Lecture 4

Speculative Instruction Execution

- **Dynamically scheduled pipelines (Ch 3.4)**

→ ✓Two-level dynamic branch prediction (Ch 3.4.3)

→ **Putting it Together:**

→ ✓ Tomasulo + Branch prediction + Speculation (Ch 3.4.4)

→ ✓ Dynamic memory disambiguation (Ch 3.4.5)

→ ✓ Register renaming techniques (Ch 3.4.6)

Two-Level Branch Prediction (Ch 3.4.3)

CHALMERS
Chalmers University of Technology

Correlating Branch Predictors 1(2)

To improve on two-bit predictors, we need to look at other branches than branches in loops.

Consider the following code snippet:

```
if (a==2) then a:=0;
if (b==2) then b:=0;
if (a!=b) then ---
```

- **If the first two conditions succeed, the 3rd will fail**
- **Thus: The 3rd branch is correlated with the first two**
- **Previous predictors track the history of each branch only**

Correlating Branch Predictors 2(2)

In general, a branch may behave differently if it is reached through different code sequences

- **The code sequence can be characterized by the outcome of the latest branches to execute**

We can use N bits of prediction and the outcomes of the last M branches to execute

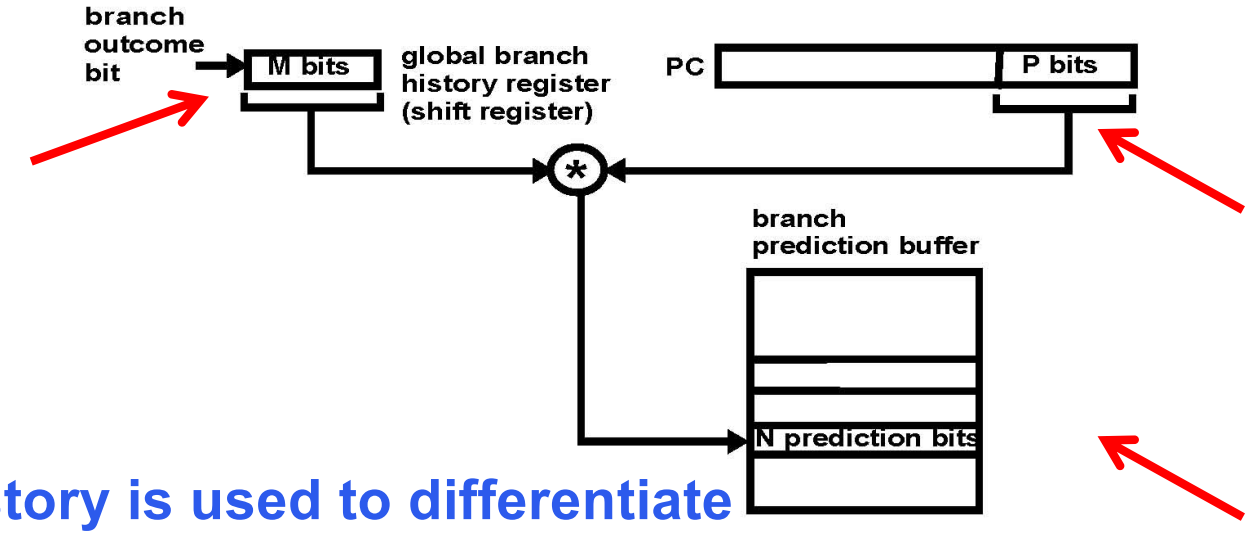
- **Global vs. local history**
- **Note that the branch itself may be part of the global history**

(M,N) BPB

Outcome of the last M branches is the global branch history

Use N bit predictor:

