California Institute of Technology
Department of Computer Science
Computer Architecture

CS184a, Fall 2000 — Assignment 3: Microcode — Monday, October 9

Due: Monday, October 16, 10:30AM

Everyone should do all problems.

You should use a schematic capture or drawing program for circuits.

For this assignment, you may assume the ALU behaves like a 74181 (functions attached) of width w=32; 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:

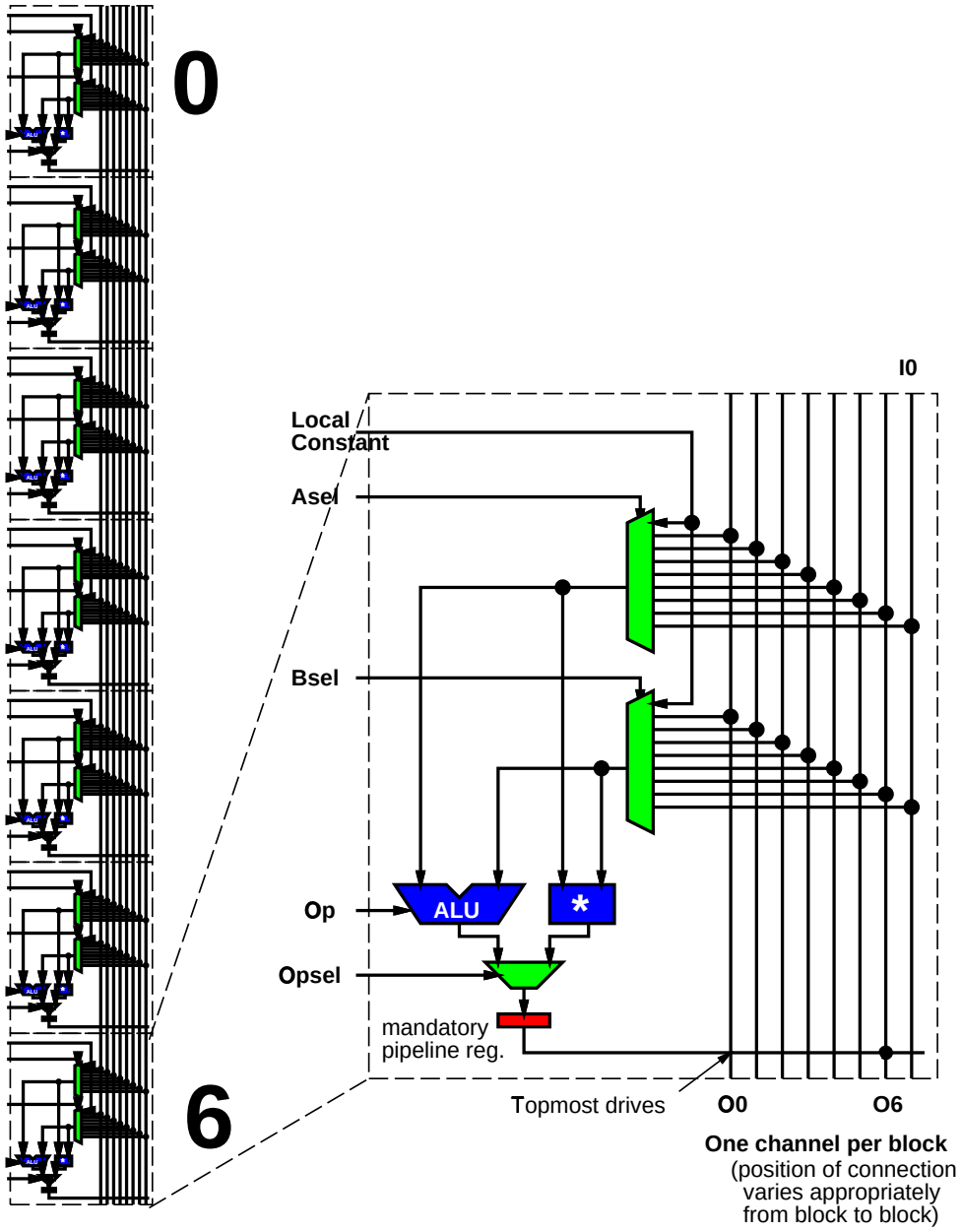
F_3	F_2	F_1	F_0	$\overline{C}$	M	A[3:0]	B[3:0]
1	0	0	1	1	0	0x01	0x02

