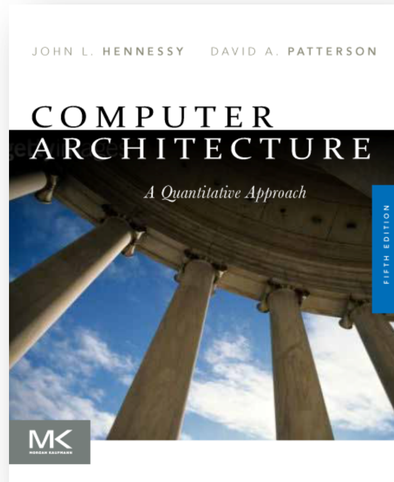


Chapter 3

Instruction-Level Parallelism and Its Exploitation

Copyright © 2012, Elsevier Inc. All rights reserved.

Introduction

- Pipelining become universal technique in 1985
  - Overlaps execution of instructions
  - Exploits "Instruction Level Parallelism"
- Beyond this, there are two main approaches:
  - Hardware-based dynamic approaches
    - Used in server and desktop processors
    - Not used as extensively in PMP processors
  - Compiler-based static approaches
    - Not as successful outside of scientific applications

MK MORGAN KAUFMANN
Copyright © 2012, Elsevier Inc. All rights reserved.

Instruction-Level Parallelism

- When exploiting instruction-level parallelism, goal is to maximize CPI
  - Pipeline CPI =
    - Ideal pipeline CPI +
    - Structural stalls +
    - Data hazard stalls +
    - Control stalls
- Parallelism with basic block is limited
  - Typical size of basic block = 3-6 instructions
  - Must optimize across branches

MK MORGAN KAUFMANN

Copyright © 2012, Elsevier Inc. All rights reserved.

Data Dependence

- Loop-Level Parallelism
  - Unroll loop statically or dynamically
  - Use SIMD (vector processors and GPUs)
- Challenges:
  - Data dependency
    - Instruction *j* is data dependent on instruction *i* if
      - Instruction *i* produces a result that may be used by instruction *j*
      - Instruction *j* is data dependent on instruction *k* and instruction *k* is data dependent on instruction *i*
- Dependent instructions cannot be executed simultaneously

MORGAN KAUFMANN

Copyright © 2012, Elsevier Inc. All rights reserved.

Data Dependence

- Dependencies are a property of programs
- Pipeline organization determines if dependence is detected and if it causes a stall

- Data dependence conveys:
  - Possibility of a hazard
  - Order in which results must be calculated
  - Upper bound on exploitable instruction level parallelism

Copyright © 2012, Elsevier Inc. All rights reserved.

Data Dependence

- Dependencies may be treated in two ways
  - Keep the dependency by avoid hazard
  - Change the code to eliminate the dependency

- Dependencies that flow through memory locations are difficult to detect
  - 20(R4) e 100(R6) may point to the same address
  - 20(R4) e 20(R4) may be different address at different times

MORGAN KAUFMANN

Copyright © 2012, Elsevier Inc. All rights reserved.

Name Dependence

- Two instructions use the same name but no flow of information
  - Not a true data dependence, *but is a problem when reordering instructions*
  - *Antidependence*: instruction j writes a register or memory location that instruction i reads
    - Initial ordering (i before j) must be preserved
  - *Output dependence*: instruction i and instruction j write the same register or memory location
    - Ordering must be preserved

```
S.D     F4, 0(R1)
DADDIU  R1, R1, #-8
```

- To resolve, use renaming techniques

MK MORGAN KAUFMANN

Copyright © 2012, Elsevier Inc. All rights reserved.

Other Factors

- Data Hazards
  - Read after write (RAW)
  - Write after write (WAW)
  - Write after read (WAR)

- Control Dependence
  - Ordering of instruction i with respect to a branch instruction
    - Instruction control dependent on a branch cannot be moved before the branch so that its execution is no longer controller by the branch
    - An instruction not control dependent on a branch cannot be moved after the branch so that its execution is controlled by the branch

Copyright © 2012, Elsevier Inc. All rights reserved.

Examples

- Example 1:

```
    DADDU R1,R2,R3
    BEQZ R4,L
    DSUBU R1,R1,R6
L:  …
    OR R7,R1,R8
```

- OR instruction dependent on DADDU and DSUBU

- Example 2:

```
    DADDU R1,R2,R3
    BEQZ R12,skip
    DSUBU R4,R5,R6
    DADDU R5,R4,R9
skip:
    OR R7,R8,R9
```

- Assume R4 isn't used after skip
  - Possible to move DSUBU before the branch

MORGAN KAUFMANN

Copyright © 2012, Elsevier Inc. All rights reserved.

Compiler Techniques for Exposing ILP

- Pipeline scheduling
  - Separate dependent instruction from the source instruction by the pipeline latency of the source instruction
- Example:

```
for (i=999; i>=0; i=i-1)
  x[i] = x[i] + s;
```

Instruction producing result	Instruction using result	Latency in clock cycles
FP ALU op	Another FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1
Load double	Store double	0

MK MORGAN KAUFMANN

Copyright © 2012, Elsevier Inc. All rights reserved.

Pipeline Stalls


```
Loop:   L.D     F0,0(R1)
        stall
        ADD.D F4,F0,F2
        stall
        stall
        S.D F4,0(R1)
        DADDUI R1,R1,#-8
        stall (assume integer load latency is 1)
        BNE R1,R2,Loop
```

Instruction producing result	Instruction using result	Latency in clock cycles
FP ALU op	Another FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1
Load double	Store double	0

Copyright © 2012, Elsevier Inc. All rights reserved.

Pipeline Stalls (Ex p158)

			issued cycle
Loop:	L.D	F0,0(R1)	1
	stall		2
	ADD.D	F4,F0,F2	3
	stall		4
	stall		5
	S.D	F4,0(R1)	6
	DADDUI	R1,R1,#-8	7
	stall		8
	BNE	R1,R2,Loop	9

Instruction producing result	Instruction using result	Latency in clock cycles
FP ALU op	Another FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1
Load double	Store double	0

Pipeline Scheduling

Scheduled code:

```
Loop:   L.D     F0,0(R1)
        DADDUI R1,R1,#-8
        ADD.D F4,F0,F2
        stall
        stall
        S.D F4,8(R1)
        BNE R1,R2,Loop
```

Instruction producing result	Instruction using result	Latency in clock cycles
FP ALU op	Another FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1
Load double	Store double	0

MORGAN KAUFMANN
Copyright © 2012, Elsevier Inc. All rights reserved.

Pipeline Scheduling

Scheduled code:

```
                                          issued cycle
Loop:   L.D       F0,0(R1)                1
        DADDUI    R1,R1,#-8
        ADD.D     F4,F0,F2
        stall
        stall
        S.D       F4,8(R1)
        BNE       R1,R2,Loop
```

- **7 cycles**
- **Only 3 useful: L.D, ADD, S.D**
- **Remaining 4 → loop overhead**

Instruction producing result	Instruction using result	Latency in clock cycles
FP ALU op	Another FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1
Load double	Store double	0

MORGAN KAUFMANN

Loop Unrolling

- Loop unrolling
  - Unroll by a factor of 4 (assume # elements is divisible by 4)
  - Eliminate unnecessary instructions

```
Loop:   L.D F0,0(R1)
        ADD.D F4,F0,F2
        S.D F4,0(R1) ;drop DADDUI & BNE
        L.D F6,-8(R1)
        ADD.D F8,F6,F2
        S.D F8,-8(R1) ;drop DADDUI & BNE
        L.D F10,-16(R1)
        ADD.D F12,F10,F2
        S.D F12,-16(R1) ;drop DADDUI & BNE
        L.D F14,-24(R1)
        ADD.D F16,F14,F2
        S.D F16,-24(R1)
        DADDUI R1,R1,#-32
        BNE R1,R2,Loop
```

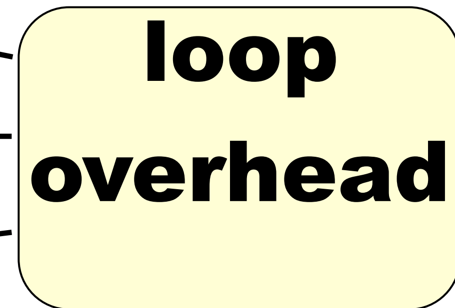
- note: number of live registers vs. original loop

Copyright © 2012, Elsevier Inc. All rights reserved.

Exmpl p159: Loop Unrolling

- Loop unrolling (on same code);
  - Unroll by a factor of 4 (assume # elements is divisible by 4)
  - Eliminate unnecessary instructions

```
Loop:   L.D F0,0(R1)
        ADD.D F4,F0,F2
        S.D F4,0(R1) ;drop DADDUI & BNE
        L.D F6,-8(R1)
        ADD.D F8,F6,F2
        S.D F8,-8(R1) ;drop DADDUI & BNE
        L.D F10,-16(R1)
        ADD.D F12,F10,F2
        S.D F12,-16(R1) ;drop DADDUI & BNE
        L.D F14,-24(R1)
        ADD.D F16,F14,F2
        S.D F16,-24(R1)
        DADDUI R1,R1,#-32
        BNE R1,R2,Loop
```

