**Not Assigned in Spring 2021**

**13 – Parallelism and Pipelining**

**Activities**

COMP256 – Computing Abstractions

Dickinson College

Spring 2021

Prof. Grant Braught

**Name:**

Today’s class built on the prior one and introduced some additional performance improvements based upon new ideas for CPU design including instruction pipelining, super scalar processors and multicore CPUs. While the implementation of each of these is mind-bendingly complex, underneath it all is really just a collection of more complicated logic circuits. But rather than trying to understand these topics at that level, we have considered them at a higher level of abstraction using metaphors. In today’s activities you will revisit each of these designs introduced and explore their operation in greater depth.

Though it is not required viewing, you might gain additional perspective by having another reasonably high-level take on the performance improvements that we have seen as well as several others. Carrie Anne’s video *Advanced CPU Designs*does this as a part of the Crash Course Computer Science series.

* <https://www.youtube.com/watch?v=rtAlC5J1U40> (12:22)

**Instruction Cycle Pipelining:**

1. Let’s investigate how much pipelining can speed up a processor and what that means in terms of today’s computers. In our example, the instruction pipeline was broken into three stages: (F)etch, (D)ecode and (E)xecute. Let’s assume that each of these stages takes 1 unit of time (e.g. one tick of the system clock). Thus, executing an instruction will take 3 time units or three ticks, one for fetch, one for decode and one for execute.

a. Assuming strictly sequential execution of instructions (i.e. no pipelining) after how many time units would a sequence of 5 instructions complete?

b. Complete the table below showing the time at which each instruction would begin to be fetched and the time at which its execution would be complete if executed with our 3-stage pipeline. Assume that the sequence of instructions contains no hazards.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  | **Instruction Number** | **Fetch Start Time** | **Execute End Time** |  |
|  | **1** | 0 | 3 |  |
|  | **2** | 1 |  |  |
|  | **3** |  |  |  |
|  | **4** |  |  |  |
|  | **5** |  |  |  |
|  |  |  |  |  |

c. Let’s now generalize parts a and b to sequences of instructions of different lengths. Assuming the same three-part fetch, decode, execute cycle, use your results from a and b, write expressions for each of the following:

i. The time tsequential for the number of time units required for the sequential (non-pipelined) execution of a sequence of n instructions.

tsequential =

ii. The time tpipelined for the number of time units required for the execution of a sequence of n instructions.

tpipelined =

iii. If we consider the case in part ii where the number of instructions (n) becomes large then the small constant that is added will become insignificant. In cases like this, engineers will often simplify by omitting the small constant. Rewrite the expression for tpipelined without the small constant term (i.e. include just the dependence on n).

tpipelined =

iv. The (S)peedup attributable to a design improvement is computed as the time required to execute a sequence of instruction without the improvement divided by the time required with the improvement. Give an expression for the speedup, S3p, for our three-stage pipeline. Be sure to simplify your expression.

S3p =

d. Different processors will have been designed with instruction pipelines that have differing numbers of stages in them. Notice that the expressions for tsequential and tpipelined are specific to the number of stages in our pipeline. Generalize these expressions and to a pipeline with k stages.

tsequential =

tpipelined =

e. Given your answer to part d, what is the speedup for pipeline with k stages?

Skp =

f. Now what does that mean in practice? Using your favorite search engine, do some research to determine how many stages are in the instruction pipeline of a recent Intel CPU design.

i. What is the name of the Intel processor that you found?

ii. How many stages are in its pipeline?

iii. Neglecting hazards, approximately how much faster would the pipelined version of this machine be than one that is equivalent except without pipelining?

It is worth noting that even if we could avoid all hazards the speedup would not be quite what you determined in question #1.f.iii above. The extra hardware required to synchronize the stages means that each step will take a little bit longer than if it was not pipelined. Thus, the execution of any given instruction actually takes a little longer on a pipelined system than it would on a non-pipelined system. However, because multiple instructions are overlapped the overall performance is still significantly higher than without pipelining.

**The Complexity/Cost vs Performance Tradeoff:**

As just mentioned, a pipelined instruction cycle requires a good bit of extra hardware to keep everything synchronized and to manage and prevent hazards. All of this extra hardware adds both complexity and cost to the system. In return for that added complexity and cost the system is able to provide greater performance. This is just one example of the Complexity/Cost vs Performance trade off. It is very frequently the case that greater performance can be achieved by adding complexity and cost, or conversely cost and complexity can be reduced by sacrificing performance.

2. Briefly explain how another design enhancement that we have seen recently is an example of the Complexity/Cost vs Performance tradeoff.

**Pipeline Hazards:**

We have seen two types of hazards that can impact the efficiency of an instruction pipeline. The following questions will explore these a little more and investigate a third type of hazard.

