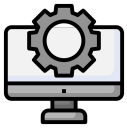
Advanced Computer Architecture



HW0

Prof :

Andreas Moshovos

Student :

Hooman Keshvari

Student Number :

1011293869

Question 1

1. I am a first year Ph.D. student under the supervision of Michael Stumm.
2. Currently, I'm only taking courses
3. I am currently searching for my ideal research path and computer architecture is one of the things that is a black-box for me, so I intend to get what does research in computer architecture really mean and what is its state-of-the-art topic
4. I still do not know yet but I am considering to stay in Academia
5. I have taken Computer Organization and Computer Security in my bachelors. I did not pick any relevant course for architecture during my first term of Ph.D. so this is practically my first one in UofT.
6. The only architecture-related course I have in this term is this coures :D

Question 2

1. (a) To provide a code which shows how well 2-way pipelined superscalar processor works, we first need to remember the fact that this processor could fetch up to 2 instructions and execute them if they do not have any dependencies.
Now that we know this we could easily analyze this simple code :

```
...
LD r5, r4
ADD r3, r1, r2
...
```

Now we hypotheticaly say that the load instruction takes 3 cycles.
This would happen in the simple one-way pipelining :

```
    Cycle 1 : LD fetch
    Cycle 2 : LD decode - ADD fetch
    Cycle 3 : LD execute - ADD decode
    Cycle 4 : LD memory - ADD execute
    Cycle 5 : LD memory - ADD stall
    Cycle 6 : LD memory - ADD stall
    Cycle 7 : LD writeback - ADD memory
    Cycle 8 : ADD writeback
```

Now with 2-way superscalar :

```
    Cycle 1 : LD fetch - ADD fetch
    Cycle 2 : LD decode - ADD decode
    Cycle 3 : LD execute - ADD execute
    Cycle 4 : LD memory - ADD memory
    Cycle 5 : LD memory - ADD writeback
    Cycle 6 : LD memory
    Cycle 7 : LD writeback
```

Now we could see that we have a one-clock cycle decrease in our overall instruction execution because the ADD would not need to wait for the memory operation to finish. Although we don't see much performance gain (only 1 cycle :D) in reality the memory operations take 100 cycles on average so the ADD instruction would have been stalled for 99 cycles.

(b) basically if we have dependencies in our operations or a ton of branching, the 2-way superscalar would not gain us alot of performance.
Look at the code below :

```
    ADD r1, r2, r3
    ADD r2, r1, r3
    ADD r3, r2, r1
    ADD r1, r2, r3
```

Because each instruction uses the output from the previous instruction, we can't do these instructions in parallel and even because of the complex control logic and overhead of the dependancy checking, the 2-way might perform slightly worse than the simple pipelined processor.

(c) We know that out-of-order execution tries to execute independant instructions. So get the code bellow as an example :

```
LD r1, r2
ADD r3, r4, r5
ADD r6, r7, r8
```

now with out-of-order execution, when the load instruction does its memory access operation, the processor can execute both ADD and SUB because they are independant but we saw that in a regular 2-way superscalar, only add could be executed in parallel to load.

(d) See the code below :

```
ADD r1, r2, r3
ADD r4, r1, r5
ADD r1, r6, r7
ADD r8, r1, r9
```

we could obviously tell that in the third instruction is write after write so the r1 value in the third instruction is independant from the outputs of the previous instructions, so if the r1 was r10 in both instructions 3 and 4, then the fake dependancy would be removed, thus more instructions could be executed in parallel

2. The instruction fetch process begins when the processor generates a virtual address. This address is simultaneously sent to two places: the 8-entry iTLB for translation to a physical address, and the instruction cache. The cache uses lower bits of the virtual address to begin access immediately, while the iTLB translates the higher bits to physical bits for tag comparison. The instruction cache, organized in 16-instruction lines, uses four sense amplifiers that can each read four adjacent instructions(0-3 4-7 8-11 12-15). Each amplifier includes a selector to choose the specific instruction needed, enabling fetches to start from any position. A rotator then arranges these selected four instructions by program order before they enter the instruction buffer.