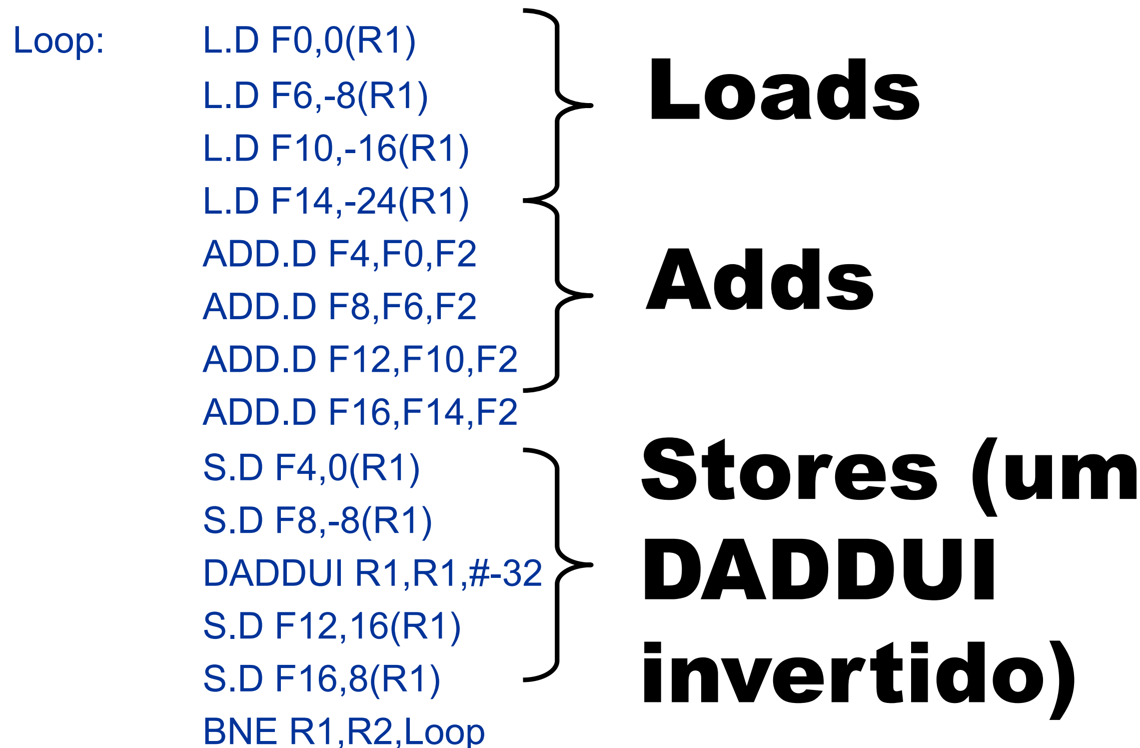


- note: number of live registers vs. original loop

- **Lots of data dependencies (solution ahead)**

Exmpl p159: Loop Unrolling + scheduling

- Pipeline schedule the unrolled loop:



- **No more data dependencies**
- **Total: 14 cycles → 3.5 cycles / iter**

Strip Mining

- Unknown number of loop iterations?
  - Number of iterations = n
  - Goal: make k copies of the loop body
  - Generate pair of loops:
    - First executes n mod k times
    - Second executes n / k times
    - “Strip mining”

MORGAN KAUFMANN

Copyright © 2012, Elsevier Inc. All rights reserved.

Branch Prediction

- Basic 2-bit predictor:
  - For each branch:
    - Predict taken or not taken
    - If the prediction is wrong two consecutive times, change prediction
- Correlating predictor:
  - Multiple 2-bit predictors for each branch
  - One for each possible combination of outcomes of preceding *n* branches
- Local predictor:
  - Multiple 2-bit predictors for each branch
  - One for each possible combination of outcomes for the last *n* occurrences of this branch
- Tournament predictor:
  - Combine correlating predictor with local predictor

MORGAN KAUFMANN

Copyright © 2012, Elsevier Inc. All rights reserved.

Correlating branch predictor

```
if (aa==2)
        aa=0;
if (bb==2)
        bb=0;
if (aa!=bb) {
```

```
        DADDIU   R3,R1,#–2
        BNEZ     R3,L1          ;branch b1    (aa!=2)
        DADD     R1,R0,R0       ;aa=0
L1:     DADDIU   R3,R2,#–2
        BNEZ     R3,L2          ;branch b2    (bb!=2)
        DADD     R2,R0,R0       ;bb=0
L2:     DSUBU    R3,R1,R2       ;R3=aa-bb
        BEQZ     R3,L3          ;branch b3    (aa==bb)
```

- branch b3 is correlated to b1 and b2.
  - B1 and b2 "not taken" → b3 "taken"
- Individual history (uncorrelating branch predictor) does not capture this behavior
- Correlating predictors or two-level predictors do

Correlating branch predictor

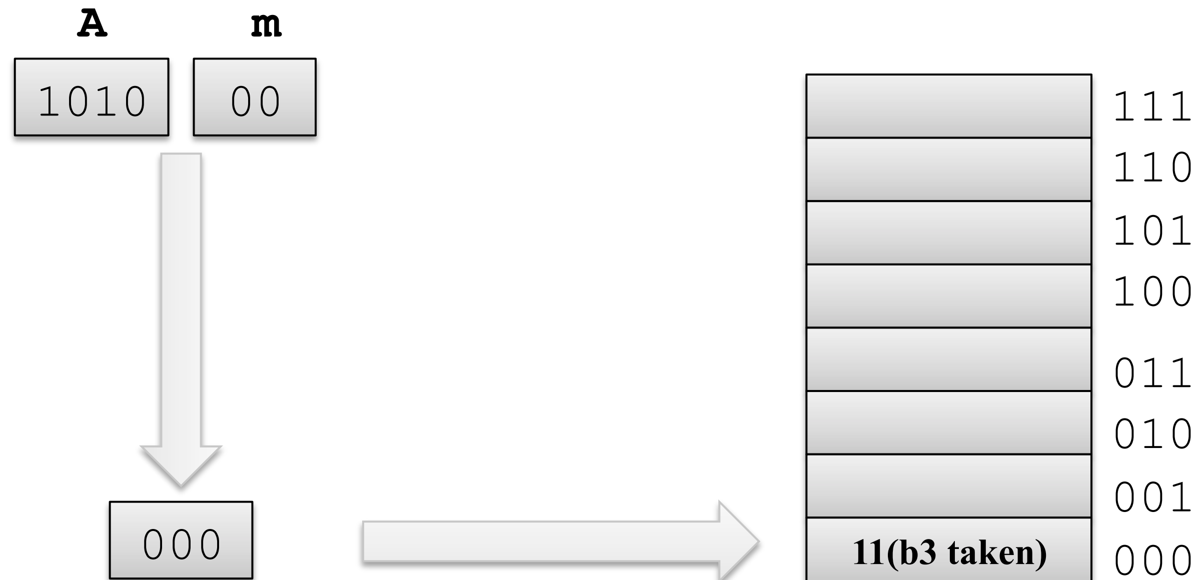
- Use information about branches to make a prediction
- ex: (1,2) uses the behavior of the last branch to choose between a pair of 2 bit-predictors
- ex: (m,n) uses the last “m” branches to pick a n-bit branch predictor among 2^m predictors (each o f which is an n-bit predictor for a single branch)
- Improves prediction and HW is simple

Correlating branch predictor

- Hardware:
  - histórico: shift register de m bits, cada bit registra se aquele branch foi tomado ou não
  - o branch prediction buffer pode ser indexado por: (lower order branch address) concat (m-bit shift register)

```
if (aa==2)
        aa=0;
if (bb==2)
        bb=0;
if (aa!=bb) {
```

```
Suppose:
    b1 at 0xF0
    b2 at 0xF8
    b3 at 0xFA
    m reg = 00
```

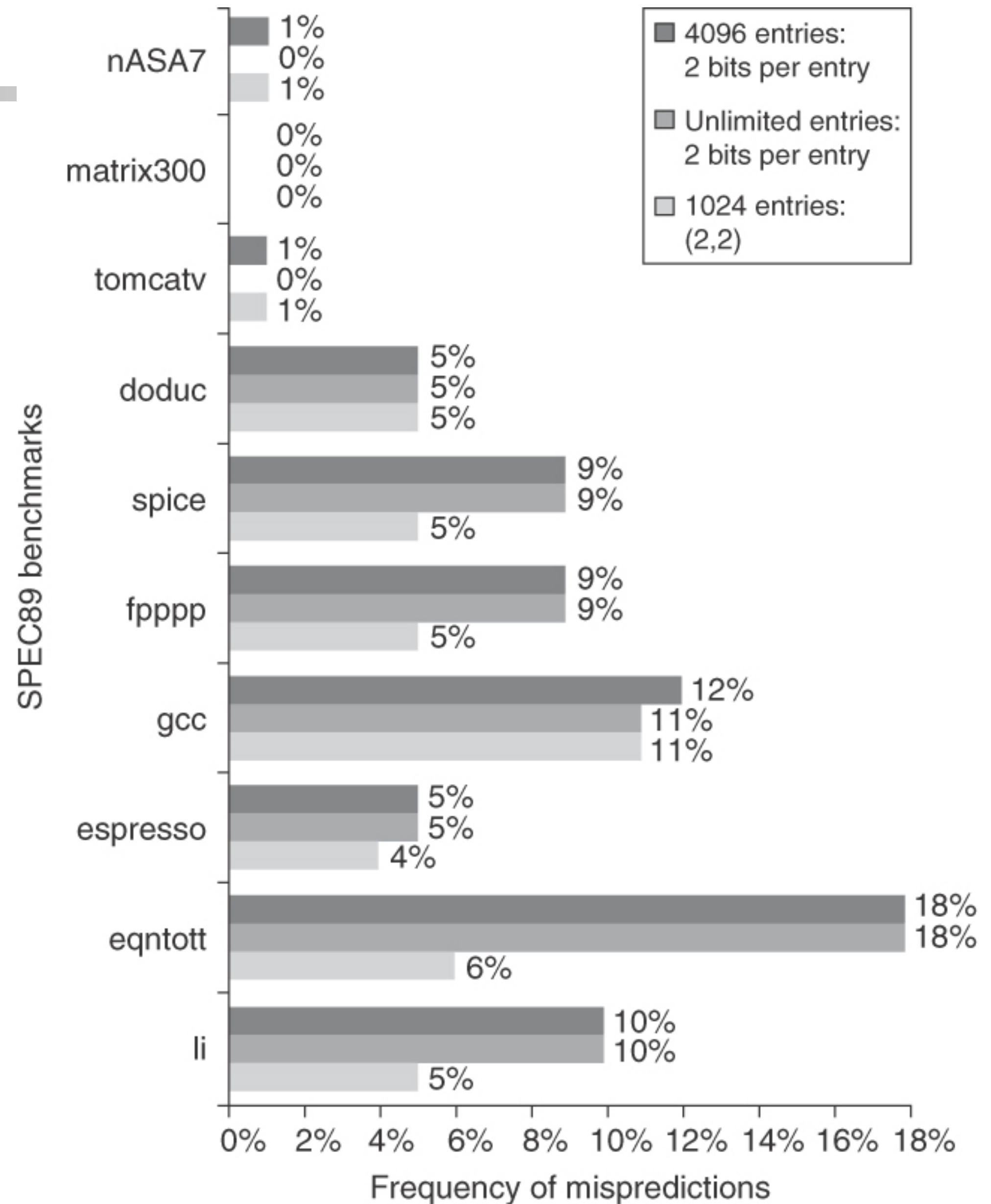


Correlating branch predictor

- Ex: Um buffer (2,2) com 64 linhas
  - índice = 6 bits = 4 bits (branch address) concat 2 bits (shift register)
- Para comparar efetividade, esquemas com mesmo hw
  - bits em (m, n) = 2^m x n x (itens selecionados pelo endereço)
- Para comparar efetividade, esquemas com mesmo hw
  - bits em (m, n) = 2^m x n x (itens selecionados pelo endereço)
- Exemplo anterior: (2,2) com 64 linhas
  - m=2, n=2, endereço de 4 bits → 2^4 = 16;
    2^2 x 2 x 2^4 = 128 (matriz de 64 linhas, 2 bits/linha)
- Exemplo1 página 164
  - (0,2) com 4k linhas? (4k precisam 12 bits de índice)
    → m=0, n=2, 12 bits → 2^0 x 2 x 2^{12} = 8k
- Exemplo2 página 164
  - (2,2) com 8k total de hardware?
    2^2 x 2 x (nº itens selecionados) = 8k
  - 8 x (nº itens selecionados) = 8k
  - nº itens selecionados = 1k → 10 lower bits do endereço

