Report Project 2

Prepared by
Fatemah Abdelwahed and Layla Mohsen

SUMMARY

- Description of Implementation
- User Guide for Simulation
- Results
    - Discussion of results

Description of Implementation

We wrote a C++ program to implement Tomasolu's algorithm on an input assembly code as well as an input memory. The memory and assembly code is input to the program as a text file that is uploaded by the user. The C++ program parses the text file into a vector of instructions and a vector for the memory. The memory is adjusted to be word addressable where each word is 16-bits. The instructions are then checked for their format whether they're I-format, R-format, SB-format, UJ-Format. The tomasolu process starts for issuing the instructions, a reservation station is built for each functional unit using the class for the reservation station. The reservation station has the label, Op, Vj, Vk, Qj, Qk, A, busy, number of cycles it takes, current cycle and a branchstall. For each instruction fetched, we check the availability of the reservation station, and we fill the station accordingly. Any dependencies are also handled. We implemented the reservation station as a vector of struct station, which has the elements aforementioned. We implemented the Reg contents using an array of struct as well with elements value and Qi. We also implemented functions for the Tomasulo algorithm including the following:

- Initialization for each type of reservation station: Add1, Add2, Add3.. etc.
- File parsing faction that takes input file and turns it into vector of instructions/map of memory
- Read next instruction function
- Checking status of reservation station function (takes the name of the station as an input)
- Setter functions: including Vq, Rd, Offset (any value calculated) + the impacted registers/memory
- Branch prediction function: implements always not taken prediction.
- Functions to check whether the prediction was a success or a fail and operate accordingly.
- Function that handles going to another state
- Function the handle operations based on Tomasolu's algorithm
- Functions checking load/store dependencies
- Function initiating cycles/updating the next cycle
- Functions to get IPC/print Status/print Reservation Stations etc.

Bonus Features

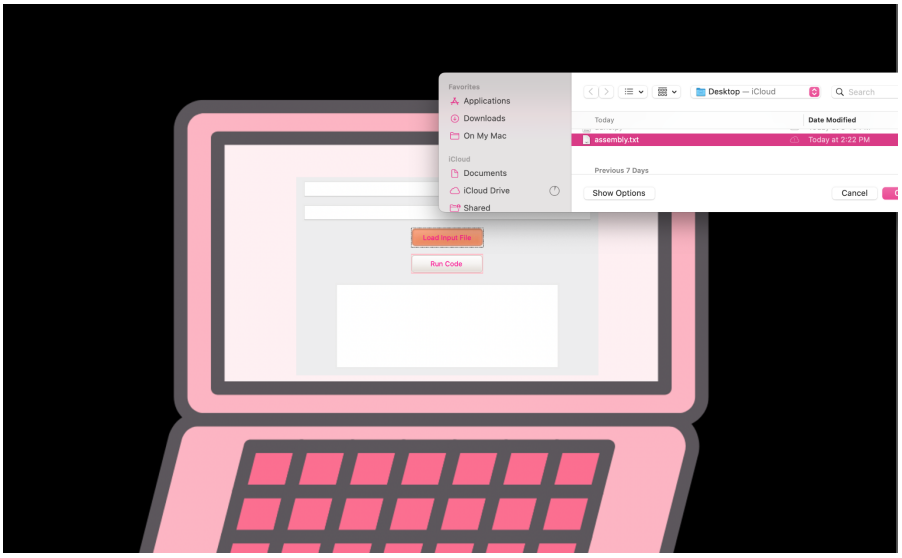
(1) Implementing and integrating a parser to allow the user to provide the input program using proper assembly language including labels for jump targets and function calls.
(2) Building a GUI using python and tkinter that takes an input assembly code file from the user and sends it to the C++ program, the C++ program creates an output file for the resulting table, which is displayed on the screen of the GUI.
(3) Support a variable hardware organization. This bonus too will be counted as two features as far as grading is concerned. If implemented the user should specify the number of

reservation stations for each class of instructions. Additionally, the user will specify the number of cycles needed by each functional unit type.

