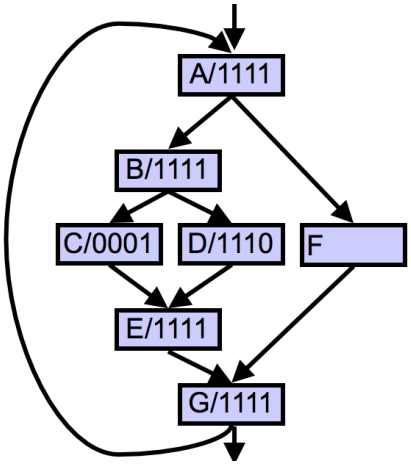
Homework 9

(Due April 14, 2020)

1. (12 points) Array processors, Vector processors, and VLIW architecture, all of them can achieve parallel execution. Please describe the difference between them. Please give the differences from both hardware (execution model) and software (program model) perspectives.

2. **(12 points)** Use brief words to describe the difference of the following terms: SIMD, Data flow, Thread(control)-level Parallelism, SIMT, SPMD.

3. **(12 points)** GPU can handle branch divergence by employing "SIMT-Stack" to record the PC and Active mask. Please read the related slides and text, and the following branch divergence flow. Then draw the "Stack Status" figure step by step when the instruction stream is from A to G.



4. **(13 points)** Solve 4.9 of the text (6th edition)

5. **(13 points)** Solve 4.13 of the text (6th edition)

6. **(13 points)** Solve 4.15 of the text (6th edition)

7. **(13 points)** Solve 4.16 of the text (6th edition)

8. **(12 points)** As Amdahl's law reveals, the speedup has an upper bound of 1/(1-f). This reminds us the early days of supercomputers. At that time, all major manufactures made supercomputers with up to 8 processors, citing Amdahl's law. However, nowadays, we have multicore processors with much more cores (Intel MIC 64 cores) on the same die.

a) What is the limitation of Amdahl's law? Is the "fixed-time" and "memory bounded" speedup model more appropriate for modern multicore design? Please answer the

questions and tell the reason.
b) What is the influence of data access in the scalability of multicore design, in terms of the number of cores?

Challenging Homework

C-1. We have introduced GPU microarchitecture. Specifically, we described different parts inside a SIMT core (Instruction Buffer, Warp Scheduler, Dispatch Unit, Register File, Different kinds of processing units, etc.), and discussed its pipeline step by step (Instruction fetch, Instruction decode, Warp scheduling, Operator collection, etc.). Please read the slides and all other related materials you can find to appreciate the complicity and the simplicity of the GPU design. GPU is so complex. It consists of all the architecture concept we have learned, pipeline, SIMD, SPMD, out-of-order-execution, multi-thread, memory hierarchy, private-global memory, etc. It is simple, in the sense it puts all these technologies together in synergy and in a manageable manner. Question: why we put so much complexity on a single chip, but not built a general/special purpose machine with a such complicity? In other words, what idea we can borrow from the GPU design to design a general HPC system?