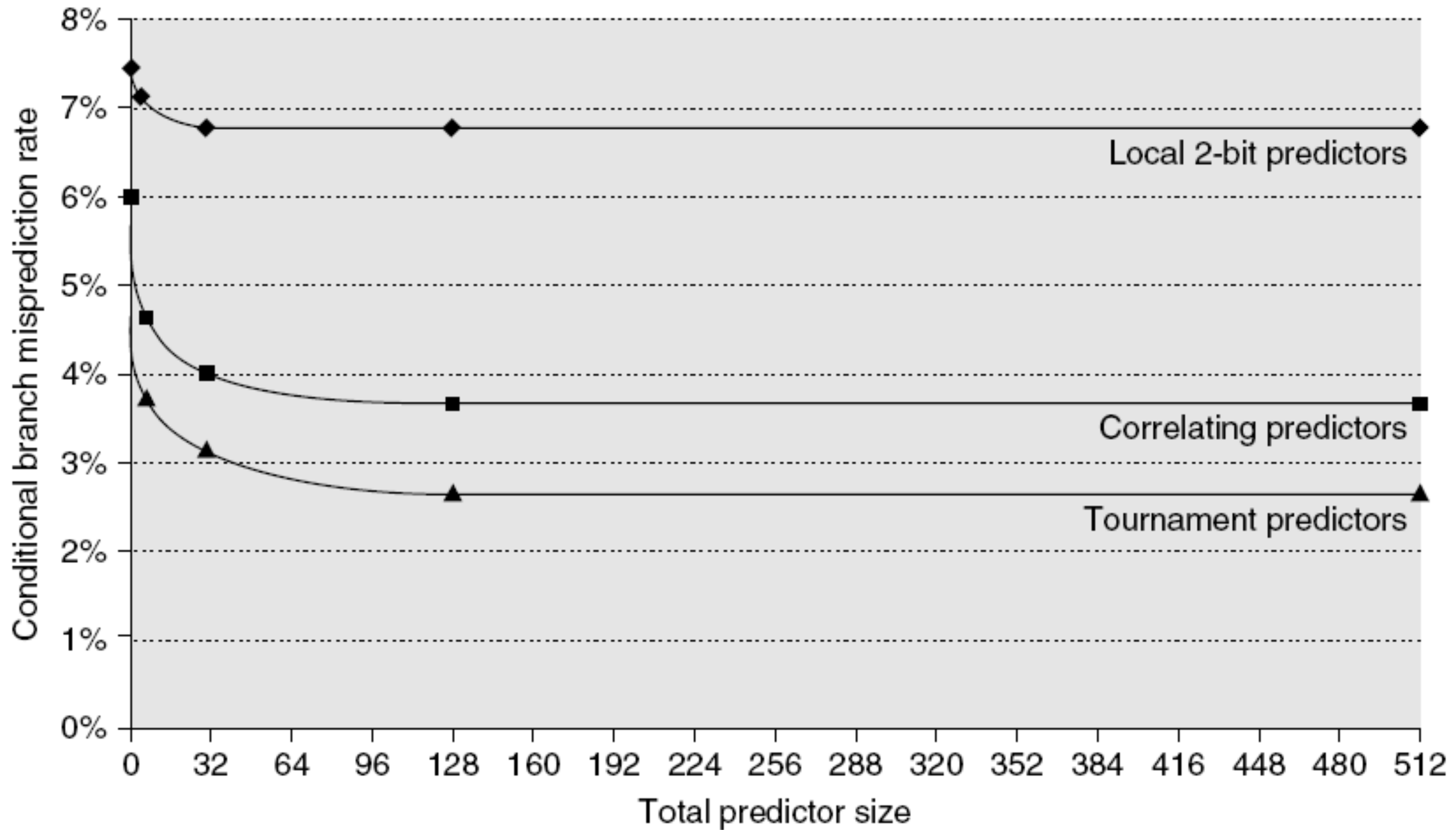
Performance

Figure 3.3 Comparison of 2-bit predictors. A **noncorrelating** predictor for 4096 bits is first, followed by a **noncorrelating** 2-bit predictor with unlimited entries and a 2-bit **correlating** predictor with 2 bits of global history and a total of 1024 entries. Although these data are for an older version of SPEC, data for more recent SPEC benchmarks would show similar differences in accuracy.



MORGAN KAUFMANN

Branch Prediction Performance



Branch predictor performance

Copyright © 2012, Elsevier Inc. All rights reserved.

Misprediction rate no Intel i7

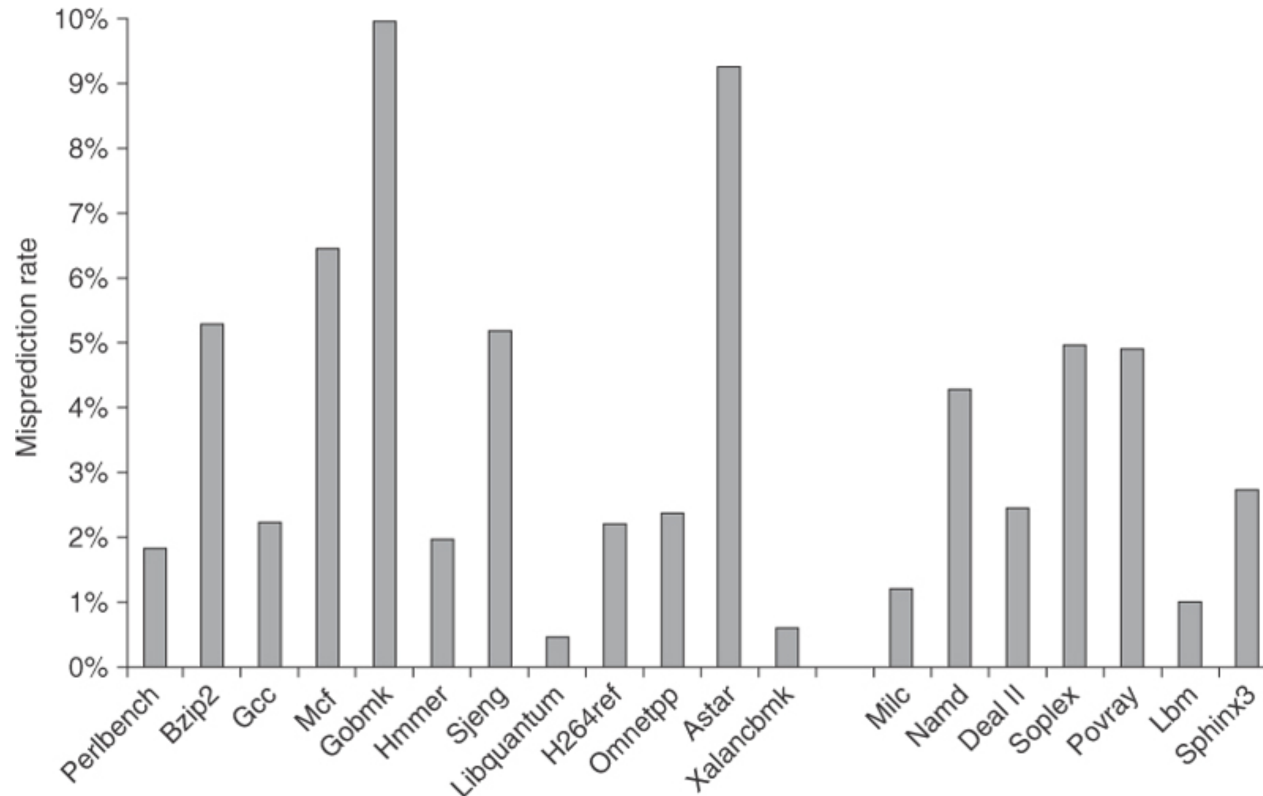


Figure 3.5 The misprediction rate for 19 of the SPECCPU2006 benchmarks versus the number of successfully retired branches is slightly higher on average for the integer benchmarks than for the FP (4% versus 3%). More importantly, it is much higher for a few benchmarks.

Dynamic Scheduling

- Rearrange order of instructions to reduce stalls while maintaining data flow

- Advantages:
  - Compiler doesn't need to have knowledge of microarchitecture
  - Handles cases where dependencies are unknown at compile time

- Disadvantage:
  - Substantial increase in hardware complexity
  - Complicates exceptions

MORGAN KAUFMANN

Copyright © 2012, Elsevier Inc. All rights reserved.

Dynamic Scheduling

- Dynamic scheduling implies:
  - Out-of-order execution
  - Out-of-order completion

- Creates the possibility for WAR and WAW hazards

- Tomasulo's Approach
  - Tracks when operands are available
  - Introduces register renaming in hardware
    - Minimizes WAW and WAR hazards

MK MORGAN KAUFMANN

Copyright © 2012, Elsevier Inc. All rights reserved.

