ou salto.

No Pipeline fica:



Reorganização do código

ALU or branch instruction	Data transf. inst	CC
loop:		

CPI: $\frac{\text{nº ciclos}}{\text{nº inst}}$ IPC = $\frac{\text{nº inst}}{\text{nº ciclos}}$

Passando a decisão do seq para o andar ID, haverá no máximo uma nova instrução no pipeline quando é decidido se o salto será efetuado

3+1=4

Multiple issue dinâmico: processador analisa as inst nos issue slots e decide quantas lançará em simultâneo. É o processador que lida com os conflitos estruturais, de dados e de controlo, garantindo a correção da execução. Um processador super escalar é processador com MI dinâmico. Em alguns casos as instruções podem ser executadas para de ordem no pipeline dynamic pipeline scheduling.

Execução especulativa de inst: consiste em decidir que inst que vão ser executadas assumindo que uma inst. anterior terá um determinado efeito. A execução especulativa permite aumentar o CPI.

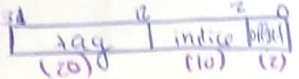
Bottlenecks: Fatores que dificultam o aproveitamento do ILP: dependências longas, acessos à memória, inst que levam muitas operações e saltos imprevisíveis (perda de tempo → erros de especulação)

one-way / redassociative = direct mapped, colocação numa posição fixa da cache.

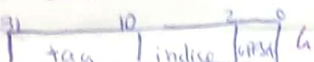
2-way set associative → conjunto de 2 posições da cache

n-way set associative → conjunto de n posições da cache

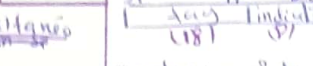
Fully Associative → colocação em qualquer posição da cache



cache direct mapped



4way



bloco com 16 palavras

Cache:
- nº de conjuntos da cache determina o conjunto em que o bloco poderá estar.
- nº blocos por conjunto determina em quantas posições um bloco poderá estar.
- dimensão de uma linha de cache, nº de palavras por bloco
- nº de bytes por palavra
- tag identifica o bloco presente em cada posição

Previsão do salto: previsão errada implica tempo

Fixa: o salto não é efetuado. O CPU continua a executar as inst. sequencialmente.

Ciclos: saltos de volta ao início do ciclo serão efetuados. Se o salto for para o início do ciclo, o CPU vai executar as inst. a partir do endereço para onde o salto é efetuado caso contrário, continua sequencialmente.

Passado: o salto terá o mesmo comportamento que da vez anterior. São guardados o endereço e destino dos saltos

Global: escolha estratégias baseada em outros saltos

Torneio: estratégia com maior sucesso, se revelar a mais precisa

Histórico: usa a história do comportamento da instrução

Cache: A info é copiada para a memória do nível superior, que é mais. Essa memória é a cache. Os conteúdos de posições de memória distintas vão ter de ocupar a mesma posição, em momentos diferentes.

Hit: o conteúdo da posição de memória acedida está em cache. índice + tag

Miss: oposto de Hit miss rate: 1 − Hit rate.

Hit time: tempo (ccc) que demora o acesso ao conteúdo de uma posição de memória na cache.

Miss penalty: tempo (ccc) que demora transferir o conteúdo de uma posição de memória de um nível inferior da hierarquia para o nível acima e re-lo para utilização (podendo substituir o seu conteúdo anterior).

Hit rate: Fração das posições de memória acedidas cujo conteúdo foi 'encontrado' na cache.

block | índice | V/B | tag | data |

V/B	tag	data	V/B	tag

repetem-se n vezes da cache (ex: 2way)

1/0 (cache ativa 1)

LRU: least recently used, escolhido elemento que não é acedido há mais tempo.

Localidade temporal: posição de memória acedida tenderá a ser acedida através em breve.

