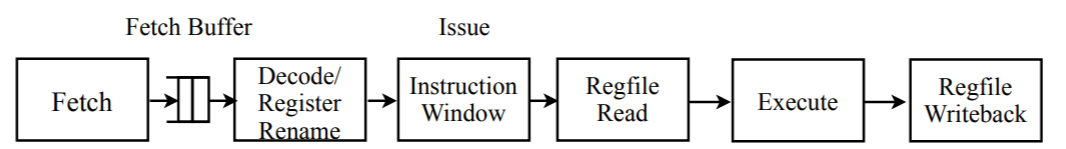
**Worksheet 9**

**Q1. OoO Processor (2011 quiz 3)**

The processor uses a split instruction window/ROB design, with a unified physical register file (similar to the MIPS R10k). A diagram of the processor is shown in the figure below.



The micro architecture can be defined as following stages: fetch, decode, reg rename, dispatch, issue, execution, complete, and commit. Answer the following questions.

1. Which stage(s) allocates entries in the ROB?
2. At which stage(s) is an entry in the ROB deallocated?
3. At which stage(s) is an instruction allocated an entry in the instruction window?
4. At which stage(s) is an instruction entry in the instruction window deallocated? What if the architecture uses instruction window for renaming instead of using a unified physical register file?
5. At which stage(s) is a store entry allocated in the LD/ST queue?
6. At which stage(s) is a load entry allocated in the LD/ST queue
7. At which stage(s) is a` store performed (i.e., sent to the memory)?
8. At which stage(s) is a load performed (i.e., when is data returned that can be used by other dependent instructions)?

**Q2. VLIW (2018 mid2)**

In this problem, we consider the execution of a code segment on a VLIW processor. The code we consider is the IMAX kernel, which finds the maximum value and its index in the list.

for (i = 0 ; i < N ; i++) {

if (max < l[i]) {

idx = i;

max = l[i];

}

}

# t0: i, s0: idx, f0: max, a0: N, a1: pointer of l[i]

|  |  |  |
| --- | --- | --- |
| loop: | fld f1, 0(a1) | # load l[i] |
|  | flt.d t1, f0, f1 | # set if max < l[i] |
|  | fmax.d f0, f0, f1 | # max = max < l[i] ? l[i] : max |
|  | beqz t1, skip | # if max >= l[i], jump to skip |
|  | addi s0, t0, 0 | # update idx |
| skip: | addi a1, a1, 8 | # bump l |
|  | addi t0, t0, 1 | # increment i |
|  | bltu t0, a0, loop | # loop |

Now we have a VLIW machine with five execution units:

* two ALU units, latency one cycle, also used for branch operations.
* one memory unit, latency two cycles, fully pipelined, each unit can perform either a store or a load.
* two FPU units, latency three cycles, fully pipelined, both can perform flt.d and fmax.d.

Assume there are no exceptions during the execution. Schedule instructions for the VLIW machine with software pipelining but without loop unrolling in table below including the prologue and the epilogue. You do not need to find the optimal scheduling. Hint: find the common path; pay attention to pointer change; make sure your loop boundary and number of loops executed is N.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Label | ALU1 | ALU2 | MEM | FPU1 | FPU2 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Q3. Multithreading (2011 quiz 4)**

Considering the following code:

addi x1, x0, 1024  
 addi x2, x0, 0  
loop:  
 fld f1, A(x2)  
 fld f2, B(x2)  
 fmul.d f3, f1, f2  
 fld f4, Y(x2)  
 fadd.d f5, f3, f4  
 fsd f5, S(x2)  
 addi x2, x2, 8  
 addi x1, x1, -1  
 bnez x1, loop

Assume the following:

* Our system does not have a cache.
* Each memory operation directly accesses main memory and takes 50 CPU cycles.
* The load/store unit is fully pipelined.
* After the processor issues a memory operation, it can continue executing instructions until it reaches an instruction that is dependent on an outstanding memory operation.
* The fmul and fadd instructions both have a use-delay of 5 cycles.

1. Now consider multithreading the pipeline. Threads are switched **every cycle** using a fixed round-robin schedule. If the thread is not ready to run on its turn, a bubble is inserted into the pipeline. Each thread executes the above code, and is calculating its own independent piece of the S array (i.e., there is no communication required between threads). Assuming an infinite number of registers, what is the **minimum** number of threads we need to fully utilize the processor? You are free to re-schedule the assembly as necessary to minimize the number of threads required.
2. Threads are switched **when there is data dependence**. Assuming an infinite number of registers, what is the **minimum** number of threads we need to fully utilize the processor? You are free to re-schedule the assembly as necessary to minimize the number of threads required.

**Q4. Vector Architecture (2011 quiz 4)**

Consider which of the following code can be vectorized.

1. for(int i = 0; i < N; i++)

A[i] = A[i] + B[i];

1. for(int i = 0; i < N; i++)

A[i] = A[i+1] + B[i];

1. for(int i = 0; i < N; i++)

A[i] = A[i-1] + B[i];