Credits: Yousif (Intel), Kubi@UCB, Asanovic @ MIT, http://csep1.phy.ornl.gov

Advanced Instruction Re-Ordering: Re-Order Buffer

Handout 13

November 2, 2004

Shoukat Ali

shoukat@umr.edu

UMR UNIVERSITY OF MISSOURI-ROLLA
The Name. The Degree. The Difference.

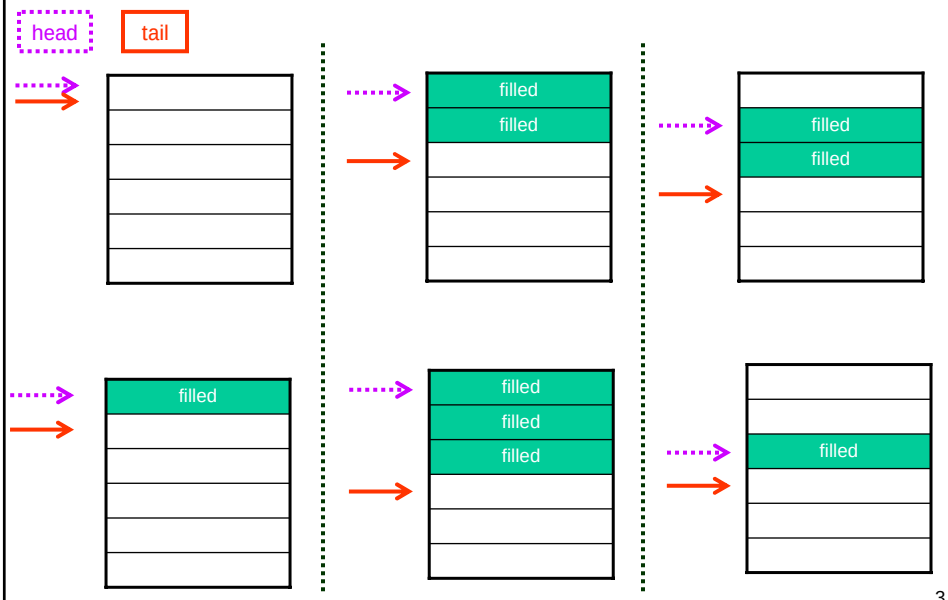
1

Digression: A Refresher On Circular Buffers

- "An area of memory used to store a continuous stream of data by starting again at the beginning of the buffer after reaching the end." *"FOLDOC" – Free Online Dictionary of Computing*
- in our case, issue stage will write to the buffer and a new "commit" stage will read from the buffer
- two addresses (pointers) are maintained
  - tail pointer points towards the next empty entry
  - head pointer points towards the "first in" element

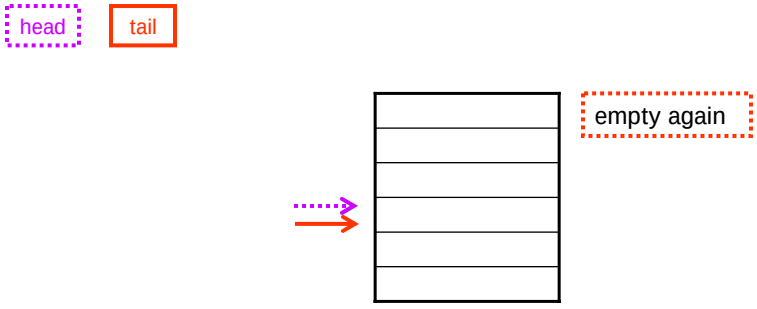
2

ROB is a FIFO circular buffer. Tail ptr points towards next empty entry. Head ptr points towards the FI element. Initially tail = head, i.e., buffer is empty



3

ROB is a FIFO circular buffer. Tail ptr points towards next empty entry. Head ptr points towards the FI element. Initially tail = head, i.e., buffer is empty