Localidade espacial: posições de memória perto de uma posição de memória acedida tenderão a ser acedidas em breve H/M sussi

- endereço: é o nº de um byte

endereço
palavra
bloco
índice
tag

palavra = $\frac{\text{endereço}}{\text{bytes por palavra}}$

bloco = $\frac{\text{palavra}}{\text{palavras por bloco}} = \frac{\text{endereço}}{\text{bytes por bloco}}$

índice/conjunto = $\frac{\text{palavra \% nº índices}}{\text{bloco \% nº de conj.}}$ $\frac{\text{palavra / nº índices}}{\text{bloco / nº conjuntos}}$

tag

O programa em execução pelo processador muda periodicamente. A execução é interrompida para se dar a mudança. Para retomar à execução do prog. é necessário saber qual a inst a executar, os valores nos registos (contexto). A mudança de contexto consiste em regular o contexto do prog. em execução e repor o contexto do prog. que o vai substituir. A mudança pode ser cooperativa (em conseq. de uma ação do prog.) ou por preempção (por decisão do sistema operativo)

Interrupções/Exceções: assinala uma circunstância excepcional, acorda atenção urgente. Pode ser origem interna ou externa ao CPU.

Capacidade total da cache? $[(VB) + (tag) + 32(1)] \times n$ (bits)

$(n - offset - indice)$

Pipeline:

- Todas as instruções têm o mesmo tamanho
- Poucos formatos diferentes de instrução e com os registos sempre nas mesmas posições, pelo que é possível iniciar a sua leitura em simultâneo com a descodificação.
- só loads e stores lidam com endereços, não na ALU.
- valores alinhados em memória permitem que um único acesso seja suficiente para os transferir.
- instruções só produzem um valor que é escrito no último andar do pipeline.

→ ciclo do relógio diminui, mas o tempo de execução não tende a aumentar se os andares não são perfeitamente equilibrados
→ todas as inst levam o mesmo tempo
→ aumenta o nº de inst. executadas por unidade de tempo (throughput)

- os registos do pipeline isolam os andares e guardam a info necessária sobre a inst. a executar em cada andar.
- os registos dos andares do pipeline também guardam os sinais de controlo da inst a executar.

conflitos estruturais: existem se não é possível executar uma ou alguma combinação de instruções no mesmo ciclo de relógio. é a razão para o MIPS ter memórias de instruções e de dados.

conflitos de dados: há-os quando uma instrução necessita de um valor produzido por uma instrução anterior, que ainda não está disponível quando a instrução lhi tenta aceder. Por exemplo, se o valor ainda não está no registo no ciclo em que instrução passa pelo andar ID.

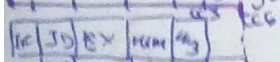
conflitos de controlo: é necessário saber o destino de um salto condicional para poder executar a próxima instrução.

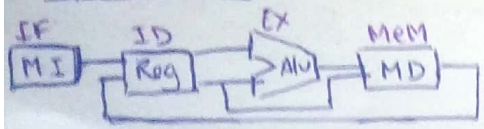
sinais de controlo do RISC

	RegDst	ALUop1	ALUop0	ALUsrc	Branch	Mem read	Mem write	Reg write	Mem to Reg
R-Format	1	1	0	0	0	0	0	1	1
lw	0	0	0	1	0	1	0	1	1
sw	x	0	0	1	0	0	1	0	x
beq	x	0	1	0	1	0	0	0	x

execution/address calculation stage control lines | memory access stage control lines | write back stage control lines

cc1 cc2 cc3 cc4 cc5 cc6 ciclos de relógio





forwarding
- WB → Mem
- Mem → Ex
- WB → Ex

sem delay slot
nops / bolha

tratamento de excepções
① - Interromper a execução do programa:
- limpar o pipeline e guardar o endereço da primeira inst que não foi completada.
② executar o código do SO que trata a excepção ocorrida (localização/gravidade)
③ retomar a execução do programa: suspensão ou abortar. (excep. software → abortar)

excepções precisas: quando lhes é associada info que indica a inst. que lhes dá origem (o seu endereço)

excepções simultâneas: quando no mesmo ciclo de relógio ocorrem múltiplas excepções, elas são atendidas, começando pela que corresponde à instrução mais avançada no pipeline

Os 3 C'S: tipos de misses:

compulsory (cold start): a primeira vez que é acedido um endereço pertencente a um bloco, ele ñ está na cache. Relacionados com o tamanho dos blocos.

capacidade: misses devidos ao bloco ter sido retirado da cache por a cache não ter capacidade para todos os blocos usados pelo programa. Relacionados com o tamanho da cache

conflito (colisão): misses devidos ao bloco ter sido retirado da cache por estar a ocupar a posição onde deverá passar a estar outro bloco e que ñ ocorreriam se a cache fosse fully associative. Relacionados com a associatividade da cache.

