COMP4211 05s1 Seminar 5: Multiple Issue & Speculation

Slides due to
David A. Patterson, 2001

Getting CPI < 1: Issuing Multiple Instructions/Cycle

- Superscalar MIPS: 2 instructions, 1 FP & 1 anything
  - Fetch 64-bits/clock cycle; Int on left, FP on right
  - Can only issue 2nd instruction if 1st instruction issues
  - More ports for FP registers to do FP load & FP op in a pair

Type	*Pipe Stages*						
Int. instruction	IF	ID	EX	MEM	WB		
FP instruction	IF	ID	EX	MEM	WB		
Int. instruction		IF	ID	EX	MEM	WB	
FP instruction		IF	ID	EX	MEM	WB	
Int. instruction			IF	ID	EX	MEM	WB
FP instruction			IF	ID	EX	MEM	WB

- 1 cycle load delay expands to 3 instructions in SS
  - instruction in right half can't use it, nor instructions in next slot

Getting CPI < 1: Issuing Multiple Instructions/Cycle

- Vector Processing: Explicit coding of independent loops as operations on large vectors of numbers
  - Multimedia instructions being added to many processors
- Superscalar: varying no. instructions/cycle (1 to 8), scheduled by compiler or by HW (Tomasulo)
  - IBM PowerPC, Sun UltraSparc, DEC Alpha, Pentium III/4
- (Very) Long Instruction Words (V)LIW: fixed number of instructions (4-16) scheduled by the compiler; put ops into wide templates
  - Intel Architecture-64 (IA-64) 64-bit address
    - » Renamed: "Explicitly Parallel Instruction Computer (EPIC)"
- Anticipated success of multiple instructions lead to Instructions Per Clock_cycle (IPC) vs. CPI

Multiple Issue Issues

- issue packet: group of instructions from fetch unit that could potentially issue in 1 clock
  - If instruction causes structural hazard or a data hazard either due to earlier instruction in execution or to earlier instruction in issue packet, then instruction does not issue
  - 0 to N instruction issues per clock cycle, for N-issue
- Performing issue checks in 1 cycle could limit clock cycle time: $O(n^2-n)$ comparisons
  - => issue stage usually split and pipelined
  - 1st stage decides how many instructions from within this packet can issue, 2nd stage examines hazards among selected instructions and those already been issued
  - => higher branch penalties => prediction accuracy important

Multiple Issue Challenges

- While Integer/FP split is simple for the HW, get CPI of 0.5 only for programs with:
  - Exactly 50% FP operations AND No hazards
- If more instructions issue at same time, greater difficulty of decode and issue:
  - Even 2-scalar => examine 2 opcodes, 6 register specifiers, & decide if 1 or 2 instructions can issue; (N-issue ~$O(N^2-N)$ comparisons)
  - Register file: need 2x reads and 1x writes/cycle
  - Rename logic: must be able to rename same register multiple times in one cycle! For instance, consider 4-way issue:

```
add r1, r2, r3          add p11, p4, p7
sub r4, r1, r2    ⇒     sub p22, p11, p4
lw  r1, 4(r4)           lw  p23, 4(p22)
add r5, r1, r2          add p12, p23, p4
```

  Imagine doing this transformation in a single cycle!
  - Result buses: Need to complete multiple instructions/cycle
    - » So, need multiple buses with associated matching logic at every reservation station.
    - » Or, need multiple forwarding paths

Dynamic Scheduling in Superscalar The easy way

- How to issue two instructions and keep in-order instruction issue for Tomasulo?
  - Assume 1 integer + 1 floating point
  - 1 Tomasulo control for integer, 1 for floating point
- Key is assigning reservation station and updating control tables
  - Issue 2X Clock Rate, so that issue remains in order
- Only loads/stores might cause dependency between integer and FP issue:
  - Replace load reservation station with a load queue; operands must be read in the order they are fetched
  - Load checks addresses in Store Queue to avoid RAW violation
  - Store checks addresses in Load Queue to avoid WAR,WAW