*Resource Hazards*

3. In a sentence of your own words, describe what causes a resource hazard.

4. Consider the K&S machine language program shown below. Note: I have given the operations that carry this out, but I have not translated them to the machine language instructions. This could be done easily – though tediously – and is unnecessary in this case.

|  |  |  |
| --- | --- | --- |
| Memory Address | Operation or  Value | Comment |
| 00000 (0) | R0 ← MM[7] |  |
| 00001 (1) | R2 ← R0 + R0 | *(2\*MM[7])* |
| 00010 (2) | MM[8] ← R2 |  |
| 00011 (3) | R2 ← R2 + R2 | *(4\*MM[7])* |
| 00100 (4) | R2 ← R2 + R2 | *(8\*MM[7])* |
| 00101 (5) | MM[9] ← R2 |  |
| 00110 (6) | Halt |  |

If we assume our three-stage instruction pipeline, then this program contains two **resource hazards**. Indicate which instructions create each of the resource hazards and briefly justify your answer.

*Control Hazards:*

5. In a sentence of your own words, describe what causes a control hazard.

6. Consider the K&S machine language program shown below.

|  |  |  |
| --- | --- | --- |
| Memory Address | Operation or  Value | Comment |
| 00000 (0) | R0 ← MM[7] |  |
| 00001 (1) | R0 ← R0 & R0 |  |
| 00010 (2) | If Negative, Branch 6 |  |
| 00011 (3) | R1 ← R1 – R1 | *(R1←0)* |
| 00100 (4) | R0 ← R1 – R0 | *(R0← -MM[7])* |
| 00101 (5) | MM[7] ← R0 | *(MM[7]= -MM[7])* |
| 00110 (6) | Halt |  |

a. The above program contains a control hazard. Indicate the instruction that creates the control hazard and the two additional instructions whose execution may be impacted by the hazard.

b. Notice that control hazards do not always cause a problem.

i. If the value in MM[7] is negative, what needs to happen to the additional instructions you identified in part a?

ii. If the value in MM[7] is positive, what needs to happen to the additional instructions you identified in part a?

**The Hazards of a Longer Pipeline:**

In designing the next revolutionary version of the K&S computer the company has decided to expand the instruction pipeline to 5 stages. They do this by dividing the execute stage into three separate operations giving a pipeline with the following stages:

* **(F)**etch - Same as before. Move the instruction in to the instruction register. Or in shorthand IR←MM[PC].
* **(D)**ecode – Same as before. The decode unit figures out what the instruction is in the instruction register and creates the micro-instruction to configure the data path.
* **(R)**ead – Read the value either from memory to memory bus, or from the registers onto the A and B bus as needed by the instruction. For example:
  + For an operation like R0←MM[12] the R stage will place the value in MM[12] on the main memory bus.
  + For an operation like R0←R1+R2 the R stage will place the values in R1 and R2 onto the A and B busses.
* **(A)**LU Operation - The ALU operation is performed and the result flags (Negative, Zero, etc) are set.
* **(W)**rite – Write either to memory via the memory bus or to the destination register via the C bus as needed by the instruction. For example:
  + For an operation like MM[12]←R0 the W stage will place the ALU result into MM[12] via the main memory bus.
  + For an operation like R0←R1+R2 the W stage will place the ALU result into R0 via the C bus.

This longer pipeline creates a new type of hazard, called a *data hazard*, that we have not yet encountered. The next few questions guide you through identifying a data hazard and then introduces one of the common strategies for mitigating them.

7. Consider the following two operations that are part of a program being executed on the new 5-stage pipeline version of the K&S.



We’ll assume that the registers begin with the following values:

* R0 = 0
* R1 = 5

a. Without considering pipelining, what value should R2 contain after executing these two instructions?

b. In which stage(F,D,R,A,W) will the first instruction write the new value of R0 into the register?

c. In which stage(F,D,R,A,W) will the second instruction read the values of R0 and R1 from the registers for the operation? Does this happen before or after the write operation in part b?

d. Based on you answers to b and c, what value will be in R0 when it is read from the registers by the second instruction?

e. Given your answer to part d, what value would be written into R2 when the second instruction completes?

f. You should have gotten different answers in parts a and e. The incorrect result in part 3 is the result of a data hazard and would need to be avoided. Summarize in sentence or two what causes a data hazard.

8. One technique that is often used for mitigating data hazards is called *out of order execution*.

a. Consider the two operations shown below:



Does the order in which these two operations are executed matter? That is, will the values in R3 and R2 be the same if the operations are performed in either order? Briefly explain why or why not.

b. Now, consider the two operations from question #7 above again. Assuming for now that they are not affected by a data hazard, does the order in which they are performed matter? Briefly explain why or why not.

Based on parts a and b, it should be clear that order in which some instructions are executed matters, while others can be reordered without changing the overall result of the computation. Sort of like getting dressed, you can put your shirt or trousers on first and the result is the same. But if you put your trousers on before your underwear the result is quite different. The idea behind out of order execution is to shuffle the order of instructions for which the order does not matter so that we avoid data hazards.

c. Consider the four operations shown below.