Escrita na memoria:

se o bloco está na cache (Hit) → atualizar só cache: write-back
→ atualizar cache + memoria: write-through

se o bloco não está na cache (miss) → ler o bloco para a cache → Hit ⇒ write allocate
→ ñ ler o bloco para a cache, é atualizada só a memoria

write-through: escritas são imediatamente propagadas para o nivel abaixo da memoria com ou sem write-allocate. Pode escrever na cache antes do bloco estar presente

write-back: escritas só se refletem no nivel abaixo da memoria quando o bloco é substituido. Usa write-allocate. Substituição de um bloco modificado (dirty) só depois de copiado para o nivel inferior (ou para um write-back buffer)

Acesso à memoria

ignorando: Tempo CPU = nº ciclos execução × 1 ciclo (T)

contando: Tempo CPU = (nº ciclos exec + nº ciclos memory stall) × 1 ciclo (T)

nº ciclos memory stall = nº ciclos read stall + nº ciclos write stall

nº ciclos read stall = nº reads × read miss rate × read miss penalty

nº ciclos write stall = (nº writes × write miss rate × write miss penalty) + write buffer stalls

latencia acesso = miss penalty

$\times$ inst/miss = miss rate

1 Hit time = 1 ciclo

1 Miss time = 1 Hit time + miss penalty

CPI global (%) = % inst H × (CPI H) + % inst Miss × (CPI M)

Average memory access time (AMAT)

AMAT = hit time + (miss rate × miss penalty)

cache multinivel: cada nivel apresenta uma miss rate local

miss rate global: % de acesso que obrigam aceder à memoria princ.

→ miss rate L1 × miss rate L2

miss rate L1: nº misses L1 / nº acessos cache L1

miss rate L2: nº miss L2 / nº acess cad L2

memoria virtual: a memoria organiza-se em paginas, blocos com 4kB a 16kB. As paginas da memoria virtual são mapeadas para paginas da memoria fisica. Uma pag. da M.V pode residir em qualquer das pag. da memoria fisica. A tabela de paginas mantem a correspondencia entre pag. vir de um programa e as pag. fisicas

Tradicional: uma entrada por cada pag. v identificando a pag. F correspon.
Pros: acesso direto
contra: necessita de demasiada memoria par espaço endere. recentes.

invertida: uma entrada por cada pag. fisica
Pros: adap ao espaço end. fisico

crescente: uma entrada para cada pag. vir, vai crescendo à medida que são inseridas mais pag.
Pros: "
contra: programas usam a pag no inicio e fim do espaço endereçam.

crescente e decrescente: igual à crescente mas em 2 partes
1. pag de ender. baixos
2. pag. end altos
Pros = "
contra: ñ se adapta a prog. que usam o espaço endereç. de forma desc.

multinivel: partes do nº da pag virtual são usadas para indicar as tabelas dos vários niveis

Pag. table (Pag)

endereço/input	Pag. virt = tag = endereço // tamanho pag
Pag. virt/tag	endereço fisico = Pag Fisica × tamanho pag + offset
TLB (H/M)	Kbyte = 2^{10} = 1024 giga = 10^9
Page table (H/M)	mibyte = 2^{20} mega = 10^6
Pag. Fisica	gibibyte = 2^{30} Tera = 10^{12}
endereço fisico	

4KiB = $2^2 \times 2^{10}$ = 2^{12} offset

1MB = 10^6 bytes

Page Fault: as pag. virtuais que ñ cabem na memoria fisica são guardadas em memoria secundaria. Se a pag. virtual acedida ñ está em memoria fisica ocorre uma page fault. Nesse caso, a pag. virtual tem de ser carregada por uma pagina fisica

