California Institute of Technology
Department of Computer Science
Computer Architecture

CS184a, Fall 2000

Assignment 4: Instructions

Monday, October 16

Due: Monday, October 23, 10:30AM

Everyone should do problems 1–7. (If any of these take you more than 30 minutes to see the path to the solution, definitely let me know.)

You should use a schematic capture or drawing program for circuits (where appropriate).

1. Consider two possible single-bit functional units: a 3-input NAND and a 3-input LUT. Consider implementing a 5-input parity function (XOR5).
   - How many instruction bits are required to specify the computation for each instruction in the two cases?
   - How many instruction cycles will it take to implement the parity function in each of these cases? (show ucode to support)
   - How many total operation instruction bits in memory are required to describe this operation in each case?
   - What additional bits exist in your full primitive instruction (pinst)?
   - Assuming the additional bits are the same in both cases, at what number of additional bits does the memory space to hold the two instruction sequences hit parity? (may not be an integer, so identify the break point)
2. Consider the spatial programmable from ucode.1. You can equally imagine a sequential (vertical) microcoded version of this which had a single functional unit and a memory instead of interconnect. It would take one cycle per operation and have a maximum throughput of one result every m cycles, where m is the number of operations needed. Compare the total number of instruction bits required in these two cases.
3. Consider your non-branch-enabled and branch-enabled datapath from problem ucode.2. Now consider implementing a multiplication on the simple ALU (like arith.b.2). Concretely, consider a 15b$\times$16b$\rightarrow$30b multiply of positive values on the 32b ALU. [I don't think you'll need the arithmetic shift ops I erroneously omitted from ucode.2a, but you're welcome to assume them here if you think they'll help.]
   - Non-branching case: how many bits in each instruction? how long is the instruction sequence? how many instruction bits in memory?
   - Branching case: how many bits in each instruction? how many instructions do you need? how many total instruction bits in memory? how many cycles to perform the multiplication?

4. Consider your final ucode.3 datapath with branching, division, and load-store. Your operation includes 6 bits to specify the ALU function plus additional bits to specify branching and memory operations. I'll assume you used 4 more bits to be concrete. Your "function" specification is now 10b wide. Do you really use all $2^{10} = 1024$ operations? with equal regularity? Now, consider limiting yourself to a 4b opcode. That means you only get 16 unique operations.

   - Identify the 16 operations you want to keep (refer to part c for goal, but keep the full power of the ucode.3 datapath), their encodings, and their expanded 10b (or whatever it is for you) decoding.
   - In terms of 2-input gates, how big is the decoder required to expand your 4b opcode into the decoded control bits for the ALU and branching hardware? (you may simply write boolean equations to justify; I don't need to see a circuit diagram. Parenthesize your boolean equations appropriately to show where the gate counts come from.)
   - Rewrite your branching multiply from the previous example in terms of this restricted opcode space. How does the number of unique instructions, cycles, and number of bits in memory change?
   - Describe how you could use a memory (or ROM) as the decoder. How large is this memory?

5. Consider again your ucode.3 datapath. Describe qualitatively the effects on instruction bits per cycle and total instruction bits per operation for each of the following. There are some effects you probably don't know off hand, so identify the key variables and write the inequalities you would use to decide which resulted in the least instruction bits in memory or instruction bits issued (cycles to compute $\times$ instr bits/cycle).

   - reducing/increasing the number of registers addressed in the register file
   - changing the number of registers appearing in the instruction
     (a) 1 op: accum = r op accum (also ops r = accum, accum=0, accum=r)
     (b) 2 op: r1 = r1 op r2
     (c) 3 op: r1 = r2 op r3

6. Describe how subroutines also serve as a technique for reducing the number of instruction bits necessary to describe a computation.

7. Let's say you have an old design which is 70% instruction memory, and you've used techniques like the ones above to reduce the instruction memory size by 35% while keeping other things the same. Assume, for simplicity, technology is continuously improving such that you get a reduction in feature size by a factor of 2 every three years. How many months of technology scaling give the same size reduction as your improved design?

extra 1 Describe another technique which can be used to decrease issued instruction width and/or total instruction bits to describe an application.