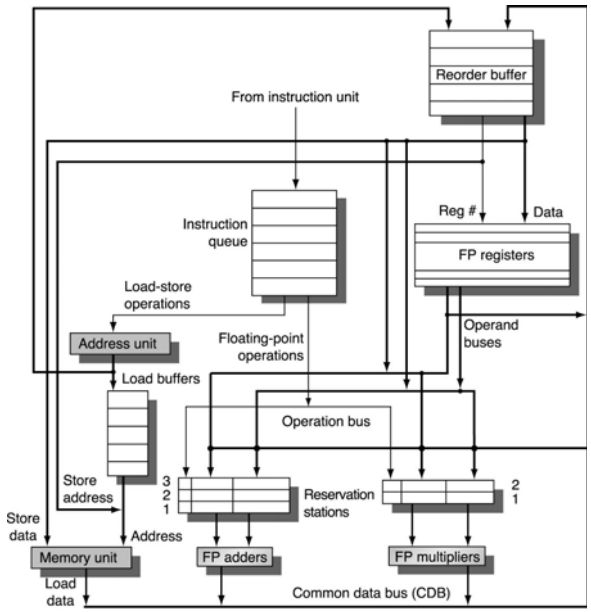
Hardware-Based Speculation

- Trying to exploit more ILP while maintaining control dependencies becomes a burden
- Overcome control dependencies by speculating on the outcome of branches and *executing* the program as if our guesses were correct
  - Need to handle incorrect guesses
- Key ideas:
  - Dynamic branch prediction
  - Speculation
  - Dynamic scheduling

Implementing Speculation

- We consider building on top of Tomasulo's algorithm
  - Must separate bypassing of results among instructions from actual completion (*write-back*) of instructions
  - Cannot allow updates to be performed that can't be undone
  - Instruction commit updates register or memory when instruction no longer speculative
  - Need to add re-order buffer
- Key idea: execute out-of-order but commit in-order

Tomasulo extended to handle speculation



Reorder buffer

- Contains 4 fields:
  1. Instruction type indicates whether branch, store, or register op
  2. Destination field memory or register
  3. Value field
  4. Ready flag indicates instruction has completed operation
- The renaming function of the reservation stations is replaced by the ROB
- Every instruction has a ROB entry until it commits
  - Therefore tag results using ROB entry number

Instruction execution

1. Issue – get instruction from instruction Q and issue if reservation station and ROB slots available – sometimes called *dispatch*
2. Execute – when both operands available at the reservation station – sometimes called *issue*
3. Write result – when result available, write to CDB tagged by ROB entry #; mark reservation station slot available
4. Commit – when instruction at head of Q ready, *writeback result* unless mispredicted branch. In latter case, flush all remaining instructions in ROB and commence fetching at target.

Register renaming, virtual registers versus Reorder Buffers

- Alternative to Reorder Buffer is a larger virtual set of registers and register renaming
- Virtual registers hold both architecturally visible registers + temporary values
  - replace functions of reorder buffer and reservation station
- Renaming process maps names of architectural registers to registers in virtual register set
  - Changing subset of virtual registers contains architecturally visible registers
- Simplifies instruction commit: mark register as no longer speculative, free register with old value
- Adds 40-80 extra registers: Alpha, Pentium,…
  - Size limits no. instructions in execution (used until commit)

How much to speculate?

- Speculation Pro: uncover events that would otherwise stall the pipeline (cache misses)
- Speculation Con: speculate costly if exceptional event occurs when speculation was incorrect
- Typical solution: speculation allows only low-cost exceptional events (1st-level cache miss)
- When expensive exceptional event occurs, (2nd-level cache miss or TLB miss) processor waits until the instruction causing event is no longer speculative before handling the event
- Assuming single branch per cycle: future may speculate across multiple branches!