Digression Complete

4

Modifying Tom's Alg For Precise Interrupts

- the modified algorithm has an additional stage, called commit
- stages are:
  - issue
  - read pending operand
  - execute
  - write result
  - commit, i.e., instruction's result in written into its destination (register or memory)
- or, we can merge RPO into execute (the way book does)
  - issue
  - execute
  - write result
  - commit

5

Changes in HW to Incorporate ROB

- each reservation station will have one new field "Dest"

RS	RS Tag	Busy	Op	Vj	Vk	Qj	Qk
Add1	1	Yes	sub	84	42	0	0

RS	Busy	Op	Vj	Vk	Qj	Qk	Dest
Add1	Yes	sub	84	42	0	0	#3

- Dest specifies the rob entry number of the instruction being housed in the RS
- symbols Qj, Qk will refer to rob entry number and NOT reservation station numbers

6

ROB Fields

head tail

entry #	busy	instruction	ready	destin-ation	value	exce-ption	PC
1	yes	MULTD	no	F0		no	1032
2	yes	SUBD	yes	F8	42	no	1036
3	yes	DIVD	no	F10		no	1040
4	no						

busy	ready	meaning
no	yes/no	vacant entry
yes	no	instruction issued but not yet completed the WR stage
yes	yes	instruction issued and completed the WR stage but not yet committed

"completed" instruction

7

Issue Stage for Modified Tomasulo's Alg

- is there a free RS and a free ROB entry (struct hazard check)
- if no, stall in issue stage
- if yes
  - reserve the ROB entry at tail
  - reserve the RS
  - set Dest field of RS to ROB entry number
    - i.e., RS will write to ROB
  - set Dest field of ROB to destination register number
    - i.e., ROB will write to register file
  - store PC
  - read data operands if available (check both reg file and ROB)
  - for a pending operand, store tag into Qj, Qk
    - tag now is the ROB entry # of producer instruction, and not the number of the producer RS

8

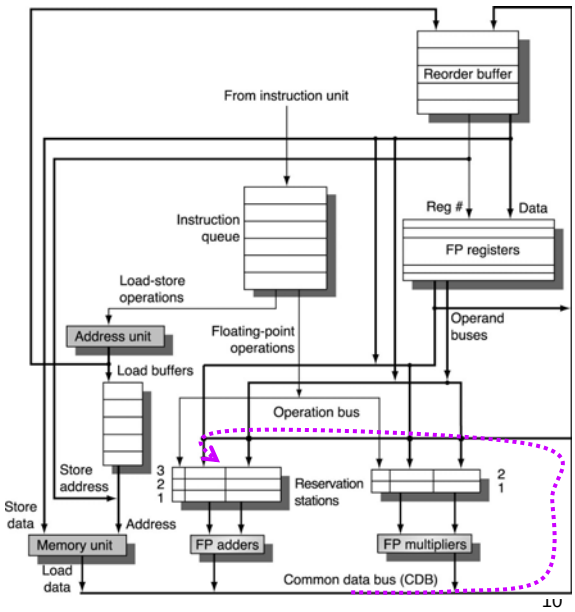
Why Check Both Register File And ROB?

- recall: re-order buffer holds the result of an instruction between the time the operation associated with an instruction completes and the time instruction commits
- consider this scenario
  - producer instruction completed but not committed
    - result is in ROB but not in register file
  - any dependent instruction must read result from ROB until the producer commits

