Computer Architecture

Quiz 3 → Guide to Reading Chapter 3

April 10, 2021

Abstract

The underlying ideas of Chapter 3 are parallelism at the instruction level

1 Vocabulary

We gain power over ideas which we can associate with a word. Instead of being fleeting collections of activations of neurons, alone, named ideas can be recalled, communicated, built upon.

Thus, we need vocabulary. Provide a definition for:

- basic block – a straight-line code sequence that has no branches in except to the entry and no branches out with the only exception being at the exit.

- stall (in pipeline) – a delay in execution of a particular instruction in order to resolve a hazard

-

- name dependence – occurs when two instructions use the same memory location or register (a name), but there isn't a flow of data between the instructions associated with the name

- register pressure – the number of hard registers that are needed to store values in pseudo registers at a given point in the compilation process which leads to more spills and reloads.

- issuing, i.e., decoding an instruction (see p. 194) – Issuing an instruction is the first part of the two-stage pipeline and requires decoding instructions, and checking for structural hazards. The decoding of an instruction allows the CPU to determine what instruction is to be performed in order to fetch the correct number of operands in order to perform the instruction. Structural hazards occur when more than, or exactly two, instructions already in a pipeline require the same resource.

-

- coarse-grained multithreading – a thread issues instructions until thread issuing stops, known as stalling. Coarse grained multi-threading switches threads when a costly stall happens, and the next thread starts issuing the instructions. At this point, a cycle is lost due to the thread switching.

- fine-grained multithreading – switches between threads on each clock cycle which causes the instruction execution from multiple threads to be interleaved.

- simultaneous multithreading – a variation of fine-grained multithreading that arises naturally when fine-grained multithreading is implemented on top of a multiple issue, dynamically scheduled processor. It uses thread-level parallelism to hide long-latency events in a processor which increases the usage of functional units.

-

- single issue pipeline – for every clock cycle, there is exactly one instruction. The pipeline gets shifted down and a new instruction is read from memory.

- very long instruction word processor – issue a fixed number of instructions that are either formatted as one large instruction or as a fixed instruction packet with parallelism explicitly specified by the instruction.

2 Relationships

Describe some relationship between the two terms, or explain why they are not related:

- branch (conditional or not) vs. higher level language flow control – these two are related. Branch instructions are used to implement control flow in higher level languages for program loops and conditionals.

-

- structural hazard vs. data hazard vs. control hazard – despite these three all being hazards, they are not related conceptually. Structural hazards occur when hardware cannot support certain combinations of instructions, data hazards happen when an instruction depends on the result of the previous instruction in the pipeline, and control hazards are caused by delays between fetching instructions and decisions regarding changes in control flow.

- loop level parallelism vs. data level parallelism – These two concepts are closely related. Loop level parallelism is concerned with parallel tasks from within loops. Data level parallelism refers to the execution of the same task concurrently across multiple processors.

- output dependence vs. register renaming – Aside from both involving registers, these two are not related. Output dependence occurs when two instructions write the same register or memory location whereas register renaming can rename a specific register either statistically or dynamically by the hardware.

-

- static scheduling vs. dynamic scheduling – These two are not directly related. Static scheduling predetermines the order, or way, in which the threads/processes are executing code whereas dynamic scheduling for threads is done by the operating system based on a potentially methodical algorithm on the OS level.