16.482 / 16.561 Computer Architecture and Design

Instructor: Dr. Michael Geiger

Spring 2014

Lecture 6:

Speculation

Midterm exam preview

Lecture outline

- Announcements/reminders
  - HW 4 due today
  - Midterm exam next week
- Today's lecture
  - Speculation
  - Midterm exam preview

Review: Dynamic scheduling

- Dynamic scheduling - hardware rearranges the instruction execution to reduce stalls while maintaining data flow and exception behavior
  - Key idea: Allow instructions behind stall to proceed
  - Allow out-of-order execution and out-of-order completion
  - We use Tomasulo's Algorithm
  - Decode stage now handles:
    - Issue—check for structural hazards and assign instruction to functional unit (via reservation station)
      - Check for register values
  - Reservation stations implicitly perform register renaming
    - Resolves potential WAW, WAR hazards
  - Results broadcast over common data bus

Speculation to greater ILP

- 3 components of HW-based speculation:

1. Dynamic branch prediction
    - Need BTB to get target in 1 cycle
2. Ability to speculate past branches
3. Dynamic scheduling

- In Tomasulo's algorithm, separate instruction completion from commit
    - Once instruction is non-speculative, it can update registers/memory
    - Reorder buffer tracks program order
        - Head of ROB can commit when ready
        - ROB supplies data between complete and commit

Reorder Buffer Entry

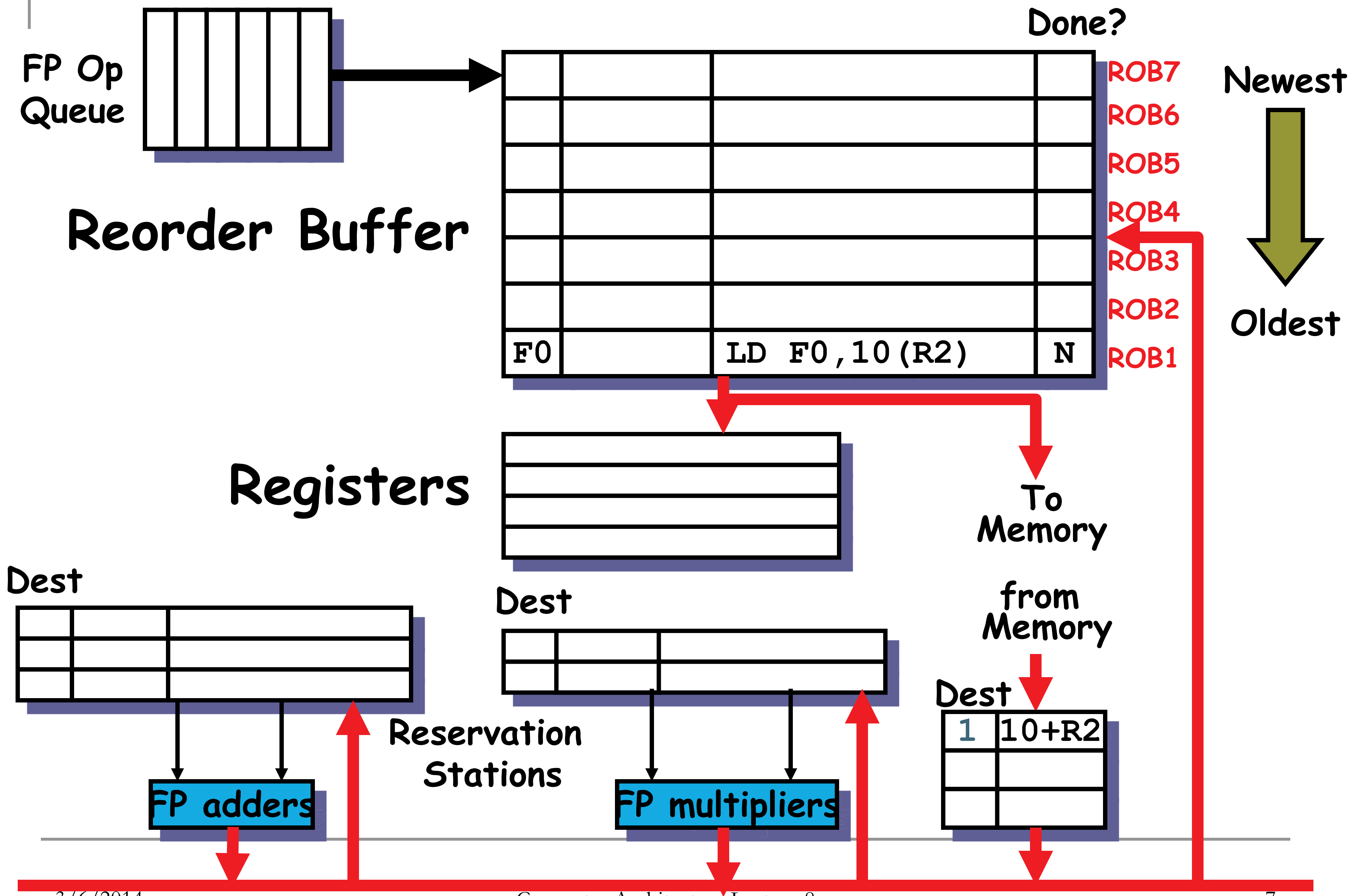
- Each entry in the ROB contains four fields:

1. Instruction type
   - a branch (has no destination result), a store (has a memory address destination), or a register operation (ALU operation or load, which has register destinations)
2. Destination
   - Register number (for loads and ALU operations) or memory address (for stores)
     where the instruction result should be written
3. Value
   - Value of instruction result until the instruction commits
4. Ready
   - Indicates that instruction has completed execution, and the value is ready

Speculative Tomasulo's Algorithm

1. Instruction fetch--get instruction from memory; place in Op Queue
2. Issue—get instruction from FP Op Queue
   - If reservation station and reorder buffer slot free, issue instr & send operands & reorder buffer no. for destination (this stage sometimes called "dispatch")
3. Execution—operate on operands (EX)
   - When both operands ready then execute; if not ready, watch CDB for result; when both in reservation station, execute; checks RAW (sometimes called "issue")
4. Memory access--if needed (MEM)
   - NOTE: Stores update memory at commit, not MEM
5. Write result—finish execution (WB)
   - Write on Common Data Bus to all awaiting FUs & reorder buffer; mark reservation station available.
6. Commit—update register with reorder result
   - When instr. at head of reorder buffer & result present, update register with result (or store to memory) and remove instr from reorder buffer. Mispredicted branch flushes reorder buffer (sometimes called "graduation")

Tomasulo's With Reorder buffer:



Revisiting stores with speculation

- With ROB, store buffers eliminated
- Why?
  - Can’t write memory until you know value is non-speculative
  - Once address is calculated, store in “destination” field of ROB entry
  - Need additional field in ROB for stores: equivalent to “Q” fields in reservation stations
    - Indicates what instruction is writing value to be stored

Reorder buffer example

- Given the following code:

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

- Walk through two iterations of the loop
- Assume
  - 2 cycles for add, load
  - 1 cycle for address calculation
  - 6 cycles for multiply
  - Forwarding via CDB

Reorder buffer example: key points

- Execution stages
  - Fetch & issue: always in order
  - Execution & completion: may be out of order
  - Commit: always in order
- Hardware
  - Reservation stations
    - Occupied from IS to WB
  - Reorder buffer
    - Occupied from IS to C
    - Used to
      - Maintain program order for in-order commit
      - Supply register values between WB and C
  - Register result status
    - Rename registers based on ROB entries

Memory hazards, exceptions

- Reorder buffer helps limit memory hazards
  - With additional logic for disambiguation (determine if addresses match)
  - WAW / WAR automatically removed
  - RAW maintained by
    - Stalling loads if store with same address is in flight
    - Ensuring that effective addresses are computed in order
- Precise exceptions logical extension of ROB
  - If instruction causes exception, flag in ROB
  - Handle exception when instruction commits

Midterm exam notes

- Allowed to bring:
  - Two 8.5" x 11" double-sided sheets of notes
  - Calculator
- No other notes or electronic devices (phone, laptop, etc.)
- Exam will last until 9:30
  - Will be written for ~90 minutes
- Covers all lectures through this week
  - Material starts with MIPS instruction set
- Question formats
  - Problem solving
  - Some short answer—may be asked to explain concepts
  - Similar to homework, but shorter
- Old exams are on website
  - Note: not all material the same
  - No performance assessment on this semester's midterm
  - Dynamic scheduling/speculation were on the final last semester