TLB: é uma cache da tabela de paginas que contem traduções de pag. virtuais em fisicas. A tabela de paginas só é consultada se a tradução nº Pag vir → nº Pag Fis ñ está presente na TLB

Acesso a um disco magnetico

tempo de acesso = seek time + latencia rotacional + tempo de transf + overhead do controlador

$$t.\ rotacional = \frac{1}{2} \times \frac{60}{\text{velocidade rotação (rpm)}}$$

$$t.\ transf = \frac{\text{bytes a transf}}{\text{taxa de transferência} \times 10^6} \qquad \text{overhead controlador} = \frac{\text{bytes a transferir}}{\text{taxa de transfer} \times 10^6}$$

ciclos acesso = t acesso / T

escalabilidade forte: evolução do speedup com o nº de CPU's

escalabilidade fraca: evolução do speedup quando a dimensão do problema aumenta com o nº CPU's

escalabilidade: modo como evolui o speedup do programa

Paralelismo:

situação ideal:

tempo depois paralel = tempo antes par / nº de processadores

Speedup = tempo antes da par / tempo depois par = nº de CPU's

tempo depois paral = (tempo antes − tempo ñ afetado) / nº de CPU's + tempo ñ afetado

Lei de Amdahl:

Speed up = tempo antes / tempo depois = 1 / ((1 − % tempo ñ afet) / nº CPU + % tem ñ afeta)

Uniforme memory access (UMA)

non-uniform memory access (NUMA)

graphics processing unit (GPU)
processador especializado para a criação de elementos graficos (imagens)

General programming GPU (GPGPU)
multiprocessador que mantem as aptidões das GPUs para o processamento de elementos graficos, mas que pode ser usado para programas genericos.

Temos processamento paralelo ou programa paralelo quando o programa utiliza multiplos processadores em simultaneo

Exercicios: calcular o tamanho total da tabela de pag. para um sistema que corre supp. e que utiliza metade da memoria disponivel

virtual access size: 32 bits
page size: 8 KiB = $2^3 \times 2^{10} = 2^{13} = 8192$
pag. table entry size: 4 bytes
tag = 32 − 13 = 19   5000 × 2 = 10 MB
offset = 19
entradas = 2^{19}
tamanho P. table = $2^{19} \times 4 = 2^{19} \times 2^2$ = 2.2 = 2MB
mas utiliza metade da memoria acessivel = 10/2 = 5MB

abordagem com 2 niveis e 256 entradas
o tamanho da T.P principal é 4 bytes.
memoria acessivel max e min
× 256 = 2^8 entradas
$2^{19} - 2^8 = 2^{11}$ entradas

$2^{11} + \left(\frac{2^8}{2} \times 4\right) = \frac{x}{2}$ = max

seja P um programa cuja execução num unico processador demora 20 s dos quais 1 s corresponde à sua parte sequencial

a) qual o speedup obtido se o programa for executado em 21 CPUs com o trabalho distribuido igualmente por todos?

b) qual o speedup obtido se um dos 21 CPU fizer 7% do trabalho?

c) qual o speedup max possivel de obter com a paralelização de P?

20 s → 1 s P. sequencial, 19 s paralelização

a) tempo paralelização (21 CPUs) = (20 − 1)/21 + 1 = 1,9

speedup = 20/1,9 = 10,5

b) 1 CPU com 7% = max(19×0,93/20, 19×0,07/1) + 1 = max(0,88; 1,33) + 1 = 2,33 s

speedup = 20/2,33 = 8,58 s

c) tempo antes melhoria = 20
situação ideal ... 1 − tempo ñ ... = 1/...

add $d, $s, $t ⇒ d = s+t	addi $t, $s, im → t = s+im	addi: add imm
and $d, $s, $t ⇒ d = s and t	andi · beq $s, $t, desl: if s = t → desl	bgez: if s ≥ 0
bne $s, $t, offset: if s != t desl	load byte: lb $t, desl($s) t = mem[s + desl] a byte is loaded into a reg from a specified address	BGTZ, BLEZ, BLTZ
lw: a word is loaded into a reg from the specified address. lw $t, offset($s) t = mem[s + off]		
slt: if s is less than t, d = 1 else d = 0	sw: $t é guardado em s: sw $t, off($s) mem[s+off]	
slt $d, $s, $t		

ll (load linked) e sc (store conditional)

A instrução sc verifica se foi escrito/modificado o valor desde que se iniciou na inst. ll pode ser computada atomicamente. Se sim, então escreve o valor na memoria e rt = 1 (true). Se o contrario ocorre, então ñ é escrito nada em memoria e rt = 0 (false).

