Computer Microarchitecture Practice Questions
COMP2300/ENGN2219/COMP6300

Total Points: 100

Important Instructions: (1) Write down the names and UIDs of each student in a group (if applicable) on the first page of your submission. (2) Submit the solution as a single pdf file.

Instructions for Q12: You can fill out the PowerPoint slide deck and convert it to a pdf document. You can then combine the pdf document with a second pdf file with responses to all other questions. Alternatively, you can copy and paste the main structures from the PowerPoint slide deck to your word document. And then submit a single pdf file. You can also print out the slide, fill the contents by hand, scan the document, and convert it to pdf.

Note about MIPS ISA: The practice questions in this homework assume MIPS ISA. Please use your favorite search engines to learn about the differences between ARM and MIPS ISA. It will not take more than 10 minutes to understand MIPS if you know ARM. Once you understand one ISA, learning about another ISA should be straightforward.

Perhaps all you need to know is that MIPS base plus offset addressing specifies the base register inside parentheses and offset outside the parentheses.

(2 Points)

Q1. Compilers impact the performance of applications in different ways. For a program, compiler **X** results in a dynamic instruction count of 1 billion instructions, and an execution time of one second. A second compiler **Y** results in an execution time of 1.5 seconds, and a dynamic instruction count of 1.2 billion instructions. For a processor with a clock cycle time of one nano seconds, find the average CPI for each of the two programs.

(8 Points)

Q2. We are interested in adding a register-memory arithmetic instruction to the MIPS architecture. The new instruction exploits the I-format for load, store, and branch instructions. The new instruction has the label *ACCM* and employs an unused opcode in the ISA (the exact opcode is irrelevant). The semantics of the ACCM instruction is shown below:

Instruction: ACCM Rt, Const(Rs)
Interpretation: Reg[Rt] = Reg[Rt] + Mem[Reg[Rs] + Const]

MIPS I-Format (shown for convenience)

Opcode	Rs	Rt	Const
6 bits	5 bits	5 bits	16 bits

A. Draw the datapath and control signals for a single-cycle implementation of the ACCM instruction. Your datapath should show the new components, control signals, multiplexers, and instruction labels. Your illustration must show every logic and memory element on the critical path.

B. Identify the critical path for the ACCM instruction. Write the equation (like the lecture slides) for the critical path. For example, use t_{ALU} and t_{MEM} for the latency of ALU and memory. List all your assumptions.

(10 Points)

Q3. Consider the following instruction sequence:

	name	dst	src1	scr2
i1:	sub	r4	r1	r0
i2:	lw	r2	16(r4)	
i3:	lw	r1	4(r1)	
i4:	and	r2	r4	r2
i5:	sw	r2	0(r4)	

A. Suppose the pipelined in-order processor does not implement forwarding or hazard detection. As a programmer, your task is to insert nops to ensure correct execution. Insert nops to ensure correct execution.

B. Suppose the processor manufacturer forgot to implement hazard detection. The processor still implements forwarding. Explain the consequences of executing the above code on the buggy processor.

C. Now consider the following scenario: the processor does not implement forwarding. How should we change the hazard detection unit to ensure correct execution? List the conditions for detecting hazards. Explain the new input and output signals we need to add to our hazard detection unit. *Note: Use the above instruction sequence as an example to explain why each input/output signal is required.*

D. Suppose there is an irrelevant instruction between i4 and i5. Is the store instruction exposed to a hazard? How can we resolve the hazard (if any)?

(10 Points)

Q4. Consider an analytics application running on top of the MIPS processor. A fraction of instructions in this application exposes a specific type of RAW hazard. We identify the type of RAW hazard by the stage that produces the result (EX or MEM) and the instruction that consumes the result (1st following instruction, 2nd instruction that follows, or both). The type of RAW hazard and the fraction of instructions are shown in the table below. Answer the questions below with the following assumptions: (1) A register write happens in the first half of the clock cycle and a register read happens in the second half, (2) CPI of the processor is one if there are no data hazards.

Assume stores are never followed by loads. All other hazards can be resolved by other tricks (RF read/write policy).