Review: MIPS addressing modes

- MIPS implements several of the addressing modes discussed earlier
- To address operands
  - Immediate addressing
    - Example: addi $t0, $t1, **150**
  - Register addressing
    - Example: sub **$t0, $t1, $t2**
  - Base addressing (base + displacement)
    - Example: lw $t0, **16($t1)**
- To transfer control to a different instruction
  - PC-relative addressing
    - Used in conditional branches
  - Pseudo-direct addressing
    - Concatenates 26-bit address (from J-type instruction) shifted left by 2 bits with the 4 upper bits of the PC

Review: MIPS integer registers

Name	Register number	Usage
`$zero`	0	Constant value 0
`$v0-$v1`	2-3	Values for results and expression evaluation
`$a0-$a3`	4-7	Function arguments
`$t0-$t7`	8-15	Temporary registers
`$s0-$s7`	16-23	Callee save registers
`$t8-$t9`	24-25	Temporary registers
`$gp`	28	Global pointer
`$sp`	29	Stack pointer
`$fp`	30	Frame pointer
`$ra`	31	Return address

- List gives mnemonics used in assembly code
  - Can also directly reference by number ($0, $1, etc.)
- Conventions
  - $s0-$s7 are preserved on a function call (callee save)
  - Register 1 ($at) reserved for assembler
  - Registers 26-27 ($k0-$k1) reserved for operating system

Review: MIPS data transfer instructions

- For all cases, calculate effective address first
  - MIPS doesn’t use segmented memory model like x86
  - Flat memory model → EA = address being accessed
- lb, lh, lw
  - Get data from addressed memory location
  - Sign extend if lb or lh, load into rt
- lbu, lhu, lwu
  - Get data from addressed memory location
  - Zero extend if lb or lh, load into rt
- sb, sh, sw
  - Store data from rt (partial if sb or sh) into addressed location

Review: MIPS computational instructions

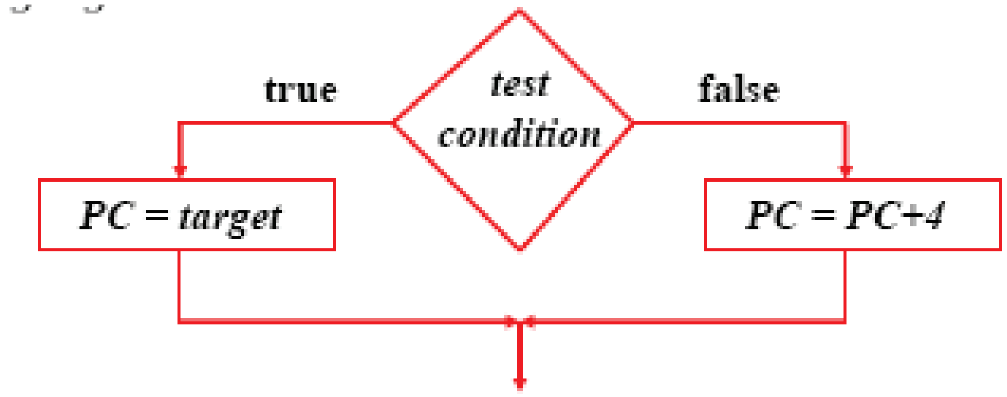
- Arithmetic
  - Signed: add, sub, mult, div
  - Unsigned: addu, subu, multu, divu
  - Immediate: addi, addiu
    - Immediates are sign-extended
- Logical
  - and, or, nor, xor
  - andi, ori, xori
    - Immediates are zero-extended
- Shift (logical and arithmetic)
  - srl, sll – shift right (left) logical
    - Shift the value in rs by shamt digits to right or left
    - Fill empty positions with 0s
    - Store the result in rd
  - sra – shift right arithmetic
    - Same as above, but sign-extend the high-order bits
  - Can be used for multiply / divide by powers of 2

Review: computational instructions (cont.)