9

Execute Stage of Modified Tomasulo's Alg

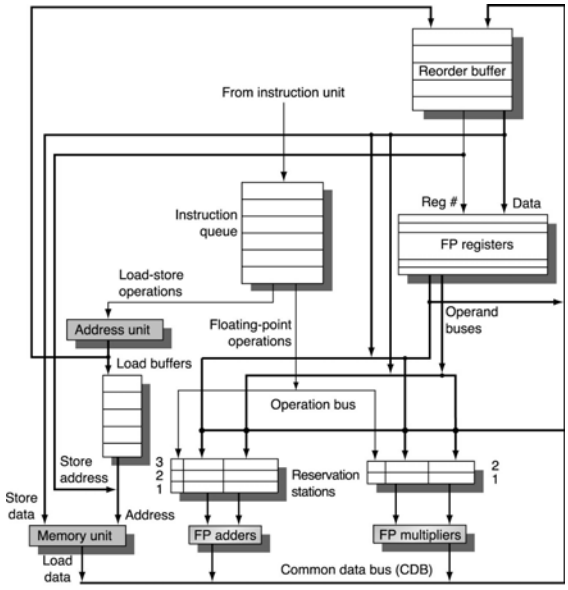
- wait until pending operands become available on the CDB (match CDB tag against operand tags)
- grab pending operands from CDB and issue to FU (if free)
- execute



10

Write Result Stage of Modified Tomasulo's Alg

- broadcast result + tag (ROB entry # of the producer instruction) on the CDB
- all reservation stations listen in, and compare the advertised ROB entry # with the one they want
  - pick up value if there is a match
- *ROB* also listens in, and writes the result in its "Value" field



Commit Stage of Modified Tomasulo's Alg - 1

- two different sequences
- examine the instruction at the head of ROB
- if "ready" field is set to "yes" (i.e., if result been written to ROB)
  - if "exception" field is set to "no" (i.e., instr did not raise exception)
    - copy the "value" field into:
      - the register (for loads and ALU instructions) if this instruction is supposed to write the register
      - memory (for stores)
    - remove the instr from ROB by setting busy to "no"

entry #	*busy*	*instruction*	*ready*	*destin-ation*	*value*	*exce-ption*	*PC*
1	yes	MULTD	yes	F0	30	no	1032
2	yes	SUBD	yes	F8	42	no	1036
3	yes	DIVD	no	F10		no	1040
4	no						

12

Commit Stage of Modified Tomasulo's Alg - 2

- two different sequences
- examine the instruction at the head of ROB
- if "ready" field is set to "yes" (i.e., if result been written to ROB)
  - ***if "exception" field is set to "yes"***
    - copy PC of offending instr somewhere
    - flush entire ROB
    - service the interrupt
    - restart execution by loading PC of the offending instruction

entry #	*busy*	*instruction*	*ready*	*destin-ation*	*value*	*exce-ption*	*PC*
1	yes	MULTD	yes	F0	30	yes	1032
2	yes	SUBD	yes	F8	42	no	1036
3	yes	DIVD	no	F10		no	1040
4	no						

13

finishing times

Instruction	j	k
LD F6	34	R2
MULTD F0	F2	F6
SUBD F8	F6	F2
ADDD F10	F8	F4

IS	RPO	EXE	WR	COM

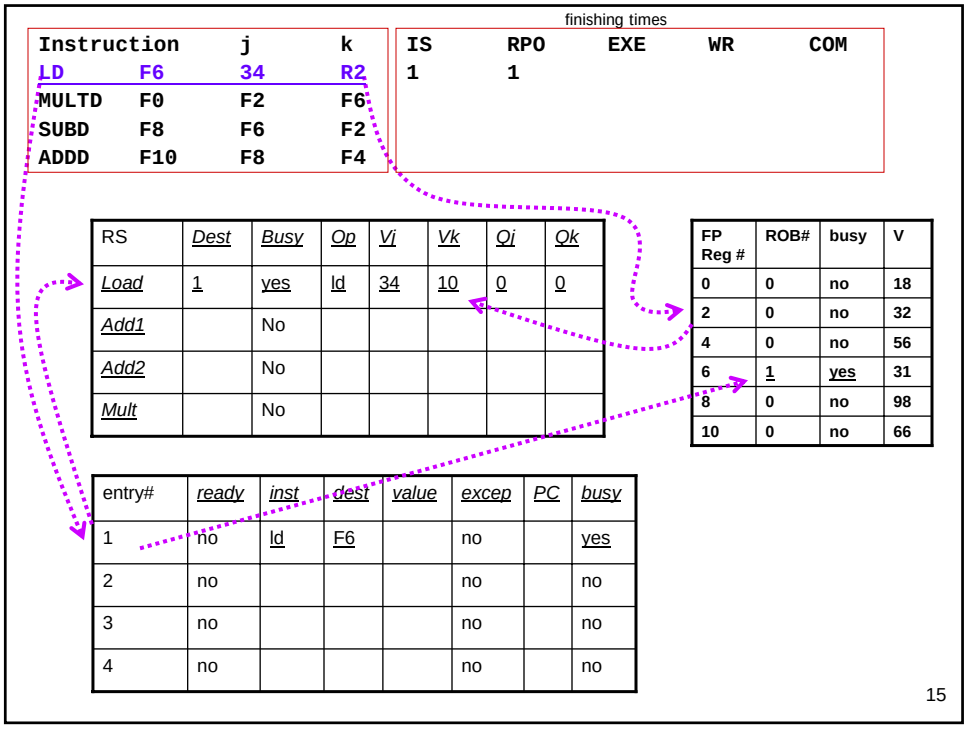
RS	*Dest*	*Busy*	*Op*	*Vj*	*Vk*	*Qj*	*Qk*
Load		No					
Add1		No					
Add2		No					
Mult		No					

