University of Pennsylvania
Department of Electrical and System Engineering
Computer Organization

ESE534, Spring 2010 Assignment 6: Defect Tolerance and Density March 15

Due: Monday, March 22, 12:00PM

1. We are trying to build a large memory out of a collection of memory banks. To focus the problem on the key effects, we will take w and d as fixed and explore row sparing (r_{spare}). Our goal for this problem is to select r_{spare} to maximize the **expected** number of yielded memory banks on a die of fixed size. All banks must provide d words of data, so a bank that does not have sufficient spares to replace its failed rows will be considered unusable. Set $d = 2^{10}$, $w = 2^{10}$, and $A_{chip} = 2^{28}$.

   We use the same basic area model as before, but expand it to include r_{spare} spare rows. We assume spare rows are more expensive than normal rows since they must have a programmable address to replace the normal rows.

   $$A_{bank} = (d + 2 \times r_{spare})(2\log_2(d) + w) + 10w \tag{1}$$

   We build an entire chip by composing banks.

   $$A_{chip} = N_{banks} \times A_{bank} \tag{2}$$

   Here we assume the area for interconnect to address the banks is captured in the bank area.

   Since we are focusing on row sparing, we will simply specify the probability of yielding a row, P_{row}. Assume both normal and spare rows yield with probability P_{row}. In practice, we might compute P_{row} based on the probability of yielding each memory bit in the row and the probability of yielding the row decoder.

   As one simplification for this problem, we are assuming no failure takes out an entire column.

(a) What is the probability of yielding a bank (P_{bank}) as a function of r_{spare}? [an equation]

(b) How many banks can you put on a chip as a function of r_{spare}? You cannot use fractional banks, so this should be an integer. [an equation]

(c) What is the expected number of repairable blocks on the chip as a function of P_{bank}? [an equation]

(d) For each of the following row yield rates, P_{row}, identify the optimal number of spares and the resulting expected number of repairable memory banks on a chip. [Describe how you arrive at your answer, include supporting data, and provide a completed table.]

P_{row}	r_{spare}	E(repairable banks/chip)
1.00		
0.9999		
0.999		
0.99		

2. For this problem, we will compare the energy of three Finite-Impulse Response (FIR) implementations with different levels of programmability.

Component	Energy
ALU	$8w$
Adder	$4w$
Equal-zero	$2w$
Multiply	$4w^2$
Memory Bank	$d(\log_2(d) + w) + 10w$
Single-Output Mux	$N_{in} + \log_2(N_{in})$

We assume program counter energy is negligible and omit it for this model.

An FIR operation is computed on a stream of input samples, x_i, to produce a stream of outputs, y_i, as follows:

$$y_i = \sum_{j=0}^{j=(n-1)} (x_{i-j} \times c_j) \tag{3}$$

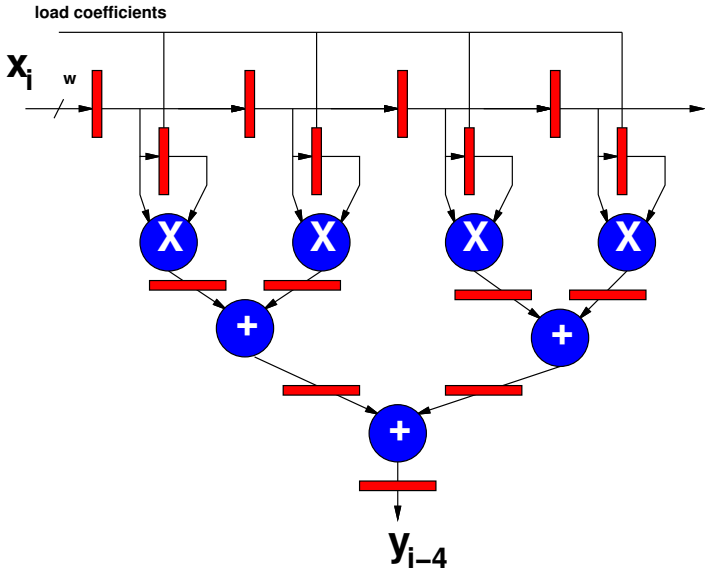
c_j's are constants that determine the particular FIR computed. That is, we perform n multiplications and $n-1$ additions to compute each output result y_i.

