CS152
Computer Architecture and Engineering
Lecture 17

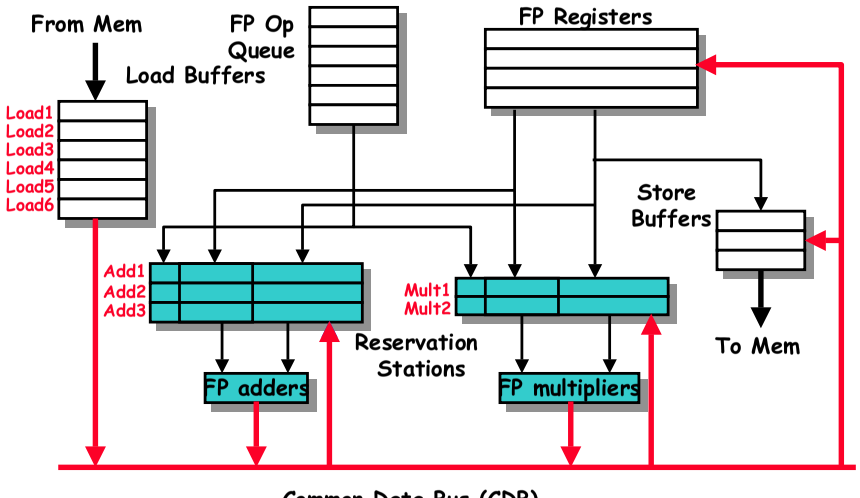
Branch Prediction, Explicit Renaming, ILP

April 5, 2004

John Kubiatowicz (http.cs.berkeley.edu/~kubitron)

lecture slides: http://inst.eecs.berkeley.edu/~cs152/

Review: Tomasulo Organization



Review: Three Stages of Tomasulo Algorithm

1. Issue—get instruction from FP Op Queue
   If reservation station free (no structural hazard), control issues instr & sends operands (renames registers).
2. Execution—operate on operands (EX)
   When both operands ready then execute; if not ready, watch Common Data Bus for result
3. Write result—finish execution (WB)
   Write on Common Data Bus to all awaiting units; mark reservation station available

° Normal data bus: data + destination ("go to" bus)

° Common data bus: data + source ("come from" bus)
- 64 bits of data + 4 bits of Functional Unit source address
- Write if matches expected Functional Unit (produces result)
- Does the broadcast

Review: Tomasulo Architecture

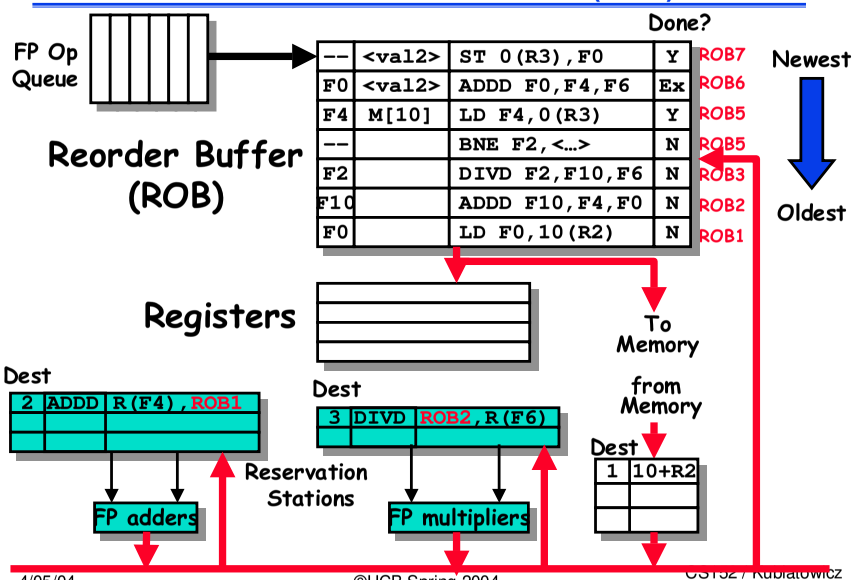
° Reservations stations:
- renaming to larger set of registers + buffering source operands
- Prevents registers as bottleneck
- Avoids WAR, WAW hazards of Scoreboard

° Not limited to basic blocks:
- integer units gets ahead, beyond branches

° Dynamic Scheduling:
- Scoreboarding/Tomasulo
- In-order issue, out-of-order execution, out-of-order commit

° Tomasulo can unroll loops dynamically in hardware!
- Need: renaming (different physical names for different iterations)
- Fast branch computation

Review: Tomasulo With Reorder buffer (ROB)



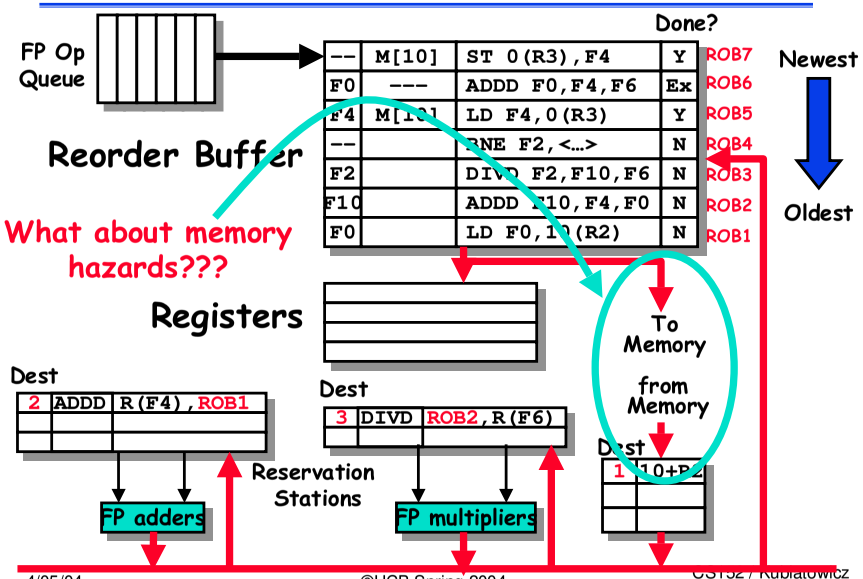
Review: Four Steps of Speculative Tomasulo Algorithm