EX to 1st Youngest	MEM to 1st Youngest	EX to 2nd Youngest	MEM to 2nd Youngest	EX to 1st and MEM to 2nd	Other RAW Hazards
7%	18%	5%	10%	10%	10%

IF	ID	EX (No FW)	EX (Full FW)	EX (FW from EX/MEM only)	EX (FW from MEM/WB only)	MEM	WB
250 ps	200 ps	220 ps	250 ps	240 ps	230 ps	220 ps	200 ps

A. What fraction of the cycles does the pipeline stalls with no forwarding?

B. What fraction of the cycles does the pipelines stalls with full forwarding?

C. What is the speedup with full forwarding versus no forwarding? *Note: Speedup is defined as the ratio of execution times with and without an optimization.*

D. To avoid the complexity of large-input multiplexers, we need to decide if it is better to forward only from the EX/MEM pipeline register or the MEM/WB pipeline register. Which option would you choose to minimize data stall cycles? (Show your calculation)

(4 points, 2, 2)

Q5. Find the longest chain of dependent instructions in the following code sequence. If maximizing IPC is the goal, should a microarchitect consider a stall-on-use in-order pipeline over a stall-on-miss in-order pipeline?

	name	**dst**	**src1**	**scr2**
i1:	add	r1	r1	r2
i2:	add	r1	r1	r3
i3:	sub	r1	r1	r4
i4:	load	r5	#0	r1
i5:	load	r7	#0	r8
i6:	add	r9	r5	r7

(12 Points)

Q6. Assume that a branch has the following sequence of taken (T) and not-taken (N) outcomes: **T,T,T,N,N,T,T,T,N,N,T,T,T,N,N**

A. What is the prediction accuracy for a 2-bit counter (Smith predictor) for this sequence assuming an initial state is strongly taken?

B. What is the minimum local history length needed to achieve perfect branch prediction for this branch outcome sequence?

C. Draw the corresponding PHT and fill in each entry with one of T (predict taken), N (predict not taken), or X (does not matter).

(6 points)

Q7.

A. Why does the register read stage must precede the issue stage in an out-of-order (OOO) processor (core) that uses an architectural register file (ARF) plus the reorder buffer to implement register renaming and hardware speculation?

B. List the reasons for separate dispatch and issue stages in an out-of-order (OOO) processor core that implements dynamic scheduling?

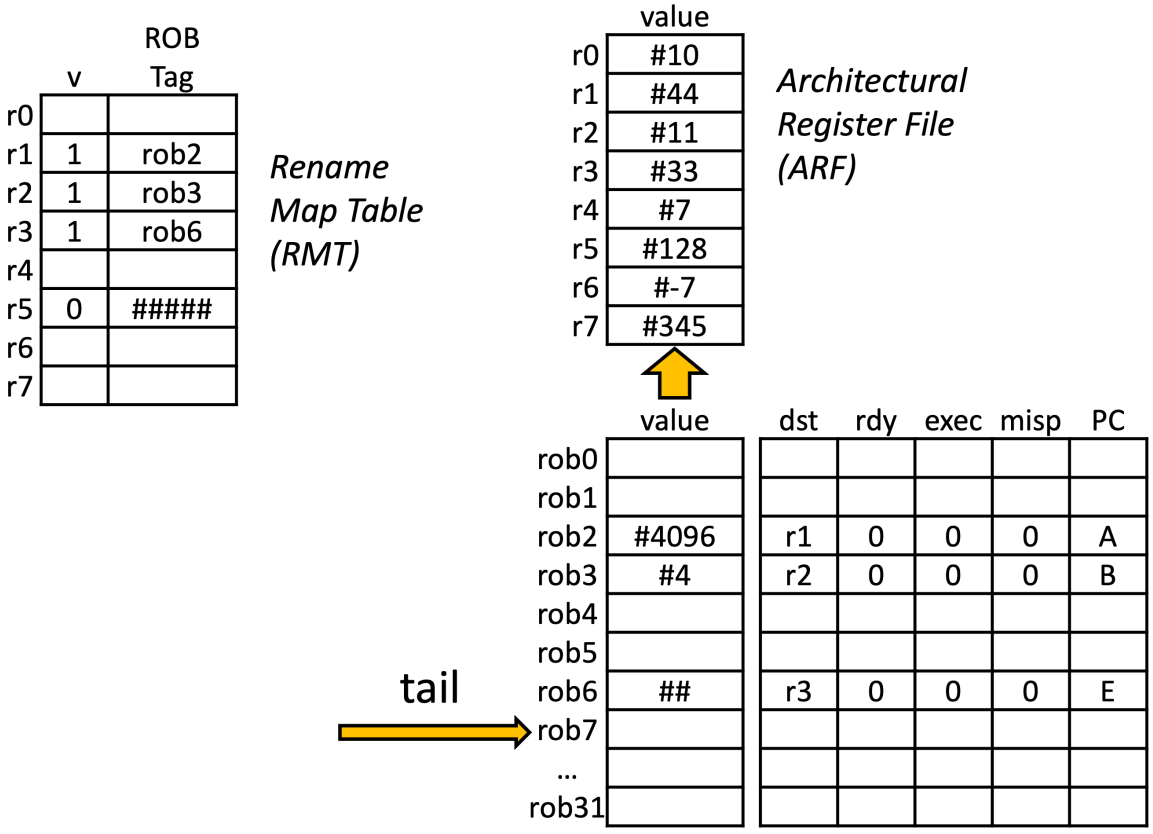
(6 points)

