Dynamic Scheduling With Real time requirements

Adijat Ajoke Sulaimon
Dynamic Scheduling with Real-Time Requirements
Hochschule Hamm Lippstadt
Email: Adijat-ajoke.sulaimon@stud.hshl.de

***Abstract*—A real-time task has very specific requirements and is plagued by deadlines, predictability, reliability, determinism, and quality of service. A real-time system is dynamic and therefore requires a scheduling strategy that matches it characteristics. In recent times, the hardware and the software tasks are scheduled separately or co-scheduled in re-configurable systems, this results in low performance and increase it's power consumption. This paper is focused on scheduling strategies for the re-use and pipelining of hardware tasks to reduce power consumption and increase performance.**

I. Introduction

The constant purpose of improving performance in real time systems has led to a broad acceptance of architectures in which complex and specialized hardware accelerators play a vital part in delivering fast leap in performance and energy efficiency. However, the constant effort of architectural heterogeneity and system improved efficiency comes at the expense of the agility of hardware/software development. High-level synthesis (HLS) has surfaced as a promising design methodology to facilitate modeling of hardware.

I have read several publications and articles in order to have broad knowledge of this topic, after which I carefully selected the papers that have connections and relevance to my proposed research. I compared, reviewed and evaluated with examples and case studies, different articles on dynamic scheduling in high level synthesis. To have an in-depth understanding of dynamic scheduling, I explained pipelining in details and with samples to give my reader a good understanding of the topic. Section 2 of this paper is mainly about dynamic scheduling with real-time requirements using tomasulo'S algorithm. Section 3 contains pipelining. Section 4 Concludes the paper with discussion and conclusion.

II. Dynamic Scheduling with Real-Time Requirements

According to Moore's Law By CARLA TARDI in MARGUERITA CHENG 2022 "The number of transistors on a microchip doubles every two years, though the cost of computers is halved" It is always like a compulsion to miniaturize components in embedded systems. To minimize power consumption and reduce heat, it is necessary to reduce the size of the transistor or processor. The most important one is the need to reduce chips to fit into single wafer. In Dynamic scheduling, the hardware chooses the task to execute as opposed to the computer making the decision as done in statically scheduled machine. Therefore, the processor is carrying out instructions out of order. Tasks do not execute based on the order at which they appear in dynamic scheduling, execution is usually based on the available resources and the requested functional units. Codes do not have to be recompiled to run accurately in dynamic scheduling, therefore this makes it versatile.

A scheduler dynamic if it makes its scheduling resolutions at run time and selects one ready task. They are called dynamic because they are flexible and easily adaptable according to the task scenario. The main idea behind dynamic scheduling and why it is used is because it allows instructions to execute out of order. They are focused on the contemporary task requests. Dynamic Scheduling is a method used by the hardware to rearrange the execution of instructions to reduce stalls and still maintain data flow and exception behaviour. There are a set of instructions referred to as the instruction window. The hardware considers these instructions and then tries to reschedule the execution their instructions in accordance to their operation availability. The hardware decides when and which of the instructions will move to another stage from their current stage. In dynamic scheduling, there is a distinguish between when an instruction begins execution and when it completes execution, the instruction is in execution between the two times. Multiple instructions to be in execution at the same time and this is the major advantage of dynamic scheduling. The dynamic scheduler initiates renaming register and abolishes WAW (write-after-write) and WAR(write-after-read) hazards in hardware. In a dynamically scheduled pipeline, all instructions can be stalled or bypass each other (read operands) but they have to first pass through the issue stage in order (in-order issue) after which they enter execution out of order. There are two ways of implementing dynamic scheduling but this paper will focus on Tomasulo's algorithm.

There are some advantages of dynamic scheduling and they are as follows:

- It makes the compiler simple
- A code compiled for one pipeline can run efficiently on another pipeline with the help of dynamic scheduling.
- When dependencies are unknown at compile time, dynamic scheduling handles the cases.
- It helps with hardware speculation which is a technique built on dynamic scheduling with significant performance advantages.