Supondo que o CPU com CPI base 1, assumindo que estamos na presença de hit na cache primária sem atrasos, e um relógio de f = 4 GHz. Assumindo que a latência de acesso de 100 ns. tem uma miss rate por inst 2%.

a) Qual o CPI real, tendo em conta os atrasos pelos acessos à memória

b) Quanto mais rápido pode o CPU ser se adicionar uma cache L2 e for grande o suficiente para reduzir a miss rate global, acesso à memória, para 0.5%

c) Qual o valor da miss rate da cache L2?

T = 0,25 ns 1 hit = 1 ciclo = 0,25 ns

miss penalty = 100 ns

1 miss = 1 hit + miss penalty = 0,25 + 100 = 100,25 ns

1 ciclo – 0,25
x – 100,25 x = 401 ciclos CPI miss = 401 ciclos

a) CPI global = % inst$_H$ × CPI$_{Hit}$ + % inst$_{mic}$ × CPI$_{Mis}$
(L1) = 0,98 × 1 + 0,02 × 401 = 9 ciclos

b) miss rate global 0,5% — L1, mem, L2

1 miss = 1 hit + miss rate
= 0,25 + 5 = 5,25

1 ciclo – 0,25
x – 5,25 x = 21 ciclos

CPI global = (0,98 × 1) + (0,02 × 21) + (0,005 × 401)
= 3,405

Speedup = $\frac{CPI\ 1}{L1+L2} = \frac{9}{3,405} = 2,64$

c) miss rate global = miss rate L1 × miss rate L2

miss rate L2 = $\frac{\text{miss rate global}}{\text{miss rate L1}} = \frac{0,005}{0,02} = 0,25$

Fully associative: colocação em qualquer posição da cache

n way associative: colocação de n posições na cache

core 1:	core 2:
ll $10, 0($4)	①
addi $10, $10, -2	② ll $12, 0($4)
sc $40, 0($4)	③ addi $17, $17, #0
lw $11, 0($4)	④
	⑤ sc $17, 0($4)
	⑥ lw $13, 0($4)

antes da execução das inst, o endereço guardado no registo $4 é 1000 0000$_{16}$ nos 2 processadores e a palavra nesse endereço é 200

Diga os valores nos registos $10, $11 do CPU e $12, $13 CPU2 e na posição de memória com endereço 1000 0000$_{16}$ depois de cada instrução.

$4 = 200

	$10	$11	$12	$13	mem[1000 0000$_{16}$]
0	x	x	x	x	200
1	200	x	x	x	200
2	198	x	200	x	200
3	1	x	210	x	198
4	1	198	210	x	198
5	1	198	0	x	198
6	1	198	0	198	198

atomic read/write

read: load linked: like a normal lw, but also saves the address in link register.

write: store conditional: check if address same as in link register. yes? normal sw, return 1
no? return 0

the idea is that you use ll to load the value stored in memory location into a register, modify it However you like there and subsequently write it back to the same place using sc. sc will only overwrite the value in memory with your modified one if no other processes has altered it while you were working on the copy in the register it has the side effect of setting a status flag to indicate whether or not it was successful. when the updated value is successfully stored, a thread can trust that its read/modify/write sequence was completed without interference from other threads.

cache: memória de nível superior de pequenas dimensões para onde a informação é copiada

endereço	
palavra	
índice	
tag	
H/M	
subs	

valid bit	tag	data
0		
1	3	M[3]