Q8. Indicate dependences and their types in the following instruction sequence. For each of the dependence types, explain the hazards that could result in the following microarchitectures: (1) single-cycle in-order, (2) pipelined in-order, and (3) pipelined out-of-order. *Assumption: No forwarding and hazard detection has been implemented yet.*

	opcode	dst	src1	scr2
i1:	and	r3	r4	r5
i2:	and	r4	r3	r6
i3:	and	r3	r3	r4

(6 points)

Q9. In this question, consider an out-of-order pipeline with an architectural register file (ARF) and a reorder buffer (ROB). The ROB has 32 entries. The tail currently points at the eighth entry of the ROB (rob7). The head of the ROB is stalled for an additional 100 cycles. The state of the ARF, the rename map table (RMT), and the ROB are shown below. Rename the destination and source register specifiers in the instruction sequence below. Identify the dependences in the original and the renamed sequence. Draw the state of the RMT after the instruction sequence is renamed.



	name	dst	src1	scr2
i1:	add	r5	r2	r1
i2:	lw	r3	4(r5)	
i3:	lw	r2	0(r2)	
i4:	or	r3	r5	r3
i5:	sw	r3	0(r5)	

(8 points, 2.5, 2.5)

Q10. Briefly explain how we can add the following features to the CDC 6600 scoreboard. (1) Register renaming. (2) Hardware speculation. Start with the scoreboard design as we studied in the lectures and briefly explain the steps required to add the two features.

(8 points, 2, 2, 2)

Q11. The complexity of processor pipelines we have encountered in the lectures vary. We rank three different pipelines with increasing complexity as follows: (1) stall-on-miss (simple) (2) stall-on-use (moderately complex) (3) ARF+ROB (very complex). For each of the following scenarios, pick the simplest pipeline that would likely deliver the highest IPC. The in-order pipelines do not use branch prediction. The OOO pipeline uses a simple one-bit branch predictor.

1. **Scenario 1:** Frequent RAW hazards, infrequent branches, negligible WAR/WAW hazards, infrequent memory operations

2. **Scenario 2:** Infrequent RAW hazards, frequent hard-to-predict branches, frequent independent memory operations, frequency of WAR/WAW hazards is unknown

3. **Scenario 3:** Same as scenario 2, but easy to predict branches, and the frequency of WAR/WAW hazards is known to be very high