Like the figure 1 above, a pipelined instruction is like a road with one lane, if the car in front stop, all cars behind it will have to stop. With dynamic scheduling, just like the second diagram in the figure, it has multi lanes, which means if the

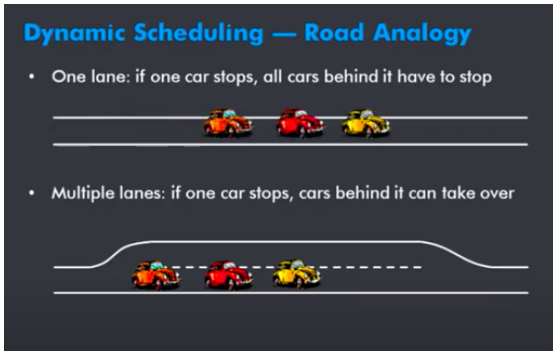

Figure 1. Road analogy of Dynamic scheduling

first car stop, the other cars can keep moving by taking other lanes.

A. *Tomasulo's algorithm*

Tomasulo received an important award called Eckert-Mauchly award in 1997 for this spectacular algorithm. Tomasulo'S algorithm was used in IBM 360/91 FPU way before caches were invented. All modern processors are based on Tomasulo's algorithm which means this algorithm is as important now as it was in 1996.

Let us look at the below instructions.

DIV.D F0,F2,F4

ADD.D F10, F0, F8

SUB.D F12, F8, F14

When the ADD.D is stalled, we want the SUB.D to proceed, but we need to get the ADD.D out of the way to do this. To achieve this, we need to buffer the ADD.D. These buffers are called 'Reservation stations' in Tomasulo's algorithm. The reservation station needs to be informed when an instruction is buffered in a reservation station and it's operand become available so that the stored instruction can proceed. The results gotten are then broadcasted on common data bus (CDB).

There are 3 main steps used by a Dynamic scheduler. They are:

• Issue • Execute • Write result

The issue step has processes such as:-

1) Obtain next order from FIFO queue
2) If reservation station (RS) is available, issue the order to the next reservation station that has operand value available.
3) If reservation station is not available, the instruction will stall because it has become a structural hazard.
4) The successive orders cannot be issued if order is not issued earlier.
5) The orders will wait in the reservation stations looking for operands at common data buses (CDBs) if operand values are unavailable.

For the execute step, the processes are:-

1) Store operand in any RS waiting for it when it becomes available.
2) The orders are executed when all operands are ready by the unit that is functional.
3) Loads and store are sustained and maintained in program instruction through an effectual address.
4) Until all branches advanced in the program have been successfully completed, no instruction will be allowed to start execution.

For the write result step,

1) Result will be written on common data bus into reservation stations and store buffers.
2) Stores must wait for value and address to be received.

The data structures maintained by the dynamic scheduler are mainly three. They are (1) The reservation station (2) A register result data structure that modifies the register (3) An instruction status data structure.

Each reservation station consist of seven fields. Namely:

1) Operation to perform (OP)
2) RS1: This contains the identifier of the reservation station that would produce the first operand value. Value 0 indicates that the source operand is available
3) RS1: This field contains the identifier of the reservation station that would produce the second operand value
4) Value1: This is the value of the first operand
5) Value2: This is the value of the second operand
6) The field contains the immediate or address. Any immediate operand the instruction has is stored here
7) Busy: This field indicates that the RS is occupied or not occupied

op	RS1	RS2	Val1	Val2	Imm/ addr	busy
add	Mul2	0	n/a	12.55	n/a	1

Figure 2. An example of the Tomasulo's dynamic scheduler

Figure 2 above is an example to explain the reservation station structure. If the instruction is an ADD and the first operand is being produced by MUl 2, and the second value operand is available in the register. RS2 is 0 as shown in the figure. Value1 is not applicable since it is being produced by Mul2 and Value2 is 12.55 for example. The immediate address field is not applicable and the busy field is set to indicate that the reservation station is occupied.

To track whether an operand value is available in a register or it's currently being produced, each register has an additional field RS. This field contains the identifier of the reservation station that would produce the result that should be stored in this register. This is blank or zero if no currently active instruction computes the result that should for this register. An example is figure 3 above, the RS field of the 1st and 3rd register are blank, which means that they currently contain valid values. The second register is currently being produced by reservation station Mul1 and the forth register is currently being produced by the reservation station Add2.