Limits to ILP

- Conflicting studies of amount
  - Benchmarks (vectorized Fortran FP vs. integer C programs)
  - Hardware sophistication
  - Compiler sophistication
- How much ILP is available using existing mechanisms with increasing HW budgets?
- Do we need to invent new HW/SW mechanisms to keep on processor performance curve?
  - Intel MMX, SSE (Streaming SIMD Extensions): 64 bit ints
  - Intel SSE2: 128 bit, including 2 64-bit Fl. Pt. per clock
  - Motorola AltaVec: 128 bit ints and FPs
  - Supersparc Multimedia ops, etc.

Limits to ILP

Initial HW Model here; MIPS compilers.

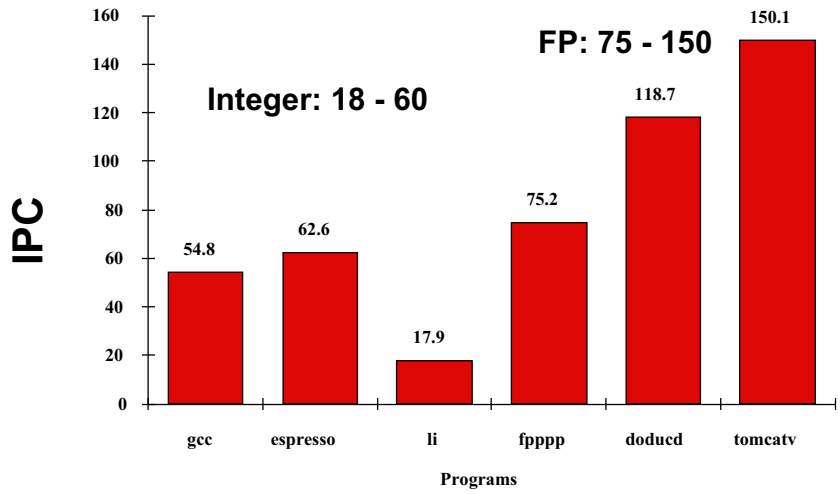
Assumptions for ideal/perfect machine to start:

1. *Register renaming* – infinite virtual registers => all register WAW & WAR hazards are avoided
2. *Branch prediction* – perfect; no mispredictions
3. *Jump prediction* – all jumps perfectly predicted 2 & 3 => machine with perfect speculation & an unbounded buffer of instructions available
4. *Memory-address alias analysis* – addresses are known & a store can be moved before a load provided addresses not equal

Also:
unlimited number of instructions issued/clock cycle;
perfect caches;
1 cycle latency for all instructions (FP *,/);

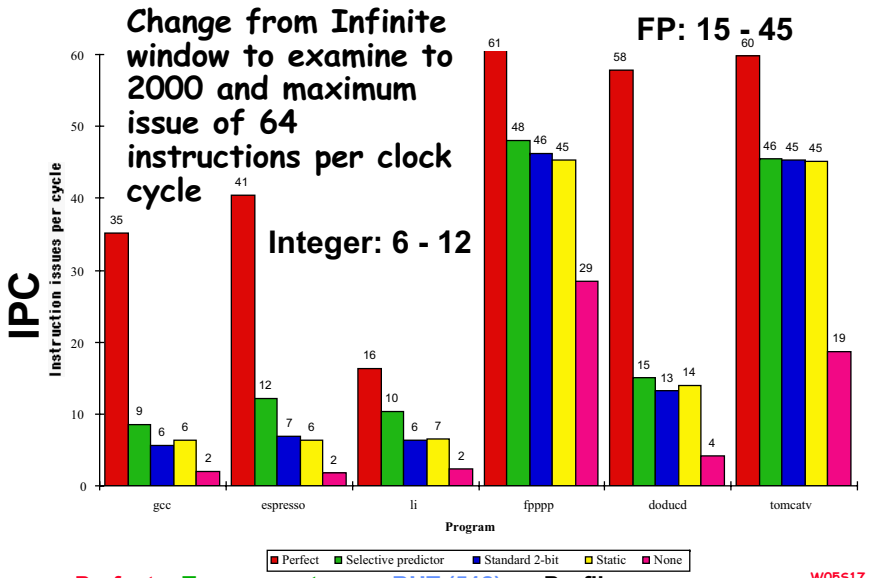
Upper Limit to ILP: Ideal Machine

(Figure 3.34, page 294)



