Instruction Level Parallelism

J. Daniel García Sánchez (coordinator)
David Expósito Singh
Javier García Blas

Computer Architecture
ARCOS Group
Computer Science and Engineering Department
University Carlos III of Madrid

1. Module structure

This module is structured in two lessons:

1. **Introduction to instruction level parallelism**. Introduces the concept of pipelined architecture, using as a use case the design of five stages basic processor. It also introduces the concept of hazard and classifies the different kinds of hazards. Finally, it introduces the idea of multicycle operations.

2. **Instruction level parallelism exploitation**. It introduces several aspects of pipelined architectures exploitation as instruction scheduling and loop unrolling. It also gives special attention to branch prediction techniques. Other aspects covered include multiple issue techniques and limits to instruction level parallelism. Finally, thread level parallelism is presented.

2. Introduction to instruction level parallelism

This lecture has the following general structure:

1. Introduction to pipelining.

2. Hazards.

3. Multicycle operations.

2.1. Introduction to pipelining

Pipelining is an implementation technique based on breaking down each instruction execution into multiple stages. This technique does not affect (or only minimally) to each instruction latency. However, it allows to increase throughput, as it allows to finish (ideally) an instruction every cycle.

In the materials details are given for the design of a five stages simplified pipelined processor:

- *Instruction Fetch* – IF. Instruction read and update program counter register.

- (*Instruction Decode* – ID. Instruction decode, registers read, sign extension in offsets, and possible branch address computation.
- *Execution* – EX. ALU operation on registers and computation of effective branch address.
- *Memory* – M. Memory read or write.
- *Write-back* – WB. Result write on register file.

2.2. Hazards

A hazard is a situation preventing next instruction to start in the expected clock cycle. These situations reduce performance of pipelined architectures. Hazards may be classified in three different kinds: structural hazards, data hazards and control hazards. The most simple approach (and less efficient) is to stall the instruction flow until the hazard has been eliminated.

Structural hazards happen when hardware cannot support all possible instruction sequences. This happens if two stages need to make use of the same hardware resource. Most common reasons are the presence of functional units that are not fully pipelined or functional units that are not fully duplicated. In general, structural hazard can be avoided during design but the increase cost of resulting hardware.

A data hazard happens when pipelining modifies the read/write access order to operands. Data hazards can be from three different kinds: RAW (*Read After Write*), WAR (*Write After Read*), and WAW (*Write After Write*). From those three, only RAW hazards may happen in MIPS type five stages architecture. RAW data hazards can be solved sometimes through the use of the forwarding technique.

A control hazard happens in a branch instruction, when the value to be used to make the branch decision is still unknown. Control hazards can be solved at compile time (static solutions) or at runtime (dynamic solutions).

Static solutions from control hazards may range from pipeline freezing or fixed prediction (always predict to taken or not taken), to using branching with delay slots.

Dynamic solutions may use a *Branch History Table* (BHT) and a state machine to perform prediction. Thus, there is a state machine associated to each entry in the table.

2.3. Multicycle operations

Allocating a single cycle for floating point operations requires an extremely long clock cycle or using a very complex floating point logic (causing high use of design resources). Alternatively, the floating point unit itself may be pipelined, so that instructions require multiple cycles for each execution stage.

3. Instruction Level Parallelism exploitation

This lesson has the following general structure:

1. Compilation techniques and ILP.
2. Branch prediction advanced techniques.
3. Introduction to dynamic scheduling.
4. Speculation.
5. Multiple issue techniques.
6. Limits of ILP.
7. Thread level parallelism.

3.1. Compilation techniques and ILP

Instruction level parallelism may be applied inside of each basic block (instruction sequence without branches). However, basic block average length is between 3 and 6 instructions which highly reduces the possibilities to take advantage from the situation. A technique to improve ILP utilization within a loop is loop unrolling.

Loop unrolling consists on interleaving execution of instructions from several loop iterations. As they are unrelated instructions they do not generate dependencies and allow for a better utilization of the pipelined architecture.

3.2. Advanced branch prediction techniques

Both loop unrolling and branch prediction allow to decrease branch impact in pipelined architectures utilization. Other alternatives are advanced branch predictions: correlated branch predictors and tournament predictors.

Correlated prediction keeps a history of the last branches to select between several predictors.

Tournament predictors combine a global predictor and a local predictor and make use of a selector to choose among both predictors.

3.3. Introduction to dynamic scheduling

Dynamic scheduling is based on instruction reordering during execution to decrease the number of stalls. Although this implies a higher hardware complexity, it also allows a higher independence from the concrete hardware architecture and allows to mange unknown dependencies at compile time as well as unpredictable delays. However, by allowing out of order execution and out of order finalization WAR and WAW hazards may be introduced.

There are two dynamic scheduling techniques: *scoreboard* and *Tomasulo* algorithm. The *scoreboard* technique stalls issued instructions until there are enough resources and no data hazards may happen. The *Tomasulo* algorithm removes WAR and WAW dependencies by renaming registers.

3.4. Speculation

To improve results in branch prediction, next step is to speculate on branch results and execute assuming that speculation was right. This requires a mechanism for handling those cases in which speculation was wrong. Basic components of speculation are branch dynamic prediction, speculation, and dynamic scheduling. To achieve this, instruction finalization is separated from passing the result from the instruction producing it to the instruction using it.

3.5. Multiple issue techniques

Multiple issue techniques are based in issuing more than one instruction per cycle with the goal to achieve a CPI below one (which is equivalent to more than one instruction per cycle). This leads to three solutions: using superscalar processors statically scheduled, superscalar processors dynamically scheduled, VLIW processors.

3.6. Limits of ILP

To evaluate limits of instruction level parallelism an ideal pipelined processor may be defined. While available ILP may be high for some applications, from a practical viewpoint it may be considered to have a value between 3 and 6.

3.7. Thread level parallelism

In a mlti-threaded processor different (hardware) execution threads share functional units in the processor, which makes replication necessary. Three approached may be used: fine grained multi-threading, coarse-grained multi-threading, and simultaneous multi-threading.

In fine grained multi-threading we switch among threads in each instruction. In coarse grained multi-threading there is switching only when there are long stalls (as L2 cache misses). In simultaneous multi-threading the execution of instructions from several threads at the same time is allowed.