Pipeline ‚Pre-load' Opcode Proposal 18.06.23

kaqu

An ISA on an actual hardware platform usually suffers from memory access timing penalties, hence the usage of (instruction) caches.
These themselves require time to fill after an enforced flush (due to branching out of bounds), with small on-chip caches (level 0/1) this tends to happen more often than wished for.
As on-chip space is an expensive resource, caches cannot grow much.

To circumvent this limitation, one shall operate several simultaneous queues by dividing the instruction cache. Four partitions seem to be a reasonable split.
Now, if there is a compiler generated (automatic) opcode, indicating the memory offset from which to load into a particular partition automatically (let's call it a ‚pipeline' from now on), this shall reduce processing stall considerably.

As usual ‚locality' heuristics do apply, that is farther jumps (beyond cache) may be weighted higher – local jumps within cache not at all etc.

Arguments against:
1. Full size better! Forget it … (K.I.S.S.)
2. switch/match/case constructs fail (as there cannot possibly be enough ‚pipelines'?)

Some examples shall clearify this (‚Address_A' assumed ‚out-of-cache', i.e. a miss):

1. Example: CALL, assumed queue #0 active

Address	**Mnemonics**	**Explanations**
B	*Some code*	Load a free instruction queue (here: #1) with code starting at ‚A' in advance (Memory access may take hundreds of cycles, on an FPGA still dozens)
B+1	LoadQueue Address_A	
…	…	
B+n	*Some code*	If (B+n+1)-(B+1)=delta is >= the # of cycles, stalling can be avoided completely!
B+n+1	CALL Address_A	
B+n+2	…	
…	…	On CALL: Maybe reload (B+n+2) to a third queue as well? May still run within queue #0 but switch over to #2 if it becomes ready?

2. Example: JMP, assumed queue #0 active

Address	**Mnemonics**	**Explanations**
B	*Some code*	Load a free instruction queue (here: #1) with code starting at ‚A' in advance (Memory access may take hundreds of cycles, on an FPGA still dozens)
B+1	LoadQueue Address_A	
…	…	
B+n	*Some code*	If (B+n+1)-(B+1)=delta is >= the # of cycles, stalling can be avoided completely!
B+n+1	JMP Address_A	
B+n+2	…	
…	…	

3. Example: JNE, assumed queue #0 active

Address	**Mnemonics**	**Explanations**
B	*Some code*	Load a free instruction queue (here: #1) with code starting at ‚A' in advance (Memory access may take hundreds of cycles, on an FPGA still dozens)
B+1	LoadQueue Address_A	
…	…	
B+n	*Some code*	If (B+n+1)-(B+1)=delta is >= the # of cycles, stalling can be avoided completely!
B+n+1	JNE Address_A	

B+n+2 Some code
… …

4. Example: RETURN, in queue <n>A return may ‚free‘ a queue immediately?
A+i RETURN

To make a reasonable judgement, it is imperative, to verify behaviour with an accordingl adapted compiler and simulator. This will allow for testing various scenarios with different cache sizes, pipeline lengths & weights (for different jump types & distances respectively).