More Realistic HW: Branch Impact

Figure 3.38, Page 300

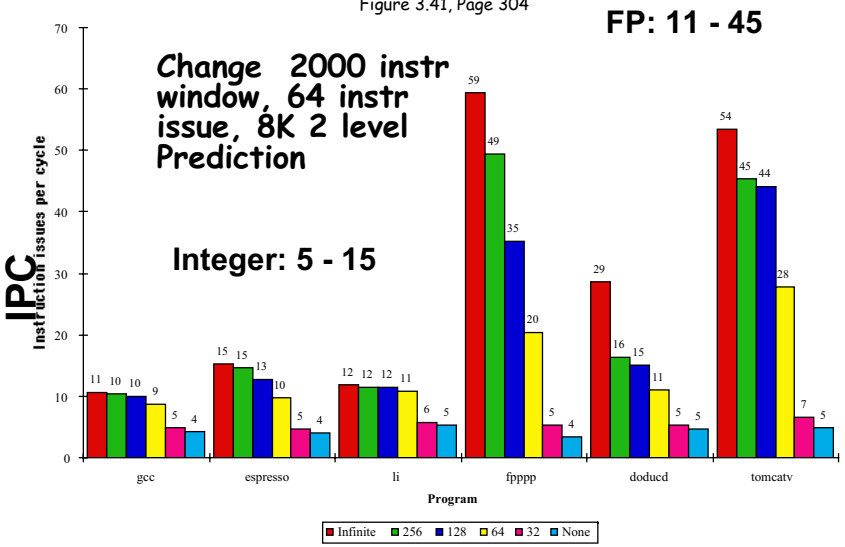


Perfect Tournament BHT (512) Profile No prediction

W05S17

More Realistic HW: Renaming Register Impact

Figure 3.41, Page 304

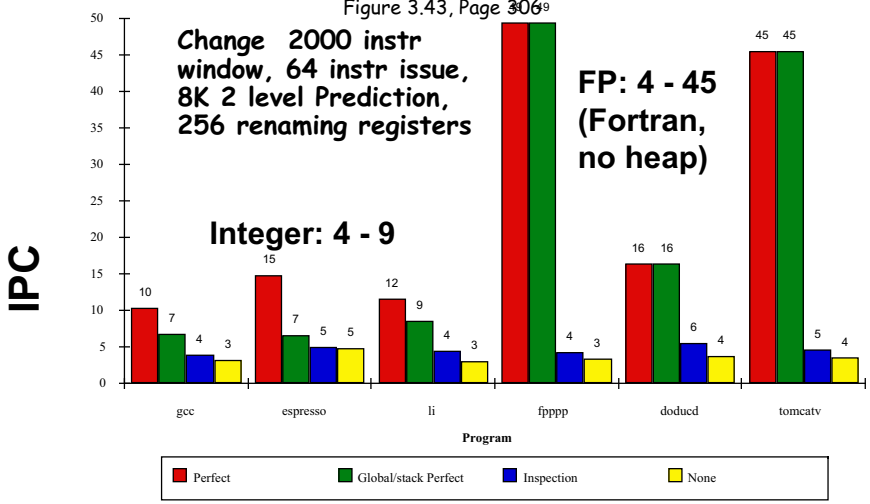


Infinite 256 128 64 32 None

W05S18

More Realistic HW: Memory Address Alias Impact

Figure 3.43, Page 306

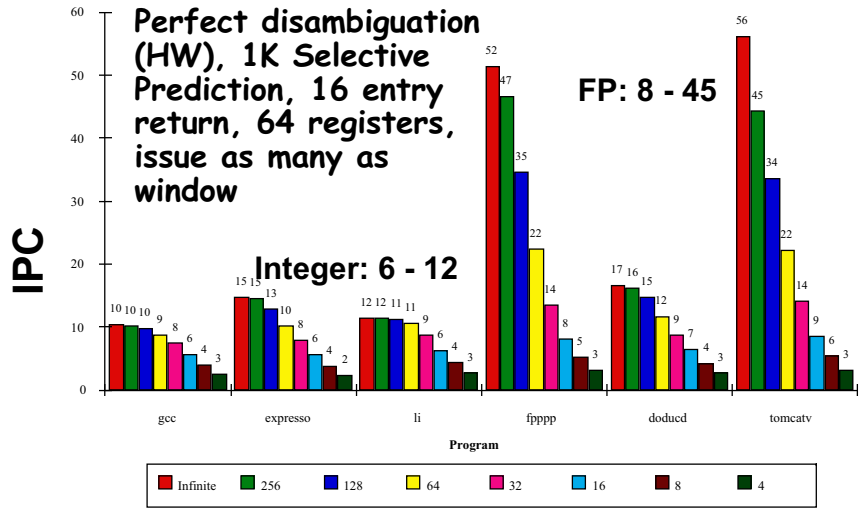


Perfect Global/Stack perf; heap conflicts Inspec. Assem. None

W05S19

Realistic HW for '00: Window Impact

(Figure 3.45, Page 309)



Infinite 256 128 64 32 16 8 4

W05S20

How to Exceed ILP Limits of this study?

- WAR and WAW hazards through memory: eliminated WAW and WAR hazards through register renaming, but not in memory usage
- Unnecessary dependences (compiler not unrolling loops so iteration variable dependence)
- Overcoming the data flow limit: value prediction, predicting values and speculating on prediction
  - Address value prediction and speculation predicts addresses and speculates by reordering loads and stores; could provide better aliasing analysis, only need predict if addresses =

Workstation Microprocessors 3/2001

Processor	Alpha 21264B	AMD Athlon	HP PA-8600	IBM Power3-II	Intel Pentium III	Intel Pentium 4	MIPS R12000	Sun Ultra-II	Sun Ultra-III
Clock Rate	833MHz	1.2GHz	552MHz	450MHz	1.0GHz	1.5GHz	400MHz	480MHz	900MHz