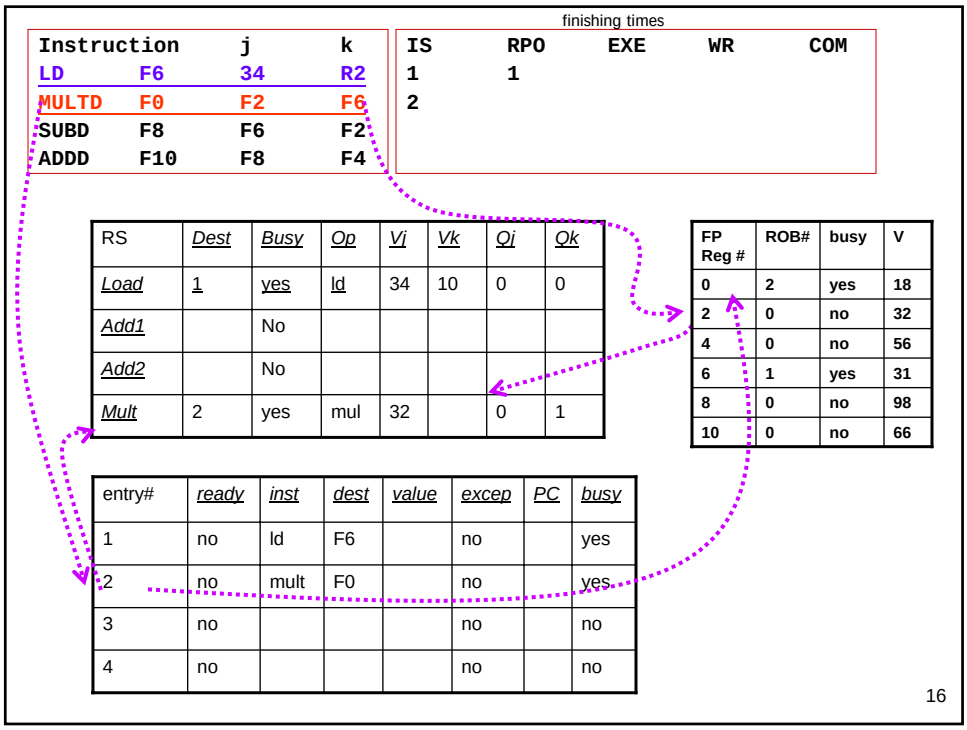
FP Reg #	ROB#	busy	V
0	0	no	18
2	0	no	32
4	0	no	56
6	0	no	31
8	0	no	98
10	0	no	66