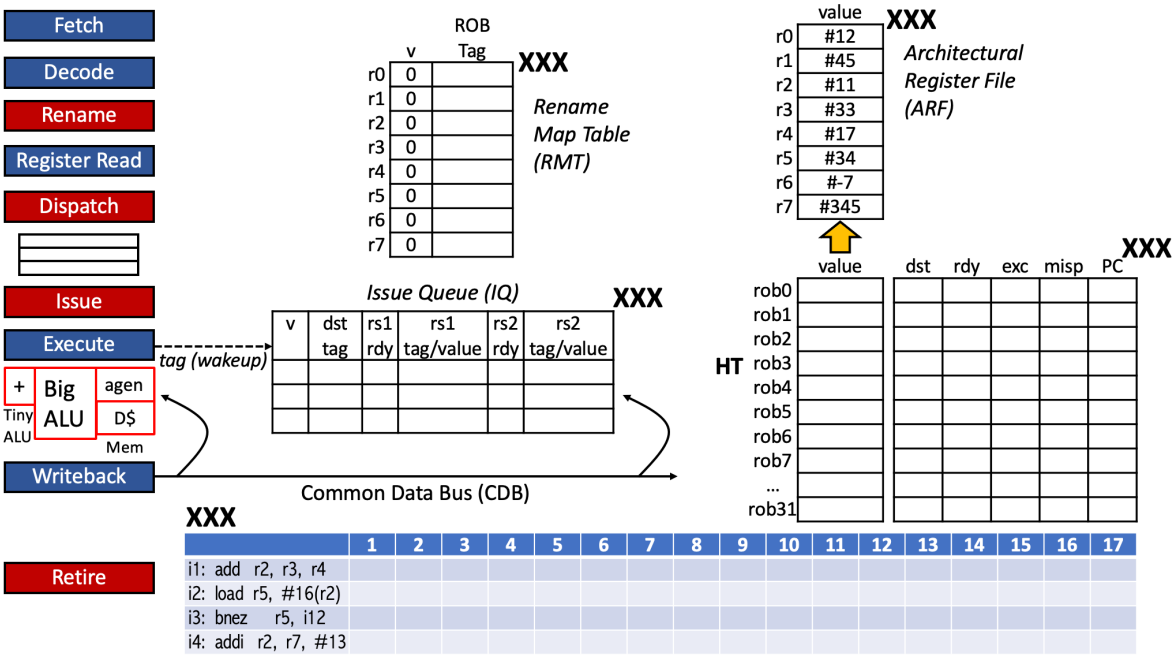
(20 points)

Q12. This question has an associated PowerPoint template slide (see next page) that you need to fill three times (for three scenarios) and attach to your final pdf submission. Consider the following instruction sequence. Suppose we run this sequence on an ARF+ROB pipeline. Your task is to fill the contents of the RMT, the issue queue, ROB, the architectural register file, and the multiple-clock-cycle diagram (bottom of the slide) at three different points in the execution of the code sequence. These structures are marked as **XXX** in the template slide.

Questions: Provide the contents of all structures marked **XXX** in the following cycles: **(1)** when the instruction i2 is in the register read stage **(2)** when the instruction i3 is in the issue stage and **(3)** when the instruction i4 is in the retire stage. In which cycle does the branch instruction sets/resets the misprediction bit in the ROB? Provide an itemized list of all the actions that take place in the pipeline during that cycle. (You should answer the last question in text, but you may fill out the PowerPoint template slide one more time for cycle # 13 and attach it to your pdf submission.)

Assumptions: Assume the processor uses the always-untaken branch prediction strategy. Also assume that branch i3 is not taken (on resolution). Assume i2 results in a data cache hit. The data cache hit latency is three cycles. Therefore, the load instruction takes one cycle for address calculation and three cycles for retrieving the value from the data cache. All other operations take one cycle to execute. The initial state of the RMT, the architectural register file, and the head/tail pointers of the ROB is shown on the slide. you should infer "when the instruction is in the issue stage" as when the instruction enters the issue stage. Same for the retire stage.

	name	**dst**	**src1**	**scr2**
i1:	add	r2	r3	r4
i2:	load	r5	#16	r2
i3:	bnez		r5	i12
i4:	addi	r2	r7	#13


Fetch
Decode
Rename
Register Read
Dispatch
Issue
Execute
+
Big
ALU
agen
D$
Tiny
ALU
Mem
Writeback
Retire
ROB
v
Tag
r0 0
r1 0
r2 0
r3 0
r4 0
r5 0
r6 0
r7 0
XXX
Rename
Map Table
(RMT)
value
r0 #12
r1 #45
r2 #11
r3 #33
r4 #17
r5 #34
r6 #-7
r7 #345
XXX
Architectural
Register File
(ARF)
Issue Queue (IQ)
XXX
v
dst
tag
rs1
rdy
rs1
tag/value
rs2
rdy
rs2
tag/value
tag (wakeup)
Common Data Bus (CDB)
HT
value
dst
rdy
exc
misp
PC
XXX
rob0
rob1
rob2
rob3
rob4
rob5
rob6
rob7
...
rob31
XXX
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
i1: add r2, r3, r4
i2: load r5, #16(r2)
i3: bnez r5, i12
i4: addi r2, r7, #13

Solutions

Q1:

Cycles(X)=1e+9, Cycles(Y)=1.5e+9, CPI(X)=1, CPI(Y)=1.25

Q2:

One approach: (1) There is an extra ALU in the datapath. (2) The second ALU is fed data from either data memory or RF. (3) There is a multiplexer in front of the second ALU that chooses an operand from either memory (ACCM) or register (other instructions). (Note: There are other ways of doing this but a second ALU and multiplexer is not avoidable. There are ways to avoid putting the extra multiplexer off the critical path. But most correct solutions report the following equation for the critical path.)

