|  |
| --- |
|  |
| **Tomasulo**  • Op: Opcode  • Qj, Qk: The reservation stations producing source operands  • Vj, Vk: The values of source operands  • Busy: Indicates if reservation station is occupied | | |  | | |
| http://wiki.expertiza.ncsu.edu/images/thumb/6/6c/MESInew.jpg/400px-MESInew.jpg  **Top:** perspective from our requests  **Bottom:** perspective of others requests | | |
| http://wiki.expertiza.ncsu.edu/images/thumb/d/d0/MSInew.jpg/600px-MSInew.jpg**Left:** perspective from our requests  **Right:** perspective of others requests | | |
| https://upload.wikimedia.org/wikipedia/commons/thumb/4/4d/MOSI_Processor_Transactions.png/287px-MOSI_Processor_Transactions.png | | | | | |
| BusRdx = ownGetx = otherGetx (own if it is current processor, other if someone else).  **Left:** perspective from our requests | **Right:** perspective of others requests | | | | | |
| https://upload.wikimedia.org/wikipedia/commons/thumb/a/a0/MOESI-Zustandsdiagramm_f%C3%BCr_aktive_CPUs.png/500px-MOESI-Zustandsdiagramm_f%C3%BCr_aktive_CPUs.pnghttps://upload.wikimedia.org/wikipedia/commons/thumb/0/0a/MOESI-Zustandsdiagramm_f%C3%BCr_passive_CPUs.png/500px-MOESI-Zustandsdiagramm_f%C3%BCr_passive_CPUs.png  https://upload.wikimedia.org/wikipedia/commons/thumb/1/12/Legende_MOESI.png/220px-Legende_MOESI.png**SNOOPING = BUS OPS | Left:** perspective from our requests | **Right:** perspective of others requests | | | | | |
| **Cache Coherence:**   * MSI: Has a lot of bus access and write backs * MESI: Better for bus accesses, but has a lot of write backs * MOSI: Better for write backs, but has a lot of bus access * MOESI: Better for both, but has larger state machine overhead | | | |  | |
| **Data Dependencies:**   * True dependency: RAW (Read after Write) * Anti-Dependency: WAR (Write after Read) * Output dependency: WAW (Write after Write) * Control dependency: an instruction execution is dependent of a branch | | **Branch Predictors**   * Correlating predictor: uses history of branches to index into a table, which corresponds to the branches address. This will have 1 or 2 bits saturating counter, which will predict. (Two images on left) * Tournament predictor: a predictor that chooses between two other predictors to use. One is usually global and one is usually local. (Image on right). | | | |
| https://upload.wikimedia.org/wikipedia/commons/thumb/c/ce/Two-level_branch_prediction.svg/420px-Two-level_branch_prediction.svg.pnghttp://www-ee.eng.hawaii.edu/~tep/EE461/Notes/ILP/Figs/predict_corr.gifhttps://scontent.xx.fbcdn.net/v/t34.0-12/17793426_1486341448083588_1338754619_n.png?oh=d761fa3a88e087aedaa27e43d0130fdc&oe=58E59112 | | | | | |
| Performance:   * **Response time (latency)** refers to how long a job takes to execute * **Throughput** – how many jobs can the machine complete in a minute * ; * Cycle time – seconds per cycle | clock rate – cycles per second | **CPI** – cycles per instruction * **Single Cycle CPU** * Cycle time is determined by the critical path of the longest instruction * One instruction per cycle is executed * **Multi-Cycle CPU** * Cycle time is determined by the longest * Instructions are broke up in to smaller parts or **State**. * MIPS – Millions of instructions per second * Higher for program using simple instructions | | | | | Pipeline: How large  ts = stage delay | Ni = No of instructions | T = time of execution  to = latch delay | a = avg degree of superscalar processing  p = No pipeline stages | tp = time between pipe stages?  T = Tbz + Tnbz  Ts = tp/p + to  Tbz(busy) = NiTs =Ni(tp/p + to) = (Ni/a)(tp/p + to)  Tnbz = NhTpipe = Nh(tp + pto) = Nh(tp + pto)(1/Nh ([summation of all Nh]Bh))  T/Ni (time per instruction) = (1/Ni)[Tbz + Tnbz]  https://scontent.xx.fbcdn.net/v/t34.0-12/17793261_1319333451481199_1768174323_n.jpg?oh=9ecabf3bf6b0ac58177774b1dc5a6961&oe=58E6C4B7 |
| Software ILP:   * Instruction scheduling: reordering instructions to reduce dependencies * Loop unrolling: unrolling a loop, allows multi-issue processing. * Prologue and epilogue might be needed * Register renaming: removes anti-dependencies * Software pipelining: achieves similar effect of loop unrolling without all of the code expansion. | | | | |
|  | | | | | |