Register Renaming

- Example:

```
DIV.D F0,F2,F4
ADD.D F6,F0,F8
S.D F6,0(R1)
SUB.D F8,F10,F14
MUL.D F6,F10,F8
```

antidependence (SUB.D F8 → ADD.D F8)

antidependence (MUL.D F6 → S.D F6)

+ name dependence with F6

Copyright © 2012, Elsevier Inc. All rights reserved.

Register Renaming – WAR e WAW

- Example (3 hazards):

```
DIV.D    F0,  F2,   F4
ADD.D    F6,  F0,   F8
S.D      F6,  0(R1)
SUB.D    F8,  F10,  F14
MUL.D    F6,  F10,  F8
```

ouput dependence (WAW em F6)

antidependence (WAR em F8)

antidependence (WAR em F6)

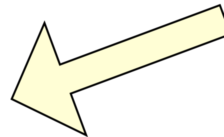
- Além disso há 3 RAW (true data dependencies)
  - F0 (DIV.D e ADD.D)
  - F8 (SUB.D e MUL.D)
  - F6 (ADD.D e S.D)

- As antidependências podem ser eliminadas por register renaming

Register Renaming

- Example:

DIV.D	F0,	F2,	F4
ADD.D	S,	F0,	F8
S.D	S,	0(R1)	
SUB.D	T,	F10,	F14
MUL.D	F6,	F10,	T



DIV.D	**F0,**	**F2,**	**F4**
ADD.D	**F6,**	**F0,**	**F8**
S.D	**F6,**	**0(R1)**	
SUB.D	**F8,**	**F10,**	**F14**
MUL.D	**F6,**	**F10,**	**F8**

- Supor existência de 2 registradores temporários S e T
- Usos posteriores de F8 devem ser substituídos por T
  - pode ser feito estaticamente pelo compilador, mas é complicado
- O algoritmo de Tomasulo trata corretamente renaming entre loops diferentes
- Now only RAW hazards remain, which can be strictly ordered

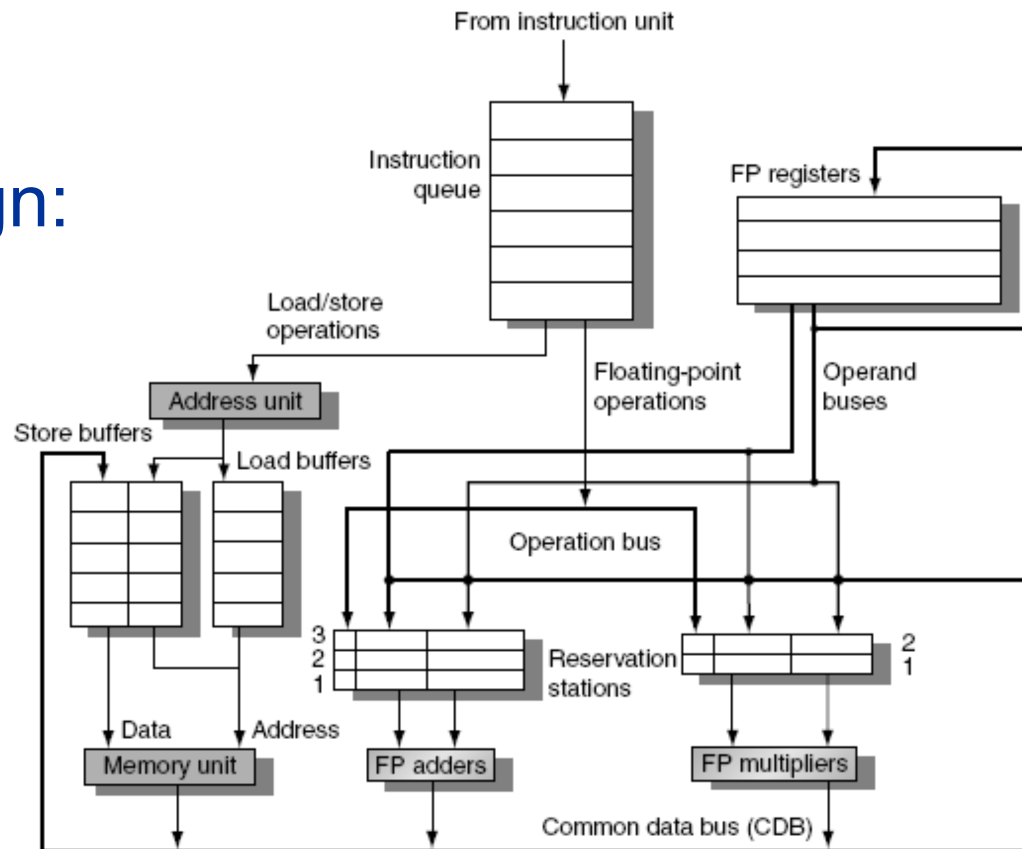
Register Renaming

- Register renaming is provided by reservation stations (RS)
  - Contains:
    - The instruction
    - Buffered operand values (when available)
    - Reservation station number of instruction providing the operand values
  - RS fetches and buffers an operand as soon as it becomes available (not necessarily involving register file)
  - Pending instructions designate the RS to which they will send their output
    - Result values broadcast on a result bus, called the common data bus (CDB)
  - Only the last output updates the register file
  - As instructions are issued, the register specifiers are renamed with the reservation station
  - May be more reservation stations than registers

Copyright © 2012, Elsevier Inc. All rights reserved.

Tomasulo's Algorithm

- Load and store buffers
  - Contain data and addresses, act like reservation stations
- Top-level design:



Copyright © 2012, Elsevier Inc. All rights reserved.

Tomasulo's Algorithm

- Three Steps:
  - Issue
    - Get next instruction from FIFO queue
    - If available RS, issue the instruction to the RS with operand values if available
    - If operand values not available, stall the instruction
  - Execute
    - When operand becomes available, store it in any reservation stations waiting for it
    - When all operands are ready, issue the instruction
    - Loads and store maintained in program order through effective address
    - No instruction allowed to initiate execution until all branches that proceed it in program order have completed
  - Write result
    - Write result on CDB into reservation stations and store buffers
      - (Stores must wait until address and value are received)

Copyright © 2012, Elsevier Inc. All rights reserved.

Tomasulo´s Alg: other aspects

- Data structures associated to RS, RF,LD/STO buffers
- Tags: names used to extend register for renaming purposes
  - indicate which RS will produce needed operand
- Issued instruction → wait source operand → refers to RS where it will be written
- Intermediate results act as renamed registers → solves WAW , WAR
- Results are broadcast (CDB), monitored by RS
  - implements forwarding and bypass
  - but, one extra cycle between producing and consuming result
- Tags refer to buffer or FU that will produce a result
  - register names are discarded as instruction → RS
  - on contrary, in scoreboarding operands stay in the registers → no register renaming

Data structures

Busy	Op	Vj	Vk	Qj	Qk	A

- RS
  - Busy: this station and accompanying FU is(not) busy
  - OP: operation to be performed on source operands S1 and S2
  - Vj, Vk: the value of the source operands
    - only one of V fields or the Q field is valid for each operand
    - in loads, Vk holds the offset
  - Qj, Qk: the RSs that will produce source operands; (zero means operand already available or not necessary)
  - A: info for memory address calculation for LD/ST
    - initially, immediate address; then, effective address
- Register File
  - Qi: # of RS that will hold the result to be stored in this register
    - if zero: no one computes results to be stored here (register value can be used)
- LD/ST buffer (each)
  - A: effective address (once 1st step of execution done)

Example

Instruction		Instruction status		
		Issue	Execute	Write Result
L.D	F6,32(R2)	√	√	√
L.D	F2,44(R3)	√	√	
MUL.D	F0,F2,F4	√		
SUB.D	F8,F2,F6	√		
DIV.D	F10,F0,F6	√		
ADD.D	F6,F8,F2	√		

	Reservation stations						
Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	No						
Load2	Yes	Load					44 + Regs[R3]
Add1	Yes	SUB		Mem[32 + Regs[R2]]	Load2		
Add2	Yes	ADD			Add1	Load2	
Add3	No						
Mult1	Yes	MUL		Regs[F4]	Load2		
Mult2	Yes	DIV		Mem[32 + Regs[R2]]	Mult1		

	Register status							
Field	F0	F2	F4	F6	F8	F10	F12	... F30
Qi	Mult1	Load2		Add2	Add1	Mult2		

MORGAN KAUFMANN

Copyright © 2012, Elsevier Inc. All rights reserved.

Example

Instruction		Instruction status Issue	Execute	Write Result
L.D	F6,32(R2)	√	√	√
L.D	F2,44(R3)	√	√	
MUL.D	F0,F2,F4	√		
SUB.D	F8,F2,F6	√		
DIV.D	F10,F0,F6	√		
ADD.D	F6,F8,F2	√		

Name	Busy	Op	Vj	Vk	Qj	Qk	A
				Reservation stations			
Load1	No						
Load2	Yes	Load					44 + Regs[R3]
Add1	Yes	SUB		Mem[32 + Regs[R2]]	Load2		
Add2	Yes	ADD			Add1	Load2	
Add3	No						
Mult1	Yes	MUL		Regs[F4]	Load2		
Mult2	Yes	DIV		Mem[32 + Regs[R2]]	Mult1		

Field	F0	F2	F4	F6	F8	F10	F12	...	F30
				Register status					
Qi	Mult1	Load2		Add2	Add1	Mult2			

Figure 3.7 Reservation stations and register tags shown when all of the instructions have issued, but only the first load instruction has completed and written its result to the CDB. The second load has completed effective address calculation but is waiting on the memory unit. We use the array Regs[ ] to refer to the register file and the array Mem[ ] to refer to the memory. Remember that an operand is specified by either a Q field or a V field at any time. Notice that the ADD.D instruction, which has a WAR hazard at the WB stage, has issued and could complete before the DIV.D initiates.

MK MORGAN KAUFMANN

Copyright © 2012, Elsevier Inc. All rights reserved.