Figure 3. An example to verify if an operand is available in a register

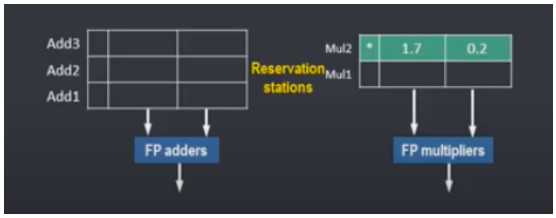

Figure 4. An example of Execute Floating Point Instruction

The figure 4 above is a simple example of instruction execution, the multiplier instruction is ready since both it's operands are available. It would therefore execute on one of it's floating point multipliers and produce the value 0.85. Finally, execution proceeds to the last stage which is 'Write result'. When the functional unit has produced the result, it is written on the CBD and from there to any reservation station and register waiting on the result.

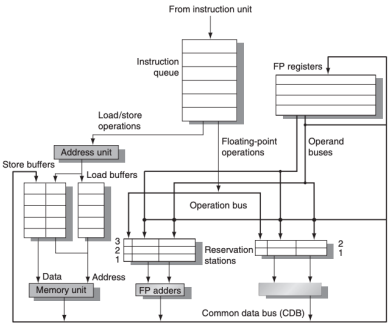

Figure 5. An example of the Tomasulo's dynamic scheduler

Figure 5 above is the organization of the Tomasulo's dynamic scheduler. according to the figure, the process starts from the instruction unit where the instructions are taken from the instruction queue and issued. The instructions are decoded and allocated an reserve station entry at the issue stage. If the reserve station did not buffer the operands if available, the reserve station entry marks the pending RS value in the Q field. Finally, the results go to the right register dictated by the register result data structure and the pending reserve stations, by passing through the CBD.

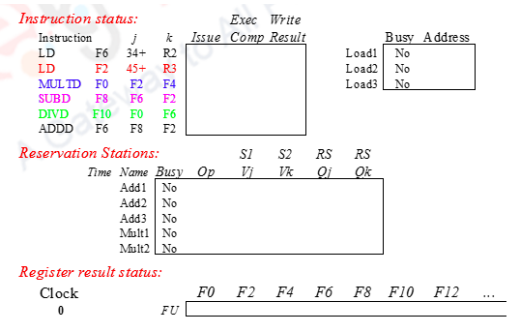

Figure 6. An example of Execute Floating Point Instruction

Figure 6 is an example of how dynamic scheduling happens for a sequence of instructions. j is the instruction, It is a basic block that has no branches. It shows the sequence of instructions reviewed with the three data structures maintained. Three Add reservation stations, two Mul reservation stations and three load buffers are available. the Add buffer is assumed to have a latency of 2, Mul has 10 and Div 40. Load 1 is issued during the first clock cycle. The allocation of the load buffer entry changed it's status to busy while the contents of R2 which is needed for the effective address calculation and the value 34 is stored in the buffer. The register shows a result proving that register F6 is to be written by the instruction spotted by load buffer 1.

III. Pipelining

The idea of pipelining is very simple. Pipelining is an implementation technique used in organizing parallel activity in a system. A good example of pipelining is the washing machine. If you place dirty clothes in the washer, as soon as the washer is finished, you load it in the dryer and then load the washer with another set of dirty clothes. You continue this steps which are called stages in pipelining. We save time by running tasks concurrently if there are resources available. Pipelining improves throughput and would not decrease the tine to complete a task as in the case of the laundry. pipelining only saves time when there are many loads or sets of laundry to be done, which means increase in throughput reduces the total time to complete the whole laundry.

The four items in figure 7 represent the washing machine, dryer,folder and the drawer for storing of the clothes. Each process, washing, drying, folding and storing, takes 30 minutes each. If done subsequently, the tasks will take 8 hours for four loads of laundry, while pipelined drying takes just three and half hours. The pipelined laundry is therefore potentially four times faster. the more tasks, the less than saved by pipelining. This is same for processors where pipelining is used. The MIPS instructions

- Get instructions from memory
- Decode instruction and read registers

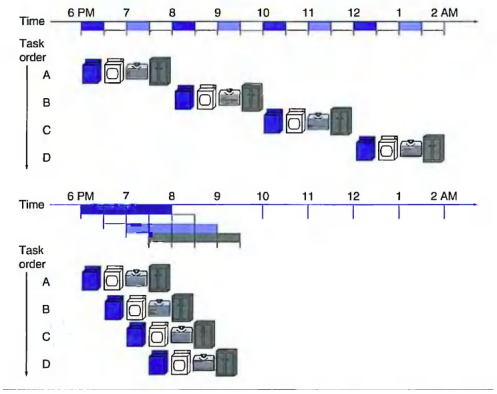

Figure 7. The laundry analogy for pipelining

- Execute the operation or address calculation
- Then finally write the result back into a register

