California Institute of Technology
Department of Computer Science
Computer Architecture

CS184b, Winter 2000 Assignment 6: VLIW/EPIC Thursday, February 8

Due: Thursday, February 15, 5:00PM

What I want to do here is to give you a chance to develop a feel for both Trace Scheduling and these organizations/architectures. Working with a small piece of code, you'll see the effects/benefits of the structural restrictions (allowances) and the code transformations.

To do this, we're going to take one small subroutine with branching (included at end). Note that M1, M2 are powers of two (convince yourself this invariant holds). I think this will be easier if you make sure to exploit this observation and use it to strength reduce the MOD and divide to be masks and shifts.

For simplicity, unless otherwise noted, we assume each "cluster" on the VLIW can perform any single operation (load/store, branch, floating or integer ALU operation including multiply) on each cycle. Assume one cycle to move a data item between clusters. Assume everything that executes in the same long instruction with branch(es) completes.

- create and show the trace flowgraph for this routine (in terms of basic cluster operations).
- schedule for a 4-issue VLIW where you resolve control flow (branches) before executing code behind the branch (*i.e.* don't use trace scheduling, multiple branches, etc.)
- identify the primary trace.
- limit yourself to a single branch per long instructions; trace-schedule the primary trace; do not schedule the off-trace portions of the flow graph, but do add patch up flowgraph nodes so that the on-trace schedule and off-trace flowgraph transitions preserve the original semantics of the program; show your resulting schedule/flowgraph.
- repeat your trace scheduling removing the single branch restriction; again show the resulting schedule/flowgraph.
- unroll the loop once and repeat schedule.
- convert this primary trace to EPIC (using additional tricks like predicated execution as you see fit...so you may not strictly be implementing the same primary trace); make instruction groups as large as you can. For sake of cycle comparison (next), assume a two-bundle-issue machine with the restrictions on syllables within a bundle given on p. 19 (simplify by not distinguishing integer and floating point; say at most two arithmetic type, at most two memory, any number of branches). I don't expect you to

write completely runnable IA-64 assembly with all the i's dotted and t's crossed, but if you find it useful to look at the IA-64 architecture references guide, I've stashed a copy in `/cs/courses/cs184/assign/b6/`.

- summarize the performance comparison (in expected cycles per loop iteration) among the 5 cases you developed above; you may need to sketch through the off-trace cases in order get a reasonable estimate:
  1. preserve code between branches
  2. trace schedule, single branch
  3. trace schedule, full branch
  4. trace schedule, unroll 2
  5. EPIC

Source from John R. Ellis, *Bulldog: A Compiler for VLIW Architectures*, p. 277.

```
C Result is put in F to prepare transform iteration.
C
       SUBROUTINE SHUFF( S, F, N )
       DIMENSION S( 1024 )
       COMPLEX F( 1024 ), CMPLX
       DO 5 IFORT = 1, N
          I = IFORT -1
          J = 0
          M2 = 1
    1     CONTINUE
              M1 = M2
              M2 = M2 + M2
              IF (MOD( I, M2 ) .LT. M1) GOTO 3
                  J = J + N / M2
    3             CONTINUE
              IF (M2 .LT. N) GOTO 1
          JFORT = J + 1
          F( IFORT ) = CMPLX( S( JFORT ).0. )
C Don't worry about the complex conversion here;
C   Treat this as just an array assignment. -- andre
    5     CONTINUE
       RETURN
       END
```