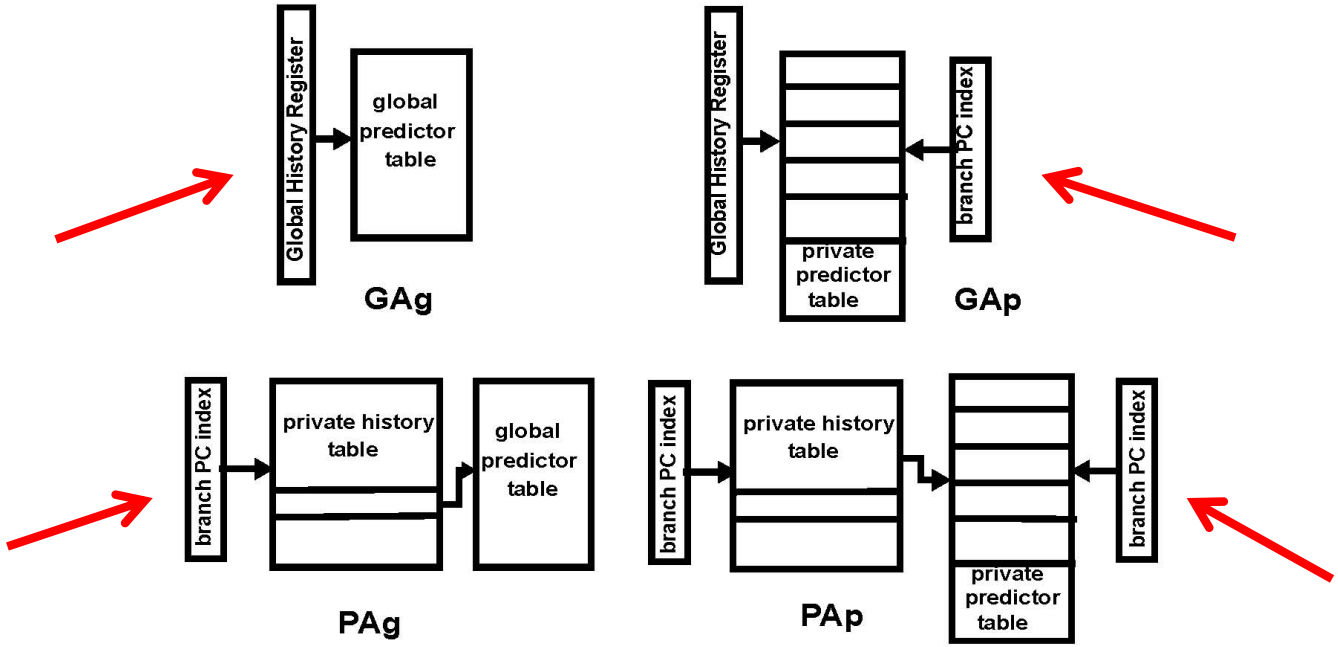
- **BPB is indexed with P bits of the branch program counter (PC) and M bits from branch history register**
- **BPB size: N x 2^M x 2^P**



Global history is used to differentiate between various behaviors of a particular branch

This can be generalized: Two level predictors

Two-level Branch Predictors



In a two-level predictor we add history bits to access the BTB

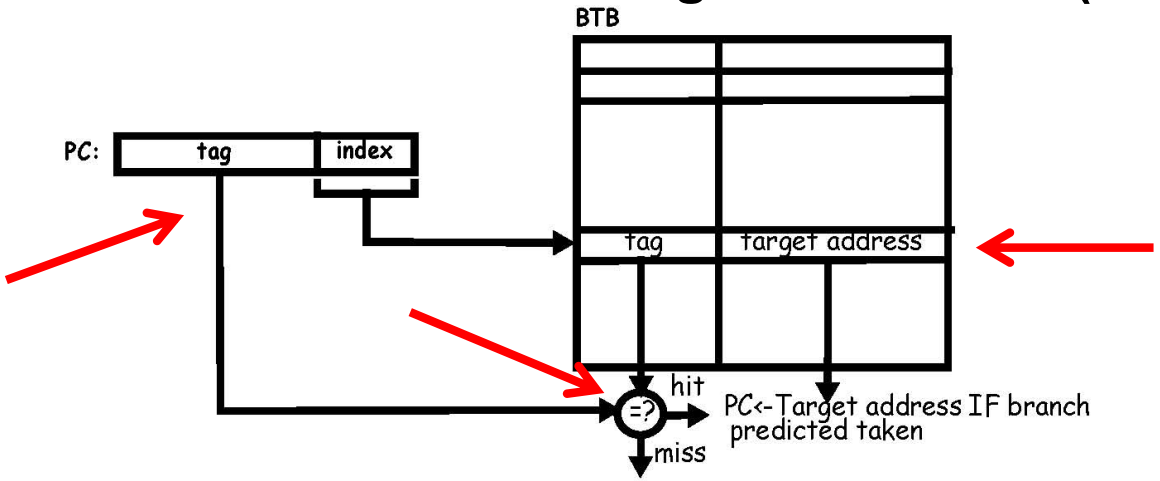
- **History can be global (all branches) or private (this branch only)**
- **Notation: The first (capital) letter refers to the type of history. The last letter refers to whether each predictor is private in the table**
- **G or g means "Global"; P or p means "Private"**

