California Institute of Technology
Department of Computer Science
Computer Architecture

CS184a, Winter 2003

Assignment 2: Space-Time Multiply

Monday, January 20

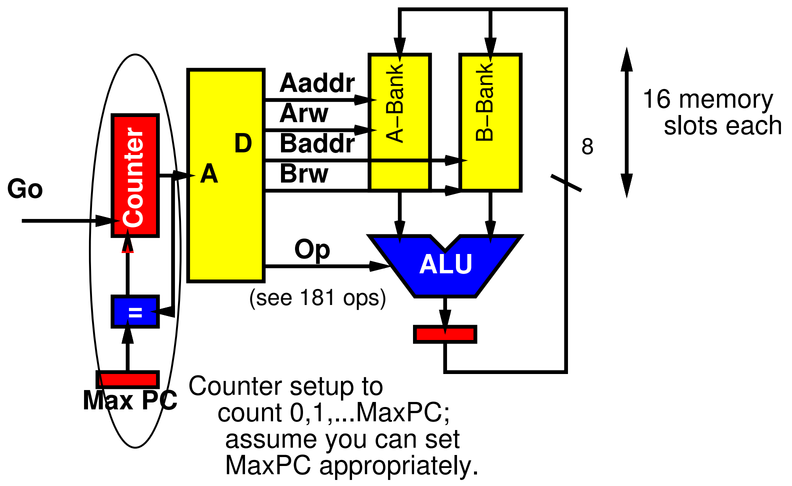
Due: Monday, January 20, 9:00AM

Everyone should do all problems.

We saw in lecture how to build various adders. In this problem, I'm asking you to review various techniques for building multipliers.

1. Show an $n \times n$ spatial multiplier built out of simple, ripple-carry adders. (may draw for $n = 4$)
   - What is the latency? (in terms of n)
   - What options exist for pipelining this design? (describe the space of designs available and the register cost-throughput tradeoff in terms of n)
2. Consider using a redundant representation for values inside the multiplier. In particular, consider that you represent a number as two binary words which, when added produce the value you are representing.
   - How does this change the latency of a single addition?
   - Show the resulting, spatial multiplier (may draw $n = 4$) which starts with numbers in standard form, but uses this representation internally. What is its resulting latency, area, and pipelineability?
   - What do you need to do on the output of the multiplier to convert the result back into normal form? What is the overall latency and area of the multiplier including this final return to standard form?
   - How do you minimize the latency of this multiplier from input to final output in normal form.
3. Show a shift-and-add multiplier design (may draw for $n = 4$):
   - Take as input two numbers and a control signal that tells it to start.
   - Produces as output the $2n$-bit product of the two numbers, along with a signal indicating that the multiplier is busy and a signal indicating when the output result is valid.
   - Uses a single adder and $O(n)$ state registers.
   - Report on latency and throughput.
   - How does this change if you use the redundant representation above.

4. Assuming the datapath below, based on the simple memory-based programmable datapath in class, show the (vertical) μcode to implement: $C = (A \times B)$. (A and B are 4b values, C should be an 8b result)



You may assume the ALU behaves like a 74181 (functions attached) of width w=8; it has outputs equal and not carry out, in addition to the w datapath output bits. For all μcode, your solution should resolve to individual bits. You may use symbolic names as long as you give definitions for the symbolic names as part of your solution.

e.g. For the ALU-Operation to add A1 and B2, you would need to specify:

F_3	F_2	F_1	F_0	$\overline{C}$	M	Arw	Brw	A[3:0]	B[3:0]
1	0	0	1	1	0	0	0	0x01	0x02

You could define:

$$\text{ADD} = F_3{:}F_2{:}F_1{:}F_0{:}\overline{C}{:}M{:}\text{Arw}{:}\text{Brw}{=}{=}10011000$$

and specify: ADD 0x01 0x02

- A starts in A-bank, slot 0;
- B starts in B-bank, slot 0;
- C should end up in A-bank slot 1;
- The ALU has an arithmetic shift right operation beyond the operations listed for the 181. Assign it a suitably unique function code.

5. Modify the datapath to allow execution to branch.

   (a) show revised datapath.

   (b) show new μcode which implements the same multiplication function now using the branching capability.

   (c) compare the cycles required with the branchless case.