Computer Architecture Project Report

**Rahul Shah**

**Peter Dunham**

**Brandon Jennings**

**November 2, 2013**

# Running the Executable

## System Requirements

* Windows
* Visual Studios

## Compiling

This code can be run in the Visual Studios environment, making compiling as simple as running the code.

## Input

The input file should be a text file structured similar to:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | # of rs | Cycles in EX | Cycles in Mem | # of FUs |
| Integer adder | 2 | 1 |  | 1 |
| FP adder | 3 | 3 |  | 1 |
| FP multiplier | 2 | 20 |  | 1 |
| Load/store unit | 3 | 1 | 4 | 1 |

ROB entries = 128

R1=10, R2=20, F2=30.1

Mem[4]=1, Mem[8]=2, Mem[12]=3.4

Add.d F1, F2, F3

Ld F4, 8(R1)

Bne R2, R3, -3

Explicit examples can be viewed in the bench mark section.

## Output

The output of the program will be displayed on the console. There are two ways to view the output. The final output of the instructions can be viewed or the output of each clock cycle can be viewed. Lines 499 and 500 can be commented to show the final output and uncommented to show the output every clock.

# Test Benchmarks

## No Dependencies

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | # of rs | Cycles in EX | Cycles in Mem | # of FUs |
| Integer adder | 2 | 1 |  | 1 |
| FP adder | 3 | 3 |  | 1 |
| FP multiplier | 2 | 20 |  | 1 |
| Load/store unit | 3 | 1 | 4 | 1 |

ROB entries = 128

R1=10, R2=20, R4=5, R5=15, R7=100, R9=21, R10=11

F2=30.1, F4=2.5, F5=4, F7=23.78, F8=11.28, F10=5, F11=5

Mem[20]=1

ld f1, 10(r1)

sd f2, 10(r2)

add r3, r4, r5

add.d f3, f4, f5

addi r6, r7, 100

sub r8, r9, r10

sub.d f6, f7, f8

mult.d f9, f10, f11

This instruction file does not contain any dependencies.

*See Appendix A for output.*

## Dependencies

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | # of rs | Cycles in EX | Cycles in Mem | # of FUs |
| Integer adder | 1 | 1 |  | 1 |
| FP adder | 3 | 2 |  | 1 |
| FP multiplier | 2 | 10 |  | 1 |
| Load/store unit | 2 | 1 | 1 | 1 |

ROB entries = 128

r7=50

f1=1, f2=5, f3=4, f4=2.5, f5=15.12, f6=3, f7=6, f11=5

Mem[150]=0.5

mult.d f1, f2, f3

mult.d f2, f1, f6

add.d f6, f11, f7

sub.d f8, f5, f2

add.d f2, f4, f11

This instruction file contains dependencies. There is a read-after-write dependency between the first and second *mult.d* instructions and also between the second *mult.d* and the *sub.d* instructions. There is a write-after-read dependency between the second *mult.d* and the first *add.d* instructions. There is a write-after-write between the second *mult.d* and the second *add.d* instructions.

*See Appendix B for output.*

## Forwarding

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | # of rs | Cycles in EX | Cycles in Mem | # of FUs |
| Integer adder | 2 | 10 |  | 1 |
| FP adder | 3 | 2 |  | 1 |
| FP multiplier | 3 | 10 |  | 1 |
| Load/store unit | 3 | 1 | 4 | 1 |

ROB entries = 128

r1=50

r2=2, r3=4, r5=15, r6=3, f4=6, f1=0.5

Mem[60]=0.5, Mem[61]=1.5, Mem[14]=2.5

add r1, r2, r3

add r4, r5, r6

sd f1, 10(r1)

sd f4, 11(r4)

ld f2, 12(r2)

This instruction file contains two *sd* instructions before a *ld* instruction. The *ld* instruction would normally be able to load into *f2* early because its destination is not dependent on a previous instruction. However, until the destination of the *sd* instructions (before the *ld*) are known, the *ld* instruction does not know where the *sd* instructions are storing, which could be the same place it is loading.