User Guide for Simulation

Step 1: Download the attached zip file for the codes.
Step 2: Open the python program GUI.py
Step 3: Correct the path for the C++ program that the python GUI runs in line 16
Step 4: Run the python program, the following output should be shown:

Step 5: Click “Load Input File”, select the text file that contains the assembly code



Step 6: Click “Run Code”

Results

Procedure

We created test programs that use the required instructions.

Test Programs & Screenshots of Simulations

program 1

```
ADDI x3, x0, 2
LOAD x2, 0(x0)
ADDI x2, x2, 1
NAND x4, x2, x3
BNE x3, x2, test
test: RET
```

Part of Output

```
0.2Instruction      Station     Issue   Execution   Write
ADDI x3, x0, 2      Add1        1       2       3       4

LOAD x2, 0(x0)      Load1       2       3       -       -

ADDI x2, x2, 1      Add2        3       4       -       -

NAND x4, x2, x3     Nand1       4       -       -       -

BNE x3, x2, test    Bne1        5       -       -       -

R1TESTETESTSADDFSADSAFS
Label Vj   Vk Busy   Qj   Qk

Load1   0   0   0
Label Vj   Vk Busy   Qj   Qk

Load2   0   0   0
Label Vj   Vk Busy   Qj   Qk

Store1   0   0   0
Label Vj   Vk Busy   Qj   Qk

Store2   0   0   0
Label Vj   Vk Busy   Qj   Qk

Bne1   3   2   1
Label Vj   Vk Busy   Qj   Qk
```

Add1 0 0 0
Label Vj Vk Busy Qj Qk

Add2 0 0 0
Label Vj Vk Busy Qj Qk

Add3 0 0 0
Label Vj Vk Busy Qj Qk

Nand1 0 0 0
Label Vj Vk Busy Qj Qk

Div1 0 0 0
Label Vj Vk Busy Qj Qk

CallRet1 32759 0 1
0.666667Instruction Station Issue Execution Write
ADDI x3, x0, 2 Add1 1 2 3 4

LOAD x2, 0(x0) Load1 2 3 5 6

ADDI x2, x2, 1 Add2 3 4 5 6

NAND x4, x2, x3 Nand1 4 5 5 6

BNE x3, x2, test Bne1 5 - - -

RET CallRet1 6 - - -

Label Vj Vk Busy Qj Qk

Load1 0 0 0
Label Vj Vk Busy Qj Qk

Load2 0 0 0
Label Vj Vk Busy Qj Qk

Store1 0 0 0
Label Vj Vk Busy Qj Qk

Store2 0 0 0
Label Vj Vk Busy Qj Qk

Bne1 0 0 0
Label Vj Vk Busy Qj Qk

Add1 0 0 0
Label Vj Vk Busy Qj Qk

```
Add2    0   0   0
Label Vj   Vk Busy    Qj   Qk

Add3    0   0   0
Label Vj   Vk Busy    Qj   Qk

Nand1    0   0   0
Label Vj   Vk Busy    Qj   Qk

Div1    0   0   0
Label Vj   Vk Busy    Qj   Qk

CallRet1    0   0   0
Instruction       Station      Issue    Execution    Write
ADDI x3, x0, 2       Add1       1       2       3       4

LOAD x2, 0(x0)       Load1       2       3       5       6

ADDI x2, x2, 1       Add2       3       4       5       6

NAND x4, x2, x3       Nand1       4       5       5       6

BNE x3, x2, test       Bne1       5       6       6       7

RET       CallRet1       6       -       -       -
```

GUI output:

You can scroll through the output box to see what you want.

In the output we see the table listing the clock cycle time at which each instruction was: issued, started execution, finished execution, and was written. As well, the code outputs the IPC, the Reservation Stations status, and the cycles. It shows the status of each reservation station at each cycle, and the status of each instruction.

Discussion of Results

Each instruction type, such as LOAD, STORE, BNE (branch), CALL/RET (call or return), ADD/ADDI (addition), NAND, and DIV (division), has a specific number of reservation station units and cycles needed for various stages of execution.