Branch Target Buffer (BTB)

Aim: To eliminate the branch penalty

- **Need to know the target address by the end of I-fetch**

The BTB: Cache for all branch target addresses (no aliasing)



- **Accessed in I-Fetch in parallel with instructions and BPB entry**
- **Relies on the fact that the target address of a branch never changes: predicting indirect jumps**
- **Procedure return is a major cause of indirect jump**
- **Use a stack to track the stack of procedure calls**

Quiz 4.1

Consider the following program:

```
for (i=0;i<100;i++)
   if (i%2==0)
        then a=1
        else a=0;
```

Consider a 6-bit branch history register for the high-lighted branch.

Which of the following statements are correct?

a) 010101

b) 110011

c) 101010

d) 001100

Hardware Supported Speculation (Ch 3.4.4)

CHALMERS
Chalmers University of Technology

Hardware Supported Speculation

Combination of three main ideas:

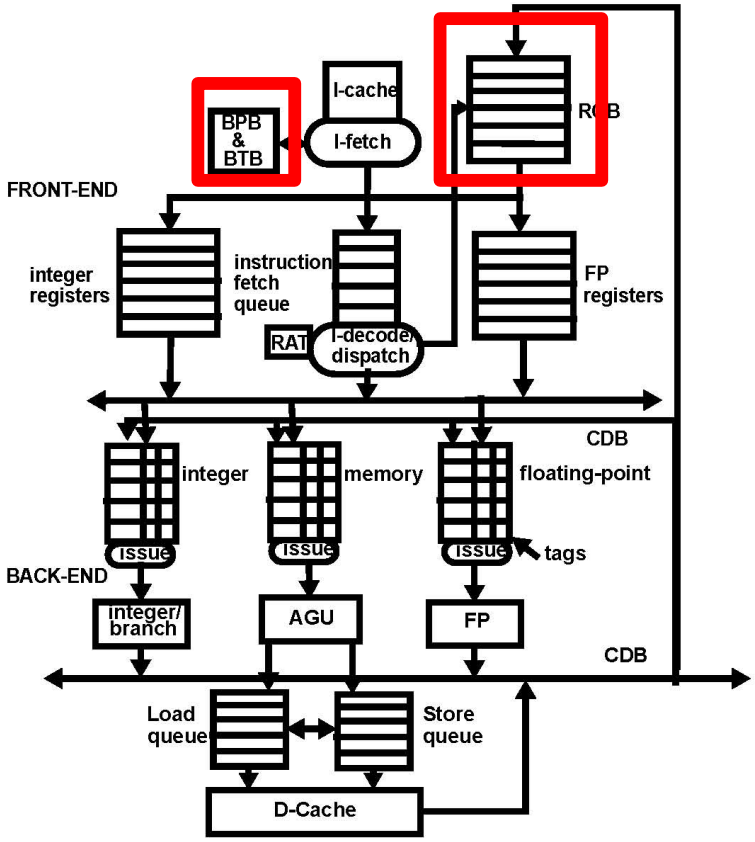
- **Dynamic OoO instruction scheduling (Tomasulo)**
- **Dynamic branch prediction, allowing scheduling across branches**
- **Speculative execution: all control dependencies are resolved**

Hardware-based speculation uses a data-flow approach: instructions execute when their operands are available, across predicted branches

Key ideas:

- **Separate the completion of instruction execution and the commit of its result**
- **Between completion and commit, results are speculative**
- **Commit the results to registers and storage in process/program order**