- Set less than
  - Used to evaluate conditions
    - Set rd to 1 if condition is met, set to 0 otherwise
  - slt, sltu
    - Condition is rs < rt
  - slti, sltiu
    - Condition is rs < immediate
    - Immediate is *sign-extended*
- Load upper immediate (lui)
  - Shift immediate 16 bits left, append 16 zeros to right, put 32-bit result into rd

Review: MIPS control instructions

- Branch instructions test a condition
  - Equality or inequality of rs and rt
    - beq, bne
    - Often coupled with slt, sltu, slti, sltiu
  - Value of rs relative to rt
    - Pseudoinstructions: blt, bgt, ble, bge
- Target address → add sign extended immediate to the PC
  - Since all instructions are words, immediate is shifted left two bits before being sign extended



Review: MIPS control instructions (cont.)

- *Jump* instructions unconditionally branch to the address formed by either
  - Shifting left the 26-bit target two bits and combining it with the 4 high-order PC bits
    - j
  - The contents of register $rs
    - jr
- *Branch-and-link* and *jump-and-link* instructions also save the address of the next instruction into $ra
  - jal
  - Used for subroutine calls
  - jr $ra used to return from a subroutine

Review: Binary multiplication

- Generate shifted partial products and add them
- Hardware can be condensed to two registers
    - N-bit multiplicand
    - 2N-bit running product / multiplier
    - At each step
        - Check LSB of multiplier
        - Add multiplicand/0 to left half of product/multiplier
        - Shift product/multiplier right
- Signed multiplication: Booth's algorithm
    - Add extra bit to left of all regs, right of prod./multiplier
    - At each step
        - Check two rightmost bits of prod./multiplier
        - Add multiplicand, -multiplicand, or 0 to left half of prod./multiplier
        - Shift product multiplier right
    - Discard extra bits to get final product

