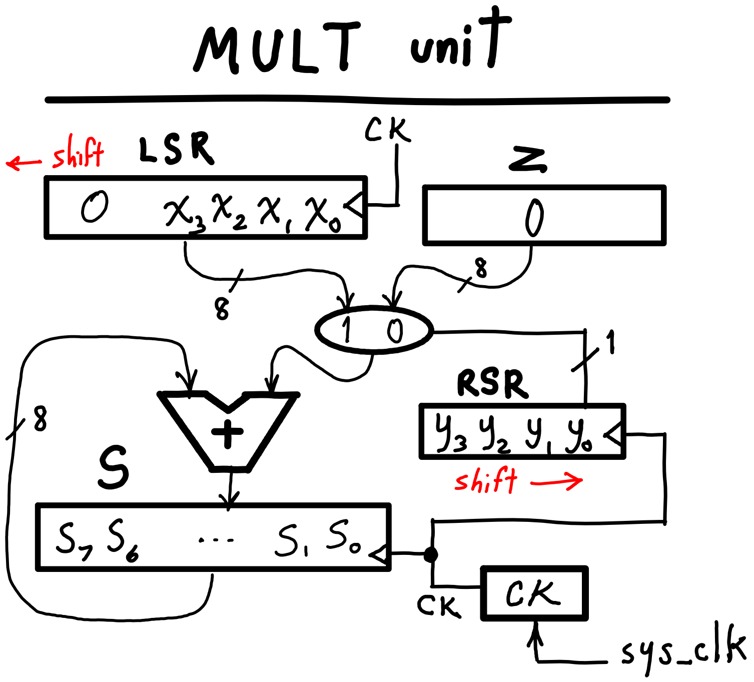
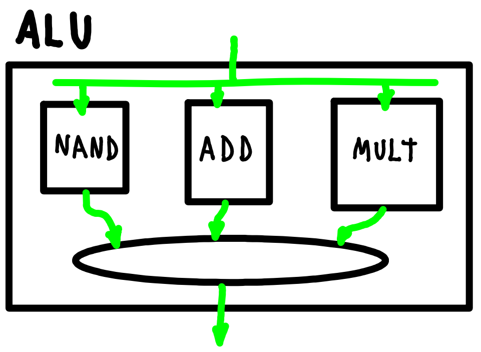
Q. At right is the multiplier unit of a 4-bit machine (4-bit x and y). MULT has its own local clock, **CK**, to control its shift-add cycles, which is started by sys_clk if the current opcode is MULT. In each **CK** cycle, the adder performs a partial sum, x shifts left, y shifts right, and the partial sum clocks into the 8-bit result register, **S**. MULT is considered to start operating just after x and y are clocked into their respective registers, and finishes just after the final result is clocked into **S**. The ADDER is a 4-stage, ripple-carry adder with a delay per stage of 4 ps. The MUX has a delay of 4 ps.



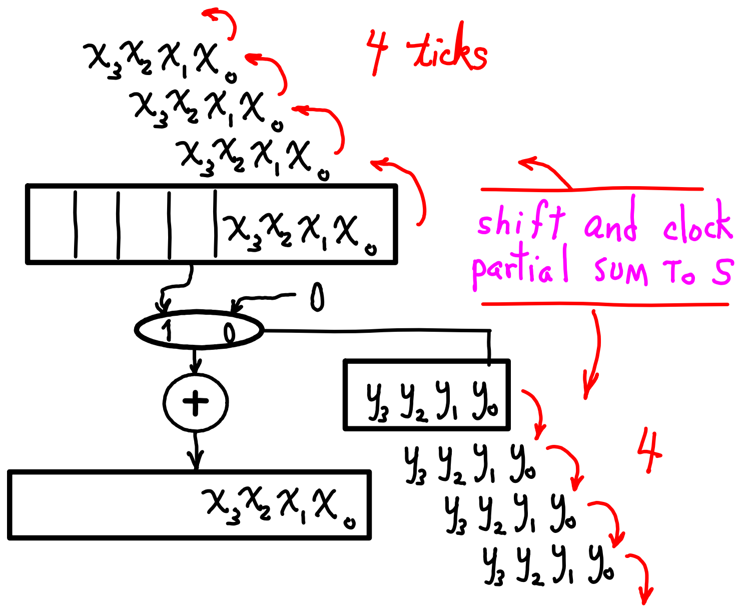
Suppose this machine's ALU consists of NAND, ADD, and MULT units, and an ALU_MUX, with these delays:

NAND:	6 ps
ADD:	25 ps
ALU_MUX:	6 ps

Q. In MULT, the product will be in **S** after how many **CK** cycles?



4 cycles. Each tick does an add and stores partial sum in S. When y_3 exits, MULT is complete.