Tomasulo with Speculative Execution



New structures:

- Reorder buffer (ROB)
- Branch prediction buffer (BPB)
- Branch target buffer (BTB)

ROB:

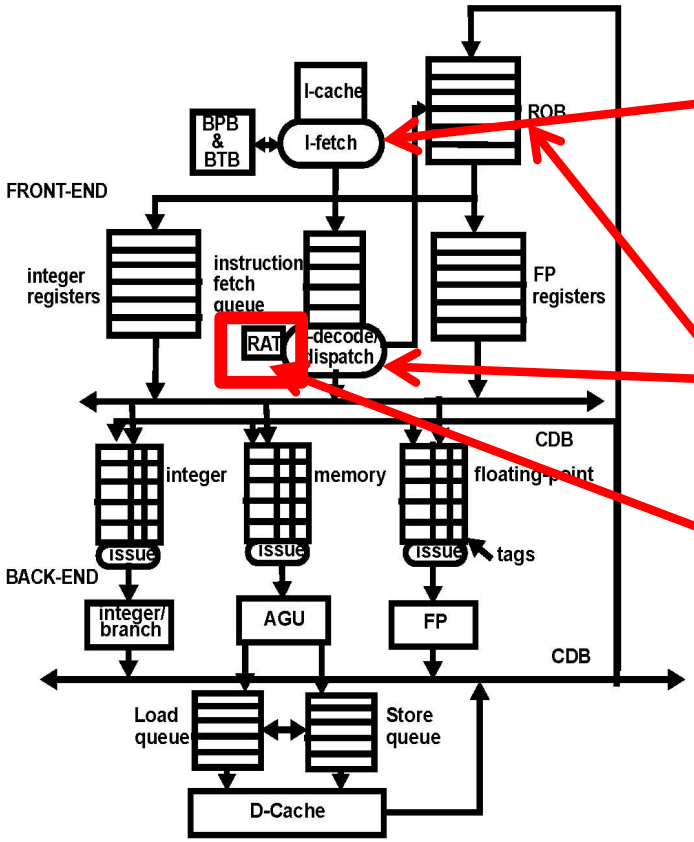
- Keeps track of process order (FIFO)
- Holds speculative results
- No more snooping by registers

Registers:

- Pending in back-end
- Speculative in ROB
- Committed in the register file

Use ROB entry ID as TAG for renaming

Speculative Tomasulo Algorithm 1(2)



Normal operation (no mispredicted branch, no exception)

1. I-FETCH

- Fetch instruction
- Predict branches and their target
- Fill the instruction fetch Q (IFQ) following the branch prediction

2. I-DECODE/DISPATCH

- Decode opcode
- Allocate 1 issue Q entry + 1 ROB entry + 1 L/S Q entry for Load/Store
- Rename destination register (TAG) with ROB entry ID, mark "pending in Register Alias Table (RAT)
- Fill input register operand field
  - If value is marked "pending" in RAT fill operand field with TAG (not ready)
  - If marked "completed" in RAT fetch value from ROB (ready)
  - If marked "committed" in RAT, fetch value from register file (ready)

Speculative Tomasulo Algorithm 2(2)

3. ISSUE

- Wait in issue Q until all inputs are ready (snoop the CDB)
- Issue in a cycle when conflicts for FU and CDB can be avoided

4. COMPLETE EXECUTION AND WRITE RESULTS

- Result is written to the ROB via the CDB
- Destination register is marked “Completed” in the RAT

5. COMMIT (OR GRADUATE OR RETIRE)

- Wait to reach the head of the ROB
- Write result to register or to memory (store)

Branch misprediction

- All instructions following the branch in the ROB must be flushed
- Wait until the branch reaches the top of the ROB and flush the ROB AND flush all instructions in the back-end
- Instructions at the correct target are fetched.