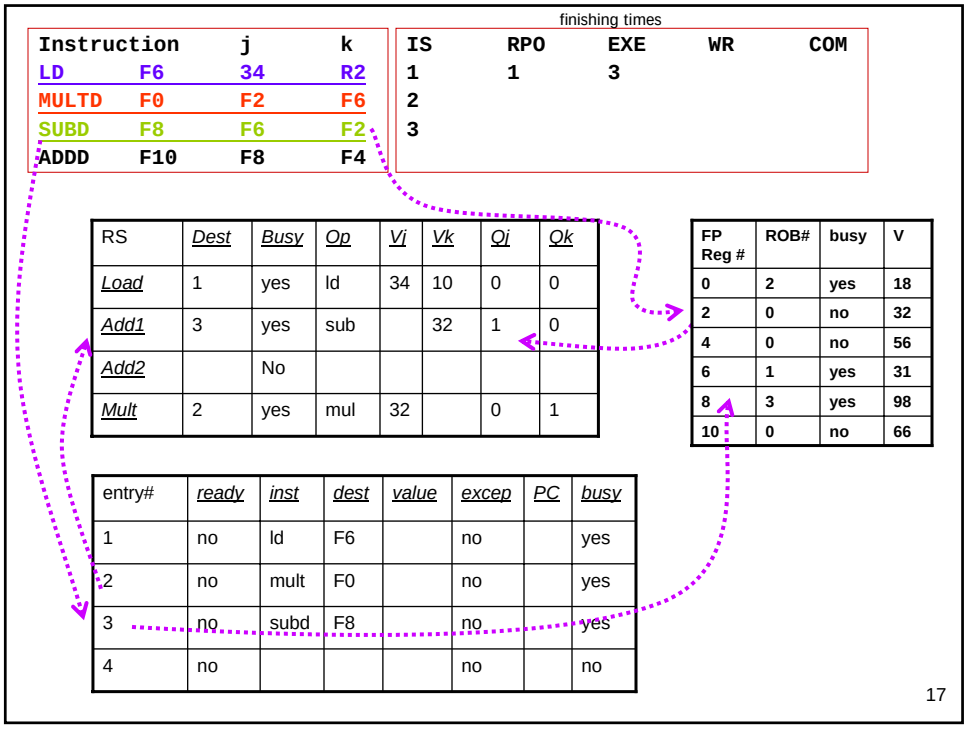
entry#	*ready*	*oper*	*dest*	*value*	*excep*	*PC*	*busy*
1	no				no		no
2	no				no		no
3	no				no		no
4	no				no		no