Hit: conteúdo da posição de memória acedida está na cache

Hit time: tempo (ciclos de relógio) que demora o acesso ao conteúdo de uma posição de memória na cache

miss: contrário do Hit

miss penalty: tempo que demora transf. o conteúdo de uma posição de memória de um nível inferior da hierarquia para o nível acima e tê-lo disponível para a utilizar

Hit Rate: fração das posições de memória acedidas cujo conteúdo foi encontrado na cache

miss rate = 1 - Hit rate

cache 8 posições

one way associative = direct mapped (1 posição => 8 conj)

2 way associative => 2 posições => 4 conj

4 way associative => 4 posições => 2 conj

8 way associative = 8 posições => 1 conj

tag (20)	índice (10)	offset bytes
31 … 12	11 … 2	1 0

tag	índice	offset
31 … (22)	10 9 … (8)	2 1 … 0

LRU: least recently used

- escolhido o elemento que não é acedido há mais tempo

bloco 16 palavras

(18)	(8)	(4)	(2)
31 … 14	13 … 6	5 … 2	1 … 0

(2) byte offset; (4) → block offset; (8) → índice; (18) → tag

$$\frac{\text{endereço}}{\text{bytes por palavra}} = \text{palavra}$$

$$\text{tag} = \text{bloco} / \text{nº conj} = \text{palavra} / \text{nº índice}$$

$$\text{bloco} = \frac{\text{palavra}}{\text{palavra por bloco}} = \frac{\text{endereço}}{\text{byte por bloco}}$$

$$\text{índice ou conjunto} = \text{bloco} \% \text{ nº de conjuntos} = \text{palavra} \% \text{ nº índices}$$

$$\text{offset bits} = \log_2(\text{nº bits do bloco})$$

$$\text{nº conjuntos / nº índice} = \frac{\text{nº bits cache}}{\text{nº bit endereço}}$$

$$\text{nº bits índice/conj.} = \log_2(\text{nº conjuntos})$$

escrita na memória

- se o bloco está em cache (Hit)
  - → atualizar só a cache (write back)
  - → atualizar cache + memória (write through)
- se o bloco não está em cache (miss)
  - → ler o bloco para a cache (passa a Hit)
  - → não ler o bloco para a cache, atualizar só a memória (write - allocate)

write - through: escritas são imediatamente propagadas para o nível abaixo da memória.

write - back: escrita só se refletem no nível abaixo da memória quando o bloco é substituído (dirty) ↑ bloco modificado

cache com 2 níveis

$$\text{miss rate } L_1 = \frac{\text{nº misses } L1}{\text{nº acessos cache } L1}$$

$$\text{miss rate } L_2 = \frac{\text{nº misses } L2}{\text{nº acessos cache } L2}$$

miss rate global = miss rate L1 x miss rate L2

acessos à memória:

t_{CPU} = (nº ciclos execução + nº ciclos memory - stall) x T

nº ciclos memory-stall → nº ciclos read - stall + nº ciclos write - stall

nº ciclos read-stall = nº reads x miss rate read x miss penalty read

nº ciclos write-stall = (nº writes x miss penalty write x miss rate write) + write - buffer stalls

AMAT = hit time + miss rate x miss penalty

Memória virtual

- os endereços usados por programa referem-se a um espaço de endereçamento virtual.
- memória organizada em páginas (blocos c/ 4 KiB até 16 KiB)
- Pag da memória virtual são mapeadas para pag da memória física
- uma pag. da m.v pode residir em qualquer das pag. m.f

endereços

endereço virtual => 32 bits

nº da Pag. virtual	offset Pag
31 … 12	11 … 0

↓ muda

nº Pag. Física	offset Pag
29 … 12	11 … 0

endereço físico => 30 bits

Pag 2^{12} bytes = 4 KiB (12 offset)

Page Fault: se a Pag virtual não está em memória física, ocorre page fault. Neste caso, a pag. virtual tem de ser carregada para uma pag. física.

endereço	
pag. vir/tag	
TLB (H/M)	
Pag. table	
endereço físico	

- endereço físico = (Pag. física x tamanho Pag) + offset
- tag = endereço // tamanho Pag.

ex: tamanho da Pag = 4 KiB

$4 \times 2^{10} = 2^2 \times 2^{10} = 2^{12}$ offset

- se TLB for Hit, não é necessário ir à tabela Pag.

