CS 250 Fall 2017 Homework 08

Due 9:00pm Thursday, Nov. 2, 2017

Submit your typewritten file in PDF format to Blackboard

1. Assume that we make an enhancement to a computer that improves some mode of execution by a factor of 10. Enhanced mode is used 50% of the time, measured as a percentage of the total execution time *when the enhanced mode is in use*.
   1. What is the speedup we have obtained from fast mode?

The un accelerated phase is 50% and the accelerated phase is 50%

Without the enhancement, the un accelerated phase would have taken as long 50%, but the accelerated phase would take 10 times as long. So, accelerated phase takes 500%. So, execution time without the enhancement would be 50% + 500% = 550%.

The speed up is

550% / 100% = 5.5

* 1. What percentage of the original execution time has been converted to fast mode?

Using Amdahls law to solve this

(((5.5\*10)-10)/((5.5\*10)-5.5))\*100

=90.9%

1. The CPU time equation is as follows, CPI means Clock cycles per instruction: CPU Time = (Instructions/Program) \* (CPI)\*(Seconds/Clock cycle). For each of the three factors in the CPU Time equation, answer the following questions:  (1) Can loop unrolling ALONE improve this factor, worsen this factor, or cannot affect this factor? (2) If loop unrolling either improves or worsens the factor, how does this occur?

For part1

(instructions/program)🡺loop unrolling will improve this factor by reducing the number of instructions

(clock cycles per instruction)🡺improved by reduction of clock cycles.

(seconds/clock cycle)🡺There will be no effect on this.

For part2

(instructions/program) Loop unrolling happens by optimizing the program and reducing the number of statements leading to improvement in the program. It will reduce clock cycles per instruction which leads to faster processing of instructions leading to an increased processor speed.

1. Once a loop has been unrolled, which factor(s) of the CPU Time equation can be improved, worsened, or cannot be affected through the application of instruction scheduling? Explain.

(instructions/program)🡺will be improved as multiple instructions will be executed in parallel

(CPI)🡺this will not be affected with the parallel processing of data

(seconds/cycle)🡺This will become worse as multiple threads will execute together and time will increase overall

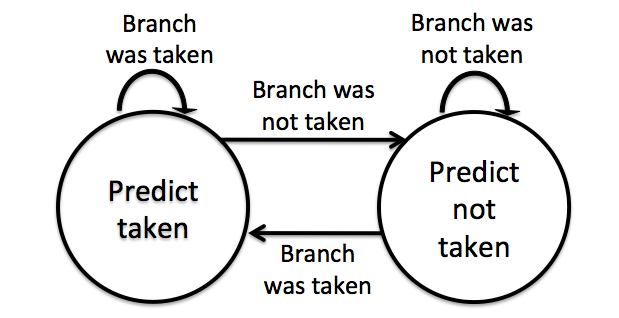
1. An 8-byte data item is stored in the byte-addressed memory of a 32-bit computer at address 0x004488cc. To load this item into registers R1 and R2 so that R1 contains the most significant bytes and R2 the least significant bytes, the processor should do what?

The processor should be executing two load word instructions. At address 0x004488cc and at 0x004488d0.

1. Refer to the following code the answer the next three questions.   
    loadi r5, 16 ; r5 🡨 16 (load immediate value into register)  
    loop: load r1, r5, 40 ; r1 🡨 memory[r5 + 40]  
    add r1, r1, r2 ; r1 🡨 r1 + r2  
    addi r5, r5, -4 ; r5 🡨 r5 – 4  
    store r1, r5, 40 ; memory[r5 + 40] 🡨 r1  
    bne r5, 0, loop ; branch to loop if r5 != 0 (r5 not equal to zero)

If the loop is unrolled 2 times, the number of instructions in the resulting loop body

* 1. must be 10
  2. need only be 8
  3. need only be 6
  4. Cannot be determined until the instructions are scheduled
  5. None of the above

1. As the code above executes, there is a true data dependence from the store to the load.
   1. True
   2. False
2. For each of the following predictors for bne in the code above, give its “win/loss/no decision” numbers. That is, say how many correct predictions, how many incorrect predictions, and how many times the predictor declines to predict. A 1-bit dynamic branch predictor is designed as follows.  
      
   Until this predictor is initialized by recording actual behavior of a machine language branch instruction, it makes no prediction for that branch.  
   1. Predict taken

Initially it makes no prediction for that branch. So now it is in prediction not taken branch.  
So if predict taken then give loss, since currently predictor in branch not taken branch.  
so Loss.

* 1. Predict not taken

Initially it makes no prediction for that branch. So prediction not taken will give Win

* 1. Predict using 1-bit branching history of bne

Predict using 1-bit branching history of bne  
It gives "no decision", because it accepts both bits.. (1-branch taken, 0-branch not taken)