1. Issue—get instruction from FP Op Queue
   - If reservation station and reorder buffer slot free, issue instr & send operands & reorder buffer no. for destination (this stage sometimes called "dispatch")
2. Execution—operate on operands (EX)
   - When both operands ready then execute; if not ready, watch CDB for result; when both in reservation station, execute; checks RAW (sometimes called "issue")
3. Write result—finish execution (WB)
   - Write on Common Data Bus to all awaiting FUs & reorder buffer
4. Commit—update register with reorder buffer (ROB) result
   - When instr. at head of reorder buffer & result present, update register with result (or store to memory) and remove instr from reorder buffer
   - Stores only commit to memory when reach head of ROB
   - Values only overwrite registers when they reach head
   - Mispredicted branch or interrupt flushes reorder buffer

NOTES:

- In-order issue, Out-of-order execution, In-order commit
- Can always throw out contents of reorder buffer (must cancel running ops)
- Precise exception point is instruction at head of buffer

Tomasulo With Reorder buffer: Memory Disambiguation



Memory Disambiguation: Handling RAW Hazards in memory

° Question: Given a load that follows a store in program order, are the two related?

- (Alternatively: is there a RAW hazard between the store and the load)?

```
Eg:   st   0(R2),R5
      ld   R6,0(R3)
```

° Can we go ahead and start the load early?

- Store address could be delayed for a long time by some calculation that leads to R2 (divide?).
- We might want to issue/begin execution of both operations in same cycle.

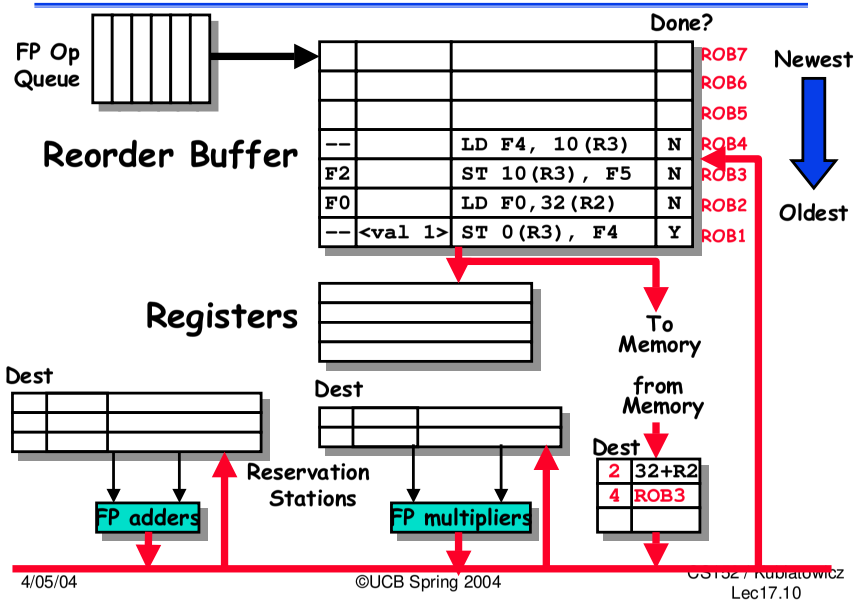
° Two techiques:

- No Speculation: we are not allowed to start load until we know *for sure* that address 0(R2) ≠ 0(R3)
- Speculation: We might guess at whether or not they are dependent (called "dependence speculation") and use reorder buffer to fixup if we are wrong.

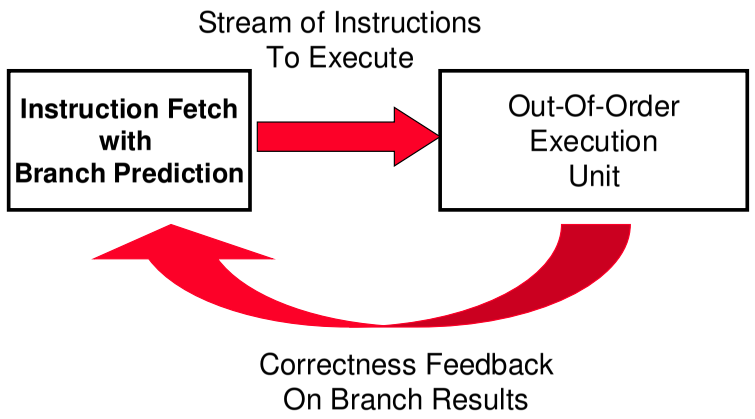
Hardware Support for Memory Disambiguation

- Need buffer to keep track of all outstanding stores to memory, in program order.
  - Keep track of address (when becomes available) and value (when becomes available)
  - FIFO ordering: will retire stores from this buffer in program order
- When issuing a load, record current head of store queue (know which stores are ahead of you).
- When have address for load, check store queue:
  - If *any* store prior to load is waiting for its address:
    - If not speculating, stall load
    - If speculating, send request to memory (predict no dependence)
  - If load address matches earlier store address (associative lookup), then we have a *memory-induced RAW hazard*:
    - store value available ⇒ return value
    - store value not available ⇒ return ROB number of source
  - Otherwise, send out request to memory
- Actual stores commit in order, so no worry about WAR/WAW hazards through memory.