Exemplo do Alg. Tomasulo (p176)

- Trecho de programa a ser executado:

```
1    L.D     F6,   34(R2)
2    L.D     F2,   45(R3)
3    MUL.D   F0,   F2,   F4
4    SUB.D   F8,   F2,   F6
5    DIV.D   F10,  F0,   F6
6    ADD.D   F6,   F8,   F2
```

RAW?: (1-4); (1-5); (2-3); (2-4); (2-6); ....
WAW?: (1-6)
WAR?: (5-6)

MORGAN KAUFMANN

Exemplo do Alg. Tomasulo

- Assumir as seguintes latências:
  - Load: 1 ciclo addr + 1 ciclo exec
  - Add 2 ciclos
  - Multiplicação: 10 ciclos
  - Divisão: 40 ciclos
- Load-Store:
  - Calcula o endereço efetivo (FU)
  - Load ou Store buffers
  - Acesso à memória (somente load)
  - Write Result
    - Load: envia o valor para o registador e/ou reservation stations
    - Store: escreve o valor na memória
    - (escritas somente no estágio “WB” – simplifica o algoritmo de Tomasulo)

Exemplo do Alg. Tomasulo

Instruções do programa

3 estágios da execução

Instruction status:

Instruction		*j*	*k*	*Issue*	*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			

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

3 Load/Buffers

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	No					

Latência (que falta) da FU

3 FP Adder R.S.
2 FP Mult R.S.

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
0	*FU*									

Clock cycle

MK MORGAN KAUFMANN

Exemplo Tomasulo: Ciclo 1

Instruction status:

Instruction		j	k	Issue	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			

	Busy	Address
Load1	Yes	34+R2
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	No					

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
1	FU				Load1					

Exemplo Tomasulo: Ciclo 2

Instruction status:

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

	Busy	Address
Load1	Yes	34+R2
Load2	Yes	45+
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	No					

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
2	FU		Load2		Load1					

Nota: pode haver múltiplos loads pendentes

Exemplo Tomasulo: Ciclo 3

Instruction status:

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

	Busy	Address
Load1	Yes	34+R2
Load2	Yes	45+R3
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	MULTD		R(F4)	Load2	
	Mult2	No					

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
3	FU	Mult1	Load2		Load1					

- Nota: nomes dos registradores são removidos ("renamed") na Reservation Stations; MULT issued
- Load1 completa; alguém esperando por Load1?

Exemplo Tomasulo: Ciclo 4

Instruction status:

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

	Busy	Address
Load1	No	
Load2	Yes	45+R3
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	Yes	SUBD	M(A1)			Load2
	Add2	No					
	Add3	No					
	Mult1	Yes	MULTD		R(F4)	Load2	
	Mult2	No					

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
4	FU	Mult1	Load2		M(A1)	Add1				

- Load2 completa; alguém esperando por Load2?

MORGAN KAUFMANN

Exemplo Tomasulo: Ciclo 5

Instruction status:

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

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
2	Add1	Yes	SUBD	M(A1)	M(A2)		
	Add2	No					
	Add3	No					
10	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
5	FU	Mult1	M(A2)		M(A1)	Add1	Mult2			

- Timer inicia a contagem regressiva para Add1, Mult1

Exemplo Tomasulo: Ciclo 6

Instruction status:

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

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
1	Add1	Yes	SUBD	M(A1)	M(A2)		
	Add2	Yes	ADDD		M(A2)	Add1	
	Add3	No					
9	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
6	*FU*	Mult1	M(A2)		Add2	Add1	Mult2			

- Issue ADDD, dependência de nome em F6?

MORGAN KAUFMANN

Exemplo Tomasulo: Ciclo 7

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6		

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
0	Add1	Yes	SUBD	M(A1)	M(A2)		
	Add2	Yes	ADDD		M(A2)	Add1	
	Add3	No					
8	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
7	FU	Mult1	M(A2)		Add2	Add1	Mult2			

- Add1 (SUBD) completa; alguém esperando por add1?

MK MORGAN KAUFMANN

Exemplo Tomasulo: Ciclo 8

Instruction status:

Instruction		j	k	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6		

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
	Add1	No					
2	Add2	Yes	ADDD	(M-M)	M(A2)		
	Add3	No					
7	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
8	*FU*	Mult1	M(A2)		Add2	(M-M)	Mult2			

Exemplo Tomasulo: Ciclo 9

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6		

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
1	Add2	Yes	ADDD	(M-M)	M(A2)		
	Add3	No					
6	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
9	FU	Mult1	M(A2)		Add2	(M-M)	Mult2			

MORGAN KAUFMANN

Exemplo Tomasulo: Ciclo 10

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
0	Add2	Yes	ADDD	(M-M)	M(A2)		
	Add3	No					
5	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
10	FU	Mult1	M(A2)		Add2	(M-M)	Mult2			

- Add2 (ADDD) completa; alguém esperando por add2?

Exemplo Tomasulo: Ciclo 11

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
4	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
11	FU	Mult1	M(A2)		(M-M+M	(M-M)	Mult2			

- Resultado de ADDD é escrito!
- Todas as instruções mais rápidas terminam neste ciclo!

MORGAN KAUFMANN

Exemplo Tomasulo: Ciclo 12

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
3	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
12	*FU*	Mult1	M(A2)		(M-M+N	(M-M)	Mult2			

MK MORGAN KAUFMANN

Exemplo Tomasulo: Ciclo 13

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
2	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
13	FU	Mult1	M(A2)		(M-M+M	(M-M)	Mult2			

MK MORGAN KAUFMANN

Exemplo Tomasulo: Ciclo 14

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
1	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
14	FU	Mult1	M(A2)		(M-M+M	(M-M)	Mult2			

Exemplo Tomasulo: Ciclo 15

Instruction status:

Instruction	j	k		Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3	15	
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
0	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
15	FU	Mult1	M(A2)		(M-M+M	(M-M)	Mult2			

- Mult1 (MULTD) completa; alguém esperando por mult1?

Exemplo Tomasulo: Ciclo 16

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3	15	16
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
40	Mult2	Yes	DIVD	M*F4	M(A1)		

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
16	FU	M*F4	M(A2)		(M-M+M	(M-M)	Mult2			

- Agora é só esperar que Mult2 (DIVD) complete

Pulando alguns ciclos (façam como exercício os ciclos faltantes?)

Exemplo Tomasulo: Ciclo 55

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3	15	16
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
1	Mult2	Yes	DIVD	M*F4	M(A1)		

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
55	*FU*	M*F4	M(A2)		(M-M+M	(M-M)	Mult2			

Exemplo Tomasulo: Ciclo 56

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3	15	16
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5	56	
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
0	Mult2	Yes	DIVD	M*F4	M(A1)		

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
56	FU	M*F4	M(A2)		(M-M+M	(M-M)	Mult2			

- Mult2 (DIVD) completa; alguém esperando por mult2?

MORGAN KAUFMANN

Exemplo Tomasulo: Ciclo 57

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3	15	16
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5	56	57
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	Yes	DIVD	M*F4	M(A1)		

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
57	*FU*	M*F4	M(A2)		(M-M+M)	(M-M)	Result			

- In-order issue, out-of-order execution e out-of-order completion.

Tomasulo's Alg simulators

- http://www.dcs.ed.ac.uk/home/hase/webhase/demo/tomasulo.html
- http://www.ecs.umass.edu/ece/koren/architecture/Tomasulo/AppletTomasulo.html
  - já está com o exemplo do CAQA5 carregado, configurável

Controle do Alg. de Tomasulo

Fig 3.9

Instruction state	Wait until	Action or bookkeeping

Issue
FP operation
	Station r empty	`if (RegisterStat[rs].Qi¦0)`
		`{RS[r].Qj ← RegisterStat[rs].Qi}`
		`else {RS[r].Vj ← Regs[rs]; RS[r].Qj ← 0};`
		`if (RegisterStat[rt].Qi¦0)`
		`{RS[r].Qk ← RegisterStat[rt].Qi`
		`else {RS[r].Vk ← Regs[rt]; RS[r].Qk ← 0};`
		`RS[r].Busy ← yes; RegisterStat[rd].Q ← r;`

Load or store	Buffer r empty	`if (RegisterStat[rs].Qi¦0)`
		`{RS[r].Qj ← RegisterStat[rs].Qi}`
		`else {RS[r].Vj ← Regs[rs]; RS[r].Qj ← 0};`
		`RS[r].A ← imm; RS[r].Busy ← yes;`

Load only		`RegisterStat[rt].Qi ← r;`

Store only		`if (RegisterStat[rt].Qi¦0)`
		`{RS[r].Qk ← RegisterStat[rs].Qi}`
		`else {RS[r].Vk ← Regs[rt]; RS[r].Qk ← 0};`

Execute FP operation	`(RS[r].Qj = 0)` and `(RS[r].Qk = 0)`	Compute result: operands are in Vj and Vk
Load/store step 1	RS[r].Qj = 0 & r is head of load-store queue	`RS[r].A ← RS[r].Vj + RS[r].A;`
Load step 2	Load step 1 complete	Read from `Mem[RS[r].A]`

Write result
FP operation
or load
	Execution complete at r & CDB available	`∀x(if (RegisterStat[x].Qi=r) {Regs[x] ← result;`
		`RegisterStat[x].Qi ← 0});`
		`∀x(if (RS[x].Qj=r) {RS[x].Vj ← result;RS[x].Qj ← 0});`
		`∀x(if (RS[x].Qk=r) {RS[x].Vk ← result;RS[x].Qk ← 0});`
		`RS[r].Busy ← no;`

Store	Execution complete at r & `RS[r].Qk = 0`	`Mem[RS[r].A] ← RS[r].Vk;` `RS[r].Busy ← no;`

MK MORGAN KAUFMANN

Exemplo: renaming em um loop

```
Loop:   L.D      F0,0(R1)
        MUL.D    F4,F0,F2
        S.D      F4,0(R1)
        DADDIU   R1,R1,-8
        BNE      R1,R2,Loop; branches if R1¦R2
```

Elementos de um vetor multiplicados por um escalar em F2

- Se “predict taken”
  - RS permitirão múltiplas execuções do loop
- Loads/Stores: podem ser executados fora de ordem desde que endereços (memória) sejam diferentes
- Se mesmo endereço:
  - se LD antes de ST, inversão causa WAR
  - se ST antes de LD, inversão causa RAW
  - inversão de ST causa WAW

Exemplo Loop p181

Instruction		From iteration	Issue	Execute	Write result
L.D	F0,0(R1)	1	√	√	
MUL.D	F4,F0,F2	1	√		
S.D	F4,0(R1)	1	√		
L.D	F0,0(R1)	2	√	√	
MUL.D	F4,F0,F2	2	√		
S.D	F4,0(R1)	2	√		

Reservation stations

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	Yes	Load					Regs[R1] + 0
Load2	Yes	Load					Regs[R1] – 8
Add1	No						
Add2	No						
Add3	No						
Mult1	Yes	MUL		Regs[F2]	Load1		
Mult2	Yes	MUL		Regs[F2]	Load2		
Store1	Yes	Store	Regs[R1]			Mult1	
Store2	Yes	Store	Regs[R1] – 8			Mult2	

Register status

Field	F0	F2	F4	F6	F8	F10	F12	...	F30
Qi	Load2		Mult2						

Figure 3.10 Two active iterations of the loop with no instruction yet completed. Entries in the multiplier reservation stations indicate that the outstanding loads are the sources. The store reservation stations indicate that the multiply destination is the source of the value to store.