Q. All ALU instructions takes **one sys_clk clock cycle**. Supposing the clocks run as fast as possible, and the ALU determines the **sys_clk** and **CK** rates, what is the machine's **sys_clk** rate, CR?

MULT is the slowest, so it determines CR. From the time CK ticks to the time S's inputs are ready is (4 adder stages) X $4\text{ps}/\text{stage}$ + 4ps MUX delay

Total ALU delay for MULT is

(4 CK ticks) X (20 ps / CK tick) = 80 ps, plus 6 ps delay for ALU_MUX = 86 ps.

which is sys_clk's cycle time. $CR = 1/86\text{ ps} = (1/86)\,10^{11}$ (cycles/sec) $\approx$ 100 GHz.

Q. Claire notices that the MULT unit could be pipelined without changing its hardware except its clocking: Eliminate CK and clock MULT with sys_clk instead. A MULT instruction would then take 4 sys_clk cycles to finish. Claire claims that if we also decrease sys_clk's cycle time, the machine's overall performance will improve. What delay now determines the CR? What is the new CR? What is the CR speedup?

The delays for NAND and ADD are (6+6) and (25+6) ps. MULT's delay is now (20+4+6) ps. So ADD's delay is now the slowest path. The new CR is (1/31 ps). $S_{CR} = CR_{new}/CR_{old} = (1/31\,ps)/(1/86\,ps) \cong 2\,3/4$.

Q. Suppose a job mix of 10% NAND instructions, 50% ADDs, and the rest MULTs. What are the average CPIs for the old machine and Claire's new machine? What is the speedup of the new machine relative to the old machine?

$$\overline{CPI}_{old} = \frac{\text{Total cycles}}{n \text{ instructions}}$$

$$= \left[n_{ADD}\left(\frac{1\text{ cycle}}{ADD}\right) + n_{NAND}\left(\frac{1\text{ cycle}}{NAND}\right) + n_{MULT}\left(\frac{1\text{ cycle}}{MULT}\right)\right] 1/n$$

$$= (\%_{ADD} + \%_{NAND} + \%_{MULT})(1\text{ cycle}/\text{instruction}) = 1\ (\text{cycle}/\text{instr.})$$

$$\overline{CPI}_{old} = \left[n_{ADD}\left(\frac{1\text{ cycle}}{ADD}\right) + n_{NAND}\left(\frac{1\text{ cycle}}{NAND}\right) + n_{MULT}\left(\frac{4\text{ cycle}}{MULT}\right)\right] 1/n$$

$$= \%_{ADD}(1) + \%_{NAND}(1) + \%_{MULT}(4)$$

$$= 0.1 + 0.5 + 0.4 \times 4 = 2.2\ (\text{cycle}/\text{instr.})$$

$$S_{new\text{-}old} = \frac{T_{old}}{T_{new}} = \frac{n\ (\text{instr.}) \times \overline{CPI}_{old} \times (1/CR_{old})}{n\ (\text{instr.}) \times \overline{CPI}_{new} \times (1/CR_{new})} = \left(\frac{1}{2.2}\right) \times \left(S_{CR} = 2\,3/4\right)$$