Memory Disambiguation:



Review: Independent "Fetch" unit



- Instruction fetch decoupled from execution
- Often issue logic (+ rename) included with Fetch

Branches must be resolved quickly

- In our loop-unrolling example, we relied on the fact that branches were under control of "fast" integer unit in order to get overlap!

```
Loop:   LD      F0    0     R1
        MULTD   F4    F0    F2
        SD      F4    0     R1
        SUBI    R1    R1    #8
        BNEZ    R1    Loop
```

- What happens if branch depends on result of multd??
  - We completely lose all of our advantages!
  - Need to be able to "predict" branch outcome.
  - If we were to predict that branch was taken, this would be right most of the time.
- Problem much worse for superscalar machines!

Handling some branches: Conditional instructions

° Avoid branch prediction by turning branches into conditionally executed instructions:

if (x) then A = B op C else NOP

- If false, then neither store result nor cause exception
- Expanded ISA of Alpha, MIPS, PowerPC, SPARC have conditional move; PA-RISC can annul any following instr.
- EPIC: 64 1-bit condition fields selected so conditional execution

° Drawbacks to conditional instructions

- Still takes a clock even if "annulled"
- Stall if condition evaluated late
- Complex conditions reduce effectiveness; condition becomes known late in pipeline
- Cannot loop!

Prediction: Branches, Dependencies, Data

° Prediction has become essential to getting good performance from scalar instruction streams.

° We will discuss predicting branches. However, architects are now predicting everything: *data dependencies, actual data,* and *results of groups of instructions*:

- At what point does computation become a probabilistic operation + verification?
- We are pretty close with control hazards already…

° Why does prediction work?

- Underlying algorithm has regularities.
- Data that is being operated on has regularities.
- Instruction sequence has redundancies that are artifacts of way that humans/compilers think about problems.

° Prediction ⇒ Compressible information streams?

Dynamic Branch Prediction

° Prediction could be "Static" (at compile time) or "Dynamic" (at runtime)

- For our example, if we were to statically predict "taken", we would only be wrong once each pass through loop
- Static information passed through bits in opcode

° Is dynamic branch prediction better than static branch prediction?

- Seems to be. Still some debate to this effect
- Today, lots of hardware being devoted to dynamic branch predictors.

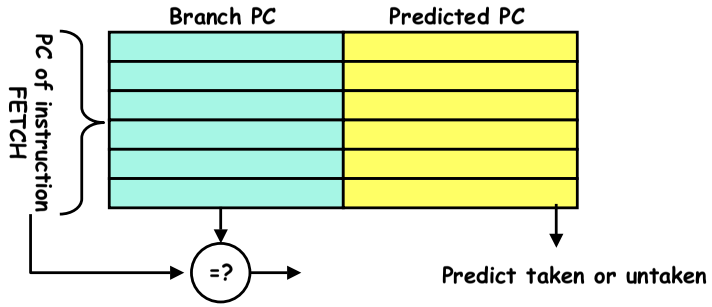
° Does branch prediction make sense for 5-stage, in-order pipeline? What about 8-stage pipeline?

- Perhaps: eliminate branch delay slots
- Then predict branches

Simple dynamic prediction: Branch Target Buffer (BTB)

° PC as branch index: get prediction AND branch address

- Must check for branch match now, since can't use wrong branch address
- Grab predicted PC from table since may take several cycles to compute

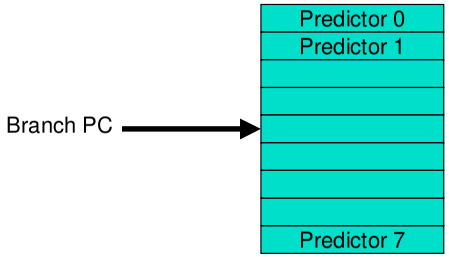


° Update predicted PC when branch is actually resolved

° Can predict branch while in Fetch stage!!!!

- *Before we know is it a branch! (Caches previous decoding of instruction)*

Branch History Table



° BHT is a table of "Predictors"
- Usually 2-bit, saturating counters
- Indexed by PC address of Branch – without tags

° In Fetch state of branch:
- BTB identifies branch
- Predictor from BHT used to make prediction

° When branch completes
- Update corresponding Predictor

Dynamic Branch Prediction (standard technologies)

° Combine Branch Target Buffer and History Tables
- Branch Target Buffer (BTB): identify branches and hold *taken addresses*
  - *Trick: identify branch before fetching instruction!*
  - *Must be careful not to misidentify branches or destinations*
- Branch History Table makes prediction
  - Can be complex prediction mechanisms with long history
  - No address check: Can be good, can be bad (aliasing)

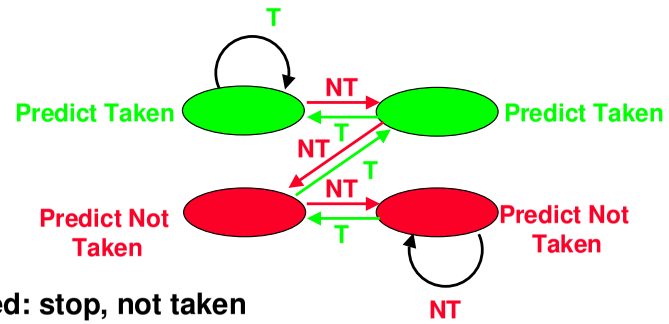
° Simple 1-bit BHT: keep last direction of branch

° Problem: in a loop, 1-bit BHT will cause two mispredictions (avg is 9 iteratios before exit):
- End of loop case, when it exits instead of looping as before
- First time through loop on *next* time through code, when it predicts exit instead of looping