You could define:

$$\text{ADD} = F_3{:}F_2{:}F_1{:}F_0{:}\overline{C}{:}M{==}100110$$

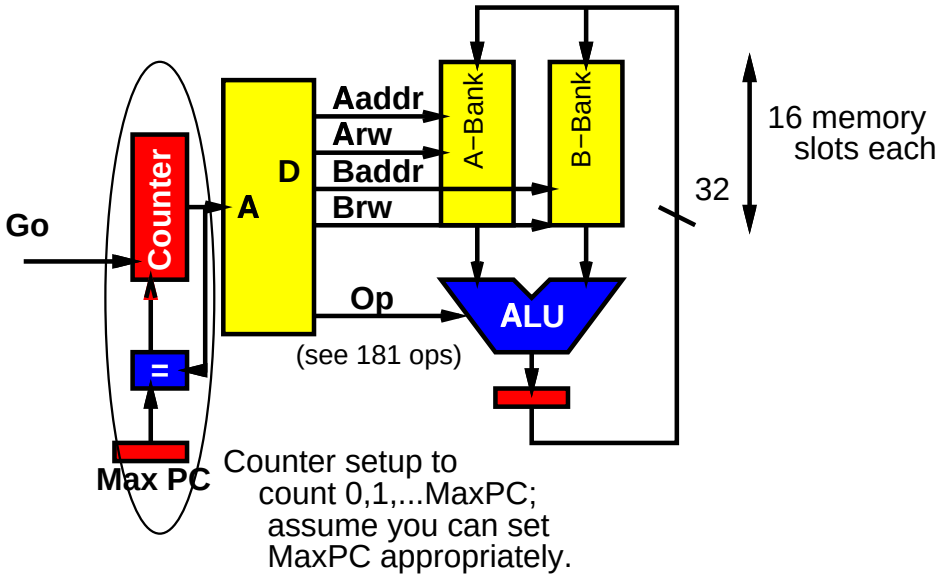
and specify: ADD 0x01 0x02

1. **Spatial Programmable (Horizontal μcode)** – Fill in the instruction bits necessary to make this programmable datapath implement the quadratic equation: $y = Ax^2 + Bx + C$. A, B, and C are constants to be embedded in the implementation (as local constants). x comes in on I0. y should come out on $O6$. Note pipeline register in datapath. Take in one x on every cycle and produce one y on every cycle. Assume all data busses are 32b wide.



2. **Branching** –

(a) Assuming the datapath below, based on the simple memory-based programmable datapath in class, show the (vertical) μcode to implement: $C = (A + B) \bmod D$.



- D is in B-bank, slot 0;
- A starts in A-bank, slot 1;
- B starts in B-bank, slot 1;
- C should end up in A-bank slot 0;
- $A \leq D$, $B \leq D$
- $0 < D < 2^{(w-1)}$
- The ALU has an arithmetic shift right operation beyond the operations listed for the 181.

(b) Modify the datapath to allow execution to branch.

i. show revised datapath.

ii. show new μcode which implements the same modulus function now using the branching capability.

iii. compare the cycles required with the branchless case.

3. **Unbounded Memory** – Implement a simple prime number seive. Show datapath and μcode.

   - Modify your datapath above to add a divider and a memory.
   - The divider should take in an A and B value like the ALU (in parallel with the ALU) and produce two outputs, the dividend and remainder. (you can assume the divider works and works in a single cycle; you don't need to show any logic for the divider.)
   - The new memory will be a single ported SRAM, a power of two in size (not to exceed 2^{30}). You can assume someone placed the size of the memory in B15 before you start.
   - The seive works by creating the list of all primes working up from 2. For each new candidate prime, you simply test if it is divisible by any of the primes already in the partially completed list of primes (*i.e.* all primes less than the current candidate prime). When you find a prime, you add it to the list.
   - Your program should stop when the memory is full or when you exhaust your integer representation (2^{31}).
   - There are almost certainly more clever algorithms; stick with this one. The idea is to design a simple datapath including a memory and control. The specific example is intended only to be motivational.