Exceptions: Similar to branch misprediction

Quiz 4.2

Which of the following statements are correct?

a) The ROB keeps instructions in the order they are executed
b) The ROB keeps instructions in the order they are issued for execution
c) The ROB keeps instructions in the order they read their operands
d) The ROB keeps instructions in the order they are dispatched

Memory Disambiguation (Ch 3.4.5)

CHALMERS
Chalmers University of Technology

Solving Memory Hazards

For Loads/Stores we use the same approach as in Tomasulo:

- Load/store instructions issue to L/S Q through the AGU
- Stores are split into 2 instructions: address + cache access
- Each sub-instruction is allocated 1 issue Q entry

All WAW and WAR hazards are automatically solved

- Because stores update the cache in process order when they reach the top of the reorder buffer

Check for RAW hazards in the L/S Q before sending the Load to cache

- Load can issue to cache as soon as it reaches the L/S Q
- However, if a store with the same address is in front of the load in the L/S Q then
  - Wait until store reaches the cache (at the top of the ROB), or
  - Return the value when it is ready

What to do when there are stores in front whose address is unknown

Dynamic Memory Disambiguation 1(2)

Figuring out if two addresses are equal to avoid hazards

- Conservative approach:
  - A ready Load must wait in the L/S Q until addresses of all Stores preceding it in the L/S Q are known
  - Problem: The situation where a Load depends on a Store in the L/S Q is quite rare
- Optimistic approach: Speculative disambiguation
  - Use the mechanisms in place for speculative execution (ROB + roll-back)

Dynamic Memory Disambiguation 2(2)

- Speculative disambiguation (cont'd)
  - If a Load is ready and Stores with unknown addresses are in front of it in the L/S Q
    - Speculatively assume that their addresses are different from the load's address
    - Issue load to cache
    - Later, when the store's address has been computed and is ready, check all following Loads in L/S Q
    - If a Load has the same address and is completed, the load and all the following instructions must be replayed (as a mispredicted branch)
- Intermediate:
  - Keep track that a load has violated in the past
  - Treat that load conservatively; treat all other loads speculatively

Example

		Dispatch	Issue	Exec start	Exec complete	Cache	CDB	Retire	COMMENTS
I1	L.S F0,0(R1)	1	2	(3)	3	(4)	(5)	6	
I2	L.S F1,0(R2)	2	3	(4)	4	(5)	(6)	7	
I3	ADD.S F2,F1,F0	3	7	(8)	12	--	(13)	14	wait for F1
I4	S.S-A F2,0(R1)	4	5	(6)	6	--	--	--	
I5	S.S-D F2,0(R1)	5	14	(15)	(15)	(16)	--	17	wait for F2
I6	ADDI R1,R1,#4	6	7	(8)	8	--	(9)	18	
I7	ADDI R2,R2,#4	7	8	(9)	9	--	(10)	19	
I8	SUBI R3,R3,#1	8	9	(10)	10	--	(11)	20	
I9	BNEZ R3,Loop	9	12	(13)	13	--	(14)	21	wait for R3
I10	L.S F0,0(R1)	10	12	(13)	13	(14)	(15)	22	CDB conflict with I9
I11	L.S F1,0(R2)	11	13	(14)	14	(15)	(16)	23	issue conflict with I10
I12	ADD.S F2,F1,F0	12	17	(18)	22	--	(23	24	wait for F1

New column: Retire

- **Load can be dispatched right after the Branch**
- **Store to cache must wait until it reaches the top of the ROB**

Quiz 3.3

Consider the following program:

I1: SW R1,0(R2)

I2: LW R4,0(R3)

I3: LW R6,0(R5)

Which of the following statements are correct

a)I1, I2 and I3 can execute in any order correctly

b)I2 can return value from store queue. I2 can read from cache before I1 when I1's address is known

c)I3 can read from cache before I1 and I2 execute if it reads from a different address than I1 and if I2 reads from same address that I1 stores

Register Renaming
(Ch 3.4.6)

CHALMERS
Chalmers University of Technology

Explicit Register Renaming

- ROB orders instruction commits and provides storage for speculative register values until they commit

- Speculative register values may also be kept in physical registers
  - Then the role of the ROB is limited to ordering instruction commits
  - Physical registers are “shadow” registers keeping non-committed results
- Large number of physical registers (more than # of architectural registers – the registers visible from the ISA)

