2190250 Comp Arch and Org

Solution to Midterm questions

1) performance

We have two different processors running a set of benchmarks. Processor A executes 10 Millions instructions, has CPI 1.7. Processor B executes 12 Millions instructions, has CPI 1.5. Assume both processors run at the same clock rate. Which one is faster? (show your calculation)

Solution

To answer the question "which one is faster?" we compare the execution time of both processors. They ran the same set of programs.

execution time = total number of instruction executed x CPI x clock rate

assume both processors run at z GHz

execution time A = 10 M x 1.7 x z = 17M z

execution time B = 12 M x 1.5 x z = 18M z

conclusion

Processor A takes LESS time to run the benchmark hence it is faster.

2) Assembly language

Write RISC-V assembly program to do this (assuming data is 32-bit integer):

Count the number of odd integer in an array of positive integer. The array is terminated by -1.

You can use any tools. Submit your code and write explanation of your code.

Solution

This array is a variable length (ako string in C). To find its length, you scan each element until find the terminator (-1). One way test for terminate is to use "branch if it is less than 0". You can also use "branch if equal -1" but you need some way to put -1 into a register.

To test an integer odd or even, check the least significant bit, if it is "0" the number is even. To do that you can "bitwise and" the number with 1, if the result is "1" then it is odd.

pseudo code

```
count = 0
i = 0
while ax[i] >=0
   if (ax[i] & 1) > 0
      count = count + 1    // count odd number
   i = i + 1
```

now translate that into RISC-V code

assume array ax[.] is at 64 (40 hexadecimal)

let x5 - count

x7 - i

x8 - base address of ax[.]

x9 - store value of ax[i]

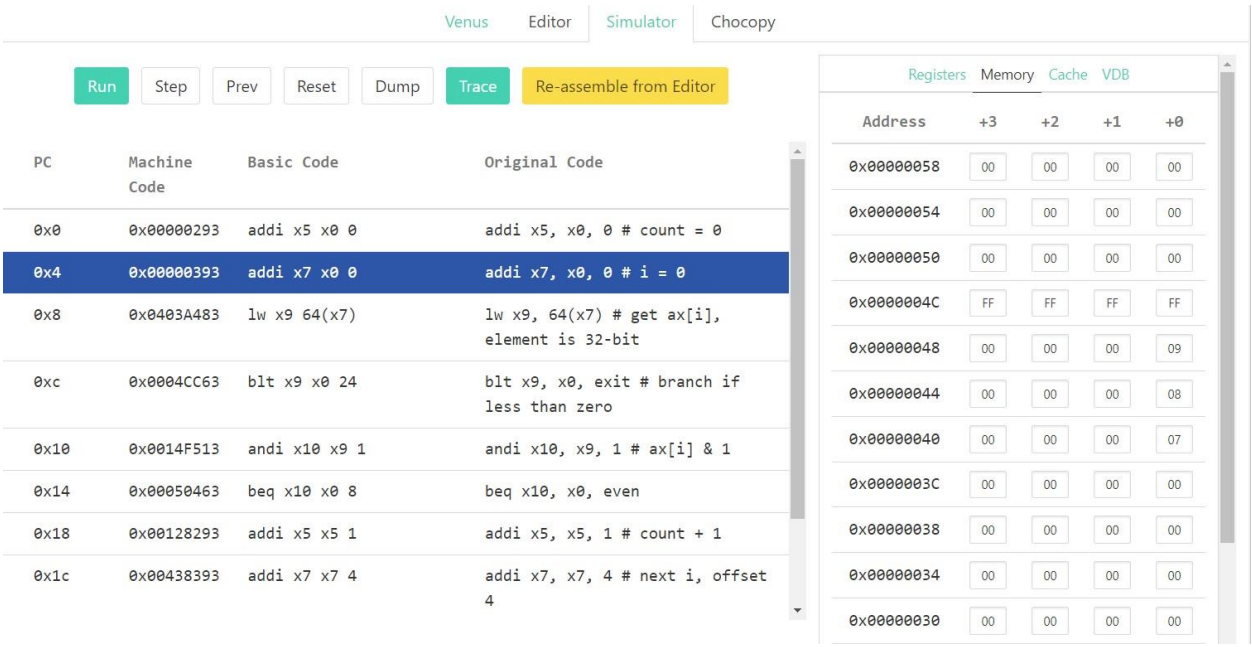
x10 - temp

```
  addi x5, x0, 0    # count = 0
  addi x7, x0, 0    #  i = 0
loop:
  lw x9, 64(x7)     #  get ax[i], element is 32-bit
  blt x9, x0, exit  #  branch if less than zero
  andi x10, x9, 1   #  ax[i] & 1
  beq x10, x0, even
  addi x5, x5, 1    #  odd, count + 1
even:
  addi x7, x7, 4    #  next i, offset 4 bytes
  j loop
exit:
```

If you want to run this program in Venus, you need to set memory (after assemble program).

To do so, go to memory tab. ax[.] starts at 40 hex, each number is 32-bit. In this picture, I set it to 7,8,9,-1. To set -1, you enter -1 in all bytes at address 4C (they will display as FF).

The above example, there are 2 odd numbers, so after the run, you get 2 in x5 and the program stops.



<figure screen of Venus>

3) pipeline

For each code sequence below, state whether it must stall, can avoid stalls using only forwarding, or can execute without stalling or forwarding.

3.1) sequence 1

```
lw x10, 0(x10)
add x11, x10, x10
```

answer

stall (after lw) because data dependency on x10, lw will get data at stage 4 (mem) but add needs this data at stage 3, forwarding will not help because forwarding can deliver data at stage 3 (alu) which (mem) did not yet get the data.

timeline in the pipeline:

```
lw x10    fet  dec  alu  mem  wb
add x11        fet  dec  alu  mem  wb
```

3.2) sequence 2

```
add x11, x10, x10
addi x12, x10, 5
addi x14, x11, 5
```

answer

between add x11 and addi x12 there is no data dependency.

between addi x12 and addi x14 there is no data dependency.

between add x11 and addi x14 there is data dependency on x11 but these instructions are already two clocks apart.

conclusion: there is no stall

in case of add x11 and addi x14, forwarding will not help because it can only forward to the next instruction (not two instructions)

timeline in the pipeline:

```
add x11    fet dec alu  mem  wb
addi x12       fet dec  alu  mem wb
addi x14           fet  dec  alu mem wb
```

3.3) sequence 3

```
addi x11, x10, 1
addi x12, x10, 2
addi x13, x10, 3
addi x14, x10, 4
addi x15, x10, 5
```

answer

all five instructions have no data dependency to each other.

conclusion: no stall