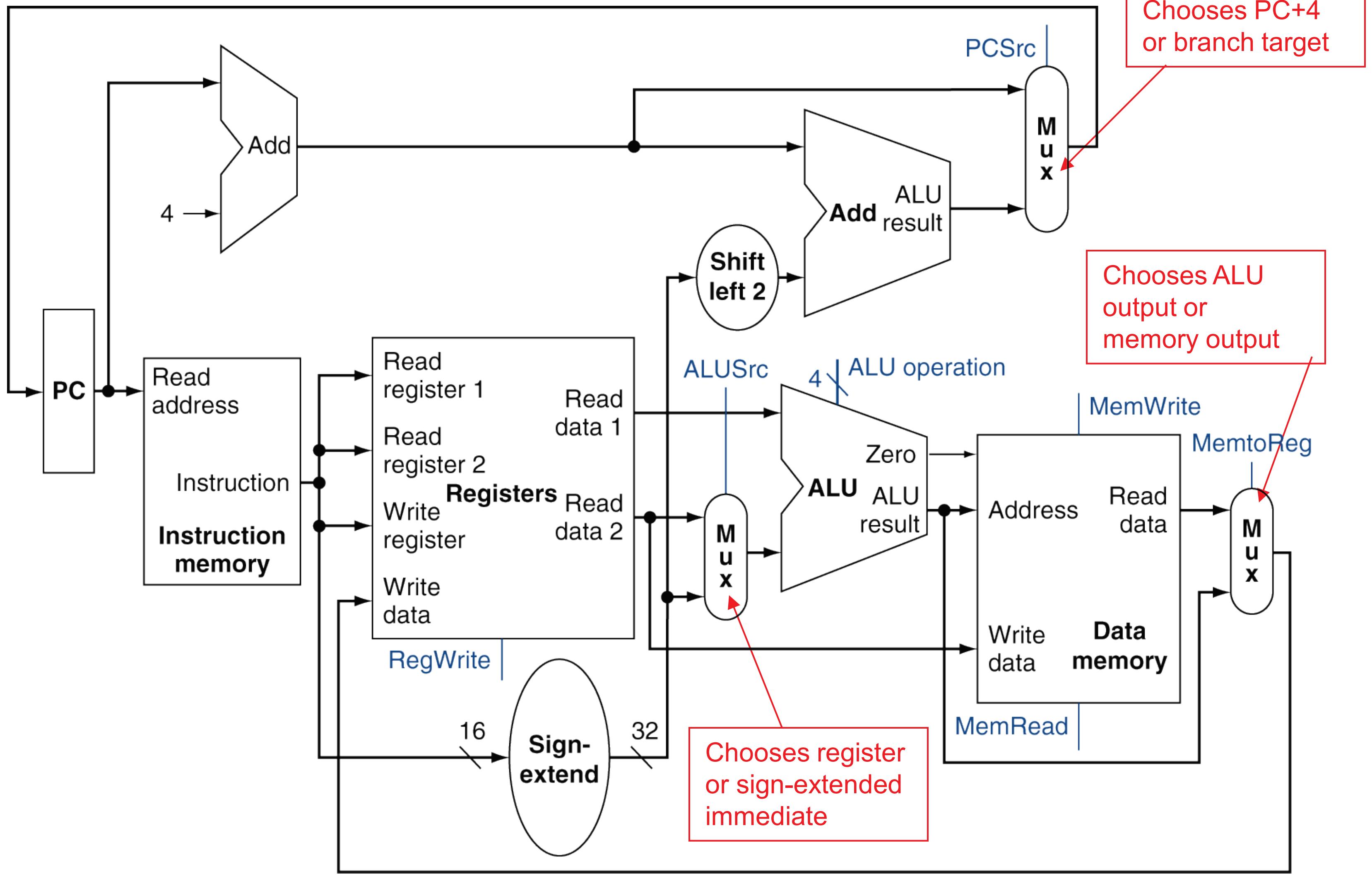
Review: IEEE Floating-Point Format

	single: 8 bits double: 11 bits	single: 23 bits double: 52 bits
S	Exponent	Fraction

$$x = (-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent}-\text{Bias})}$$

- S: sign bit (0 $\Rightarrow$ non-negative, 1 $\Rightarrow$ negative)
- Normalize significand: 1.0 ≤ |significand| < 2.0
  - Significand is Fraction with the "1." restored
- Actual exponent = (encoded value) - bias
  - Single: Bias = 127; Double: Bias = 1023
- Encoded exponents 0 and 111 ... 111 reserved
- FP addition: match exponents, add, then normalize result
- FP multiplication: add exponents, multiply significands, normalize results

Review: Simple MIPS datapath



Review: Pipelining

- Pipelining → low CPI and a short cycle
  - Simultaneously execute multiple instructions
  - Use multi-cycle “assembly line” approach
  - Use staging registers between cycles to hold information
- Hazards: situation that prevents instruction from executing during a particular cycle
  - Structural hazards: hardware conflicts
  - Data hazards: dependences cause instruction stalls; can resolve using:
    - No-ops: compiler inserts stall cycles
    - Forwarding: add hardware paths to ALU inputs
  - Control hazards: must wait for branches
    - Can move target, comparison into ID → only 1 cycle delay

