Final Project

"Tomasulo Simulator/Solver"

Lalitha Dwarapudi - 861310053
Keerthana Ashok Bidarakoppa - 861309124

1. Introduction

Tomasulo's algorithm enables the processor to handle out-of-order execution of instructions and avoid structural hazards to maximize the utilization of the functional units in the processor. Tomasulo's algorithm, shown in Figure1, has multiple Reservation Stations "RS" for each functional unit. All the modules in Tomasulo's architecture will be connected through a Common-Data Bus "CDB". However, the implementation of Tomasulo's algorithm in modern system is infeasible. One reason for that would be the expense of implementing a common bus that has to be routed to all the modules in the system.

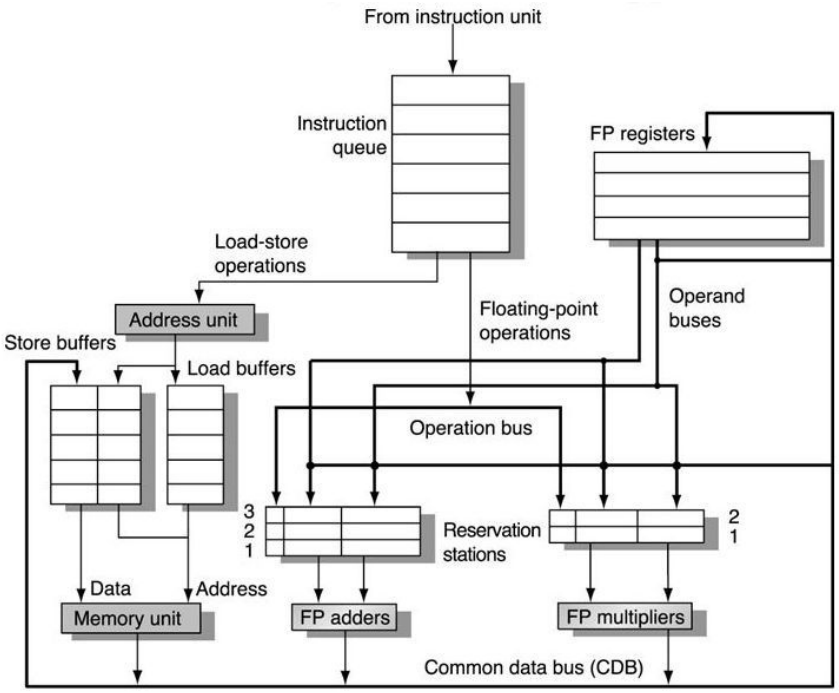

Figure 1. Tomosulo's Algorithm

2. Implementation

In our project, each instruction goes through 4 stages

1. Issue stage

2. Execute stage
3. Memory stage
4. Write back stage

The architecture contains Reservation Stations, Functional units and a CDB in which the following parameters are configurable:

1. **In every cycle, a single instruction is written into the Common data bus (CDB).**
2. **Number of Reservation stations for each execution unit resource.**
3. **Number of Functional Units.**
4. **Number of Memory units**
5. **Execution time for each instruction type.**
6. **Static branch prediction (always Taken or Not Taken). If Taken, branch exits after 4 iterations including the first loop.**

Every instruction has the following parameters:
Destination – dest
Source1 – src1
Source2 – src2
Type – ADD/SUB/MULT/DIV/LW/SW/BNEZ

Reservation stations (RS): These are the intermediate stations which hold data before execution of the instructions and for checking dependencies between the instructions.
We are maintaining separate reservation stations for Load/Store, Add/Sub, Mult/Div and Branch instructions. Based on the user input, reservation stations are created for each execution unit resource. The status of each RS is "AVAILABLE" initially. $Q_j$and Q_k are special variables associated with each Reservation station which hold the dependencies of src1 and src2. Each RS has a timer set to 0. We also maintain the type of instruction in the RS.
Other status maintained in RS are "RESULT_READY", "MEM", "BUSY".

Functional units (FU): These are the units in which the execution of the instruction takes place. Based on the user input, functional units are created. The status of the Functional unit is "EMPTY" initially.

CDB: Common data bus holds the destination of the executed instruction in every cycle. SW and BNEZ instructions do not write into the CDB.