Cache (I/D/L2)	64K/64K	64K/64K/256K	512K/1M	32K/64K	16K/16K/256K	12K/8K/256K	32K/32K	16K/16K	32K/64K
Issue Rate	4 issue	3 x86 instr	4 issue	4 issue	3 x86 instr	3 x ROPs	4 issue	4 issue	4 issue
Pipeline Stages	7/9 stages	9/11 stages	7/9 stages	7/8 stages	12/14 stages	22/24 stages	6 stages	6/9 stages	14/15 stages
Out of Order	80 instr	72ROPs	56 instr	32 instr	40 ROPs	126 ROPs	48 instr	None	None
Rename regs	48/41	36/36	56 total	16 int/24 fp	40 total	128 total	32/32	None	None
BHT Entries	4K x 9-bit	4K x 2-bit	2K x 2-bit	2K x 2-bit	>= 512	4K x 2-bit	2K x 2-bit	512 x 2-bit	16K x 2-bit
TLB Entries	128/128	280/288	120 unified	128/128	32I / 64D	128I/65D	64 unified	64I/64D	128I/512D
Memory B/W	2.66GB/s	2.1GB/s	1.54GB/s	1.6GB/s	1.06GB/s	3.2GB/s	539 MB/s	1.9GB/s	4.8GB/s
Package	CPGA-588	PGA-462	LGA-544	SCC-1088	PGA-370	PGA-423	CPGA-527	CLGA-787	1368 FC-LGA
IC Process	0.18µ 6M	0.18µ 6M	0.25µ 2M	0.22µ 6m	0.18µ 6M	0.18µ 6M	0.25µ 4M	0.29µ 6M	0.18µ 7M
Die Size	115mm²	117mm²	477mm²	163mm²	106mm²	217mm²	204mm²	126 mm²	210mm²
Transistors	15.4 million	37 million	130 million	23 million	24 million	42 million	7.2 million	3.8 million	29 million
Est mfg cost*	$160	$62	$330	$110	$39	$110	$125	$70	$145
Power(Max)	75W*	76W	60W*	36W*	30W	55W(TDP)	25W*	20W*	65W
Availability	1Q01	4Q00	3Q00	4Q00	2Q00	4Q00	2Q00	3Q0	4Q00