- miss rate:

Tabela Pag

P.V	valid	P.Física

TLB

valid	tag	nº pág

tempo acesso = seek time + latência rotacional + tempo transf. + overhead do CC

latência rotacional = ½ x 60 / rotações

ALUOp (bits)	Operação
[illegible]	[illegible]

Controlo da operação da ALU

Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

ALUOp		Funct field						Operation
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Instruction	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp0
R-format	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
lw	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
sw	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
beq	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Input or output	Signal name	R-format	lw	sw	beq
Inputs	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Outputs	RegDst	[illegible]	[illegible]	[illegible]	[illegible]
	ALUSrc	[illegible]	[illegible]	[illegible]	[illegible]
	MemtoReg	[illegible]	[illegible]	[illegible]	[illegible]
	RegWrite	[illegible]	[illegible]	[illegible]	[illegible]
	MemRead	[illegible]	[illegible]	[illegible]	[illegible]
	MemWrite	[illegible]	[illegible]	[illegible]	[illegible]
	Branch	[illegible]	[illegible]	[illegible]	[illegible]
	ALUOp1	[illegible]	[illegible]	[illegible]	[illegible]
	ALUOp0	[illegible]	[illegible]	[illegible]	[illegible]

Organização dos sinais pelo andar em que são usados

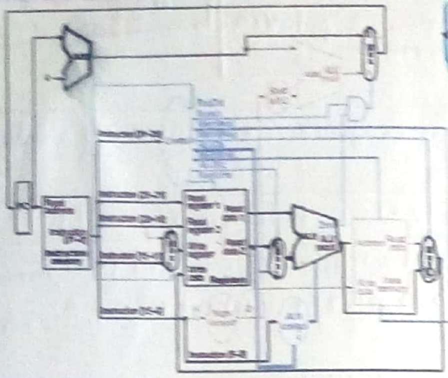
Instruction	Execution/address calculation stage control lines				Memory access stage control lines			Write-back stage control lines	
	RegDst	ALUOp1	ALUOp0	ALUSrc	Branch	Mem Read	Mem Write	Reg Write	Memto Reg
R-format	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
lw	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
sw	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
beq	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load word (lw)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (sw)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (add, sub, AND, OR, slt)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (beq)	200 ps	100 ps	200 ps			500 ps

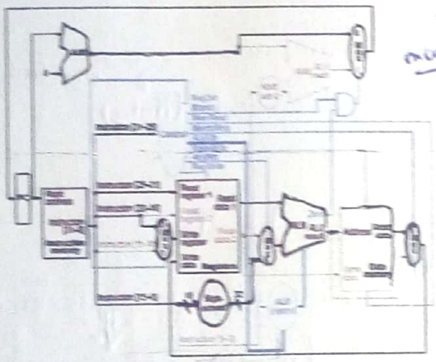
A instrução com a duração mais longa é lw