Different stages in the algorithm:

In every cycle, the instruction passes through 4 stages.

ISSUE: The instructions are read from an input file. Each instruction is sent to its particular type of reservation station by reading the instruction type. If the reservation stations are "AVAILABLE", the

instruction is put in the RS. If not "AVAILABLE", the instruction is not issued. When an instruction is put in the RS, status is changed to "BUSY" and the dependencies are checked with previous instructions that are issued and put in Q_j and Q_k.

EXECUTE: In this stage, if the instruction has status as "BUSY" in the RS and has no dependencies on the previous instructions (Q_j = null and Q_k=null), the timer is set to the latency of that instruction and starts executing if the functional unit is "EMPTY" and the status of FU is changed to "FULL". Depending on the number of functional units, the instructions in the same RS are executed in the same cycle. If only 1 functional unit exists and if there are multiple instructions in the RS ready to execute, the instruction which is issued first is executed. If the timer becomes 0, the status of the RS is changed to "RESULT_READY" which means the output is ready to be written into the CDB in the next cycle and the status of FU is made "EMPTY". If instructions in different type of RS have the status as "BUSY" and has no dependencies, they also execute in the same cycle because we have different functional units for each RS. For LW/SW instructions, once execution completes, the status of the RS is changed to "MEM".

MEMORY: The load and store operations are performed in this stage. If the status of LW/SW instructions in LW/SW RS is "MEM" and if the memory unit is "EMPTY", then the instruction completes the Memory stage and changes the status to "RESULT_READY" for LW instruction and "AVAILABLE" for SW instruction because SW doesn't write into CDB.

WRITE_BACK: In this stage, we check all the RS with status as "RESULT_READY". If many instructions have status as "RESULT_READY", we insert all those instructions in the RS into a Priority Queue to find which instruction has to write into the CDB.

From the Queue, we extract the instruction with minimum instruction 'line number' and that particular instruction is written on to the CDB. After writing into CDB, the RS which holds the instruction that is written on to the CDB will be selected and its status will be made as "AVAILABLE". The destination of the instruction which is written into the CDB has to be maintained to check the dependencies.

Tomasulo simulator takes the above values as command line arguments and prints the "Number of cycles" taken to run the algorithm for all instructions.

3. Running the Simulator

We have implemented from scratch a Java based Tomasulo algorithm simulator (Tomasulo).

Tomasulo simulator takes the following parameters as command line arguments.

1. Filename – trace file containing the instructions
2. num_add_rs – number of reservation stations for ADD/SUB
3. num_mult_rs - number of reservation stations for MULT/DIV
4. num_ld_rs - number of reservation stations for LW/SW

5. num_add_unit – number of functional units for ADD/SUB
6. num_mult_unit - number of functional units for MULT/DIV
7. num_ld_unit - number of functional units for LW/SW
8. num_br_unit - number of functional units for BNEZ
9. add_latency – execution time for ADD
10. sub_latency - execution time for SUB
11. mult_latency - execution time for MULT
12. div_latency - execution time for DIV
13. ld_latency - execution time for LW
14. st_latency - execution time for SW
15. br_latency - execution time for BNEZ
16. num_mem_unit – number of memory units
17. branch_result – "T" for Branch taken and "NT" for Branch Not Taken

Compilation and running the project:

1. Go to command prompt.
2. Compilation:
   Go to Tomasulo/src in the "Tomasulo" folder submitted.
   Tomasulo/src$ javac Tomasulo/Tomasulo.java
3. Run:
   Tomasulo/src$ java Tomasulo.Tomasulo instruction.txt 3 2 2 1 1 1 1 1 1 1 1 1 1 1 NT

For e.g 1: (**Branch Not Taken, multiple RS**)

Let's say input trace file is instruction.txt, number of RS for ADD/SUB are 3, number of RS for MULT/DIV are 2, number of RS for LW/SW are 2, number of RS for BNEZ is 1, number of Functional units are all 1, execution time for all instructions is 1 cycle, number of memory units are 1, Branch is "Not taken":

1. Compilation:
   Go to Tomasulo/src in the "Tomasulo" folder submitted.
   Tomasulo/src$ javac Tomasulo/Tomasulo.java
2. Run:
   Tomasulo/src$ java Tomasulo.Tomasulo instruction.txt 3 2 2 1 1 1 1 1 1 1 1 1 1 1 NT