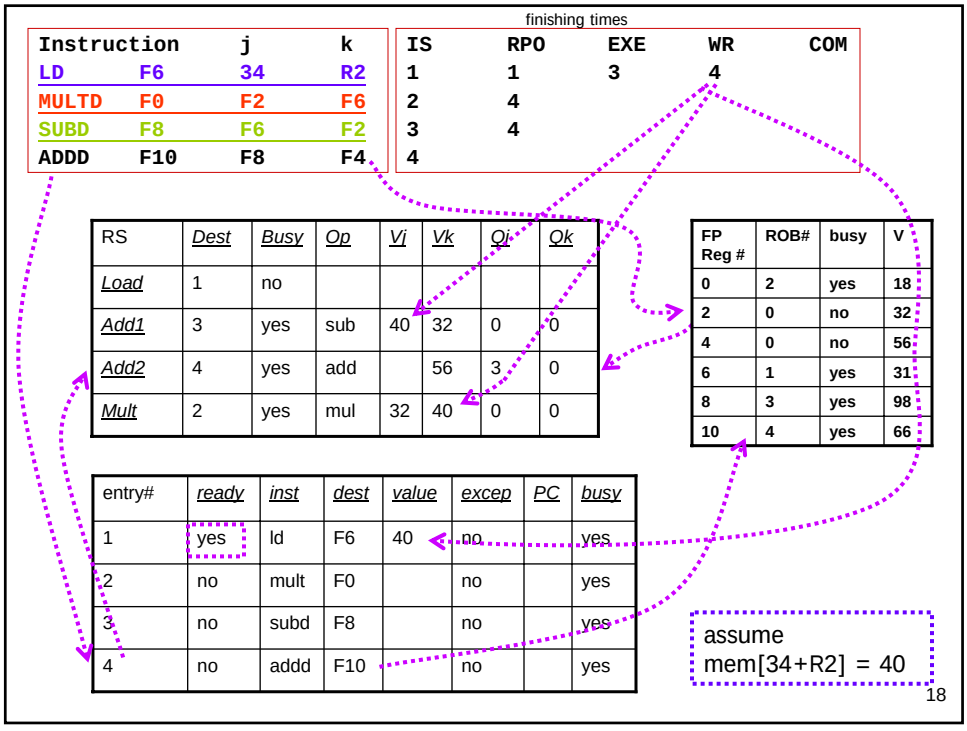
assume R2 = 10

14


finishing times
Instruction j k
LD F6 34 R2
MULTD F0 F2 F6
SUBD F8 F6 F2
ADDD F10 F8 F4
IS RPO EXE WR COM
1 1
RS Dest Busy Op Vj Vk Qj Qk
Load 1 yes ld 34 10 0 0
Add1 No
Add2 No
Mult No
FP Reg # ROB# busy V
0 0 no 18
2 0 no 32
4 0 no 56
6 1 yes 31
8 0 no 98
10 0 no 66
entry# ready inst dest value excep PC busy
1 no ld F6 no yes
2 no no
3 no no
4 no no
15


finishing times
Instruction j k
LD F6 34 R2
MULTD F0 F2 F6
SUBD F8 F6 F2
ADDD F10 F8 F4
IS RPO EXE WR COM
1 1
2
RS Dest Busy Op Vj Vk Qj Qk
Load 1 yes ld 34 10 0 0
Add1 No
Add2 No
Mult 2 yes mul 32 0 1
FP Reg # ROB# busy V
0 2 yes 18
2 0 no 32
4 0 no 56
6 1 yes 31
8 0 no 98
10 0 no 66
entry# ready inst dest value excep PC busy
1 no ld F6 no yes
2 no mult F0 no yes
3 no no
4 no no
16


finishing times
Instruction j k
LD F6 34 R2
MULTD F0 F2 F6
SUBD F8 F6 F2
ADDD F10 F8 F4
IS RPO EXE WR COM
1 1 3
2
3
RS Dest Busy Op Vj Vk Qj Qk
Load 1 yes ld 34 10 0 0
Add1 3 yes sub 32 1 0
Add2 No
Mult 2 yes mul 32 0 1
FP Reg # ROB# busy V
0 2 yes 18
2 0 no 32
4 0 no 56
6 1 yes 31
8 3 yes 98
10 0 no 66
entry# ready inst dest value excep PC busy
1 no ld F6 no yes
2 no mult F0 no yes
3 no subd F8 no yes
4 no no
17


finishing times
Instruction j k
LD F6 34 R2
MULTD F0 F2 F6
SUBD F8 F6 F2
ADDD F10 F8 F4
IS RPO EXE WR COM
1 1 3 4
2 4
3 4
4
RS Dest Busy Op Vj Vk Qj Qk
Load 1 no
Add1 3 yes sub 40 32 0 0
Add2 4 yes add 56 3 0
Mult 2 yes mul 32 40 0 0
FP Reg # ROB# busy V
0 2 yes 18
2 0 no 32
4 0 no 56
6 1 yes 31
8 3 yes 98
10 4 yes 66
entry# ready inst dest value excep PC busy
1 yes ld F6 40 no yes
2 no mult F0 no yes
3 no subd F8 no yes
4 no addd F10 no yes
assume
mem[34+R2] = 40
18

finishing times

Instruction	j	k
LD	F6	34 R2
MULTD	F0	F2 F6
SUBD	F8	F6 F2
ADDD	F10	F8 F4

IS	RPO	EXE	WR	COM
1	1	3	4	5
2	4			
3	4			
4				

if instr has a register dest

```
h = rob head
d = Rob[h].dest
if ROB[h].excep = "yes"
   copy PC, flush ROB, service interrupt, restart from PC
if ROB[h].excep = "no"  and Reg[d].ROB# = h
   Reg[d].V = ROB[h].value
   Reg[d].busy = no; Reg[d].ROB# = 0;
   ROB[h].busy = no;
}
```

