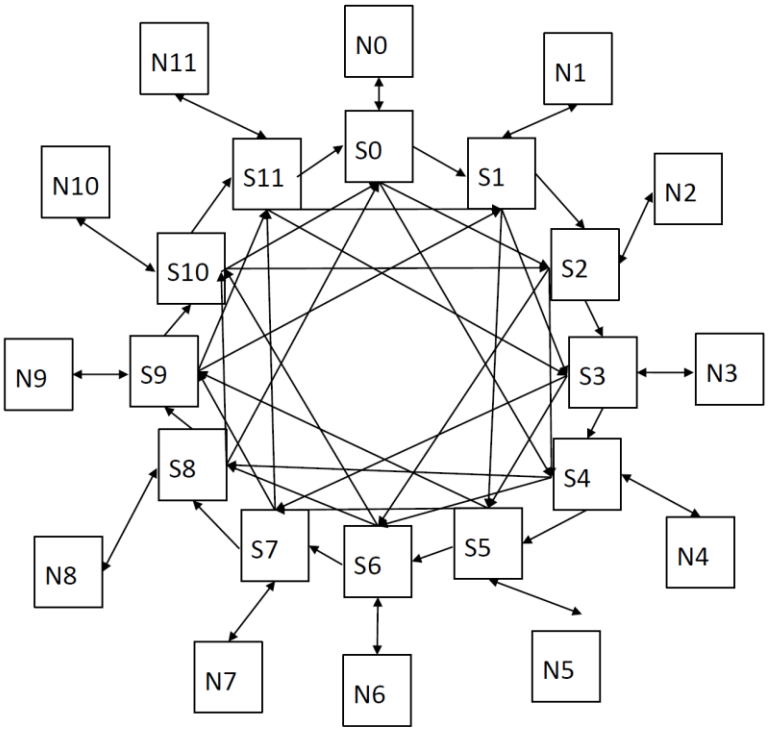
3. NoC: Consider the ring network shown in the figure below for the case of N=12 nodes / switches. Links between each node and its corresponding switch are full-duplex; other links are half-duplex. Each switch is connected to its corresponding node and 6 other switches: the two next to it, the two at distance 2, and the two at distance 4 (precisely, there are half-duplex links from any switch with index i to switch (i + 1) mod N, switch (i+2) mod N, and switch (i+4) mod N). Packets are always routed along a shortest route. The network uses wormhole switching with backpressure and no virtual channels; each input port buffers at most 2 flits.



a. The presented network architecture can be generalized to any larger number of nodes N that is a multiple of 4 . Compute as a function of N: (1) maximal shortest route; (2) the total number of flit buffers in nextwork switches. *[4P]*

The maximal shortest route (e.g. from switch 0 to switch N-1) is 2 (node to switch, switch to node) + N/4 – 1 (distance 4 links) + 1 (distance 2 link) + 1 (distance 1 link) = N/4 + 3.

Each switch has 4 input ports and 4 output ports. The number of flit buffers is thus 8N.

b. Assume that the network runs at 1Ghz clock speed with a link width of 16 bits. Each packet consists of one head flit, plus a number of data flits sufficient to send 16 bytes of data, plus a tail flit (the head and tail are used for routing and flow control). Under the assumption that only one node transmits, compute the throughput B_{single} of the network. Note: the throughput refers to the amount of data sent per unit of time. Also compute the minimum and maximum latency of a packet in ns as a function of N. *[4P]*

Throughput: W = 16 bytes; data length = 16/2 = 8 flits; packet length is 8 + 2 = 10 flits. The throughput is thus 8/10*16* 1 Ghz = 12.8 Gbit/s. (2P).

Latency: 10 flits + H - 1 cycles, where H is the hop count. For a network of size N, the minimum H is 3 and the maximum is N/4+3. Hence min is 12 ns and max is N/4 + 12. (1P for minimum, 1P for max)

c. Compute the total per-node throughput B_{total} of the network (remember: this assumes that all nodes transmit simultaneously in the most optimistic configuration of sender, receiver pairs). *[1P]*

Each note can simply send to the node next to it. Hence $B_{total} = B_{single}$

d. Compute again the total per-node throughput of the network as in part c; however, this time under the additional assumption that node i transmits to node (i + N – 1) mod N (i.e., node 0 transmits to node N-1; node 1 transmits to node 0; and so on). *[2P]*

Because nodes must use distance 4 links, the throughput is reduced by a factor of N/4 (that is, only 4 nodes can transmit simultaneously), down to 51.2/N Gbit/s.

e. **Bonus question:** consider the case N = 16. Assume that we relax the constraint on shortest route (i.e., nodes are allowed to select a non-shortest route if they want). Compute again B_{total} under the assumption of part d. *[2P]*

4. Energy/power optimization: Consider the following system:

- The memory hierarchy comprises a 32KB L1 data cache, a 256KB L2 cache, and DRAM main memory.
- HitTimeL1 = 1 cycle; HitTimeL2 = 5 cycles; HitTimeDRAM = 40 cycles.
- A program has a 80% hit rate in L1 and 75% hit rate in L2.
- You have the option of increasing the size of L2 cache. The size must be a power of 2. Quadrupling the L2 size reduces the L2 miss rate by a factor of 2 but increases the HitTimeL2 by a similar factor.