The critical path is: Tpc + 2*Tmem* + *Trf-read* + *2*Talu + 2*Tmux + Trf-setup

Q3:

(A) 2 nops b/w i1 and i2, 1 nop b/w i3 and i4, 2 nops b/w i4 and i5

(B) Code executes correctly. There is no load-use hazard

(C) We need to detect dependency with first/second youngest instruction.

The instruction that is currently in the ID stage needs to be stalled if it depends on a value produced by the instruction in the EX or the instruction in the MEM stage. So we need to check the destination register of these two instructions. For the instruction in the EX stage, we need to check Rd for R-type instructions and Rd for loads. For the instruction in the MEM stage, the destination register is already selected (by the Mux in the EX stage) so we need to check that register number (this is the bottommost output of the EX/MEM pipeline register).

First youngest:
ID/EX.RegWrite and
((ID/EX.RegisterRt = IF/ID.RegisterRs) or
(ID/EX.RegisterRt = IF/ID.RegisterRt))

Second youngest:
EX/MEM.RegWrite and
((EX/MEM.RegisterRt = IF/ID.RegisterRs) or
(EX/MEM.RegisterRt = IF/ID.RegisterRt))

(D) We now need forwarding from Mem to Ex stage (or insert an extra nop).

Q4:

(A) Dependences to the 1st next instruction result in 2 stall cycles, and the stall is also 2 cycles if the dependence is to both 1st and 2nd next instruction. Dependences to only the 2nd next instruction result in one stall cycle. We have: 1 + (7%+18%+10%)*2 + (5%+10%)*1 = 1.85 (46%)

(B) With full forwarding, the only RAW data dependences that cause stalls are those from the MEM stage of one instruction to the 1st next instruction. Even this dependences causes only one stall cycle, so we have: 1+(18%)*1=1.18 (15%)

(C) We first need to compute the execution times using CPI and cycle times.

Execution time with no forwarding = 1.85 * 250 = 463ps

Execution time with forwarding = 1.18 * 250 = 295ps

Speedup = 463/295 = 1.57

(D) First, we need to calculate the CPI with each option.

CPI with forwarding from EX/MEM = 1 + (18%)*2 + (10%+10%)*1 = 1.56

CPI with forwarding from MEM/WB = 1 + (7%+10%)*2 + (18%+5%) = 1.57

Forwarding from EX/MEM is better.

Q5:

i1—i2—i3—i4—i6

With stall-on-use, we can issue i4 and i5 in parallel

Q6:

(A) 7/15 (47%)

(B) 3

PHT:

000: X
001: T
010: X

011: T
100: T
101: X
110: N
111: N

Q7:

A: The values of destination registers reside either in ARF or ROB. If an instruction in the rename/register-read stage does not capture the values from ROB, then the value may move to the ARF by the time instruction is ready to execute. This is the reason why we must have register read stage before the issue stage in ARF+ROB.

B: The issue stage enables dynamic scheduling of instruction out of the original program order. The instruction enter the issue queue in program order (dispatch) and are selected by the dynamic scheduler (issue) for execution out of order.

Q8:

(1) no hazards (2) RAW only (3) All hazards are possible (need renaming to avoid WAR and WAW)

i1->i2: RAW (r3)
i1->i3: RAW (r3)
i2->i3: RAW (r4)
i1->i2: WAR (r4)
i2->i3: WAR (r3)
i1->i3: WAW (r3)

Q9:

No dependences in the renamed sequence (purpose of renaming)

Dependences in the original sequence:

```
i1->i2: RAW (r5)
i1->i4: RAW (r5)
i1->i5: RAW (r5)
i1->i3 WAR (r2)
i2->i4: WAW (r3)
i2->i4: RAW (r3)
i4->i5: RAW (r3)
```

Renamed Sequence:

```
add rob7, rob3, rob2
```

```
lw rob8, 4(rob7)
lw rob9, 0(rob3)
or rob10, rob7, rob8
sw rob10, 0(rob7)
```

RMT:

```
r0
r1 1 rob2
r2 1 rob9
r3 1 rob10
r4
r5 1 rob7
```

Q10:

Register renaming: (1) Add RMT, (2) Add an extended set of registers (e.g., via ROB)

Hardware speculation: Use ROB

Q11:

(1) Stall-on-miss

(2) Stall-on-use

(3) ARF+ROB

Q12:

See next few pages.