Tomasulo: conclusão

- Algoritmo caiu no esquecimento desde o IBM/360
- Motivações para retorno
  - O algoritmo foi desenvolvido antes das caches. Mas caches hoje tem latências imprevisíveis → oportunidade para execução fora de ordem (esconder penalidades)
  - Processadores atuais mais agressivos: dificuldade de static scheduling, necessidade de dynamic scheduling, register renaming, especulação
  - O algoritmo permite grande desempenho sem exigir que o compilador produza código personalizado para uma determinada estrutura de pipeline: valuable property in the era of shrink-wrapped mass market software

Hardware-Based Speculation

- Execute instructions along predicted execution paths but only commit the results if prediction was correct
- Instruction commit: allowing an instruction to update the register file when instruction is no longer speculative
- Need an additional piece of hardware to prevent any irrevocable action until an instruction commits
  - I.e. updating state or taking an execution

MK MORGAN KAUFMANN

Copyright © 2012, Elsevier Inc. All rights reserved.

Reorder Buffer

- Reorder buffer – holds the result of instruction between completion and commit

- Four fields:
  - Instruction type: branch/store/register
  - Destination field: register number
  - Value field: output value
  - Ready field: completed execution?

- Modify reservation stations:
  - Operand source is now reorder buffer instead of functional unit

Copyright © 2012, Elsevier Inc. All rights reserved.

Reorder Buffer

- Register values and memory values are not written until an instruction commits
- On misprediction:
  - Speculated entries in ROB are cleared

- Exceptions:
  - Not recognized until it is ready to commit

MORGAN KAUFMANN
Copyright © 2012, Elsevier Inc. All rights reserved.

Tomasulo + ROB

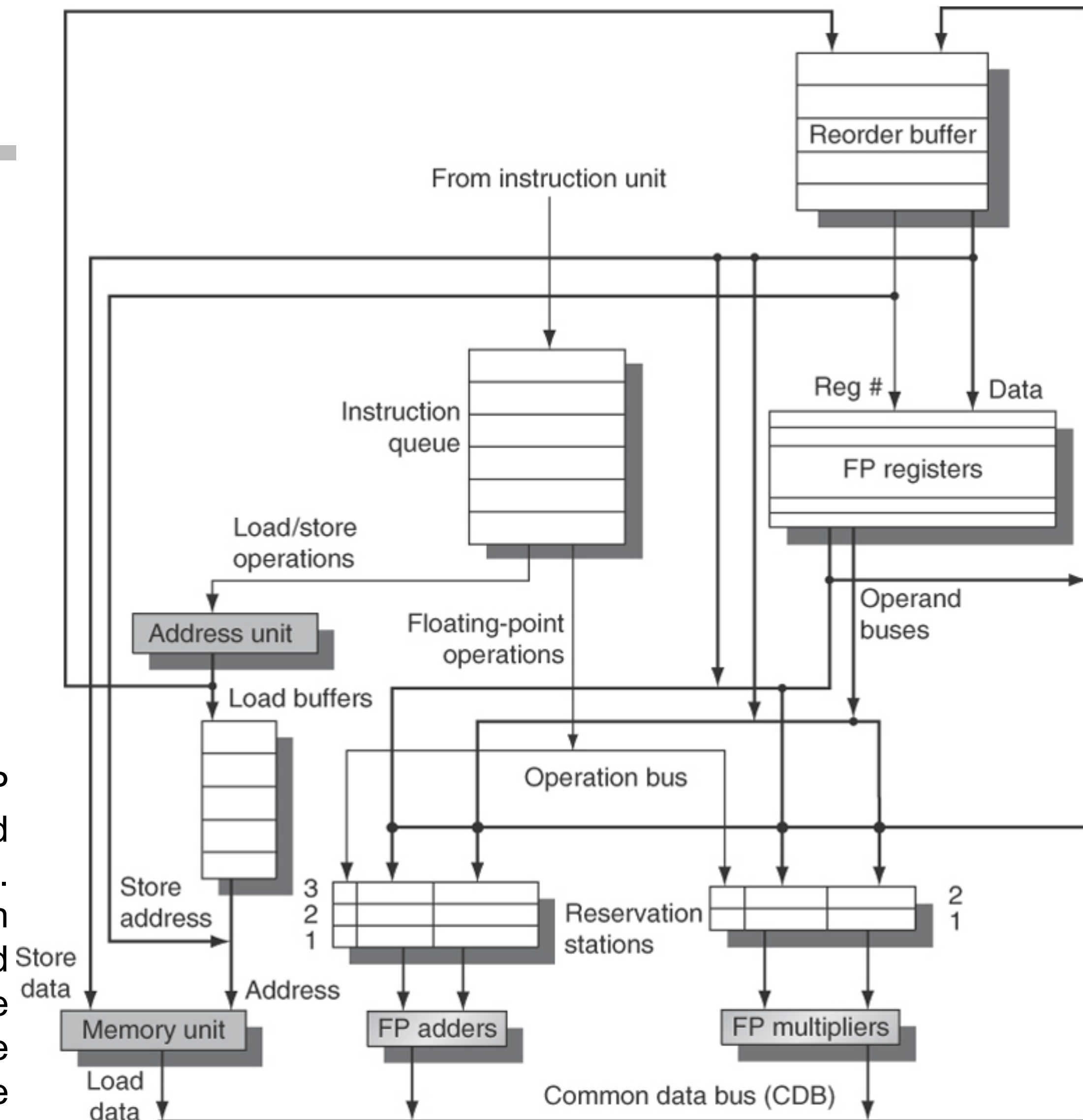


Figure 3.11 The basic structure of a FP unit using Tomasulo's algorithm and extended to handle speculation. Comparing this to Figure 3.6 on page 173, which implemented Tomasulo's algorithm, the major change is the addition of the ROB and the elimination of the store buffer, whose function is integrated into the ROB. This mechanism can be extended to multiple issue by making the CDB wider to allow for multiple completions per clock.

Tomasulo+ROB

- Issue:
  - Se houver espaço no ROB e RS; envia operandos p RS; envia p RS nº linha do ROB
- Execute: como antes; monitora CDB e operandos
- Write result:
  - Broadcast no CDB, junto com o nº da linha do ROB; libera RS;
- Commit: (completion / graduation); depende da inst.
  - se normal commit: instrução chega ao top do ROB e valor presente → atualiza registrador e libera ROB
  - se store: idem, mas atualiza memória
  - branch: se correto → finish; se incorreto → flush ROB e re-inicia a execução da instrução correta

MORGAN KAUFMANN

Exmpl p187: uso do ROB

Example Assume the same latencies for the floating-point functional units as in earlier examples: add is 2 clock cycles, multiply is 6 clock cycles, and divide is 12 clock cycles. Using the code segment below, the same one we used to generate Figure 3.8, show what the status tables look like when the MUL.D is ready to go to commit.

```
L.D      F6,32(R2)
L.D      F2,44(R3)
MUL.D    F0,F2,F4
SUB.D    F8,F2,F6
DIV.D    F10,F0,F6
ADD.D    F6,F8,F2
```

Answer Figure 3.12 shows the result in the three tables. Notice that although the SUB.D instruction has completed execution, it does not commit until the MUL.D commits. The reservation stations and register status field contain the same basic information that they did for Tomasulo's algorithm (see page 176 for a description of those fields). The differences are that reservation station numbers are replaced with ROB entry numbers in the Qj and Qk fields, as well as in the register status fields, and we have added the Dest field to the reservation stations. The Dest field designates the ROB entry that is the destination for the result produced by this reservation station entry.

Exmpl p187: uso do ROB (cont)

Vantagem s/ Tomasulo:

Precise exception possível se MUL → exception

(pós mul ñ commit)

Reorder buffer

Entry	Busy	Instruction		State	Destination	Value
1	No	L.D	F6,32(R2)	Commit	F6	Mem[32 + Regs[R2]]
2	No	L.D	F2,44(R3)	Commit	F2	Mem[44 + Regs[R3]]
3	Yes	MUL.D	F0,F2,F4	Write result	F0	#2 × Regs[F4]
4	Yes	SUB.D	F8,F2,F6	Write result	F8	#2 – #1
5	Yes	DIV.D	F10,F0,F6	Execute	F10	
6	Yes	ADD.D	F6,F8,F2	Write result	F6	#4 + #2

Reservation stations

Name	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Load1	No							
Load2	No							
Add1	No							
Add2	No							
Add3	No							
Mult1	No	MUL.D	Mem[44 + Regs[R3]]	Regs[F4]			#3	
Mult2	Yes	DIV.D		Mem[32 + Regs[R2]]	#3		#5	

FP register status

Field	F0	F1	F2	F3	F4	F5	F6	F7	F8	F10
Reorder #	3						6		4	5
Busy	Yes	No	No	No	No	No	Yes	. . .	Yes	Yes

Figure 3.12 At the time the MUL.D is ready to commit, only the two L.D instructions have committed, although several others have completed execution. The MUL.D is at the head of the ROB, and the two L.D instructions are there only to ease understanding. The SUB.D and ADD.D instructions will not commit until the MUL.D instruction commits, although the results of the instructions are available and can be used as sources for other instructions. The DIV.D is in execution, but has not completed solely due to its longer latency than MUL.D. The Value column indicates the value being held; the format #X is used to refer to a value field of ROB entry X. Reorder buffers 1 and 2 are actually completed but are shown for informational purposes. We do not show the entries for the load/store queue, but these entries are kept in order.