The datapath includes an input path where the sample value (x_i) can be loaded and an output into which to load the resulting y_i. The instruction should include an output-write bit to signify when y_i should be loaded, and one multiplexer selecting sources into the data memory should support the x_i input.

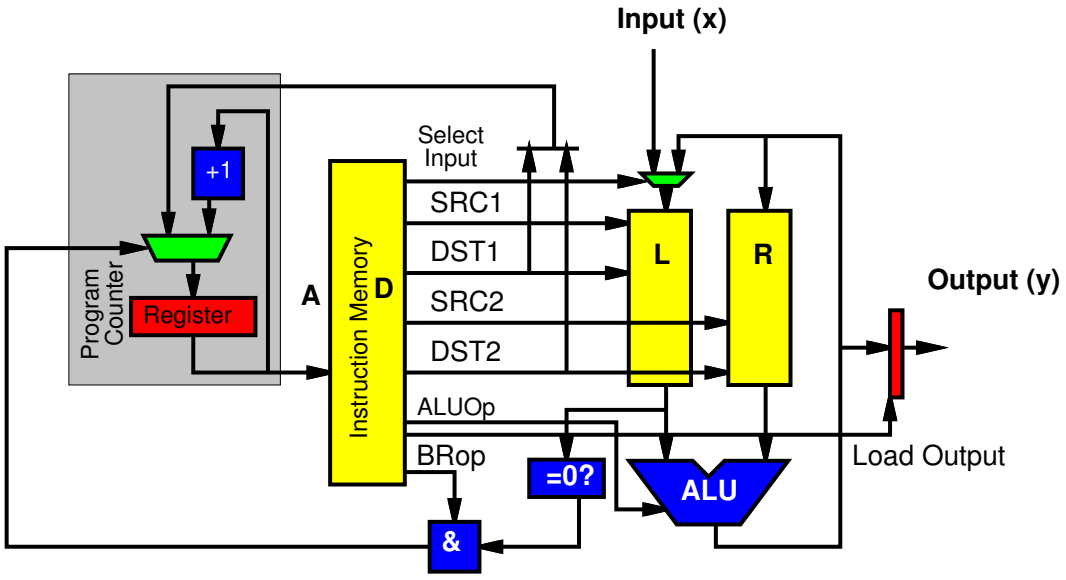
The multiplication on two w-bit values produces a $2w$-bit result. The sum of n w-bit values may require $w+\log_2(n)$ bits to represent. As a result, we might want multiple bit widths in our datapath and memories. For simplicity in this problem, we will assume we select a single, uniform w for the multiplier, data memory, and ALU widths. This may mean that we store some values (c_j's and x_i's) that are smaller than w to avoid overflowing the result of the multiplication and summation.

For this problem, we consider three cases:

(1) Hardwired FIR – only the coefficients are programmable. As shown, coefficients can be loaded using the same shift path for x_i inputs. For modeling, assume all registers and adders are w-bits wide and the multiplier is a $w \times w$ multiply. Figure shows an $n = 4$ hardwired FIR.



(2) ALU-based programmable datapath – single ALU datapath similar to previous assignments. We add a branch-zero instruction bit that allows the PC to be conditionally loaded from the instruction memory based on a value read from the data memory.