Output: Number of cycles taken to run is 12

For e.g 2: (**Branch Taken, multiple RS**)

Let's say input trace file is instruction.txt, number of RS for ADD/SUB are 3, number of RS for MULT/DIV are 2, number of RS for LW/SW are 2, number of RS for BNEZ is 4, number of Functional units are all 1, execution time for all instructions is 1 cycle, number of memory units are 1, Branch is "Taken":

1. Compilation:
   Go to Tomasulo/src in the "Tomasulo" folder submitted.
   Tomasulo/src$ javac Tomasulo/Tomasulo.java
2. Run:
   Tomasulo/src$ java Tomasulo.Tomasulo instruction.txt 3 2 2 1 1 1 1 1 1 1 1 1 1 1 T

Output: Number of cycles taken to run is 34

For e.g 3: (**Branch Not taken, multiple RS, multiple FU's, multiple execution cycles, multiple Memory units)**

Let's say input trace file is instruction.txt, number of RS for ADD/SUB are 3, number of RS for MULT/DIV are 2, number of RS for LW/SW are 2, number of RS for BNEZ is 1, number of Functional units are all 2, execution time for all instructions is 2 cycles, number of memory units are 2, Branch is "Not Taken":

1. Compilation:
   Go to Tomasulo/src in the "Tomasulo" folder submitted.
   Tomasulo/src$ javac Tomasulo/Tomasulo.java
2. Run:
   Tomasulo/src$ java Tomasulo.Tomasulo instruction.txt 3 2 2 2 2 2 2 2 2 2 2 2 2 2 NT

Output: Number of cycles taken to run is 17

For e.g 4: **(Branch Not taken, multiple RS**)

Let's say input trace file is instruction2.txt, number of RS for ADD/SUB are 3, number of RS for MULT/DIV are 2, number of RS for LW/SW are 2, number of RS for BNEZ is 1, number of Functional units are all 1, execution time for all instructions is 1 cycle, number of memory units are 1, Branch is "Not Taken":

1. Compilation:
   Go to Tomasulo/src in the "Tomasulo" folder submitted.
   Tomasulo/src$ javac Tomasulo/Tomasulo.java
2. Run:
3. Tomasulo/src$ java Tomasulo.Tomasulo instruction.txt 3 2 2 1 1 1 1 1 1 1 1 1 1 1 NT

Output: Number of cycles taken to run is 15

For e.g 5: (**Branch Taken, multiple RS, multiple FU's, multiple execution cycles, multiple Memory units**)

Let's say input trace file is instruction.txt, number of RS for ADD/SUB are 3, number of RS for MULT/DIV are 2, number of RS for LW/SW are 2, number of RS for BNEZ is 2, number of Functional units are all 2, execution time for all instructions is 2 cycles, number of memory units are 2, Branch is "Taken":

1. Compilation:
   Go to Tomasulo/src in the "Tomasulo" folder submitted.
   Tomasulo/src$ javac Tomasulo/Tomasulo.java
2. Run:
   Tomasulo/src$ java Tomasulo.Tomasulo instruction.txt 3 2 2 2 2 2 2 2 2 2 2 2 2 2 T

Output: Number of cycles taken to run is 21

Simulator Output: We have performed rigorous testing with various configurations and found that the simulator was working as expected. Below are few test cases we have performed.

If instruction.txt -> filename=1

If instruction2.txt -> filename=2

filename	add_rs	mult_rs	ld_rs	Add_unit	Mult_unit	Ld_unit	Br_unit	Add_time	Sub_time	Mult_time	Div_time	Ld_time	St_time	Br_time	Mem_unit	Branch_res	cycles
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	NT	13
1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	NT	17
1	3	2	2	1	1	1	1	1	1	1	1	1	1	1	1	NT	12
1	3	2	2	1	1	1	1	1	1	1	1	1	1	1	1	T	34
2	3	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	44
2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	NT	19
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	NT	21
2	3	2	2	2	2	2	2	2	2	2	2	2	2	2	2	NT	21
2	3	2	2	2	1	1	1	1	1	1	1	1	1	1	1	T	15
2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	T	19