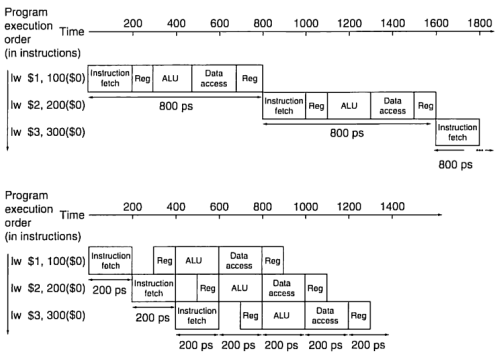

Figure 8. Non-pipelined VS Pipelined execution

From figure 8, even though both execution uses the same components, it took the non-pipelined implementation 800ps for each instruction but only 200ps for a pipelined implementation.

A. Pipeline Hazards

A major limitation of pipelining is that it uses an in-order instruction execution. Instructions are executed concurrently and if an instruction is stalled or delayed in the pipeline, no subsequent instruction can be executed. Therefore, a task depending on another who is stalled in the pipeline will lead to what is called a 'Hazard'. An example is the code below:

DIV.D F0,F2,F4

ADD.D F10, F0, F8

SUB.D F12, F8, F14

If task DIV.D is stalled, ADD.D will not be able to execute and if ADD.D did not execute, SUB.D will also be stalled. This is referred to as hazard.

Below are the types of hazards:

1) Data Hazards
2) Structural Hazards
3) Control Hazards

Data hazards occur when an instruction is dependent on a previous instruction in the pipeline, that is a process is waiting for another process before proceeding.

Data hazards can be classified into three

- Read After Write (RAW)
- Write After Read (WAR)
- Write After Write (WAW)

Structural hazards occur when we want to execute a combination of instructions in the same clock cycle which the hardware cannot support. That is if two or more instructions in the pipeline need the same resource for execution.

Control hazards are hazards caused by delay in decision making by the result of an instruction while other instructions are executing. A good example is having to decide if to change the laundry soap for your white shirts.

There are several solutions for the different types of hazards,pipeline stalls or bubbling. The Tomasulo's algorithm is one of those.

IV. Discussion and Conclusion

I have discussed dynamic scheduling with example and enumerated it's advantages. We also showed how instructions pass through a dynamically scheduled pipeline and to further understand dynamic scheduling and it's importance, we went further to discuss the basics of pipelining with examples and it's hazards. I have observed that pipelining is convergence execution of instructions and that the more sets of instructions, the more the time saved by pipelining. Pipelined instructions if stalled or delayed in the pipeline, will affect all subsequent instructions to be executed. This can be solved by dynamic scheduling. The basic idea of dynamic scheduling is to execute instructions out of order.

V. Affidavit

I Adijat Ajoke Sulaimon hereby declare that I wrote this paper independently and have not used any sources or aid other than those presented. The paper in the same or similar form has not been submitted to any examination body and has not been published.

Adijat Ajoke Sulaimon
June 26, 2022

REFERENCES

[1] Enabling Adaptive Loop Pipelining in High-Level Synthesis Steve Dai, Gai Liu, Ritchie Zhao, Zhiru Zhang School of Electrical and Computer Engineering, Cornell University, Ithaca, NY. Email: hd273,gl387,rz252,zhiruz@cornell.edu

[2] Computer Architecture Subtitle:Engineering And Technology Author:Dr Ranjani Parthasarathi Computer Architecture by INFLIBNET Centre is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License, except where otherwise noted. July 24, 2018

[3] Moore's Law By CARLA TARDI Updated February 26, 2022 Reviewed by MARGUERITA CHENG https://smartfinance.suaramasa.com/host-https-www.investopedia.com/terms/m/mooreslaw.asp

[4] Prof. Dr. Ben Juurlink Embedded Systems Architecture Institute of Computer Engineering and Micro-electronics School of Electrical Engineering and Computer Science TU Berlin https://www.youtube.com/watch?v=y-N0Dsc9LmU

[5] Computer Organization and Design – The Hardware / Software Interface, David A. Patterson and John L. Hennessy, 4th Edition, Morgan Kaufmann, Elsevier, 2009.

[6] Chapter: Advanced Computer Architecture : Instruction Level Parallelism https://www.brainkart.com/article/Dynamic-Scheduling$_8$832/