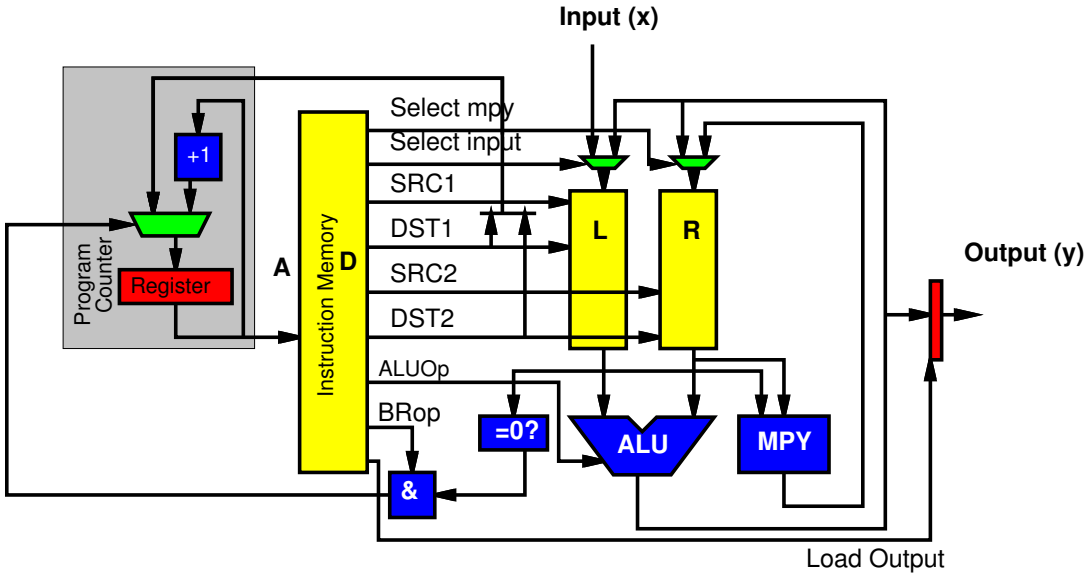


Here is the code snippet for performing a single multiplication:

init-constants:	L4,R1←R1-R1	// R1=0, L4=0
	R1←L4+1	// R1=1
		// Assume operands in L1, L2
	L3←L3-L3	// L3=0, clear sum
mpy-top:	if (L2==0) PC=mpy-complete	// exit if done
	else PC=PC+1	
	R2←L2 AND R1	// extract L2[0]
	R2←L4-R2	// if (L2[0]==0) R2=0 else R2=-1=0xffffffff
	R2←L1 AND R2	// if (L2[0]==0) R2=0 else R2=L1
	L3←L3+R2	// conditionally add shifted operation into sum
	L1←L1<<1	// shift L1 up for significance of next L2 bit
	L2←L2>>1	// shift L2 down so next bit is in zero position
	if (L4==0) PC=mpy-top	// unconditional branch to continue mpy
	else PC=PC+1	
mpy-complete:		// result in L3; continue with next operation

You do not have to use this multiply routine. Nonetheless, this does suggest a sample format for expressing your instructions. Li refers to a data in the left (L) memory bank, and Ri refers to one in the right (R) memory bank. Note that the datapath allows you to write values to either or both banks, hence the first instruction above exploits that to write a 0 into different slots in each of the banks at once. Assume you can write to a non-existent bank address when you do not want to write the value to any slot in the bank.

(3) Programmable datapath with a single, hardwired multiplier – add a hardwired $w \times w$ multiplier to the datapath (in parallel with the ALU) with additional data routing to select the output of the multiplier.



(a) Write the code to perform the FIR operation on the multiplier-less programmable datapath (2). Try to minimize the energy for this computation. As you will calculate, energy depends on the number of instructions in the instruction memory, the number of data items stored in the data memory, and the number of cycles required to perform the task.

(b) Write the code to perform the FIR operation on the programmable datapath with the multiplier (3). Try to minimize energy for this computation.

(c) Identify the minimum (i) size of the instruction and (ii) data memories necessary to support the code written above and (iii) the number of cycles required to compute the n-point FIR on a w-bit datapath (you may assume the datapath width and task width are matched). [an equation with w and n as parameters].

(d) Give equations for the total energy to perform an n-point FIR on w-bit samples (x_i) with w-bit coefficients (c_j) for each of the three cases identified above. [an equation with w and n as parameters].

(e) For $w = 16$, $n = 8$, calculate and report the energy for each of the three cases identified above.