For example, LOAD and STORE instructions, each requiring 2 units, are issued in three cycles. The execution begins in the subsequent cycle (X + 1), with the computation of the

address. By the end of cycle (X + 3), the instruction has completed execution, and the result is ready to be written back.

Similarly, branch instructions like BNE, taking 1 unit, are issued in cycle Z and complete execution by cycle (Z + 1). CALL/RET instructions also take 1 cycle, issued at cycle W and finishing execution by cycle (W + 1).

Arithmetic instructions like ADD/ADDI, requiring 3 units, are issued in cycle P, start execution in the next cycle (P + 1), and finish execution by cycle (P + 3).

NAND instructions, with 1 unit, are issued at cycle Q, start execution at cycle (Q + 1), and finish by cycle (Q + 2).

Notably, DIV instructions require 1 unit but have a longer execution time, spanning 11 cycles. They are issued at cycle R, start execution at cycle (R + 1), and complete by cycle (R + 11).

These execution timelines provide an overview of the dynamic scheduling and execution of different instruction types in the Tomasulo algorithm.

program 2

```
ADDI x3, x0, 5
ADDI x4, x0, 10
ADDI x5, x0, 7
L1: ADD X2, X3, X1
DIV x7, x5, x4
CALL EXIT
EXIT:DIV x8, x3, x5
```

Instruction	Station	Issue	Execution	Write
Add1	1	2	3	4
Add2	2	3	4	5
Add3	3	4	5	6
Add1	4	5	6	7
Div1	5	6	15	16
Div1	16	17	-	-

Label Vj Vk Busy Qj Qk

Load1 0 0 0
Label Vj Vk Busy Qj Qk

Load2 0 0 0
Label Vj Vk Busy Qj Qk

Store1 0 0 0
Label Vj Vk Busy Qj Qk

Store2 0 0 0
Label Vj Vk Busy Qj Qk

Bne1 0 0 0
Label Vj Vk Busy Qj Qk

Add1 0 0 0
Label Vj Vk Busy Qj Qk

Add2 0 0 0
Label Vj Vk Busy Qj Qk

Add3 0 0 0
Label Vj Vk Busy Qj Qk

Nand1 0 0 0
Label Vj Vk Busy Qj Qk

Div1 3 5 1
Label Vj Vk Busy Qj Qk

CallRet1 0 0 0
0.192308Instruction Station Issue Execution Write
Add1 1 2 3 4

Add2 2 3 4 5

Add3 3 4 5 6

Add1 4 5 6 7

Div1 5 6 15 16

Div1 16 17 - -

Label Vj Vk Busy Qj Qk

Load1 0 0 0

Label Vj Vk Busy Qj Qk

Load2 0 0 0
Label Vj Vk Busy Qj Qk

Store1 0 0 0
Label Vj Vk Busy Qj Qk

Store2 0 0 0
Label Vj Vk Busy Qj Qk

Bne1 0 0 0
Label Vj Vk Busy Qj Qk

Add1 0 0 0
Label Vj Vk Busy Qj Qk

Add2 0 0 0
Label Vj Vk Busy Qj Qk

Add3 0 0 0
Label Vj Vk Busy Qj Qk

Nand1 0 0 0
Label Vj Vk Busy Qj Qk

Div1 0 0 0
Label Vj Vk Busy Qj Qk

CallRet1 0 0 0

Instruction	Station	Issue	Execution	Write
Add1	1	2	3	4
Add2	2	3	4	5
Add3	3	4	5	6
Add1	4	5	6	7
Div1	5	6	15	16
Div1	16	17	26	27


/Users/layla/Desktop/Tomasulosalgorithm/instructions.txt
Load Input File
Run Code
CallRet1 0 0 0
Instruction Station Issue Execution
Write
ADDI x3, x0, 2 Add1 1 2 3
4
LOAD x2, 0(x0) Load1 2 3 5
6