Kevin Martin
Syracuse University
CIS655 – Summer 2020, Tuesday @ 9:00pm EST
Final Exam – Online portion

Background:

In order to detect pipelining hazards from a group of MIPS instructions, I used a fairly standard Python implementation. However, to help make the reading of each instruction easier, I turn each set of MIPS instructions into a dataframe using the Pandas library. This also helps ease the import of the instructions as well as there is a built-in "read from CSV" function. In this case, the user enters all the MIPS instruction into a regular CSV file (I use Microsoft Excel) and then everything else is processed by my program.

There is only one dependency check function that the program needs, and it checks and marks for all three situations (no solution, no forwarding, and forwarding). Each issue is flagged with a 1 or a 2, depending on how many stall cycles are needed. The program calls this function for each situation and prints the proper execution order and any required stalls. The spacing for the stalls is taken care of by secondary "stall" flags, and the total spaces are calculated using a running total. There are two separate stall counters, one for no forwarding unit and one with a forwarding unit.

The methodology is to detect the various types of dependencies, and then add a flag for how many stalls are needed. There are flags for instances both with and without a forwarding unit. Additionally, because the inputs are only add, lw, sw, and sub, no control dependencies are checked. This leaves instances for both data dependencies and name dependencies, which the program checks for.

I added a couple of additional/advanced features. The first being that, in the case a lw shares a register with an add or subtract **two** lines below it, the program will detect this. This specific circumstance is not really "additional" as it is a very normal dependency, but the program is advanced enough to not simply check the instruction immediately above. Additionally, I added functionality for both beq and bne. The difference here is that the registers which may have the dependency are the first and second, as opposed to something like add or sub, where the second and third registers are the ones being checked.

My program requires at least two instructions, and, because it uses a loop to look for the last instruction, has no upper limit. The user does not need to modify anything, just update the CSV file "input.csv" and save. For my sample program, I built upon the example from the midterm and added a few more instructions to show a few scenarios. I will show the sample input first, highlight the dependencies, show the output in the terminal, and finally the full code. Note that all three situations are printed in the terminal: hazard type/instruction number/register involved, solution without forwarding unit, and solution with forwarding unit.

Sample input (red indicates data dependency and green indicates name dependency):

action	reg1	reg2	reg3
lw	$r1	0($s2)	
addi	$t0	$r1	5
sw	$t0	0($s4)	
add	$r1	$r2	$r3
sub	$r4	$r1	$r5
lw	$t0	0($s2)	
add	$r6	$r7	$r8
beq	$r6	$r1	done
lw	$t6	0($s2)	
lw	$t7	4($s2)	
add	$t3	$t6	$t1

Note the dependency in the final instruction, the data hazard with $t6 comes from the first lw two instructions ahead. This requires only a single stall cycle. Additionally, the beq instruction checks the first register.

Output:


mips_pipeline
C:\Users\Captain\miniconda3\envs\tensorflow\python.exe C:/Users/Captain/Syracuse/Syracuse/CIS655/FinalOnline/mips_pipeline.py
****WITHOUT ANY SOLUTION****
Data Dependency with $r1 from instruction 1 and 0
Name Dependency with $t0 from instruction 2 and 1
Data Dependency with $r1 from instruction 4 and 3
Data Dependency with $r6 from instruction 7 and 6
Data Dependency with $t6 from instruction 10 and 8
****SOLUTION 1 - NO FORWARDING UNIT****
IF ID EX M W
IF S S ID EX M W
IF S S ID EX M W
IF ID EX M W
IF S S ID EX M W
IF ID EX M W
IF ID EX M W
IF S S ID EX M W
IF ID EX M W
IF ID EX M W
IF S ID EX M W
****SOLUTION 2 - WITH FORWARDING UNIT****
IF ID EX M W
IF S ID EX M W
IF ID EX M W
IF ID EX M W
IF ID EX M W
IF ID EX M W
IF ID EX M W
IF ID EX M W
IF ID EX M W
IF ID EX M W
Process finished with exit code 0

Full code:

mips_pipeline.py

```
import pandas as pd

no_stall = 'IF  ID  EX  M   W'
one_stall = 'IF  S   ID  EX  M   W'
two_stall = 'IF  S   S   ID  EX  M   W'
reg_space = '    '
path = './input.csv'
df = pd.read_csv(path, sep=',')
df['stall'] = 0
df['space'] = 0
df['fstall'] = 0
df['fspace'] = 0


def dependency_check(row):
    if df.action[row - 1] != 'sw' and df.action[row-1] != 'lw' and df.action[row] != 'sw':
        if df.action[row] == 'beq' or df.action[row] == 'bne':
            if df.reg1[row - 1] == df.reg1[row] or df.reg1[row - 1] == df.reg2[row]:
                df.at[row, 'stall'] = 2
                print('Data Dependency with', df.reg1[row - 1], 'from instruction', row, 'and', row - 1)
        elif df.reg1[row - 1] == df.reg2[row] or df.reg1[row - 1] == df.reg3[row]:
            df.at[row, 'stall'] = 2
            print('Data Dependency with', df.reg1[row - 1], 'from instruction', row, 'and', row - 1)
    elif df.action[row] == 'sw':
        if df.reg1[row - 1] == df.reg1[row]:
            df.at[row, 'stall'] = 2
            print('Name Dependency with', df.reg1[row - 1], 'from instruction', row, 'and', row - 1)
    elif df.action[row-1] == 'lw' and df.action[row] != 'sw':
        if df.reg1[row - 1] == df.reg2[row] or df.reg1[row - 1] == df.reg3[row]:
            df.at[row, 'stall'] = 2
            df.at[row, 'fstall'] = 1
            print('Data Dependency with', df.reg1[row - 1], 'from instruction', row, 'and', row - 1)
    if row > 2 and df.action[row - 2] == 'lw' and df.action[row] == 'add' or df.action[row] == 'sub':
        if df.reg1[row - 2] == df.reg2[row] or df.reg1[row - 2] == df.reg3[row]:
            df.at[row, 'stall'] = 1
            print('Data Dependency with', df.reg1[row - 2], 'from instruction', row, 'and', row - 2)

```

```
38
39  print(' \n****WITHOUT ANY SOLUTION****')
40  for i in range(1, df.shape[0]):
41      dependency_check(i)
42
43
44  print(' \n****SOLUTION 1 - NO FORWARDING UNIT****')
45  print('', no_stall)
46  for i in range(1, df.shape[0]):
47      df.at[i, 'space'] = df.space[i-1] + df.stall[i-1]
48      if df.stall[i] == 2:
49          print((df.space[i] + i) * reg_space, two_stall)
50      if df.stall[i] == 1:
51          print((df.space[i] + i) * reg_space, one_stall)
52      elif df.stall[i] == 0:
53          print((df.space[i] + i) * reg_space, no_stall)
54
55  print(' \n****SOLUTION 2 - WITH FORWARDING UNIT****')
56  print('', no_stall)
57  for i in range(1, df.shape[0]):
58      # df_for.at[i, 'stall'] = forward_check(i)
59      df.at[i, 'fspace'] = df.fspace[i-1] + df.fstall[i-1]
60      if df.fstall[i] == 2:
61          print((df.fspace[i] + i) * reg_space, two_stall)
62      if df.fstall[i] == 1:
63          print((df.fspace[i] + i) * reg_space, one_stall)
64      elif df.fstall[i] == 0:
65          print((df.fspace[i] + i) * reg_space, no_stall)
66
```