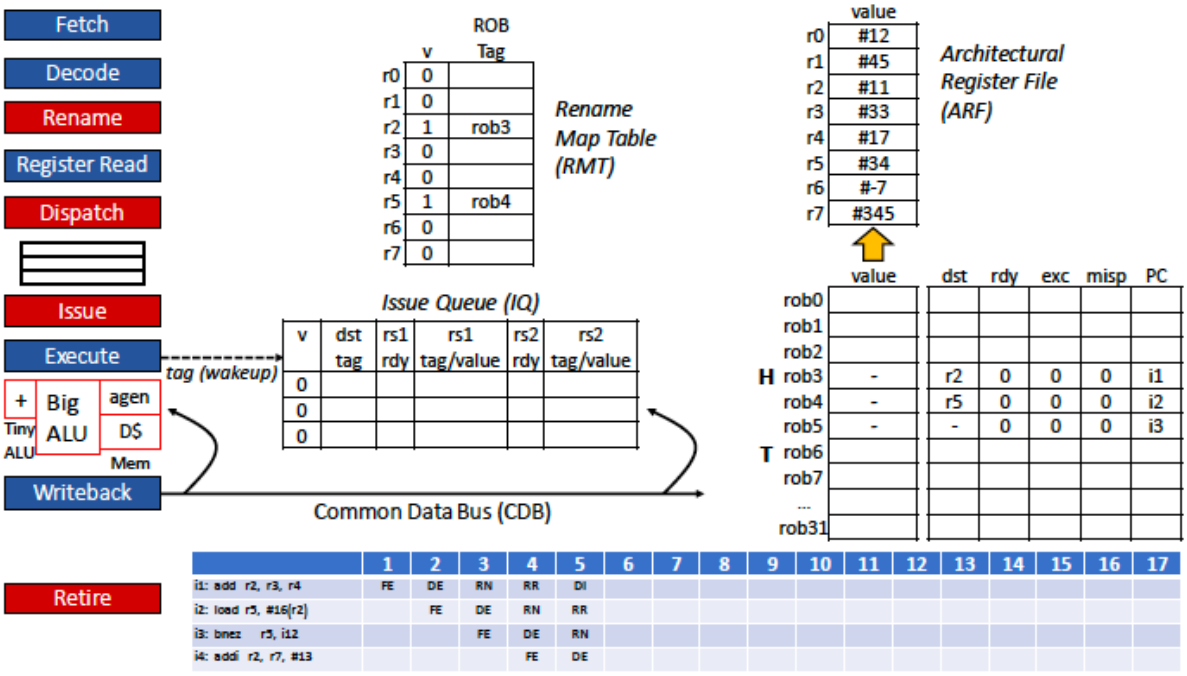


Figure 3: The (micro)architectural state when `i2` is in its Register Read stage

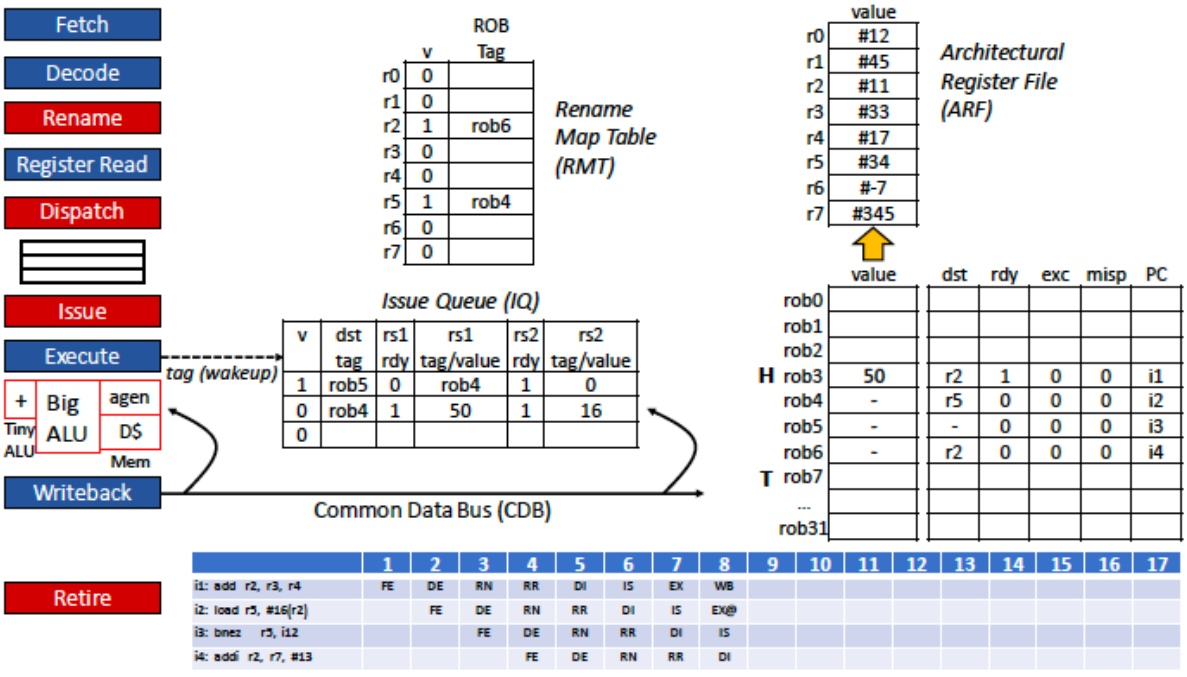


Figure 4: The (micro)architectural state when `i3` is in its Issue stage

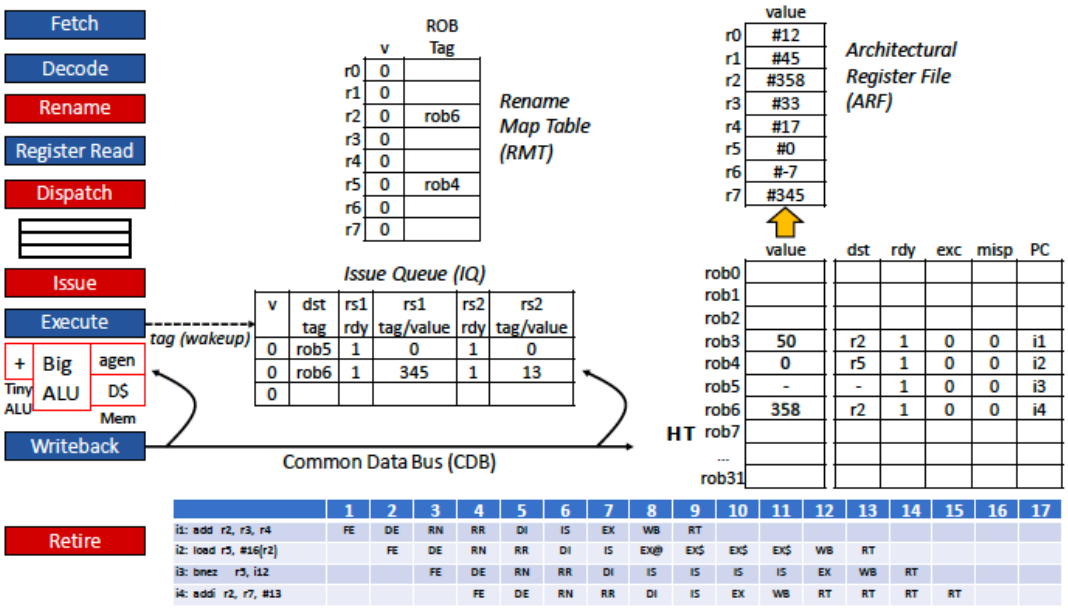


Figure 5: The (micro)architectural state when `i4` is in its Retire stage

The branch instruction sets/resets the misprediction bit in the writeback stage, so in this case it will be cycle 13. During this cycle the following actions occur:

- `i2` retires:
  - writes its result to ARF[5] (i.e. `r5`)
  - increments the ROB head
  - clears the valid bit of RMT[5] (i.e. `r5`)
- `i3` writes back:
  - `rob5` marked as ready
  - if there was a misprediction, then it would set the mispredicted bit for `rob5`. However, in this case we don't have a misprediction, so the bit is left unset
  - `rob5` tag and result value is broadcast on the CDB (nothing can depend on `rob5` though, as it's a branch)