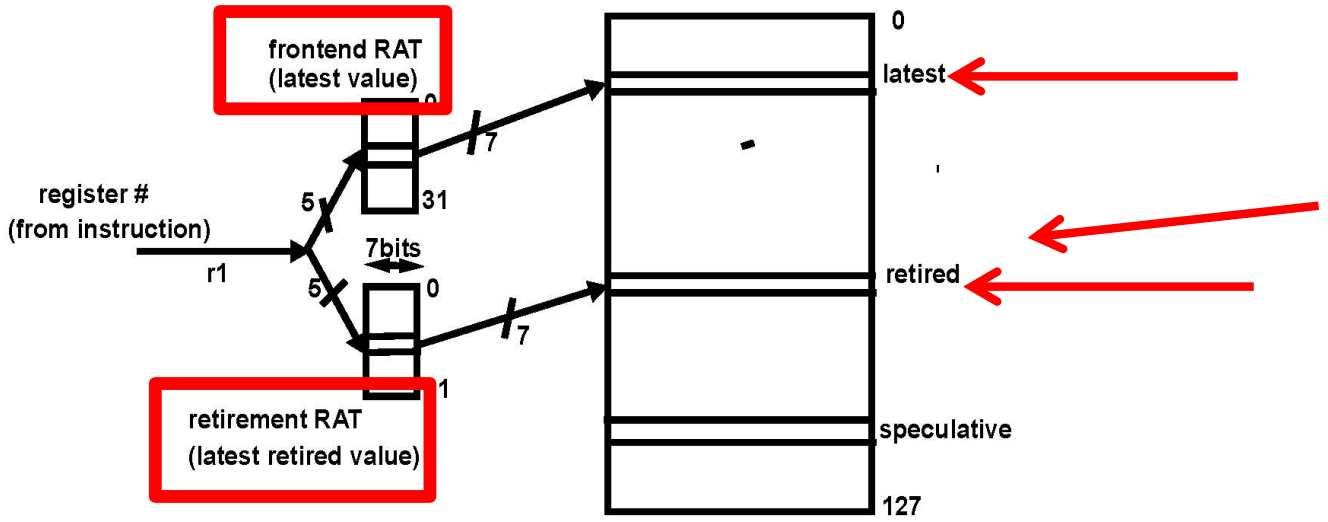
Physical vs architectural registers

Architectural registers are mapped to physical registers dynamically

- At any one time one architectural register may map to multiple physical registers
- Architectural registers are renamed to physical at dispatch
- Whenever a new value is stored in a register, a new physical register is allocated and the mapping is changed to point to this latest value
- One physical register must hold the latest committed (retired) value of each architectural register
- Physical registers must be reclaimed
- ROB entry carries the mapping of architectural to physical number

Can't dispatch if all physical registers are allocated

Register Renaming: Structures



- Frontend RAT points to most recent value. Retirement RAT points to most recent retired value (could be the same)
- More speculative values of the same architectural register may be in the register file
- "Updating register" now means "Updating the pointer in the retirement RAT"

Register Renaming: Algorithm

When an instruction is dispatched its operands are renamed

- A physical register is allocated to the destination register
- The frontend RAT is updated
- The physical register number mapped by the frontend is now the TAG used in Tomasulo algorithm (not the ROB entry #)
- Input register operands are mapped to their physical register through the frontend RAT
- If value is ready it is dispatched to the issue Q. Otherwise the physical register number is dispatched (Not Ready)
- When an instruction retires, its destination physical register has the retired value and the retirement RAT is updated
  - Previous physical register mapped by the retirement RAT is reclaimed

Quiz 3.4

Consider the following program

I1: ADD R1,R2,R3 (STATUS: COMMITTED)

I2: ADD R1,R4,R5 (STATUS: RETIRED)

I3: ADD R1,R6,R7 (STATUS: UNDER EXECUTION)

Which of the following statements are incorrect?