- Max issue: 4 instructions (many CPUs)
  Max rename registers: 128 (Pentium 4)
  Max BHT: 4K x 9 (Alpha 21264B), 16Kx2 (Ultra III)
  Max Window Size (OOO): 126 intructions (Pent. 4)
  Max Pipeline: 22/24 stages (Pentium 4)

Source: Microprocessor Report, www.MPRonline.com

SPEC 2000 Performance 3/2001 *Source: Microprocessor Report, www.MPRonline.com*

Processor	Alpha 21264B	AMD Athlon	HP PA-8600	IBM Power 3-II	Intel PIII	Intel P4	MIPS R12000	Sun Ultra-II	Sun Ultra-III
System or Motherboard	Alpha ES40 Model 6	AMD GA-7ZM	HP9000 j6000	RS/6000 44P-170	Dell Prec. 420	Intel D850GB	SGI 2200	Sun Enterprs 450	Sun Blade 1000
Clock Rate	833MHz	1.2GHz	552MHz	450MHz	1GHz	1.5GHz	400MHz	480MHz	900MHz
External Cache	8MB	None	None	8MB	None	None	8MB	8MB	8MB
164.gzip	392	n/a	376	230	545	553	226	165	349
175.vpr	452	n/a	421	285	354	298	384	212	383
176.gcc	617	n/a	577	350	401	588	313	232	500
181.mcf	441	n/a	384	498	276	473	563	356	474
186.crafty	694	n/a	472	304	523	497	334	175	439
197.parser	360	n/a	361	171	362	472	283	211	412
252.eon	645	n/a	395	280	615	650	360	209	465
253.perlbmk	526	n/a	406	215	614	703	246	247	457
254.gap	365	n/a	229	256	443	708	204	171	300
255.vortex	673	n/a	764	312	717	735	294	304	581
256.bzip2	560	n/a	349	258	396	420	334	237	500
300.twolf	658	n/a	479	414	394	403	451	243	473
SPECint_base2000	518	n/a	417	286	454	524	320	225	438
168.wupside	529	360	340	360	416	759	280	284	497
171.swim	1,156	506	761	279	493	1,244	300	285	752
172.mgrid	580	272	462	319	274	558	231	226	377
173.applu	424	298	563	327	280	641	237	150	221
177.mesa	713	302	300	330	541	553	289	273	469
178.galgel	558	468	569	429	335	537	989	735	1,266
179.art	1,540	213	419	969	410	514	995	920	990
183.equake	231	236	347	560	249	739	222	149	211
187.facerec	822	411	258	257	307	451	411	459	718
188.ammp	488	221	376	326	294	366	373	313	421
189.lucas	731	237	370	284	349	764	259	205	204
191.fma3d	528	365	302	340	297	427	192	207	302
200.sixtrack	340	256	286	234	170	257	199	159	273
301.aspi	553	278	523	349	371	427	252	189	340
SPECfp_base2000	590	304	400	356	329	549	319	274	427

1.5X 3.8X 1.2X 1.6X 1.7X

Conclusion

- 1985-2000: 1000X performance
  - Moore's Law transistors/chip => Moore's Law for Performance/MPU
- Hennessy: industry been following a roadmap of ideas known in 1985 to exploit Instruction Level Parallelism and (real) Moore's Law to get 1.55X/year
  - Caches, Pipelining, Superscalar, Branch Prediction, Out-of-order execution, …
- ILP limits: To make performance progress in future need to have explicit parallelism from programmer vs. implicit parallelism of ILP exploited by compiler, HW?
  - Otherwise drop to old rate of 1.3X per year?
  - Less than 1.3X because of processor-memory performance gap?
- Impact on you: if you care about performance, better think about explicitly parallel algorithms vs. rely on ILP?