ECE 571 – Advanced Microprocessor-Based Design Lecture 6

Vince Weaver

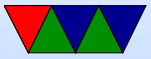
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

14 September 2020

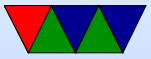
Announcements

- Homework #2 reminder



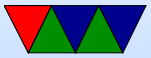
Exploiting Parallelism

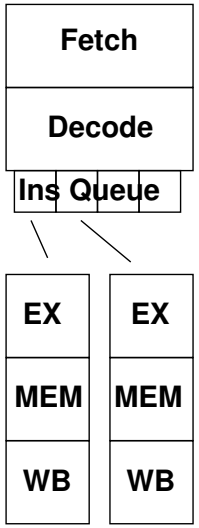
- How can we take advantage of parallelism in the control stream?
- Can we execute more than one instruction at a time?

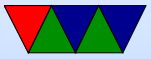


Multi-Issue (Super-Scalar)

- Decode up to X instructions at a time, and if no dependencies issue at same time.
- Types
  - Static Multi-Issue – at compile time, VLIW
  - Dynamic
- Dual issue example. Can have theoretical IPC of 2.0
- Can have unequal pipelines.



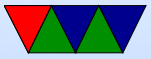

Fetch
Decode
Ins Queue
EX
EX
MEM
MEM
WB
WB



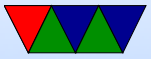
Register Renaming

- Loop unrolling
- If only a "name" dependence
- Architectural register doesn't have to be updated until written to
- Once written to it is essentially a separate register despite the same name

```
ldr     r1,[1024]          ; ldr    r10
add     r1,#5              ; add    r10
str     r1,[2048]          ; str    r10
```

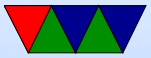


```
ldr     r1,[1025]           ; ldr    r10
add     r1,#5               ; add    r10
str     r1,[2049]           ; str    r10
```



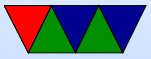
Out-of-Order

- Tries to exploit instruction-level parallelism
- Instead of being stuck waiting for a resource to become available for an instruction (cache, multiplier, etc) keep executing instructions beyond as long as there are no dependencies
- Need to insure that instructions commit in order
  Need to make sure loads/stores happen in order.
- Precise exceptions (skid?)
- What happens on exception? (interrupt, branch



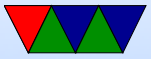
mispredict, etc)

- Register Renaming
- Re-order buffer
- Speculative execution / Branch Prediction?



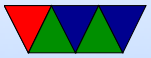
Perf Counters related to Stalls

- Front-end stalls – fetch, decode, icache misses
- Back-end stalls – memory accesses

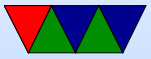


Instruction Level Parallelism

- Using super-scalar and/or OoO (Out of Order) execution try to find parallelism within your serial code
- Chip companies want to speed up existing code. Why? (it's a pain to change, you might not have source, etc.)

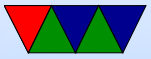


Other Ways to get better Parallelism



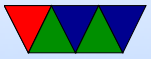
SIMD / Vector Instructions

- x86: MMX/SSE/SSE2/AVX/AVX2
  semi-related FMA
- MMX (mostly deprecated), AMD's 3DNow! (deprecated)
- PowerPC Altivec
- ARM: Neon / A64 SIMD / "Scalabale Vector Extensions" SVE



SSE / x86

- SSE (streaming SIMD): 128-bit registers XMM0 - XMM7, can be used as 4 32-bit floats
- SSE2 : 2*64bit int or float, 4 * 32-bit int or float, 8x16 bit int, 16x8-bit int
- SSE3 : minor update, add dsp and others
- SSSE3 (the s is for supplemental): shuffle, horizontal add
- SSE4 : popcnt, dot product



AVX / x86

- AVX (advanced vector extensions) – now 256 bits, YMM0-YMM15 low bits are the XMM registers. Now twice as many.
  Also adds three operand instructions a=b+c
- AVX2 – 3 operand Fused-Multiply Add, more 256 instructions
- AVX-512 – version used on Xeon Phis (knights landing) and Skylake – now 512 bits, ZMM0-XMM31