° Performance = *ƒ*(accuracy, cost of misprediction)
- Misprediction ⇒ Flush Reorder Buffer

Dynamic Branch Prediction

° Solution: 2-bit scheme where change prediction only if get misprediction *twice:* (Figure 4.13, p. 264)



° Red: stop, not taken

° Green: go, taken

° Adds *hysteresis* to decision making process

2-bit BHT Accuracy

° Mispredict because either:
- Wrong guess for that branch
- Got branch history of wrong branch when index the table

° 4096 entry table misprediction varies from:
- 1% (nasa7, tomcatv)
- 9% (spice)
- 12% (gcc)
- 18% (eqntott)

° 4096 about as good as infinite table (in Alpha 211164)

Correlating Branches

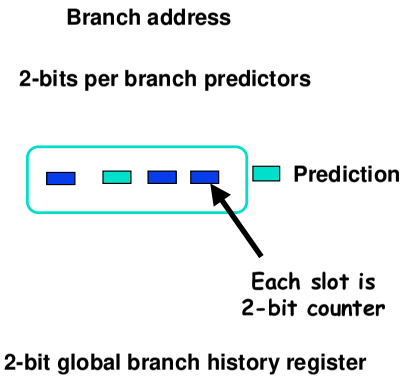
- Hypothesis: behavior of recently executed branches affects prediction of current branch
- Two possibilities; Current branch depends on:
  - Last m most recently executed branches anywhere in program
    Produces a "GA" (for "global address") in the Yeh and Patt classification (e.g. GAg)
  - Last m most recent outcomes of same branch.
    Produces a "PA" (for "per address") in same classification (e.g. PAg)
- Idea: record m most recently executed branches as taken or not taken, and use that pattern to select the proper branch history table entry
  - A single history table shared by all branches (appends a "g" at end), indexed by history value.
  - Address is used along with history to select table entry (appends a "p" at end of classification)
  - If only portion of address used, often appends an "s" to indicate "set-indexed" tables (I.e. GAs)

Correlating Branches

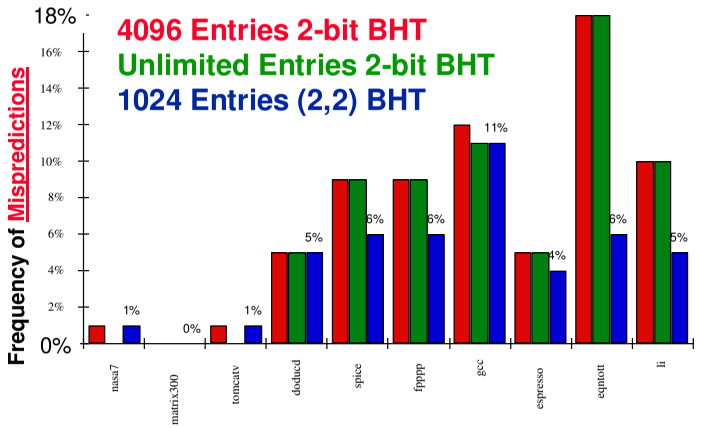
- For instance, consider global history, set-indexed BHT. That gives us a GAs history table.

(2,2) GAs predictor

- First 2 means that we keep two bits of history
- Second means that we have 2 bit counters in each slot.
- Then behavior of recent branches selects between, say, four predictions of next branch, updating just that prediction
- Note that the original two-bit counter solution would be a (0,2) GAs predictor
- Note also that aliasing is possible here...



Accuracy of Different Schemes



Administrivia

- Lab 4: due tonight:
  - Lab report must be submitted by midnight
  - You must demo a working system to your TA sometime today (work it out with them).
  - Your TA will be running code that prints out contents of memory with a series of break instructions + address/contents through memory-mapped I/O
  - You will need to demo single-stepping to your TA as well
- Lab 5: out very soon (tonight/tomorrow)
  - Building a DRAM memory controller/cache controller
  - Need to get the stalls right!!
  - Design document: due Wednesday by 9pm
  - Problem 0: also due Wednesday to Tas by email
    - This is very important: may take off points if you do not submit
- Midterm II: Wednesday 5/5

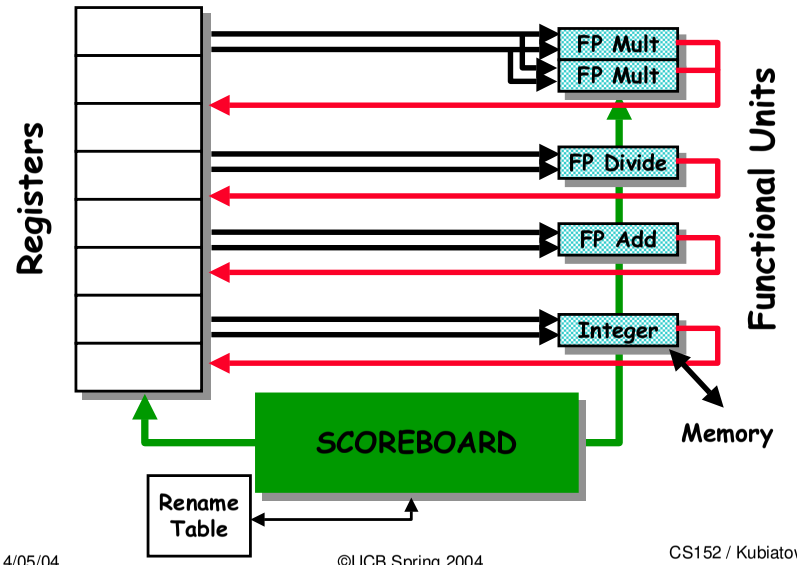
Explicit Register Renaming

- Make use of a *physical* register file that is larger than number of registers specified by ISA
- Key insight: Allocate a new physical destination register for every instruction that writes
  - Very similar to a compiler transformation called Static Single Assignment (SSA) form — but in hardware!
  - Removes all chance of WAR or WAW hazards
  - Like Tomasulo, good for allowing full out-of-order completion
  - Like hardware-based dynamic compilation?
- Mechanism? Keep a translation table:
  - ISA register ⇒ physical register mapping
  - When register written, replace entry with new register from freelist.
  - Physical register becomes free when not used by any active instructions