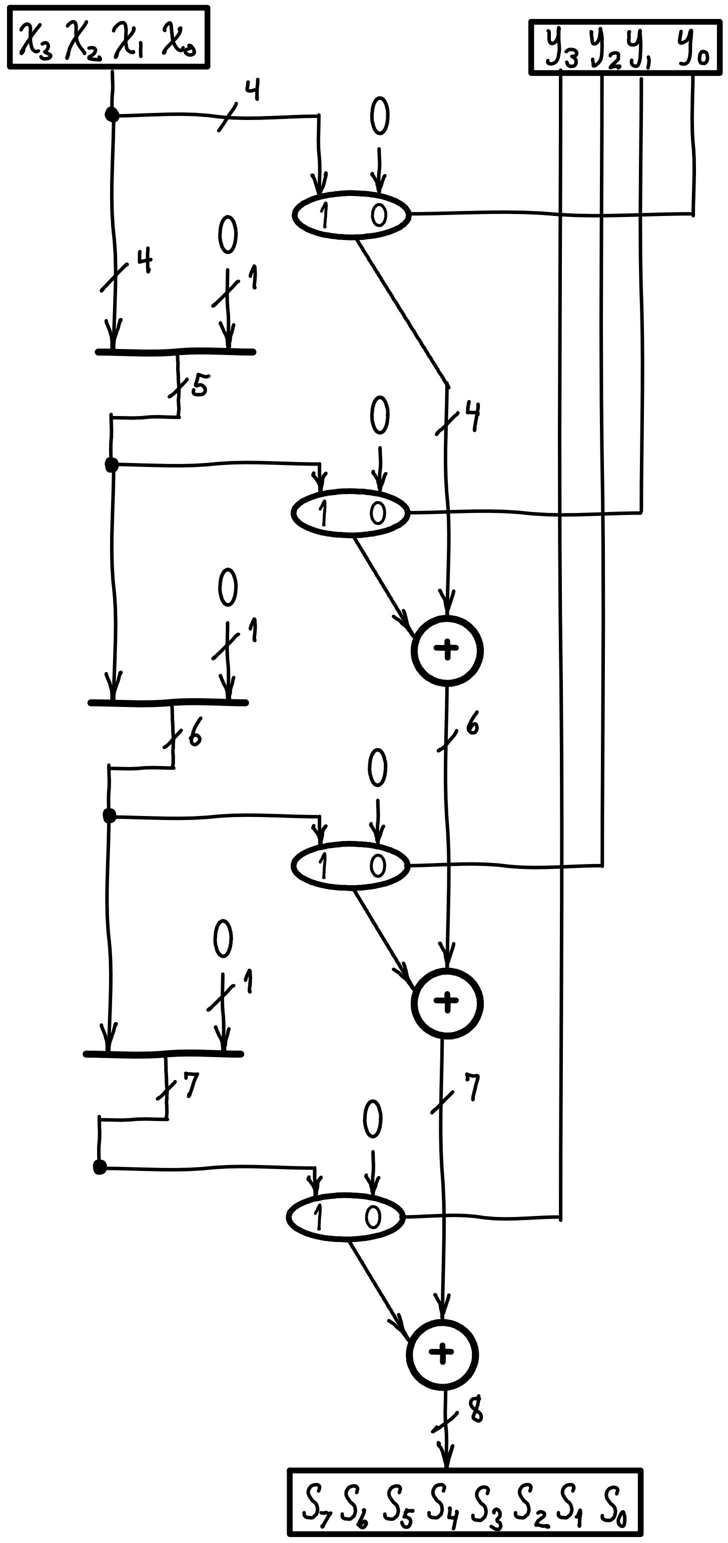
$\approx$ 25% faster

Q. What job mix gives the maximum speedup? The minimum? What are the speedups?

Max mix = 100% (ADD, NAND) $\Rightarrow \overline{CPI}_{new} = 1 \Rightarrow S_{new\text{-}old} = \left(\frac{1}{1}\right)(2\,3/4) = 2\,3/4$

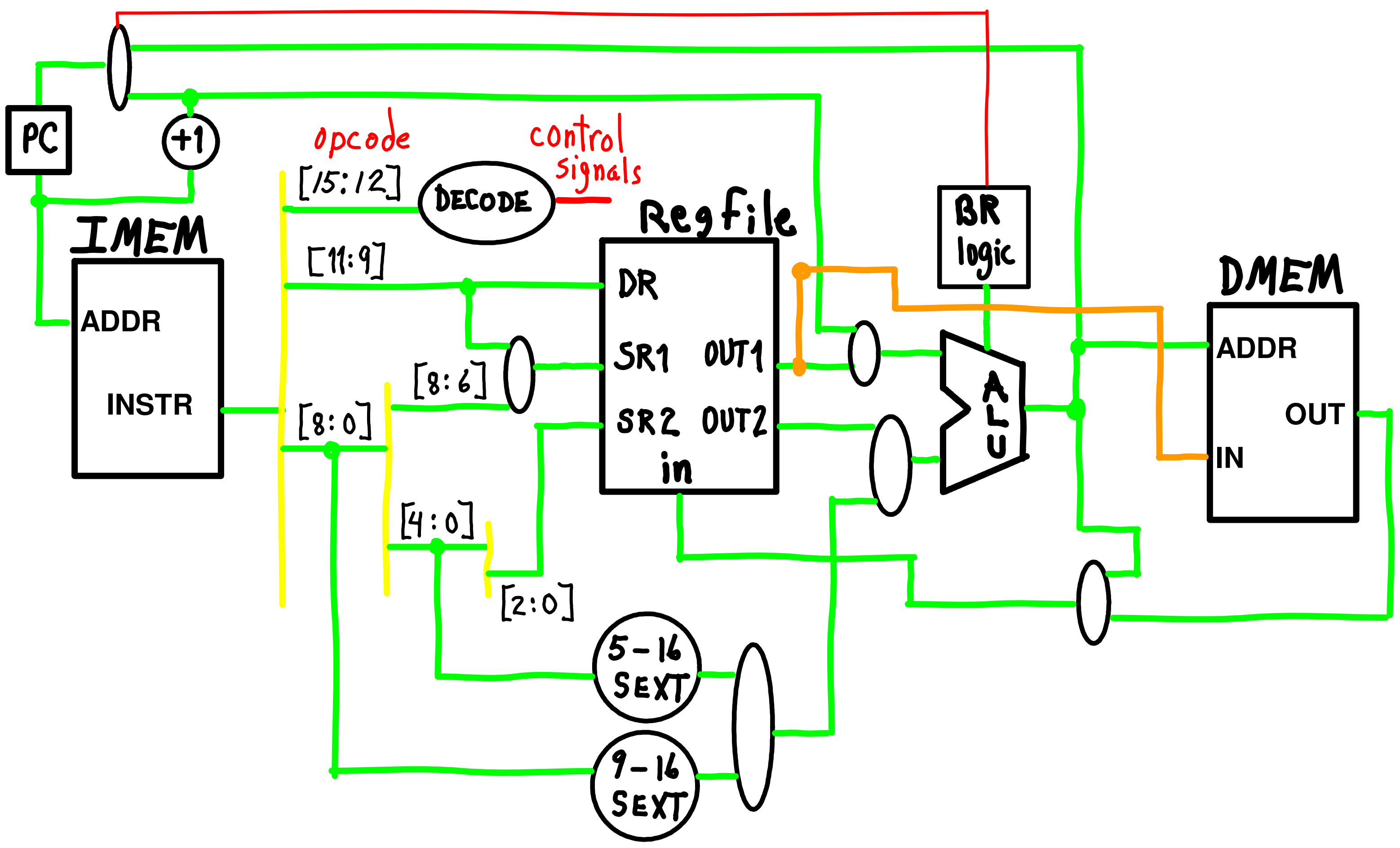
Min mix = 100% (MULT) $\Rightarrow \overline{CPI}_{new} = 4 \Rightarrow S_{new\text{-}old} = \left(\frac{1}{4}\right)(2\,3/4) = 11/16 \approx 2/3$