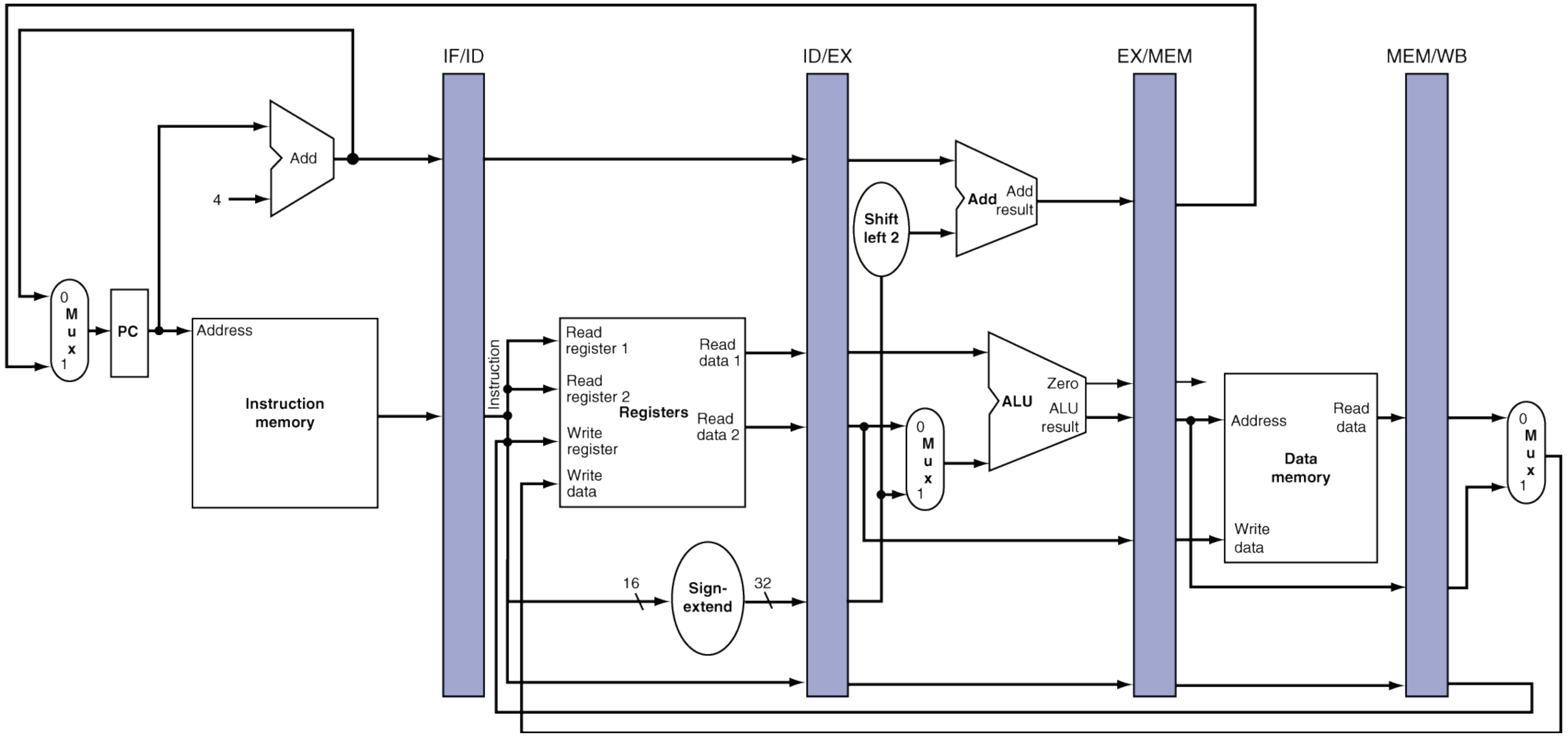
Review: Pipeline diagram

	Cycle 1	2	3	4	5	6	7	8
`lw`	IF	ID	EX	MEM	WB			
`add`		IF	ID	EX	MEM	WB		
`beq`			IF	ID	EX	MEM	WB	
`sw`				IF	ID	EX	MEM	WB

- Pipeline diagram shows execution of multiple instructions
  - Instructions listed vertically
  - Cycles shown horizontally
  - Each instruction divided into stages
  - Can see what instructions are in a particular stage at any cycle

Review: Pipeline registers

- Need registers between stages for info from previous cycles
- Register must be able to hold all needed info for given stage
  - For example, IF/ID must be 64 bits—32 bits for instruction, 32 bits for PC+4
- May need to propagate info through multiple stages for later use
  - For example, destination reg. number determined in ID, but not used until WB