Advantages of Explicit Renaming

- Decouples *renaming* from *scheduling:*
  - Pipeline can be exactly like "standard" DLX pipeline (perhaps with multiple operations issued per cycle)
  - Or, pipeline could be tomasulo-like or a scoreboard, etc.
  - Standard forwarding or bypassing could be used
- Allows data to be fetched from single register file
  - No need to bypass values from reorder buffer
  - This can be important for balancing pipeline
- Many processors use a variant of this technique:
  - R10000, Alpha 21264, HP PA8000
- Another way to get precise interrupt points:
  - All that needs to be "undone" for precise break point is to undo the table mappings
  - Provides an interesting mix between reorder buffer and future file
    - Results are written immediately back to register file
    - Registers *names* are "freed" in program order (by ROB)

Can we use explicit register renaming with scoreboard?



Stages of Scoreboard Control With Explicit Renaming

- Issue—decode instructions & check for structural hazards & allocate new physical register for result
  - Instructions issued in program order (for hazard checking)
  - Don't issue if no free physical registers
  - Don't issue if structural hazard
- Read operands—wait until no hazards, read operands
  - All real dependencies (RAW hazards) resolved in this stage, since we wait for instructions to write back data.
- Execution—operate on operands
  - The functional unit begins execution upon receiving operands. When the result is ready, it notifies the scoreboard
- Write result—finish execution
- Note: No checks for WAR or WAW hazards!

Scoreboard With Explicit Renaming

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2				
LD	F2	45+	R3				
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
	Mult1	No								
	Add	No								
	Divide	No								

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
	FU	P0	P2	P4	P6	P8	P10	P12		P30

- Initialized Rename Table

Renamed Scoreboard 1

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1			
LD	F2	45+	R3				
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	Yes	Load	P32		R2				Yes
	Int2	No								
	Mult1	No								
	Add	No								
	Divide	No								

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
1	FU	P0	P2	P4	P32	P8	P10	P12		P30

- Each instruction allocates free register
- Similar to single-assignment compiler transformation

Renamed Scoreboard 2

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2		
LD	F2	45+	R3	2			
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	Yes	Load	P32		R2				Yes
	Int2	Yes	Load	P34		R3				Yes
	Mult1	No								
	Add	No								
	Divide	No								

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
2	FU	P0	P34	P4	P32	P8	P10	P12		P30

Renamed Scoreboard 3

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	
LD	F2	45+	R3	2	3		
MULTD	F0	F2	F4	3			
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	Yes	Load	P32		R2				Yes
	Int2	Yes	Load	P34		R3				Yes
	Mult1	Yes	Multd	P36	P34	P4	Int2		No	Yes
	Add	No								
	Divide	No								

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
3	FU	P36	P34	P4	P32	P8	P10	P12		P30

Renamed Scoreboard 4

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	
MULTD	F0	F2	F4	3			
SUBD	F8	F6	F2	4			
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	Yes	Load	P34		R3				Yes
	Mult1	Yes	Multd	P36	P34	P4	Int2		No	Yes
	Add	Yes	Sub	P38	P32	P34		Int2	Yes	No
	Divide	No								

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
4	*FU*	P36	P34	P4	P32	P38	P10	P12		P30

Renamed Scoreboard 5

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3			
SUBD	F8	F6	F2	4			
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
	Add	Yes	Sub	P38	P32	P34			Yes	Yes
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
5	*FU*	P36	P34	P4	P32	P38	P40	P12		P30

Renamed Scoreboard 6

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6		
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
10	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
2	Add	Yes	Sub	P38	P32	P34			Yes	Yes
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
6	*FU*	P36	P34	P4	P32	P38	P40	P12		P30

Renamed Scoreboard 7

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6		
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
9	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
1	Add	Yes	Sub	P38	P32	P34			Yes	Yes
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
7	*FU*	P36	P34	P4	P32	P38	P40	P12		P30

Renamed Scoreboard 8

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6	8	
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
8	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
0	Add	Yes	Sub	P38	P32	P34			Yes	Yes
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
8	*FU*	P36	P34	P4	P32	P38	P40	P12		P30

Renamed Scoreboard 9

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
7	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
	Add	No								
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
9	*FU*	P36	P34	P4	P32	P38	P40	P12		P30

Renamed Scoreboard 10

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10			

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
6	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
	Add	Yes	Addd	P42	P38	P34			Yes	Yes
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

WAR Hazard gone!

