EECS 470 Winter 2024
Homework 5

Due Tuesday April 23rd by ***10:00pm***.

This is an individual assignment; all of the work should be your own. Assignments that are difficult to read will lose at least 50% of the possible points and we may not grade them at all. If you use references other than the text and class notes, be sure to cite them!

---

1. A compiler for IA-64 has generated the following sequence of three instructions:

```
        L.D F0,0(R1)    ; F0=Mem[R1+0]
(p1)    DADD R1,R2,R3   ; if (p1) then R1=R2+R3
(p2)    DSUB R5,R1,R4   ; if (p2) then R5=R1-R4
```

   where p1 and p2 are two predicate registers that are set earlier in the program. Assume that the three instructions are to form a bundle. What are the possible templates that the compiler could use for the bundle? Under what circumstances would each template be chosen? Think about relations that might be known at compile time between p1 and p2. **[20 points]**

   See http://www.eecs.umich.edu/eecs/courses/eecs470/HW/AppG.pdf for information on bundles.

2. Say you are the head architect at "Computers R Us" and your main product is a processor that executes 15 BIPS (billions of instructions per second) and runs at 100 Watts. You have a few teams of interns that have proposed some improvements to the architecture of this processor in order to save power. Which of these might be worth considering and why? For purposes of this question assume voltage scaling by X reduces performance by X and power by X^3. **[15 points]**
   a. A fairly trivial change to the cache system that drops performance to 14 BIPS while it drops power to 90 Watts. **[5]**
   b. A fairly complex change to the out-of-order core that drops performance to 12 BIPS while it drops power to 50 Watts. **[5]**
   c. A very trivial change to the predictor (making it smaller) and the L2 cache (reducing leakage current of some transistors) that drops performance to 13.0 BIPS while it drops power to 50 Watts. **[5]**

3. Consider the following C code:

```
for ( i = 0; i < MAX; i++ )
{
   a[ i ] = a[ i ] + b[ i ];
}
```

That C code is translated into the following x86-like assembly code:
(note: the ++ indicates the autoincrement addressing mode.)

```
        mov r1, addr( a )     -- address of a[ 0 ] into r1
        mov r2, addr( b )     -- address of b[ 0 ] into r2
        mov rx, MAX           -- Number of iterations into rx
l1:     ld r3, (r1)           -- load indirect into r3 through r1
        ld r4, (r2)++         -- what r2 points to loaded in r4
        fadd r5, r3, r4       -- r5 holds sum of two elements
        st r5, (r1)++         -- store result and post-increment
        loop l1               -- does an autodecrement (by 1) of rx
                              -- if rx isnt zero branches to l1
```

And then that assembly code is software pipelined.

```
                                 -- Initialization:
        mov r0, addr( a )     -- r0 is pointer to a[0]
        mov r1, r0            -- copy address of a[0] into r1
        mov r2, addr( b )     -- r2 is pointer to b[0]
        ___blank A______
        ___blank B______
        ___blank C______
        fadd r5, r3, r4
        ld r3, (r1)++
        ld r4, (r2)++
l2:     st r5, (r0)++
        fadd r5, r3, r4
        ld r3, (r1)++
        ld r4, (r2)++
        loop l2               -- decrement rx, if != 0 jump to l2
        ___blank D______
        fadd r5, r3, r4
        st r5, (r0)
```

Answer the following **[23 points]**

a. Supply the missing code for each blank **[18]**
b. If, in the original C code, MAX is less than _____ the software-pipelined loop will behave incorrectly. **[5]**

4. IBM (*IanBraMus Inc*). has just released a new 4-core processor that uses a shared snoopy bus. Each core has a 4-way associative, 64KB cache with 32-byte cache lines and keeps data in the M, S or I state. There is a shared, on-chip, L2. On a given benchmark the following is true of each core:
    - One fourth of the processor's memory transactions to the cache are stores (the rest being loads).
    - 95% of loads and 95% of stores don't generate a bus transaction.
    - 20% of all evictions are of dirty data.
    - Each core sends 95 million transactions on the bus per second. 5 million of those are BILs.

    Answer the following questions. You are to assume that the cores aren't bandwidth limited. **[22 points]**
    a. How many transactions per second would you expect of each of BRIL, BRL, and BWL on the bus? **[10]**
    b. If we were to add an "E" state to the processor, for which transactions types would the rate of transactions be impacted? How would they be impacted (go up or go down)? Explain your answer. **[6]**
    c. A co-worker sees that the "E" state is helpful but that your current product line doesn't support it. They propose to use MSI but to go to the "M" state when MESI would go to the "E" state (where MSI generally goes to the S state). For which transactions types would the rate of transactions be impacted? How would they be impacted (go up or go down)? Explain your answer. **[6]**

5. Read On Pipelining Dynamic Instruction Scheduling Logic and answer the following questions: **[20 points]**
    a. In your own words, explain what problem this paper is trying to solve. **[5]**
    b. Consider figure 4. Describe what this is doing using the terminology we've used in class. **[4]**
    c. Consider figure 7 and 8. Explain, in your own words: **[6]**
        - What is happening in the "Reg Read" stage and how that differs from how register reading was dealt with in class.
        - How the SUB can wakeup before the XOR completes execution
        - What "Execute/Bypass" means.
        - What the difference is between figure 7 and 8.
    d. For your project, you have two basic options: **[5]**
        - Do the select/issue/execute/CDB in one cycle
        - Pipeline this process.

    Which of those two did your group do? If the first, is this on your critical path? If the second, do you deal with back-to-back dependent instructions?