*See Appendix C.1 for output.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | # of rs | Cycles in EX | Cycles in Mem | # of FUs |
| Integer adder | 2 | 5 |  | 1 |
| FP adder | 3 | 2 |  | 1 |
| FP multiplier | 3 | 10 |  | 1 |
| Load/store unit | 3 | 1 | 4 | 1 |

ROB entries = 128

r1=50, r2=8, r3=4, r5=15, r6=3, r7=10

f4=6, f1=0.5

Mem[60]=0.5, Mem[61]=1.5, Mem[14]=2.5

add r1, r2, r3

add r4, r5, r6

sd f1, 10(r7)

ld f2, 12(r2)

This instruction file contains forwarding between the *ld* and *sd* instructions. The *sd* instruction stored to the location the *ld* instruction was loading from. As soon as it finished, the data from *sd* was forwarded to *ld*, hence *ld* instruction can begin after the *sd* instruction executes.

*See Appendix C.2 for output.*

## Structure Hazards

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | # of rs | Cycles in EX | Cycles in Mem | # of FUs |
| Integer adder | 1 | 1 |  | 1 |
| FP adder | 1 | 3 |  | 1 |
| FP multiplier | 1 | 10 |  | 1 |
| Load/store unit | 1 | 1 | 4 | 1 |

ROB entries = 128

R1=10, R2=20, R4=5, R5=15, R7=100, R9=21, R10=11

F2=30.1, F4=2.5, F5=4, F7=23.78, F8=11.28, F10=5, F11=5

Mem[20]=1

ld f1, 10(r1)

sd f2, 10(r2)

add r3, r4, r5

add.d f3, f4, f5

addi r6, r7, 100

sub r8, r9, r10

sub.d f6, f7, f8

mult.d f9, f10, f11

## Loop

# Responsibilities

# Appendix

## Appendix A

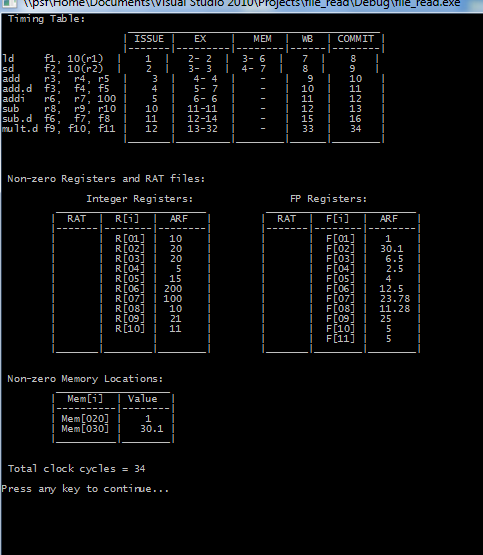


Figure : Output from instruction file containing no dependencies.