Determine the best L2 size for the program. [4P]

Let X be the best factor, such that newsize/256KB = X^2. (1P)

The average time is then HitTimeL1 + MissRateL1*(HitTimeL2*X + MissRateL2/X*DRAMTime) (1P)

Differentiating by X we obtain:

HitTimeL2 = MissRateL2*DRAMTime/X^2 (1P)

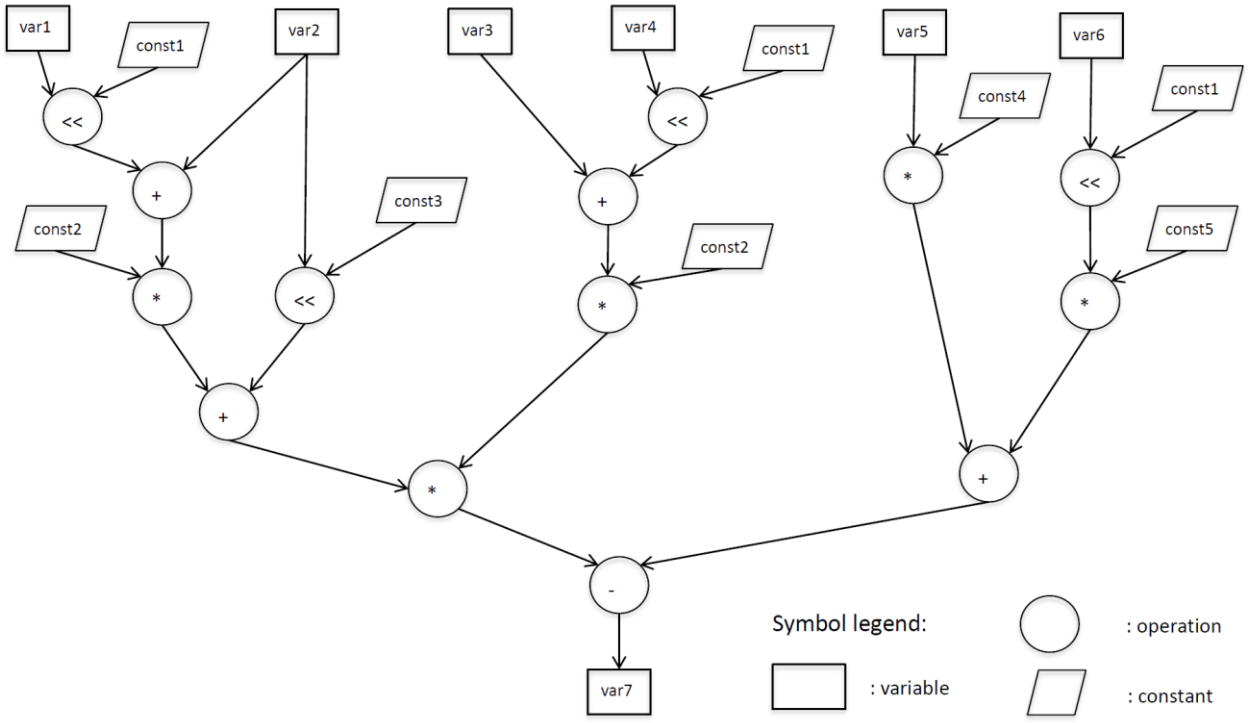
X^2 = MissRateL2*DRAMTime/HitTimeL2 = 0.25*40/5 = 2

Hence newsize = 512KB. (1P)

5.Custom Instruction: Consider the Data Flow Graph for basic block PROC shown below. The processor runs at 50Mhz, and **executes a single operation every clock cycle**. Assume the following timing in hardware:

- ADD, SUB, RSHIFT by a variable: 4ns
- RSHIFT by a constant: 0ns
- MUL: 14ns
- Ignore delays for registers

The processor can implement custom instructions, each with 2 reads / 1 write.



a. Determine the custom instruction that results in the highest speedup for the basic block. You can draw the cut corresponding to the custom instructions directly on the figure. Determine the execution time of the basic block in nanoseconds with or without the custom instruction; briefly show your computation/rationale below. *[4p]*

2P for best one (reusable), 1P for cut with speedup 3; 2P for the execution time

The best cut is the one that includes CONST1, CONST2, the <<, + and *; the cut is repeated twice, with a critical path length of 18 (one cycle), so it saves 4 cycles total. 14 cycles (280ns) software, 10 cycles (200ns) hardware.

b. Next, assume that the processor can support two custom instructions rather than a single one. Determine the set of two custom instructions that result in the highest speedup for the basic block, and determine the execution time of the basic block in nanoseconds. As before, you can draw the cuts directly on the figure; also briefly show your computation/rationale below. *[3p]*

2P for best cuts, 1P for suboptimal solution, 1P for the execution time

The best two cuts is the same as for part a, plus the cut on the right. This save 4 + 3 = 7 total, so the cut takes 14-7 = 7 cycles (140 ns).

c. Finally, assume that the processor executes the program with the following CFG. The number within each basic block denotes its execution time in ns. The number on each backward edge determines the number of times the edge is taken in the program. Determine the speedup for the program achieved by selecting the two custom instructions for the PROC basic block at point b. *[3p]*

The total time without custom instructions is 300 + 200 + (40+60)*5 + (20+40+280)*50 = 18 us

The time saved by the custom instructions is 140*50 = 7 us

Hence the speedup is 18/(18-7) -> 63.6%.

Note: since the specification regarding the number of inner loop iteration was confusing, we also admitted solutions that consider 10, instead of 50, inner loops. This results in total time: 300 + 200 + (40+60)*5 + (20+40+280)*10 = 4.4 us, and saved time of 1.4 us, with speedup 4.4/(4.4-1.4) = 46.67%.