Computer Architecture (COL216)
Minor Examination

Assignment Report

Submitted by
Pratyush Saini
2019CS10444

Preface

The Assignment involves the enhancement of a basic MIPS interpreter handling a subset of the ISA that was developed in Assignment 3.

The additional features included are:

1. Developed a model for the Main Memory and integrate into the basic interpreter.

2. The memory access should be non-blocking (subsequent instructions don't always wait for the previous instructions to complete).

Input

1. MIPS assembly language program as a text file.

2. DRAM timing values ROW_ACCESS_DELAY and COL_ACCESS_DELAY in cycles. Default values taken are 10 cycles and 2 cycles respectively.

Reading values (lw instructions)

The Program is able to handle all lw instruction forms, example:

```
lw $t1, 12($s0)
lw $t1, variable
lw $t1, address
```

While Reading values from a memory address and storing in the destination register, first we check whether the row corresponding to the input memory address is already loaded in the row buffer.
In such a case, we just copy the data at the column offset to the data bus after time COL_ACCESS_DELAY and no need to load any new row in the row buffer.

In case the row is different from the one currently located in the row buffer, then:

1. Write the current row in the row buffer back to DRAM in time ROW_ACCESS_DELAY.

2. Load the new row in the row buffer in time ROW_ACCESS_DELAY.

3. Copy the data at the column offset to the data bus after time COL_ACCESS_DELAY.

Saving values (sw instructions)

The Program is able to handle all sw instruction forms, example:

```
sw $t1, 12($s0)
sw $t1, variable
sw $t1, address
```

While Reading values from a Register and storing in the given memory address, first we check whether the row corresponding to the input memory address is already loaded in the row buffer.
In such a case, we just write the data from register to the column offset to the data bus after time COL_ACCESS_DELAY and no need to load any new row in the row buffer.

In case the row is different from the one currently located in the row buffer, then:

1. Write the current row in the row buffer back to DRAM in time ROW_ACCESS_DELAY.

2. Load the new row in the row buffer in time ROW_ACCESS_DELAY.

3. Update the data in the row buffer in time COL_ACCESS_DELAY.

Other Instructions

Other Instructions are implemented in the same way as in Assignment3.

Non Blocking Memory Address

While Implementing the DRAM we notice that the processor slows down significantly because of DRAM access delays. To improve the efficiency the following enhancement has been made in the program.

After Issuing some DRAM Operation :

If in any upcoming cycle when the DRAM is running, any instruction that is encountered is independent of the sw/lw instruction for which DRAM access is issued, then that instruction will be executed in the coming cycle.

In case the instruction is dependent on the sw/lw instruction for which DRAM access is issued, the program halts at the stage and waits for loading row in the row buffer, and then executes the upcoming instructions.

How to check Dependency?

In order to check dependency, the following features are incorporated in the program.

a) When DRAM instruction is issued for lw instructions

Eg. lw Rdest, address

If in any subsequent instruction in which Rdest value is read or updated is considered a dependent statement.

b) When DRAM instruction is issued for sw instructions

Eg. sw Rdest, offset($Src)

If in any subsequent instruction in which $Src value is read or updated is considered a dependent statement.
However in this case, if the instruction is dependent on Rdest, then that statement is considered an independent statement. In a processor we read the contents of the register first and then proceed to memory. In such a case, the register value of Rdest is already read in the previous cycle. So, altering the value in Rdest will not make much of a difference.

RowBuffer Data Structure

Holds the attributes of current row loaded in the row Buffer.

Assumptions Taken

1. Instructions may be in memory, but are not fetched from DRAM (consider instruction access delay to be zero). Only lw/sw instructions result in DRAM accesses.

2. The same architectural and ISA assumptions as in Assignment 3 are used.

Output:

At every clock cycle, the following outputs are displayed:

a) The current clock cycle.
b) Current Instruction being executed
c) Modification made in any register in that clock cycle.
d) Modified made in memory address in that clock cycle.
e) Activity in the DRAM, which can be
    i) Issued the DRAM Instruction
    ii) Writing row buffer back to the DRAM
    iii) Reading row buffer from the DRAM
    iv) Accessing the column from the DRAM
f) After every DRAM activity is completed, the modification made is displayed at the last clock cycle of the corresponding DRAM activity.

After all the executions are completed, the program outputs:

a) Contents of all the Registers.
b) Total clock cycles.
c) Number of times each instruction was executed.
d) Number of row Buffer updates.
e) Memory content at the end of off executions.