- **Tcpu**=(instruções*cpi)/f(Hz)
- **Tcpu**=instruções*CPI*T
- **Tcpu**=Nºciclos*T
- **Tcpu**=Nºciclos/f
- **Ciclos**=instruções*cpi=Tcpu*f
- **IPS**=instruções/Tcpu
- **Instruções**=ciclos/CPI
- **Ciclos**=instruções*CPI=Tcpu*f
- **f**=(instruções*cpi)/Tcpu
- **global CPI**=∑% * CPI class
- **Mips p** =instruções p/(Tcpu p *10^6)
- **CPI**=Tcpu/(instruções*T)
- **Tcpu** = instruções*CPI*T
- **Speedupx/y**=desempenhox /desempenhoy=Tcpu y/Tcpu x = n
- **Speedup depois/antes**=desempenho depois/desempenho antes=T antes/T depois= T antes/ (Tafetado/melhoria)+T não afetado
- **Tempo depois da melhoria**=(Tempo afetado pela melhoria/Valor da melhoria) + Tempo não afetado pela melhoria
- KMGTPEZY 1024 bytes
- **Picossegundos(Ps)->**Ps*10^-12 s
- **GHz->**GHz * 10^9 Hz
- **ns->**ns*10^-9 s

Partes usadas pela situação Type-R:

- PC,PC+4,Instructor Memory, Register, ALU, Mux (ALUSrc), Mux(MemToReg)

Em **Lw:**

-PC,PC+4,Instructor Memory, Register, Immgen, Mux(ALUSrc),ALU,Mux(MemToReg).

Em **beq:**

-PC,PC+4,Instructor Memory,Register,ImmGen,ALU, Mux(Branch).

-O desempenho associa-se diretamente ao tempo de execução, precisa-se do CPI

-CPI em Risc-v é 1.

-O CPI depende da implementação do processador e do programa

-**Métricas que permitem concluir desempenho de um processador->**CPI

IF -> Instruction fetch -> Memória de instruções
ID -> Instruction decode -> Banco de registos (leitura)
EX -> Execute -> ALU
MEM -> Memory access -> Memória de dados
WB -> Write back -> Banco de registos (escrita)

Tempo entre 2 instruções pipelined=Tempo entre 2 Não Instruções/Número de andares do pipeline

Instruções não pipelined vs Número de andares no pipeline: Duração do ciclo de relógio diminui I Tempo para executar uma instrução não diminui I Tende a aumentar, sobretudo se os andares do pipeline não são perfeitamente equilibrados I Todas as instruções demoram o mesmo tempo I Aumenta o número de instruções executadas por unidade de tempo (throughput)

- Num processador pipelined, o período do relógio não pode ser inferir à latência do andar que tem maior latência.
- Num processador não pipelined, o período não pode ser inferior ao máximo dos somatórios das latências das operações que são efetuadas por alguma instrução. **Tnão pipelined**>= ∑
- **CPI**=ciclos/instruções=((andares-1)+instruções)/instruções
- **Pipeline** – 4 andares
- **Tlw pipelined** = andares*Tpipelined
- **Tlw não pipelined**=Tnão pipelined lw>=Tnão pipelined
- **ciclos**=(andares-1)+instruções
- **IPC**=instruções/ciclos
- **speedup**=Texecução não pipelined/Nº andares

Context switch:

- O programa em execução pelo processador muda periodicamente
- A execução é interrompida para se dar a mudança
- A mudança de contexto consiste em guardar o contexto do programa em execução e repor o contexto do programa que vai passar a ser executado.
- A mudança é sempre controlada pelo sistema operativo.

Excepções:

Assinalam uma circunstancia excepcional, ocorrida durante o processamento, que requer atenção com urgência| Constituem um mecanismo para passar o controlo ao sistema operativo (SO)|Uma exceção pode ter origem interna ou externa ao processador|As excepções externas ao processador são também conhecidas
como interrupções (interrupts)

- Uma pipeline é uma forma de ILP. Várias instruções estão em execução em cada ciclo de relógio. Cada instrução está numa diferente da execução.
- **Pipeline scheduling dinâmico** -> Apesar de a ordem de execução das instruções poder não corresponder à ordem das instruções no programa, os efeitos visíveis das instruções tem de corresponder aos da sua execuções puramente sequencial.

- **Hit Rate** = nº de hits/nº acessos
- **Miss Rate**=nº misses/nº acessos =1-hit rate
- **nºacessos**=nºbits+nº acessos
- **Localidade Temporal** é uma posição de memoria acedida tenderá a ser acedida outra vez em breve
- **Localidade espacial** as posições de memoria perto de uma posição de memoria acedida tenderão a ser acedidas em breve
- **Palavra**=endereço/bytes por palavra
- **Bloco**=Palavra/palavra por bloco=endereço/bytes por bloco
- **Indice ou conjunto**=bloco % nº conjuntos
- **tag**=bloco/nº conjuntos
- **Tempo de CPU** = ciclos execução × duração de 1 ciclo

<u>Contando com os acessos à memória:</u>

- Tempo de CPU =(ciclos execução + ciclos memory-stall) × duração de 1 ciclo

- ciclos memory-stall = ciclos read-stall + ciclos write-stall

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

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

Tempo medio de accesso à memória (Average memory access time):

-AMAT = hit time + miss rate × miss penalty

Com caches com vários niveis:

-Miss rate global= ∏miss rate

Acesso a um disco magnético

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

- **latência rotacional**=0.5* (60/velocidade de rotação)

Desafio do paralelismo

-Tempo depois da paralelização=Tempo antes da paralelização/ Número de processadores

-Speedup=(Tempo antes da paralelização/Tempo depois da paralelização)=Número de processadores

Lei de Amdhal (no contexto da paralelização)

-Tempo depois da paralelização=(Tempo afetado pela paralelização/Número de processadores)+Tempo não afetado pela paralelização

Speedup=Tempo antes/ ((Tempo afetado/Numero de processadores)+tempo não afetado)

(**Tempo antes da paralelização** = Tempo afectado + Tempo não afectado)

Acessos atómicos à memória em RISC-V

-**lr.w rd, (rs1)** (instrução load-reserved)

-> Funciona como lw rd, 0(rs1)

-> Associa uma marca ao endereço acedido

-sc.w rd, rs2, (rs1) (instrução store-conditional)

Se for executada pela mesma thread, no mesmo processador em que foi executado o lr.w, sobre o mesmo endereço, e a marca ainda lhe está associada

-> Escreve o conteúdo de rs2 no endereço pretendido

-> Remove a marca

-> Põe o valor 0 em rd

Caso contrário

-> Não escreve nada na memória

-> Remove qualquer marca pertencente à thread

-> Põe um valor diferente de 0 em rd

Organização da memória em sistemas SMP

Uniform memory access (UMA)

-> Todos os processadores acedem igualmente a toda a memória

-> Competição no acesso à memória limita número de processadores

Non-uniform memory access (NUMA)

-> Cada processador acede a uma zona da memória com menor latência do que às outras

-> Diminui o tráfego no bus de acesso à memória

-> Permite mais processadores

-> Programas devem ter em conta diferenças nos acessos à memória