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
10	*FU*	P36	P34	P4	P42	P38	P40	P12		P30

- Notice that P32 not listed in Rename Table
  - Still live. Must not be reallocated by accident

Renamed Scoreboard 11

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10	11		

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
5	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
2	Add	Yes	Addd	P42	P38	P34			Yes	Yes
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
11	*FU*	P36	P34	P4	P42	P38	P40	P12		P30

Renamed Scoreboard 12

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10	11		

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
4	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
1	Add	Yes	Addd	P42	P38	P34			Yes	Yes
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
12	*FU*	P36	P34	P4	P42	P38	P40	P12		P30

Renamed Scoreboard 13

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10	11	13	

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
3	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
0	Add	Yes	Addd	P42	P38	P34			Yes	Yes
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
13	*FU*	P36	P34	P4	P42	P38	P40	P12		P30

Renamed Scoreboard 14

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10	11	13	14

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
2	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
	Add	No								
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
14	*FU*	P36	P34	P4	P42	P38	P40	P12		P30

Renamed Scoreboard 15

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10	11	13	14

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
1	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
	Add	No								
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
15	*FU*	P36	P34	P4	P42	P38	P40	P12		P30

Renamed Scoreboard 16

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6	16	
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10	11	13	14

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
0	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
	Add	No								
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
16	*FU*	P36	P34	P4	P42	P38	P40	P12		P30

Renamed Scoreboard 17

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6	16	17
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10	11	13	14

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
	Mult1	No								
	Add	No								
	Divide	Yes	Divd	P40	P36	P32	Mult1		Yes	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
17	*FU*	P36	P34	P4	P42	P38	P40	P12		P30

Renamed Scoreboard 18

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6	16	17
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5	18		
ADDD	F6	F8	F2	10	11	13	14

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
	Mult1	No								
	Add	No								
40	Divide	Yes	Divd	P40	P36	P32	Mult1		Yes	Yes

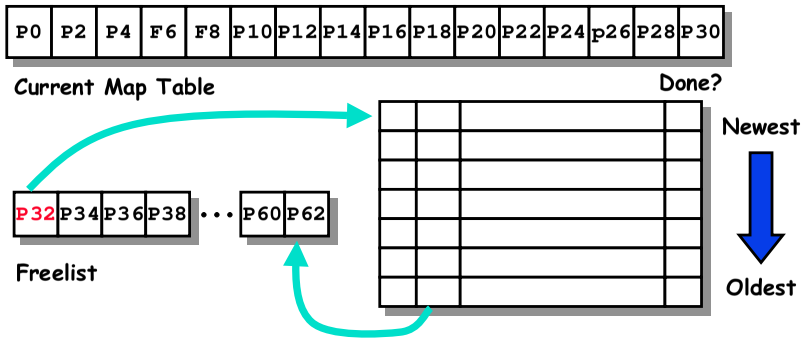
Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
18	*FU*	P36	P34	P4	P42	P38	P40	P12		P30

Explicit Renaming Support Includes:

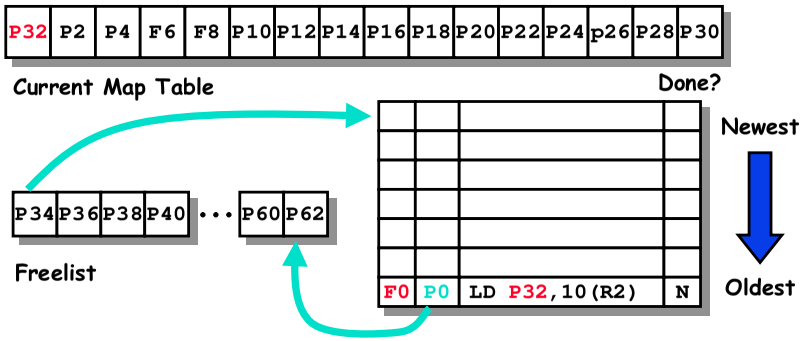
- Rapid access to a table of translations
- A physical register file that has more registers than specified by the ISA
- Ability to figure out which physical registers are free.
  - No free registers ⇒ stall on issue
- Thus, register renaming doesn't require reservation stations. However:
  - Many modern architectures use explicit register renaming + Tomasulo-like reservation stations to control execution.
- **Two Questions:**
  - **How do we manage the "free list"?**
  - **How does Explicit Register Renaming mix with Precise Interrupts?**

Explicit register renaming: (MIPS R10000 Style)



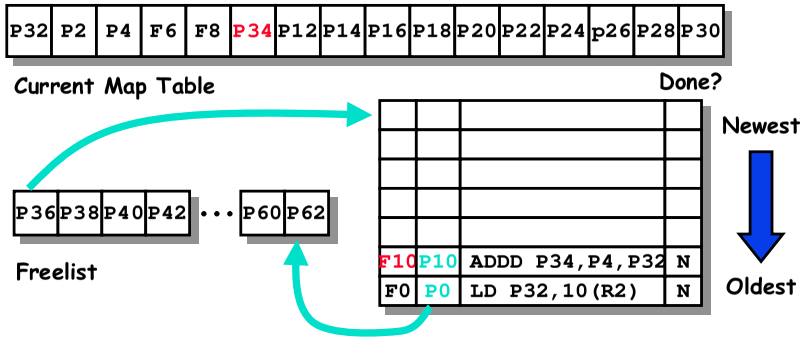
- Physical register file larger than ISA register file
- On issue, each instruction that modifies a register is allocated new physical register from freelist

Explicit register renaming: (MIPS R10000 Style)

