CSE 490/590 Computer Architecture

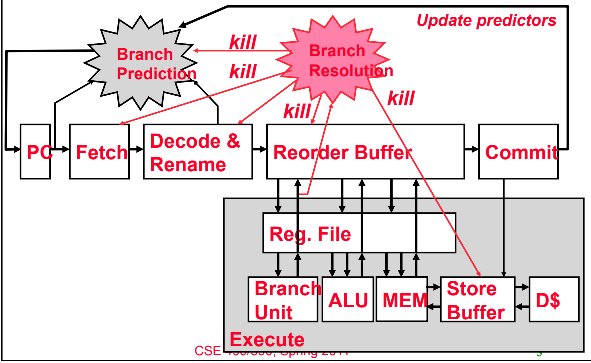
VLIW

Steve Ko
Computer Sciences and Engineering
University at Buffalo

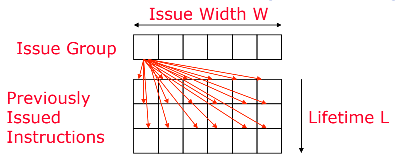
Last time…

- BTB allows prediction very early in pipeline
- In practice, use BHT and BTB together
- Speculative store buffer holds store values before commit to allow load-store forwarding
- Can execute later loads past earlier stores when addresses known, or predicted no dependence

Datapath: Branch Prediction and Speculative Execution



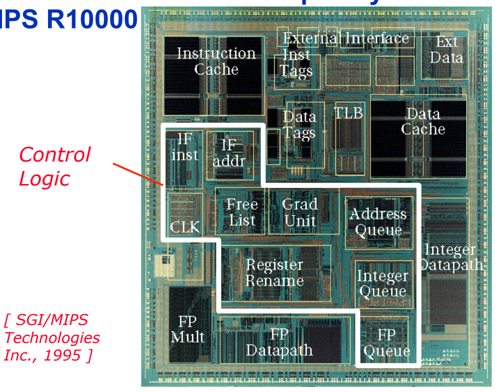
Superscalar Control Logic Scaling



- Each issued instruction must somehow check against W*L instructions, i.e., growth in hardware ∝ W*(W*L)
- For in-order machines, L is related to pipeline latencies and check is done during issue (interlocks or scoreboard)
- For out-of-order machines, L also includes time spent in instruction buffers (instruction window or ROB), and check is done by broadcasting tags to waiting instructions at write back (completion)
- As W increases, larger instruction window is needed to find enough parallelism to keep machine busy => greater L

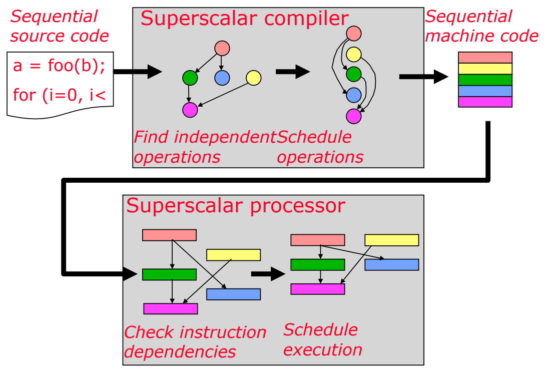
=> Out-of-order control logic grows faster than W^2 (~W^3)

Out-of-Order Control Complexity: MIPS R10000

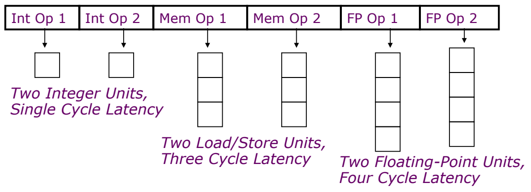


[SGI/MIPS Technologies Inc., 1995]

Sequential ISA Bottleneck



VLIW: Very Long Instruction Word


Int Op 1
Int Op 2
Mem Op 1
Mem Op 2
FP Op 1
FP Op 2
Two Integer Units, Single Cycle Latency
Two Load/Store Units, Three Cycle Latency
Two Floating-Point Units, Four Cycle Latency

- Multiple operations packed into one instruction
- Each operation slot is for a fixed function
- Constant operation latencies are specified
- Architecture requires guarantee of:
  - Parallelism within an instruction => no cross-operation RAW check
  - No data use before data ready => no data interlocks

VLIW Compiler Responsibilities

- Schedules to maximize parallel execution
- Guarantees intra-instruction parallelism
- Schedules to avoid data hazards (no interlocks)
  - Typically separates operations with explicit NOPs

Early VLIW Machines

- FPS AP120B (1976)
  - scientific attached array processor
  - first commercial wide instruction machine
  - hand-coded vector math libraries using software pipelining and loop unrolling
- Multiflow Trace (1987)
  - commercialization of ideas from Fisher's Yale group including "trace scheduling"
  - available in configurations with 7, 14, or 28 operations/instruction
  - 28 operations packed into a 1024-bit instruction word
- Cydrome Cydra-5 (1987)
  - 7 operations encoded in 256-bit instruction word
  - rotating register file

CSE 490/590 Administrivia

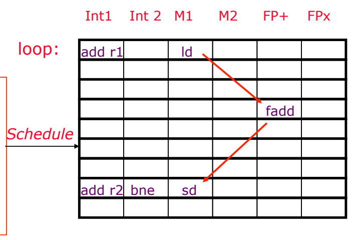
- HW2 is out
- Midterm solution will be up today
- Quiz 2 (next Friday 4/8)

Loop Execution

```
for (i=0; i<N; i++)
   B[i] = A[i] + C;
```

Compile

```
loop:  ld f1, 0(r1)
       add r1, 8
       fadd f2, f0, f1
       sd f2, 0(r2)
       add r2, 8
       bne r1, r3, loop
```



How many FP ops/cycle?