a)I3's R1 value is the latest value

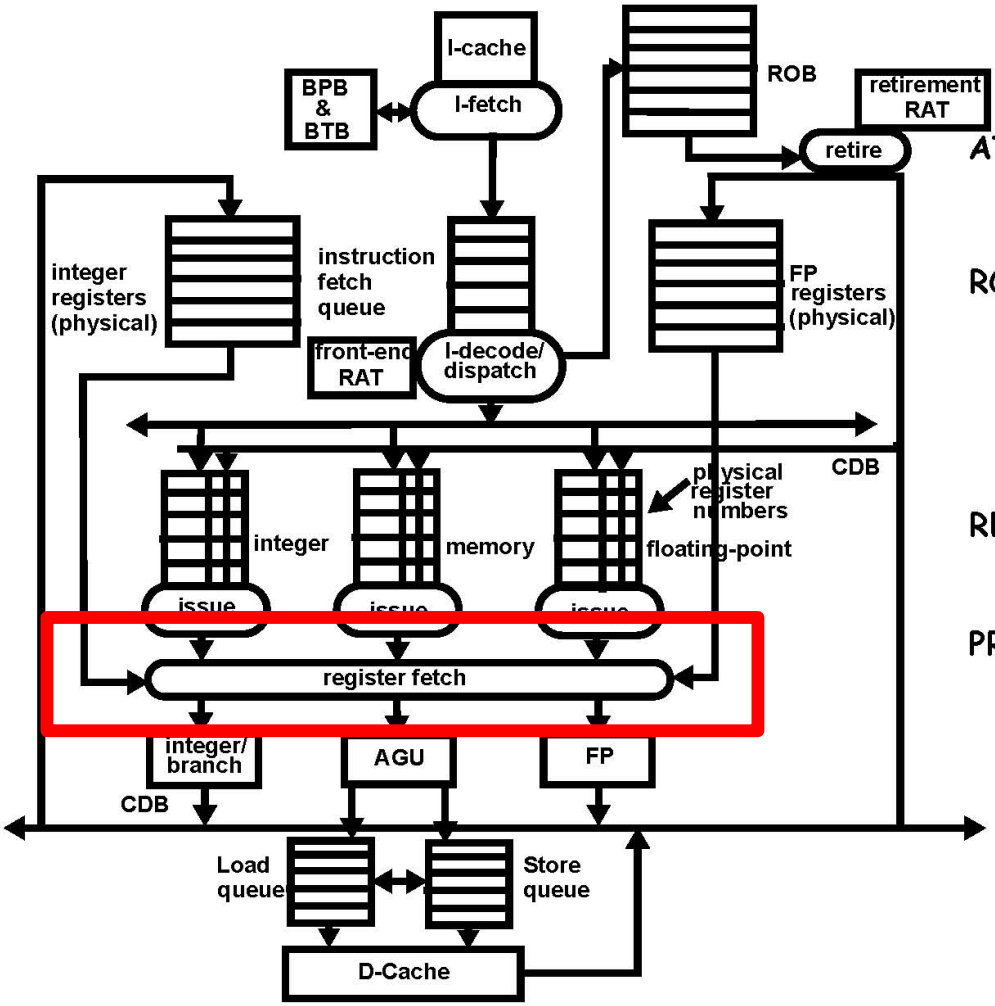
b)I1's R1 value is the latest value

c)I2's R1 value is the latest value

d)The register file contains the R1 value produced by I2

e)The registerfile contains the R1 value produced by I1

Register Fetch After Issue



AT DISPATCH INPUT REGISTER # (NOT ITS VALUE) IS DISPATCHED WITH READY BIT

ROLE OF THE CDB:

- TRANSFER VALUES TO REGISTER
- AWAKE INSTRUCTIONS BY SETTING READY BITS IN ISSUE Qs

REGISTERS ARE FETCHED RIGHT BEFORE EXECUTION

PROBLEMS:

- EFFECTIVE LATENCY MUCH HIGHER
- SIMILAR PROBLEM IN TOMASULO

What you should know by now

- **Dynamic branch prediction**
  - Simple 1- and 2-bit predictors
  - Correlated branch prediction
  - Two level branch prediction
    - Local/global branch history and predictors
- **Speculative execution**
  - Combination of Tomasulo + branch prediction + speculation
  - Structures needed for speculation
  - Speculative Tomasulo algorithm
  - Techniques for dynamic memory disambiguation
  - Register renaming techniques