FP Reg #	ROB#	busy	V
0	2	yes	18
2	0	no	32
4	0	no	56
6	0	no	40
8	3	yes	98
10	4	yes	66

entry#	ready	inst	dest	value	excep	PC	busy
1	yes	ld	F6	40	no		no
2	no	mult	F0		no		yes
3	no	subd	F8		no		yes
4	no	addd	F10		no		yes

19

Commit Stage For a Store Instruction

```
h = rob head
d = Rob[h].dest = some address

if ROB[h].excep = "yes" {
copy PC, flush ROB, service interrupt, restart from PC
}

if ROB[h].excep = "no"  {
   Mem[d] = ROB[h].value
   Reg[d].busy = no; Reg[d].ROB# = 0;
   ROB[h].busy = no
}
```

20

Hardware Speculation

The Grandest Concept of Them All

21

ROB Allows Speculative Execution

- result from a completed but not committed instruction X is used by later consumer instructions speculatively
- if X raises an exception, the entire ROB is flushed
  - i.e., results of all later instructions (possibly speculatively executed) are discarded
- the process of using an instruction's result before knowing if it is valid is called "speculative execution"

22

Other Forms of Speculation - 1

I realize we have not done branch prediction schemes yet. Will cover soon.

- branch outcome speculation:
  - assume the HW predicted that a particular branch will be taken
  - based on the prediction, HW executed a few instructions after the branch
    - that is, some speculative execution was done
  - then HW finds out that the branch was not taken!!!
  - *open question to class: How can we use re-order buffer to ensure that incorrect code execution will not be done in the above case?*
    - you do not need to know the methods for predicting branches to answer this question

23

Commit Stage For Predicted Branches

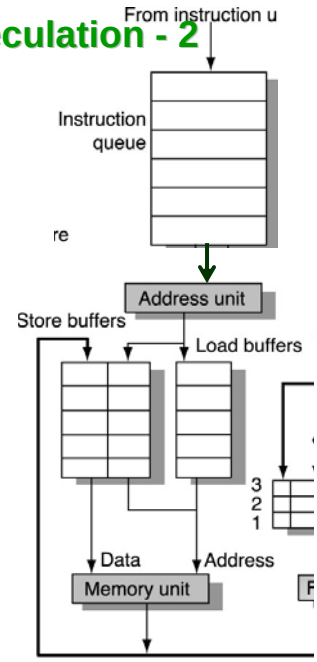
- add a "branch prediction" field to the ROB entry
- set to "nok" when it is known that branch was incorrectly predicted
- set to "ok" when it is known that branch was correctly predicted

- assume that a branch instruction is at the head of ROB
- examine the ***branch*** instruction
  - if the "branch prediction" is set to "ok"
    - commit branch, i.e., "accept" the results of the instructions speculatively executed on the predicted path
  - if the "branch prediction" is set to "nok"
    - flush entire ROB
    - restart execution by loading _____

24

Other Forms of Speculation - 2

- SW-LW dependency speculation



HW Detection of Store-to-Load Hazard

- before sending a load instruction to its reservation station (called a load buffer), do this:
  1. calculate the memory address of the load
  2. compare it with the memory address of all store instructions currently in the store buffers (reservation stations for store instructions)
  3. if there is an address match, do not send the load instruction to load buffer
- above three steps performed by the address unit
- essentially, a load instruction waits in the instruction queue until all stores issued before have completed

25

SW-LW Dependency Speculation

- as soon as the load's effective address is calculated, the load is sent to memory
  - without waiting for earlier issued stores to complete
- *open question to class: How can we use re-order buffer to ensure that incorrect code execution will not be done in the above case?*

26