Q. Claire suggests that by adding hardware to MULT, it can be pipelined so that multiple MULTs can execute simultaneously. To get started, a one-cycle implementation of MULT is given below. Add pipeline registers where appropriate. Give the speedup for the job mixes above.



The MIPS instruction set is designed to accomodate pipelining. The LC3 instruction set is very similar. In fact, it is almost a 16-bit version of MIPS with a very reduced instruction set. We will reduce the LC3 instruction set even further to only LD, BR, and ADD. (These instructions are specified in our lecture notes "Lec-1c-hardware.pdf", and in our archive "projects/LC3trunk/docs/LC3-3-PP-Append-A.pdf".) The one-cycle MIPS implementation (see Patterson & Hennessy, Chapter 4) has two memories: IMEM (for instructions) and DMEM (for data). Below we show a similar implementation for the LC3 ISA. It is different from MIPS in that the ALU is used to produce the BR target address. (Although we are not including any store instructions, a path is provided (orange)). Delays are: IMEM and DMEM, 200 ps; RegFile 100 ps; ALU, 150 ps; Writeback, 100 ps.

Q. Provide pipeline registers to implement pipelining for the LC3 circuit shown below.
Q. For the instruction mix (50% ADD, 30% LD, 20% BR), calculate the new CPI and speedup. You can ignore pipe fill and drain overhead.
Q. What problem do you see arising if we tried to include LDI in our implementation?



ADD DR, SR1, SR2
ADD DR, SR1, imm5

Encodings

15 12	11 9	8 6	5	4 3	2 0
0001	DR	SR1	0	00	SR2

15 12	11 9	8 6	5	4 0
0001	DR	SR1	1	imm5

Operation

```
if (bit[5] == 0)
      DR = SR1 + SR2;
else
      DR = SR1 + SEXT(imm5);
setcc();
```

LDI DR, LABEL

Encoding

15 12	11 9	8 0
1010	DR	PCoffset9

Operation

```
DR = mem[mem[PC† + SEXT(PCoffset9)]];
setcc();
```

LD DR, LABEL

Encoding

15 12	11 9	8 0
0010	DR	PCoffset9

Operation

```
DR = mem[PC† + SEXT(PCoffset9)];
setcc();
```

BRn	LABEL	BRzp	LABEL
BRz	LABEL	BRnp	LABEL
BRp	LABEL	BRnz	LABEL
BR†	LABEL	BRnzp	LABEL

Encoding

15 12	11	10	9	8 0
0000	n	z	p	PCoffset9

Operation

```
if ((n AND N) OR (z AND Z) OR (p AND P))
   PC = PC‡ + SEXT(PCoffset9);
```