$T \geq 800\,ps \equiv f \leq 1.25\,GHz$

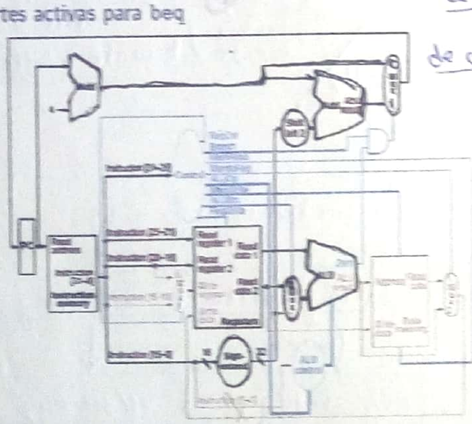
Partes activas para instruções tipo-R



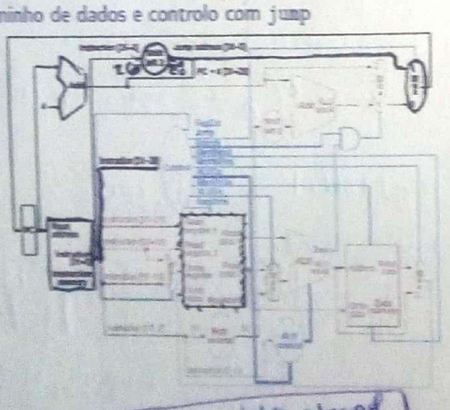
Partes activas para lw



Partes activas para beq



Caminho de dados e controlo com jump



desempenho

quanto menor for o tempo de desempenho (execução) de um programa no PC$_x$ maior (CPU) o desempenho;

$$desempenho_x = \frac{1}{temp.\ execução_x}$$

$$speedup_{x\leftarrow y} = \frac{desempenho_x}{desempenho_y} = \frac{t.\ exec._y}{t.\ exec._x} = n$$ x é n vezes mais rápido do que y; n≥1 | n<1 slowdown

$t_{CPU} = n^o\ ciclos \times duração\ ciclo$

$t_{CPU} = \frac{n^o\ ciclos}{f}$

$T = 250\,ps = 250 \times 10^{-12}\,s = 0,25 \times 10^{-9}\,s$

$f = \frac{1}{T} = \frac{1}{0,25 \times 10^{-9}} = 4 \times 10^{9}\,s^{-1} = 4 \times 10^{9}\,Hz = 4\,GHz$

$CPI = \frac{n^o\ ciclos}{n^o\ instruções}$ | $CPI_{global} = \sum_i (\%)_i \times CPI_i$ | $MIPS = \frac{n^o\ instruções}{temp.\ execução \times 10^6}$

pipeline:
- todas as instruções têm o mesmo tamanho;
- poucos formatos diferentes de instrução e/registos sempre nas mesmas posições;
- só loads/stores lidam c/ endereços;

vs. monociclo

1. duração ciclo de relógio diminui
2. tempo de execução de instrução diminui
3. tende a aumentar o tempo entre instruções, sobretudo se os andares não são perfeitamente alinhados.
4. aumenta o nº de instruções executadas por unidade tempo (throughput)

conflitos:

estruturais: quando não é possível executar uma combinação de instruções no mesmo ciclo;

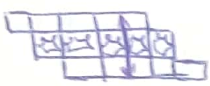
de dados: quando uma instrução necessita de um valor produzido por uma instr. anterior e ainda ñ disponível;

de controlo: necessário saber o destino de um salto condicional para poder executar a próx. inst.

1. add $s0, $t0, $t1
2. sub $t2, $s0, $t3

mem → ex

1. lw $s0, 20($t1)
2. sub $t2, $s0, $t3



WB → ex

forwarding

$$speedup = \frac{t_{não\ pipelined}}{t_{pipelined}}$$

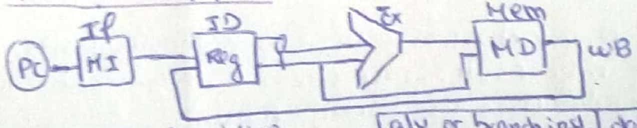
$t_{processar}$ pipelined = tempo do andar com > tempo

$t_{processar}$ ñ pipelined = somatório das (%)

$t_{instrução}$ no pipeline = 5x tempo do andar com > tempo

$$CPI = \frac{n^o\ ciclos + inst.}{instr.} (4)$$

$$IPC = \frac{n^o\ inst}{n^o\ ciclos}$$



desenrolamento do código

	alu or branch inst.	data transf. inst
loop:		

excepções (tratamento)
- interromper a execução do programa corrente
- executar o código do SO que lida c/ excepção ocorrida
- retomar a execução do código do programa ou abortá-lo.

$$ciclos\ acesso = \frac{t_{acesso}}{T}$$

$$t_{transf} = \frac{bytes\ a\ transf}{taxa\ transf}$$

$$overhead = \frac{bytes\ a\ transf}{taxa\ transf \times 10^6}$$

→ acesso discos ← 1MB = 10^6 bytes

Scanned by CamScanner