MORGAN KAUFMANN

Mesmo exemplo, sem especulação → comparar

Aqui sub e add completadas → se mul exception → imprecise

Instruction	Instruction status: Issue	Execute	Write result
L.D F6,32(R2)	√	√	√
L.D F2,44(R3)	√	√	√
MUL.D F0,F2,F4	√	√	
SUB.D F8,F2,F6	√	√	√
DIV.D F10,F0,F6	√		
ADD.D F6,F8,F2	√	√	√

Reservation stations

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	No						
Load2	No						
Add1	No						
Add2	No						
Add3	No						
Mult1	Yes	MUL	Mem[44 + Regs[R3]]	Regs[F4]			
Mult2	Yes	DIV		Mem[32 + Regs[R2]]	Mult1		

Register status

Field	F0	F2	F4	F6	F8	F10	F12	...	F30
Qi	Mult1					Mult2			

Figure 3.8 Multiply and divide are the only instructions not finished.

MK MORGAN KAUFMANN

Exmpl p189: ROB e loop

Example Consider the code example used earlier for Tomasulo's algorithm and shown in Figure 3.10 in execution:

```
Loop:   L.D      F0,0(R1)
        MUL.D    F4,F0,F2
        S.D      F4,0(R1)
        DADDIU   R1,R1,#-8
        BNE      R1,R2,Loop    ;branches if R1¦R2
```

Assume that we have issued all the instructions in the loop twice. Let's also assume that the L.D and MUL.D from the first iteration have committed and all other instructions have completed execution. Normally, the store would wait in the ROB for both the effective address operand (R1 in this example) and the value (F4 in this example). Since we are only considering the floating-point pipeline, assume the effective address for the store is computed by the time the instruction is issued.

Answer Figure 3.13 shows the result in two tables.

Exmpl p189: ROB e loop (cont

Reorder buffer					
Entry	**Busy**	**Instruction**	**State**	**Destination**	**Value**
1	No	L.D F0,0(R1)	Commit	F0	Mem[0 + Regs[R1]]
2	No	MUL.D F4,F0,F2	Commit	F4	#1 × Regs[F2]
3	Yes	S.D F4,0(R1)	Write result	0 + Regs[R1]	#2
4	Yes	DADDIU R1,R1,#-8	Write result	R1	Regs[R1] – 8
5	Yes	BNE R1,R2,Loop	Write result		
6	Yes	L.D F0,0(R1)	Write result	F0	Mem[#4]
7	Yes	MUL.D F4,F0,F2	Write result	F4	#6 × Regs[F2]
8	Yes	S.D F4,0(R1)	Write result	0 + #4	#7
9	Yes	DADDIU R1,R1,#-8	Write result	R1	#4 – 8
10	Yes	BNE R1,R2,Loop	Write result		

FP register status									
Field	**F0**	**F1**	**F2**	**F3**	**F4**	**F5**	**F6**	**F7**	**F8**
Reorder #	6				7				
Busy	Yes	No	No	No	Yes	No	No	...	No

Figure 3.13 Only the L.D and MUL.D instructions have committed, although all the others have completed execution. Hence, no reservation stations are busy and none is shown. The remaining instructions will be committed as quickly as possible. The first two reorder buffers are empty, but are shown for completeness.

Exmpl p. 189: ROB e loop (cont)

- Se predição errada (por ex: na 1ª e na última iter)
  - clear ROB: todas as instruções depois do branch
  - permite que as instruções anteriores ao branch commit
  - re-inicia o fetch do endereço correto
- Processadores com especulação
  - Custo de misprediction é maior
  - Qualidade de branch prediction deve ser maior
- Exceções
  - não reconhecidas até o momento do commit
  - registro da exceção fica no ROB até o commit
    - se instrução causadora está em ramo errado de branch, simplesmente flush
    - se ela chega do top do ROB, então commit e trata exceção

Hazards e o ROB

- WAW e WAR through memory
  - ok com especulação e commit em ordem

- RAW through memory, ok com duas restrições
  - Não permitir 2º passo de Load se existir Store no ROB com “destination field” = campo A do Load
  - Cálculo de endereços efetivo de Loads feito em ordem com relação a todos os Stores anteriores

  - Essas 2 restrições garantem que qualquer Load que acesse posição de memória escrita por Store anterior aguarde até que o Store tenha escrito o valor

Multiple Issue and Static Scheduling

- To achieve CPI < 1, need to complete multiple instructions per clock

- Solutions:
  - Statically scheduled superscalar processors
  - VLIW (very long instruction word) processors
  - dynamically scheduled superscalar processors

Copyright © 2012, Elsevier Inc. All rights reserved.

Multiple Issue

Common name	Issue structure	Hazard detection	Scheduling	Distinguishing characteristic	Examples
Superscalar (static)	Dynamic	Hardware	Static	In-order execution	Mostly in the embedded space: MIPS and ARM, including the ARM Coretex A8
Superscalar (dynamic)	Dynamic	Hardware	Dynamic	Some out-of-order execution, but no speculation	None at the present
Superscalar (speculative)	Dynamic	Hardware	Dynamic with speculation	Out-of-order execution with speculation	Intel Core i3, i5, i7; AMD Phenom; IBM Power 7
VLIW/LIW	Static	Primarily software	Static	All hazards determined and indicated by compiler (often implicitly)	Most examples are in signal processing, such as the TI C6x
EPIC	Primarily static	Primarily software	Mostly static	All hazards determined and indicated explicitly by the compiler	Itanium

MK MORGAN KAUFMANN

Copyright © 2012, Elsevier Inc. All rights reserved.

VLIW Processors

- Package multiple operations into one instruction
- Example VLIW processor:
  - One integer instruction (or branch)
  - Two independent floating-point operations
  - Two independent memory references
- Must be enough parallelism in code to fill the available slots

Copyright © 2012, Elsevier Inc. All rights reserved.

VLIW Processors

- Disadvantages:
  - Statically finding parallelism
  - Code size
  - No hazard detection hardware
  - Binary code compatibility

MORGAN KAUFMANN

Copyright © 2012, Elsevier Inc. All rights reserved.

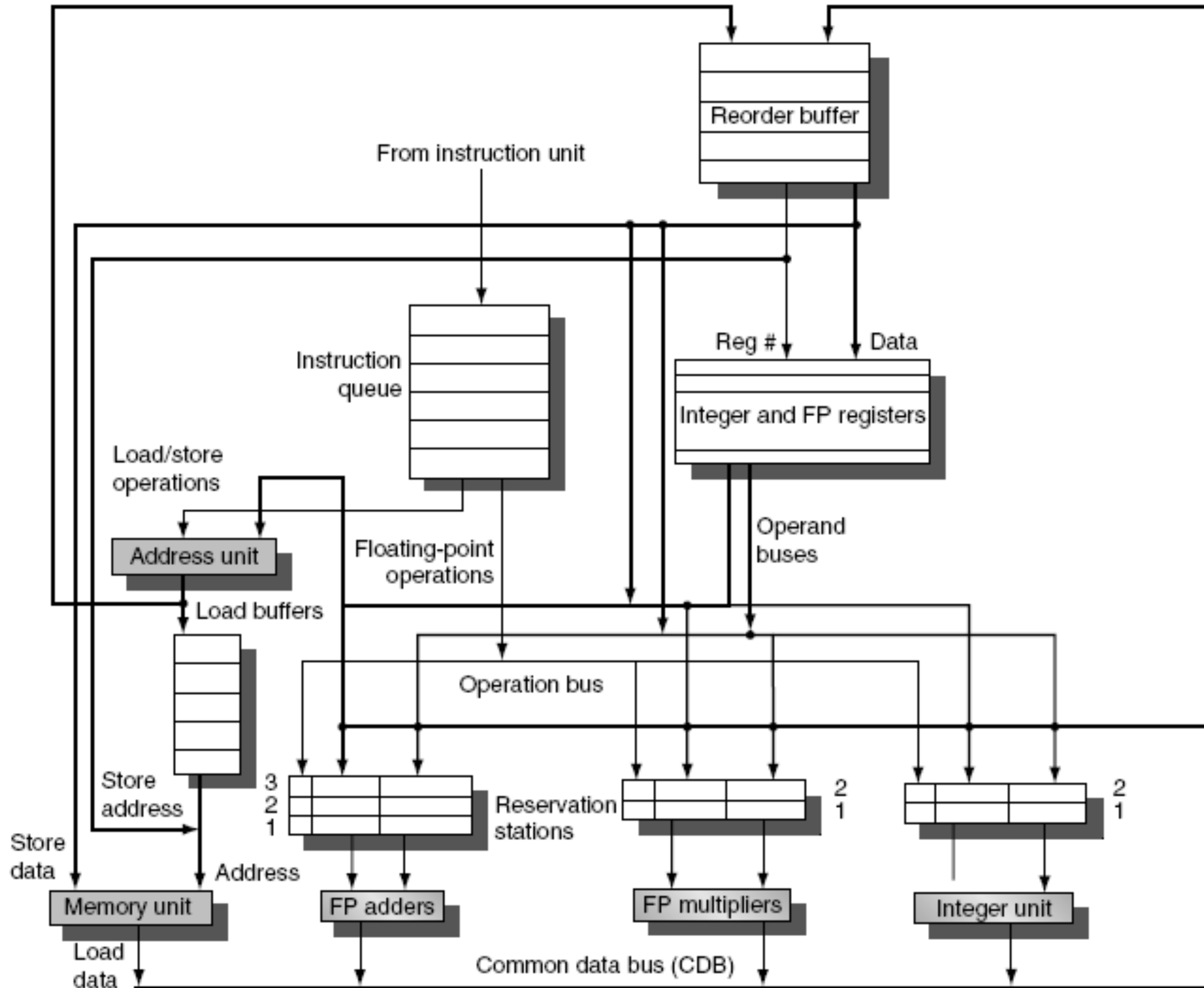
Dynamic Scheduling, Multiple Issue, and Speculation

- Modern microarchitectures:
  - Dynamic scheduling + multiple issue + speculation
- Two approaches:
  - Assign reservation stations and update pipeline control table in half clock cycles
    - Only supports 2 instructions/clock
  - Design logic to handle any possible dependencies between the instructions
  - Hybrid approaches
- Issue logic can become bottleneck

MK MORGAN KAUFMANN

Copyright © 2012, Elsevier Inc. All rights reserved.