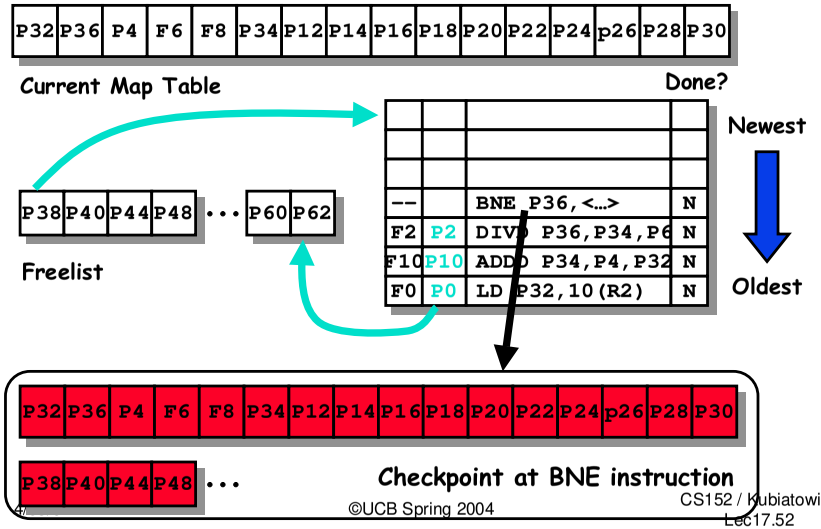


- Note that physical register P0 is "dead" (or not "live") past the point of this load.
  - When we go to commit the load, we free up

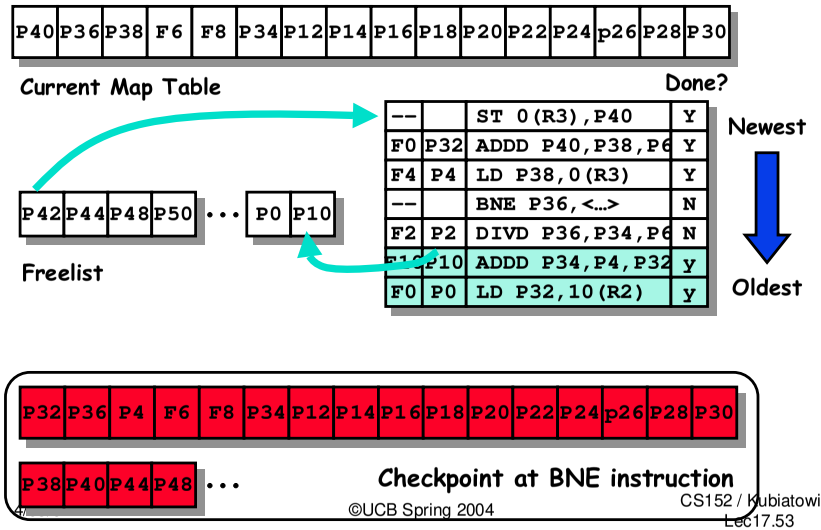
Explicit register renaming: (MIPS R10000 Style)



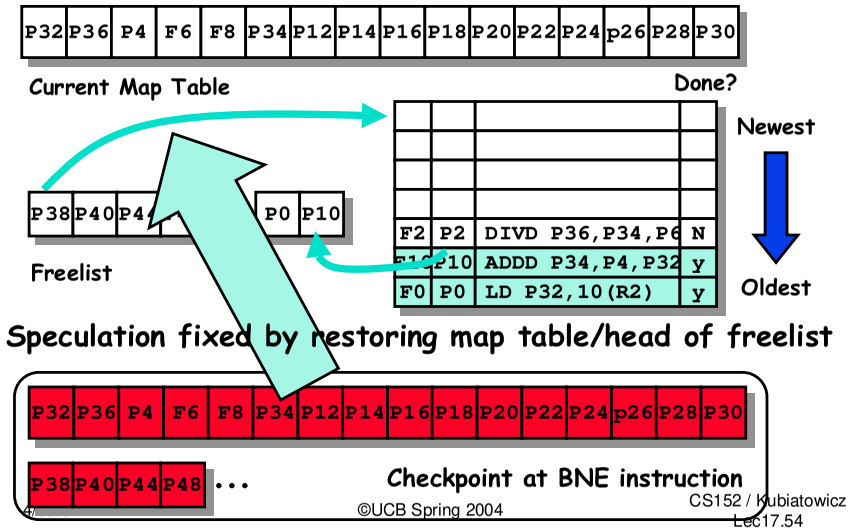
Explicit register renaming: (MIPS R10000 Style)



Explicit register renaming: (MIPS R10000 Style)



Explicit register renaming: (MIPS R10000 Style)



Limits to Multi-Issue Machines

° **Multi-issue: simple matter of accounting**
- Must do dataflow analysis across multiple instructions simultaneously
- Rename table updated as if instructions happened serially!

° **Inherent limitations of ILP**
- 1 branch in 5: How to keep a 5-way superscalar busy?
- Latencies of units: many operations must be scheduled
- Need about Pipeline Depth x No. Functional Units of independent instructions to keep fully busy
- Increase ports to Register File
  - VLIW example needs 7 read and 3 write for Int. Reg. & 5 read and 3 write for FP reg
- Increase ports to memory
- Current state of the art: Many hardware structures (such as issue/rename logic) has delay proportional to square of number of instructions issued/cycle

Limits to ILP

Conflicting studies of amount
- Benchmarks (vectorized Fortran FP vs. integer C programs)
- Hardware sophistication
- Compiler sophistication

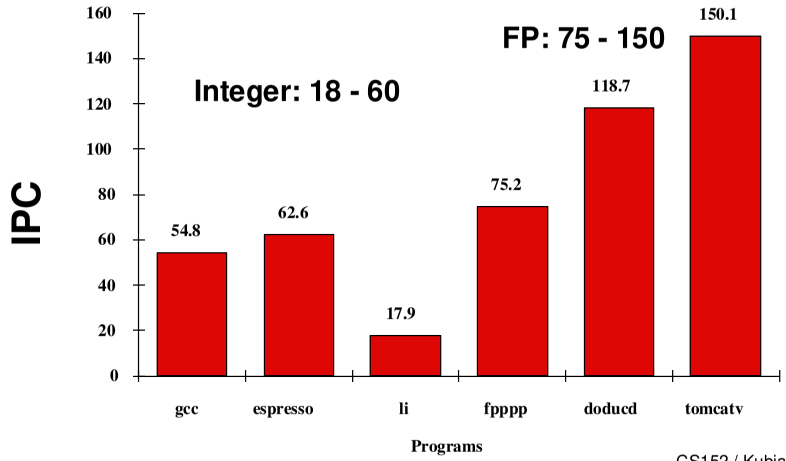
Initial HW Model here; MIPS compilers.