1 fadd / 8 cycles = 0.125

Loop Unrolling

```
for (i=0; i<N; i++)
   B[i] = A[i] + C;
```

Unroll inner loop to perform 4 iterations at once

```
for (i=0; i<N; i+=4)
{
   B[i]   = A[i] + C;
   B[i+1] = A[i+1] + C;
   B[i+2] = A[i+2] + C;
   B[i+3] = A[i+3] + C;
}
```

Need to handle values of N that are not multiples of unrolling factor with final cleanup loop

Scheduling Loop Unrolled Code

Unroll 4 ways

```
loop: ld f1, 0(r1)
      ld f2, 8(r1)
      ld f3, 16(r1)
      ld f4, 24(r1)
      add r1, 32
      fadd f5, f0, f1
      fadd f6, f0, f2
      fadd f7, f0, f3
      fadd f8, f0, f4
      sd f5, 0(r2)
      sd f6, 8(r2)
      sd f7, 16(r2)
      sd f8, 24(r2)
add r2, 32
      bne r1, r3, loop
```

Schedule →

loop:

Int1	Int 2	M1	M2	FP+	FPx
		ld f1			
		ld f2			
		ld f3			
add r1		ld f4		fadd f5	
				fadd f6	
				fadd f7	
				fadd f8	
		sd f5			
		sd f6			
		sd f7			
add r2	bne	sd f8			

How many FLOPS/cycle?

4 fadds / 11 cycles = 0.36

Software Pipelining

Unroll 4 ways first

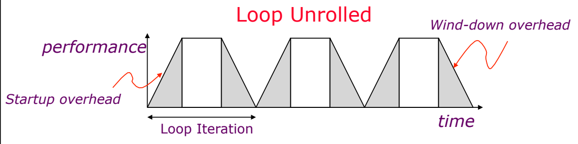
```
loop: ld f1, 0(r1)
      ld f2, 8(r1)
      ld f3, 16(r1)
      ld f4, 24(r1)
      add r1, 32
      fadd f5, f0, f1
      fadd f6, f0, f2
      fadd f7, f0, f3
      fadd f8, f0, f4
      sd f5, 0(r2)
      sd f6, 8(r2)
      sd f7, 16(r2)
      add r2, 32
      sd f8, -8(r2)
      bne r1, r3, loop
```

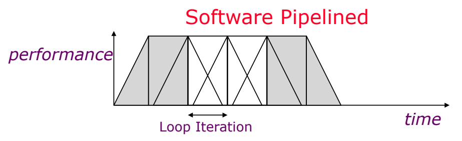
		Int1	Int 2	M1	M2	FP+	FPx
prolog				ld f1			
				ld f2			
				ld f3			
		add r1		ld f4			
				ld f1		fadd f5	
				ld f2		fadd f6	
				ld f3		fadd f7	
		add r1		ld f4		fadd f8	
iterate	loop:			ld f1	sd f5	fadd f5	
				ld f2	sd f6	fadd f6	
			add r2	ld f3	sd f7	fadd f7	
		add r1	bne	ld f4	sd f8	fadd f8	
epilog					sd f5	fadd f5	
					sd f6	fadd f6	
			add r2		sd f7	fadd f7	
			bne		sd f8	fadd f8	
					sd f5		

How many FLOPS/cycle?

4 fadds / 4 cycles = 1

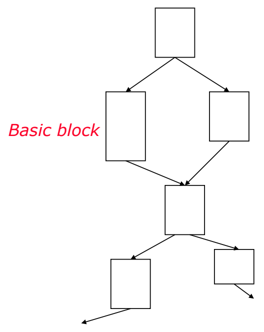
Software Pipelining vs. Loop Unrolling





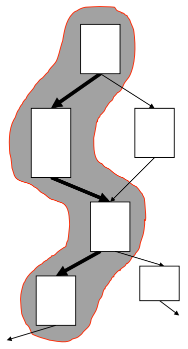
Software pipelining pays startup/wind-down costs only once per loop, not once per iteration

What if there are no loops?



- Branches limit basic block size in control-flow intensive irregular code
- Difficult to find ILP in individual basic blocks

Trace Scheduling *[Fisher,Ellis]*

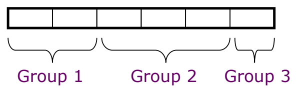


- Pick string of basic blocks, a *trace*, that represents most frequent branch path
- Use profiling feedback or compiler heuristics to find common branch paths
- Schedule whole "trace" at once
- Add fixup code to cope with branches jumping out of trace

Problems with "Classic" VLIW

- Object-code compatibility
  - have to recompile all code for every machine, even for two machines in same generation
- Object code size
  - instruction padding wastes instruction memory/cache
  - loop unrolling/software pipelining replicates code
- Scheduling variable latency memory operations
  - caches and/or memory bank conflicts impose statically unpredictable variability
- Knowing branch probabilities
  - Profiling requires an significant extra step in build process
- Scheduling for statically unpredictable branches
  - optimal schedule varies with branch path

VLIW Instruction Encoding



- Schemes to reduce effect of unused fields
  - Compressed format in memory, expand on I-cache refill
    - » used in Multiflow Trace
    - » introduces instruction addressing challenge
  - Mark parallel groups
    - » used in TMS320C6x DSPs, Intel IA-64
  - Provide a single-op VLIW instruction
    - » Cydra-5 UniOp instructions

Acknowledgements

- These slides heavily contain material developed and copyright by
  - Krste Asanovic (MIT/UCB)
  - David Patterson (UCB)
- And also by:
  - Arvind (MIT)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)

- MIT material derived from course 6.823
- UCB material derived from course CS252