Overview of Design



MORGAN KAUFMANN

Copyright © 2012, Elsevier Inc. All rights reserved.

Multiple Issue

- Limit the number of instructions of a given class that can be issued in a “bundle”
  - I.e. on FP, one integer, one load, one store
- Examine all the dependencies amoung the instructions in the bundle
- If dependencies exist in bundle, encode them in reservation stations
- Also need multiple completion/commit

MORGAN KAUFMANN

Copyright © 2012, Elsevier Inc. All rights reserved.

Example

```
Loop:  LD R2,0(R1)          ;R2=array element
       DADDIU R2,R2,#1      ;increment R2
       SD R2,0(R1)          ;store result
       DADDIU R1,R1,#8      ;increment pointer
       BNE R2,R3,LOOP       ;branch if not last element
```


Copyright © 2012, Elsevier Inc. All rights reserved.

Example (No Speculation)

Iteration number	Instructions	Issues at clock cycle number	Executes at clock cycle number	Memory access at clock cycle number	Write CDB at clock cycle number	Comment
1	LD R2,0(R1)	1	2	3	4	First issue
1	DADDIU R2,R2,#1	1	5		6	Wait for LW
1	SD R2,0(R1)	2	3	7		Wait for DADDIU
1	DADDIU R1,R1,#8	2	3		4	Execute directly
1	BNE R2,R3,LOOP	3	7			Wait for DADDIU
2	LD R2,0(R1)	4	8	9	10	Wait for BNE
2	DADDIU R2,R2,#1	4	11		12	Wait for LW
2	SD R2,0(R1)	5	9	13		Wait for DADDIU
2	DADDIU R1,R1,#8	5	8		9	Wait for BNE
2	BNE R2,R3,LOOP	6	13			Wait for DADDIU
3	LD R2,0(R1)	7	14	15	16	Wait for BNE
3	DADDIU R2,R2,#1	7	17		18	Wait for LW
3	SD R2,0(R1)	8	15	19		Wait for DADDIU
3	DADDIU R1,R1,#8	8	14		15	Wait for BNE
3	BNE R2,R3,LOOP	9	19			Wait for DADDIU

MORGAN KAUFMANN

Copyright © 2012, Elsevier Inc. All rights reserved.

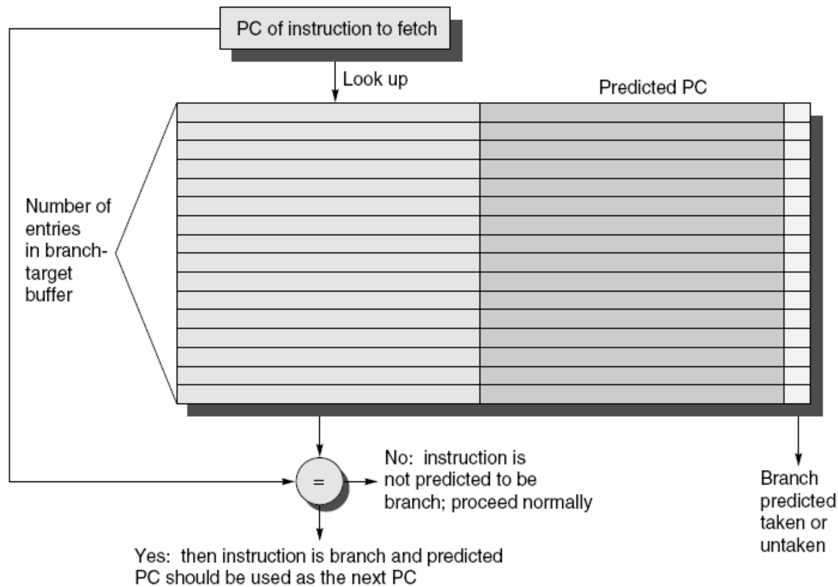
Example

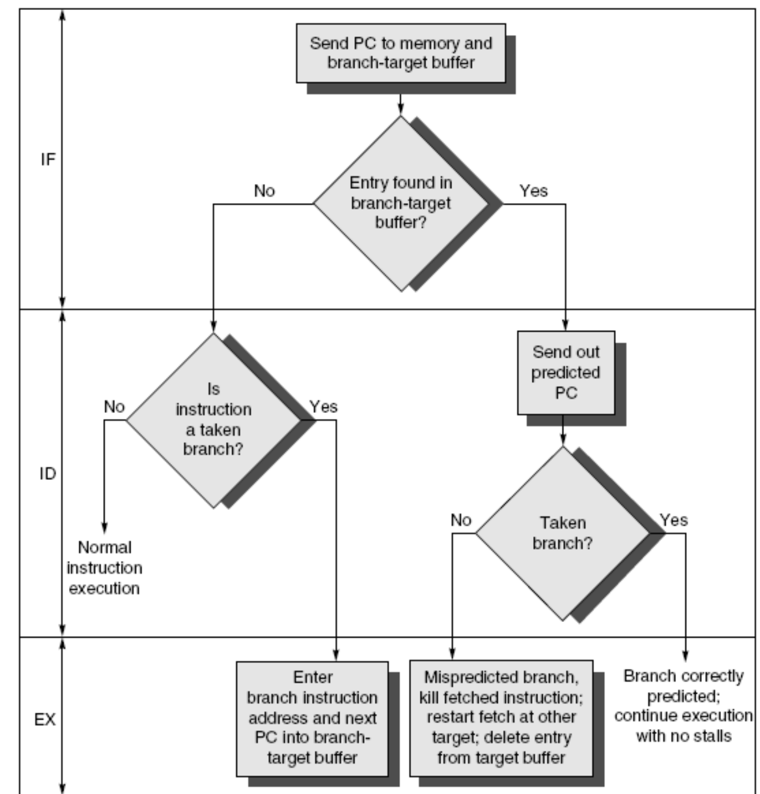
Iteration number	Instructions		Issues at clock number	Executes at clock number	Read access at clock number	Write CDB at clock number	Commits at clock number	Comment
1	LD	R2,0(R1)	1	2	3	4	5	First issue
1	DADDIU	R2,R2,#1	1	5		6	7	Wait for LW
1	SD	R2,0(R1)	2	3			7	Wait for DADDIU
1	DADDIU	R1,R1,#8	2	3		4	8	Commit in order
1	BNE	R2,R3,LOOP	3	7			8	Wait for DADDIU
2	LD	R2,0(R1)	4	5	6	7	9	No execute delay
2	DADDIU	R2,R2,#1	4	8		9	10	Wait for LW
2	SD	R2,0(R1)	5	6			10	Wait for DADDIU
2	DADDIU	R1,R1,#8	5	6		7	11	Commit in order
2	BNE	R2,R3,LOOP	6	10			11	Wait for DADDIU
3	LD	R2,0(R1)	7	8	9	10	12	Earliest possible
3	DADDIU	R2,R2,#1	7	11		12	13	Wait for LW
3	SD	R2,0(R1)	8	9			13	Wait for DADDIU
3	DADDIU	R1,R1,#8	8	9		10	14	Executes earlier
3	BNE	R2,R3,LOOP	9	13			14	Wait for DADDIU

Copyright © 2012, Elsevier Inc. All rights reserved.

Branch-Target Buffer

- Need high instruction bandwidth!
  - Branch-Target buffers
    - Next PC prediction buffer, indexed by current PC





Copyright © 2012, Elsevier Inc. All rights reserved.

Branch Folding

- Optimization:
  - Larger branch-target buffer
  - Add target instruction into buffer to deal with longer decoding time required by larger buffer
  - "Branch folding"

Copyright © 2012, Elsevier Inc. All rights reserved.

Return Address Predictor

- Most unconditional branches come from function returns
- The same procedure can be called from multiple sites
  - Causes the buffer to potentially forget about the return address from previous calls
- Create return address buffer organized as a stack

Copyright © 2012, Elsevier Inc. All rights reserved.

Integrated Instruction Fetch Unit

- Design monolithic unit that performs:
  - Branch prediction
  - Instruction prefetch
    - Fetch ahead
  - Instruction memory access and buffering
    - Deal with crossing cache lines

Copyright © 2012, Elsevier Inc. All rights reserved.

Register Renaming

- Register renaming vs. reorder buffers
  - Instead of virtual registers from reservation stations and reorder buffer, create a single register pool
    - Contains visible registers and virtual registers
  - Use hardware-based map to rename registers during issue
  - WAW and WAR hazards are avoided
  - Speculation recovery occurs by copying during commit
  - Still need a ROB-like queue to update table in order
  - Simplifies commit:
    - Record that mapping between architectural register and physical register is no longer speculative
    - Free up physical register used to hold older value
    - In other words: SWAP physical registers on commit
  - Physical register de-allocation is more difficult

Copyright © 2012, Elsevier Inc. All rights reserved.

Integrated Issue and Renaming

- Combining instruction issue with register renaming:
  - Issue logic pre-reserves enough physical registers for the bundle (fixed number?)
  - Issue logic finds dependencies within bundle, maps registers as necessary
  - Issue logic finds dependencies between current bundle and already in-flight bundles, maps registers as necessary

MORGAN KAUFMANN

Copyright © 2012, Elsevier Inc. All rights reserved.

How Much?

- How much to speculate
  - Mis-speculation degrades performance and power relative to no speculation
    - May cause additional misses (cache, TLB)
  - Prevent speculative code from causing higher costing misses (e.g. L2)
- Speculating through multiple branches
  - Complicates speculation recovery
  - No processor can resolve multiple branches per cycle

MK MORGAN KAUFMANN

Copyright © 2012, Elsevier Inc. All rights reserved.

Energy Efficiency

- Speculation and energy efficiency
  - Note: speculation is only energy efficient when it significantly improves performance
- Value prediction
  - Uses:
    - Loads that load from a constant pool
    - Instruction that produces a value from a small set of values
  - Not been incorporated into modern processors
  - Similar idea--*address aliasing prediction*--is used on some processors

Copyright © 2012, Elsevier Inc. All rights reserved.