Assumptions for ideal/perfect machine to start:

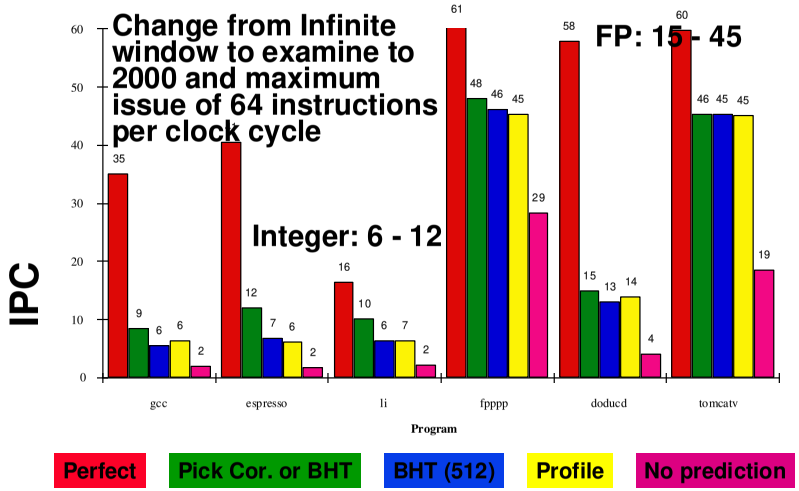
1. *Register renaming*–infinite virtual registers and all WAW & WAR hazards are avoided
2. *Branch prediction*–perfect; no mispredictions
3. *Jump prediction*–all jumps perfectly predicted => machine with perfect speculation & an unbounded buffer of instructions available
4. *Memory-address alias analysis*–addresses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

One cycle latency for all instructions; unlimited number of instructions issued per clock cycle

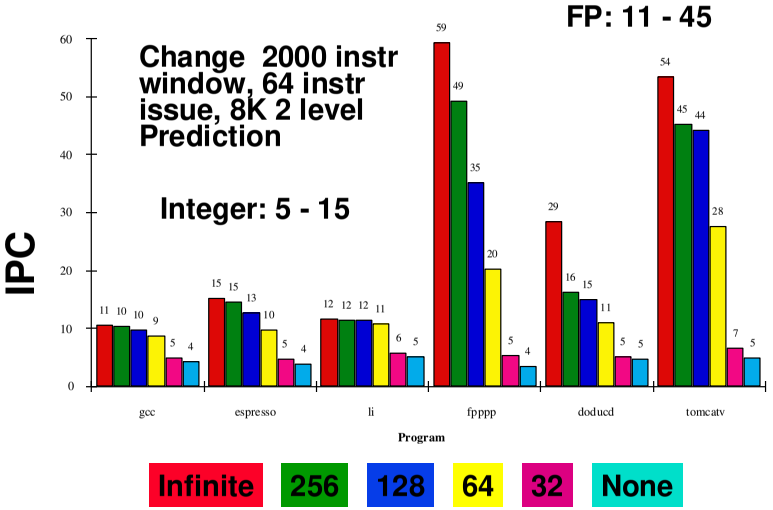
Upper Limit to ILP: Ideal Machine



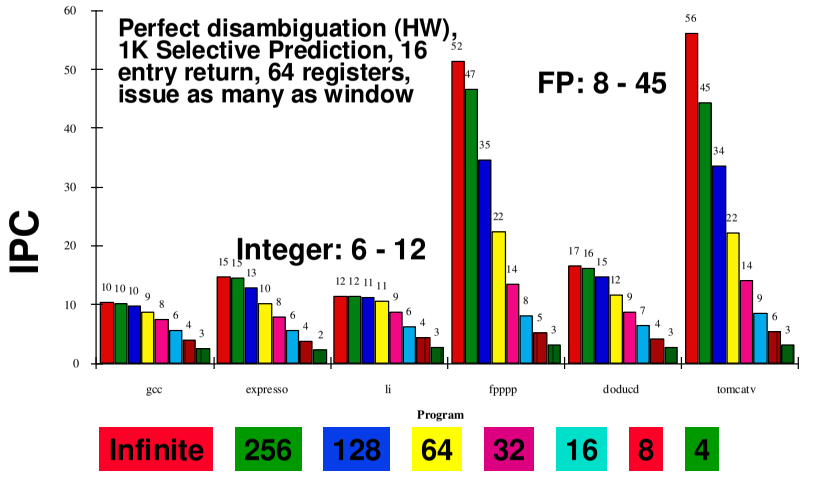
More Realistic HW: Branch Impact



More Realistic HW: Register Impact (rename regs)



Realistic HW for '9X: Window Impact



Summary:

- Modern computer architects predict everything:
  - Branches/Data Dependencies/Data!
- Fairly simple hardware structures can do a good job of predicting branches:
  - Branch Target Buffer (BTB) identifies branches and branch offsets
  - Branch History Table (BHT) does prediction
- More Sophisticated prediction: Correlation
  - Different branches depend on one another!
- Explicit Renaming: more physical registers than ISA.
  - Separates *renaming* from *scheduling*
    - Opens up lots of options for resolving RAW hazards
  - Rename table: tracks current association between architectural registers and physical registers
  - Potentially complicated rename table management
- Parallelism hard to get from real hardware.