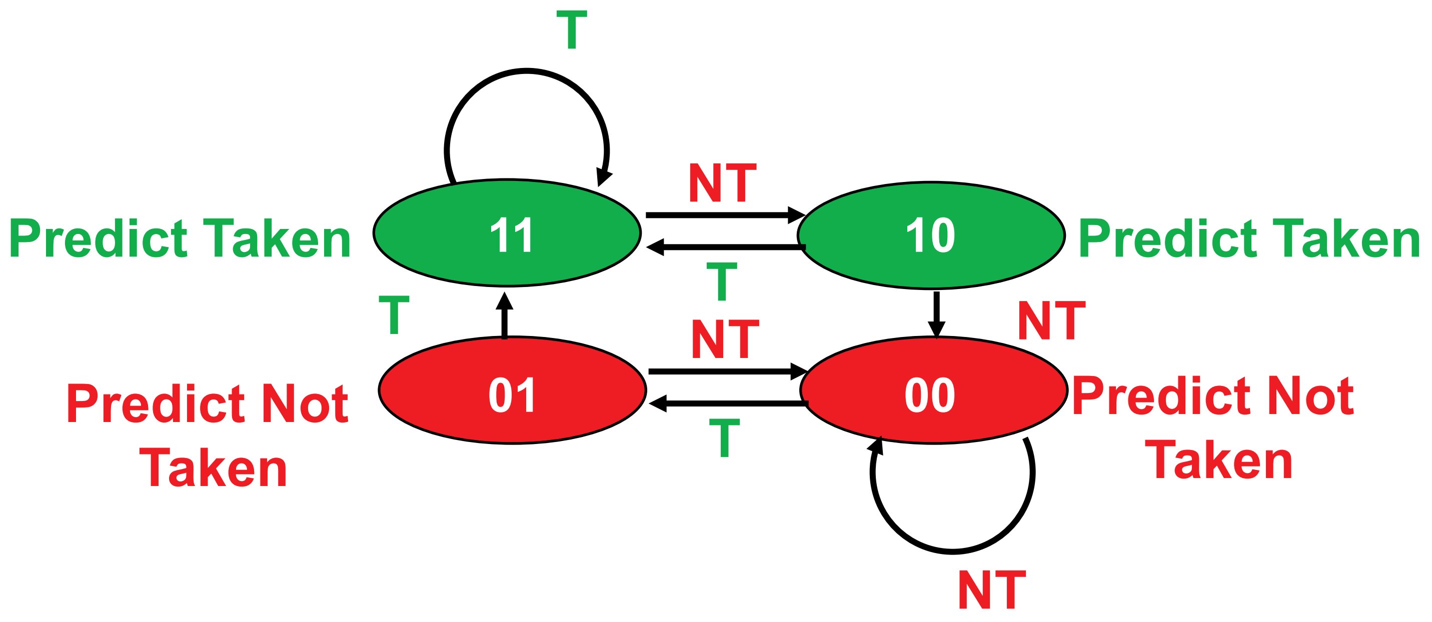


Review: Dynamic Branch Prediction

- Want to avoid branch delays
- Dynamic branch predictors: hardware to predict branch outcome (T/NT) in 1 cycle
  - Use branch history to determine predictions
  - Doesn't calculate target
- Branch history table: basic predictor
  - Which line of table should we use?
    - Use appropriate bits of PC to choose BHT entry
    - # index bits = $\log_2$(# BHT entries)
  - What's prediction?
  - How does actual outcome affect next prediction?

Review: BHT

- Solution: 2-bit scheme where change prediction only if get misprediction twice



- Red: “stop” (branch not taken)
- Green: “go” (branch taken)

Review: Correlated predictors, BTB

- Correlated branch predictors
  - Track both individual branches and overall program behavior (global history)
    - Makes some branches easier to predict
  - To make a prediction
    - Branch address chooses row
    - Global history chooses column
    - Once entry chosen, make prediction in same way as basic BHT (11/10 → predict T, 00/01→predict NT)
- Branch target buffers
  - Save previously calculated branch targets
  - Use branch address to do fully associative search

Review: Dynamic scheduling

- Dynamic scheduling - hardware rearranges the instruction execution to reduce stalls while maintaining data flow and exception behavior
  - Key idea: Allow instructions behind stall to proceed
  - Allow out-of-order execution and out-of-order completion
  - We use Tomasulo's Algorithm
  - Decode stage now handles:
    - Issue—check for structural hazards and assign instruction to functional unit (via reservation station)
      - Check for register values
  - Reservation stations implicitly perform register renaming
    - Resolves potential WAW, WAR hazards
  - Results broadcast over common data bus

Review: Speculation

- Hardware speculation, a technique with significant performance advantages, builds on dynamic scheduling
  - Assume branch predictions are correct
    - Speculate past control dependences
  - Allow instructions to execute and complete out-of-order, but must commit in order
    - Extra stage: when instructions commit, they update the register file or memory
    - In-order commit also allows precise exceptions
    - Reorder buffer maintains program order
      - Also effectively replaces register file for in-flight instructions—instructions first check reorder buffer for operand values

Final notes

- Next time
  - Midterm exam
- Reminders
  - HW 4 due today