## Appendix B

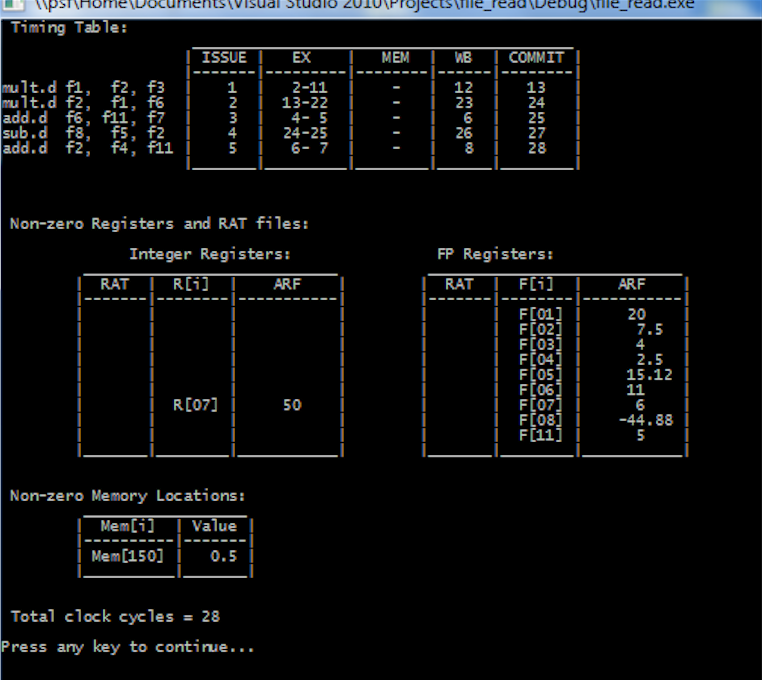


Figure : Output from instruction file containing dependencies.

## Appendix C

#### Appendix C.1

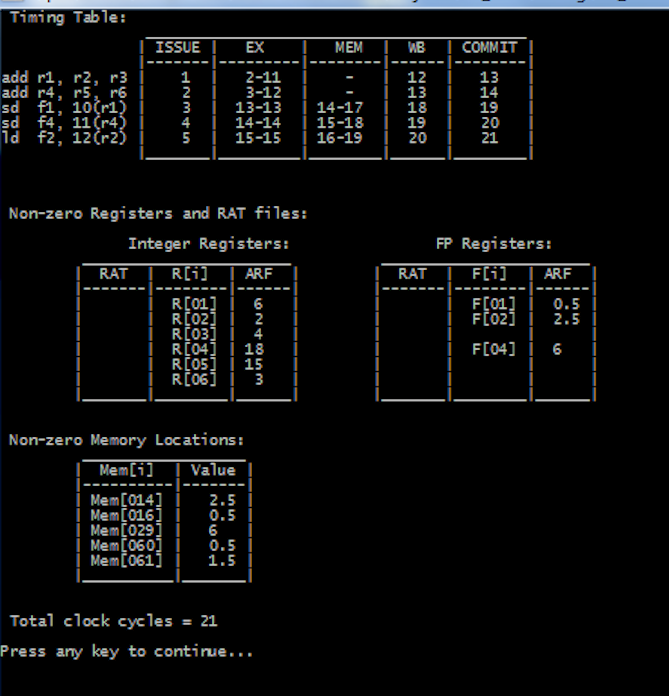


Figure : Output from instruction file containing a load instruction waiting for store instructions.

#### Appendix C.2

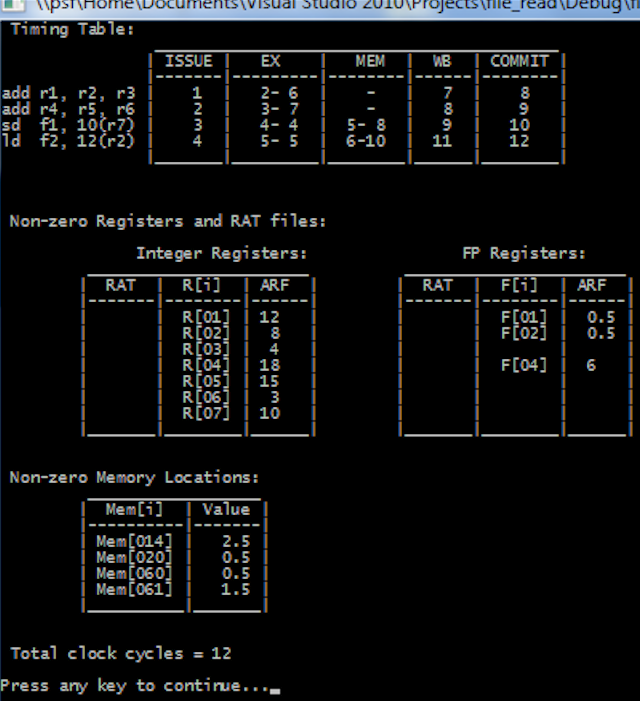


Figure : Output from instruction file containing forwarding between load/store.

## Appendix D

## Appendix E