Briefly describe the data hazard that exists in the above sequence of operations.

d. Rewrite the operations from part c into a new order in the table below so that the overall result of the computation is the same, but that the data hazard is removed.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | | | | | | | |  |
|  | **Operation** | **Pipeline Stage** | | | | | | | |  |
|  |  | F | D | R | A | W |  |  |  |  |
|  |  |  | F | D | R | A | W |  |  |  |
|  |  |  |  | F | D | R | A | W |  |  |
|  |  |  |  |  | F | D | R | A | W |  |
|  |  |  |  |  |  |  |  |  |  |  |

**Multicore Processors:**

We briefly discussed the multicore processor below in today’s class:

A picture containing parking, side, street, parked

Description automatically generated

The following questions dig a little deeper into the operation of multicore processors.

9. In a multicore processor each processor core is essentially a full CPU with its own registers, ALU and control unit. The control unit in each core of course has its own program counter and its own instruction register. Thus, as each core goes through its fetch/decode/execute cycle it can be fetching instructions from different parts of the main memory. Thus, the different cores could be running different parts of the same program or each core could be running a different program. Use these ideas to explain why it is advantageous for each core to have its own independent L2 cache. Hint: Remember that cache works based on the principle of locality.

10. From your answer to question 9, it should not be that surprising that each core has its own independent L1 cache as well. This is just a smaller faster more expensive cache that is located physically closer to the registers than the L2 cache, and thus can be accessed even faster. But the L1 cache for each processor has also been divided into separate caches for data and instructions. These caches behave the same as any other (holding copies of things we are likely to need soon), except that instruction cache will only hold program instructions and the data cache will only hold data used by the program. Explain why it is advantageous to have separate caches for instructions and data? Hint: If its about cache… it’s about locality!!

So based on questions 9 and 10 it makes sense that each core has its own independent caches. So then why is the L3 cache is shared between the cores? The answer is that while multiple cores each with their own cache works great for reading data, there are issues that must be dealt with when one core changes a piece of data that may also be being used by another core. This is called the problem of *cache coherency* and having a L3 cache that is shared by all of the cores helps with it. The next question explores the cache coherency problem.

11. For this problem, imagine two cores A and B, and that these cores are running different parts of the same program. Because A and B are running the same program they will have access to the same data. For example, if core A reads from memory address 25 and gets the value 5, then when core B reads from address 25 it should also get 5.

a. Imagine core A reads from memory address 25 and gets 5. How many copies of that 5 will exist in the system and where will they be? Briefly justify your answer. Hint: Remember that cache levels are checked in order from fastest to slowest and every miss ultimately results in having the data copied into that cache.

b. A little while later B reads from address 25 and gets 5.

i. Where would this value have been retrieved from: B’s L1 cache? B’s L2 cache? The L3 cache? Main memory? Briefly explain your answer.

ii. How many copies of the 5 from address 25 now exist in the system? Where are they?

c. Imagine now, that A changes the value at address 25 to be 10 and that the value is updated in A’s L1 data cache. This is great for A, if it reads address 25 again, it will have a hit in its L1 data cache and get the value 10. But what value will B get if it reads address 25 again? Where would it get this value?

d. Briefly propose an idea for what could be done when A changes the value at address 25 that will ensure that when B reads address 25 it obtains the proper value.

The solution you came up with in part d is one approach to managing cache coherency. There are a number of different approaches that have been implemented in different machines at different times. Covering them is beyond the scope of what I want to do here. Rather, you should just be aware of the issue of cache coherency at this point and have an idea of what would need to do to address it. It is not required, but if you are curious there are a few good videos on Cache Coherence from the Georgia Tech High Performance Computer Architectures course:

* *Cache Coherence Problem:* <https://www.youtube.com/watch?v=TMJj015C93A> (3:21)
* *How to Get Cache Coherence:* <https://www.youtube.com/watch?v=In8RZg345pM> (5:18)

**Final Thoughts:**

We can now see that even thinking about the fetch/decode/execute cycle is an abstraction. If we think about instructions being fetched, decoded and executed one-by-one, as is done by the Knob & Switch computer, we have a way of understanding what the result of running a machine language program will be. Generally that is the relevant information. Being able to write or study a machine language program and know what it will do. What is not relevant at that level of abstraction of is exactly how that program is actually executed. It may be run though a 3-stage or a 15-stage pipeline. Its instructions may be run out of order to minimize data hazards. Or multiple instructions may be run in parallel on multiple functional units in a super scalar processor, or on multiple processor cores. On a modern machine, it’s is likely all of those! It is really quite astonishing to imagine all of that happening each time you run a program.

**Above and Beyond:**

It is not required, but if thinking about design improvements like these is something you find fascinating you might enjoy reading Modern Processors: A 90-Minute Guide! by Jason Robert Carry Patterson of Lighterra. It discusses many of the biggest ideas in processor design and is, though not an easy read, written at a level that is pretty accessible.

* <http://www.lighterra.com/papers/modernmicroprocessors/>

Optional: To help me improve and scope these activities for future semesters please consider providing the following feedback.

a. Approximately how much time did you spend on this activity outside of class time?

b. Please comment on any particular challenges you faced in completing this activity.