Superscalar

Computer Architecture: Design and Simulation
CMU 15-346, Spring 2021

How to go below 1 CPI

- Pipelining has its limitations
  - Latch delays
  - Dependencies

- Fundamentally, fetching 1 per cycle is a limitation

Simple first

- Duplicate the pipeline
  - VLIW
  - Compiler groups multiple instructions into bundles
  - Can the compiler make big enough bundles?
  - (maybe more on this later)

Fetching More

- To execute more than 1, the processor needs to fetch more
- P1: Instruction bundle crosses cache line
  - Multiport cache
  - Banking
- P2: Branches
  - Access predictor / BTB for all four instructions (banking)
  - Chain results to determine fetch bundle
  - Trace cache

New Pipeline

- Fetch -> Decode -> Schedule -> Execute -> State Update

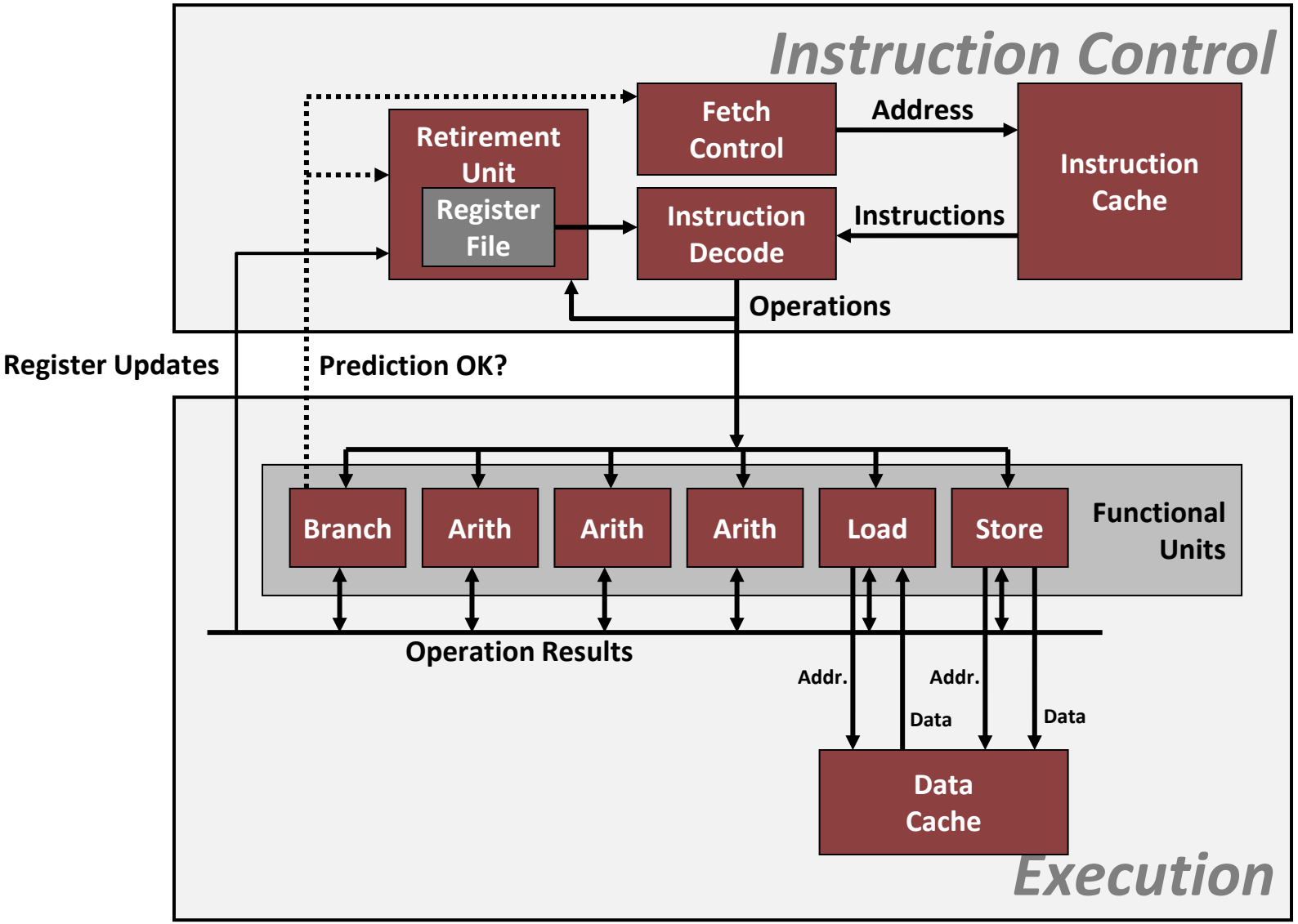
Old ID

- Welcome to queuing theory ;)

The new ID stage(s)

- In-order flow of fetched instructions
  - Checks dependencies
  - Dispatches instructions to specific function units

Modern CPU Design



Dynamic Scheduling

- Can processors find independence in the instruction stream? (ILP)
  - Especially, can we hide memory latency?

- Interesting things:
  - Dependencies / Registers
  - Exceptions

Approaches

- Fire (or issue) in-order or out-of-order
- Complete in-order or out-of-order

- FICI – 5-stage from before
- FICO – Coming up
- FOCO – Stay tuned
- FOCI – No.

FICO picture

- Still combined dispatch / schedule stage
- Register file with busy and value
- Scoreboard for each FU with busy

A dynamic scheduling algorithm: Simple interlocking (FICO)

1. [*Dispatch/Scheduling unit (combined unit)*]

    (a) for all FU's do:

        if (FU[i] *is pipelined*) then Scoreboard[i].Busy = False

    (b) for each Inst in SchedQueue do:

        if (!Regs[Inst.Src1].Busy

            AND !Regs[Inst.Src2].Busy

            AND !Regs[Inst.Dest].Busy

            AND !Scoreboard[Inst.FU].Busy)

        then

            Scoreboard[Inst.FU].Busy = True

            Regs[Inst.Dest].Busy = True

            // Issue instruction to function unit Inst.FU

        else

            exit loop *// halt issuing*

2. [*Execution unit, at completion of instruction* Inst]

    (a) Regs[Inst.Dest].Busy = False

    (b) if (Inst.FU *pipeline advances and now first stage is free*) then

        Scoreboard[Inst.FU].Busy = False

Examples

- ADD R1, R2, R3
- MUL R4, R1, R5
- ADD R3, R1, R2
- ADD R2, R4, R1

- MUL R1, R2, R3
- ADD R2, R3, R1
- ADD R4, R5, R6

FOCO

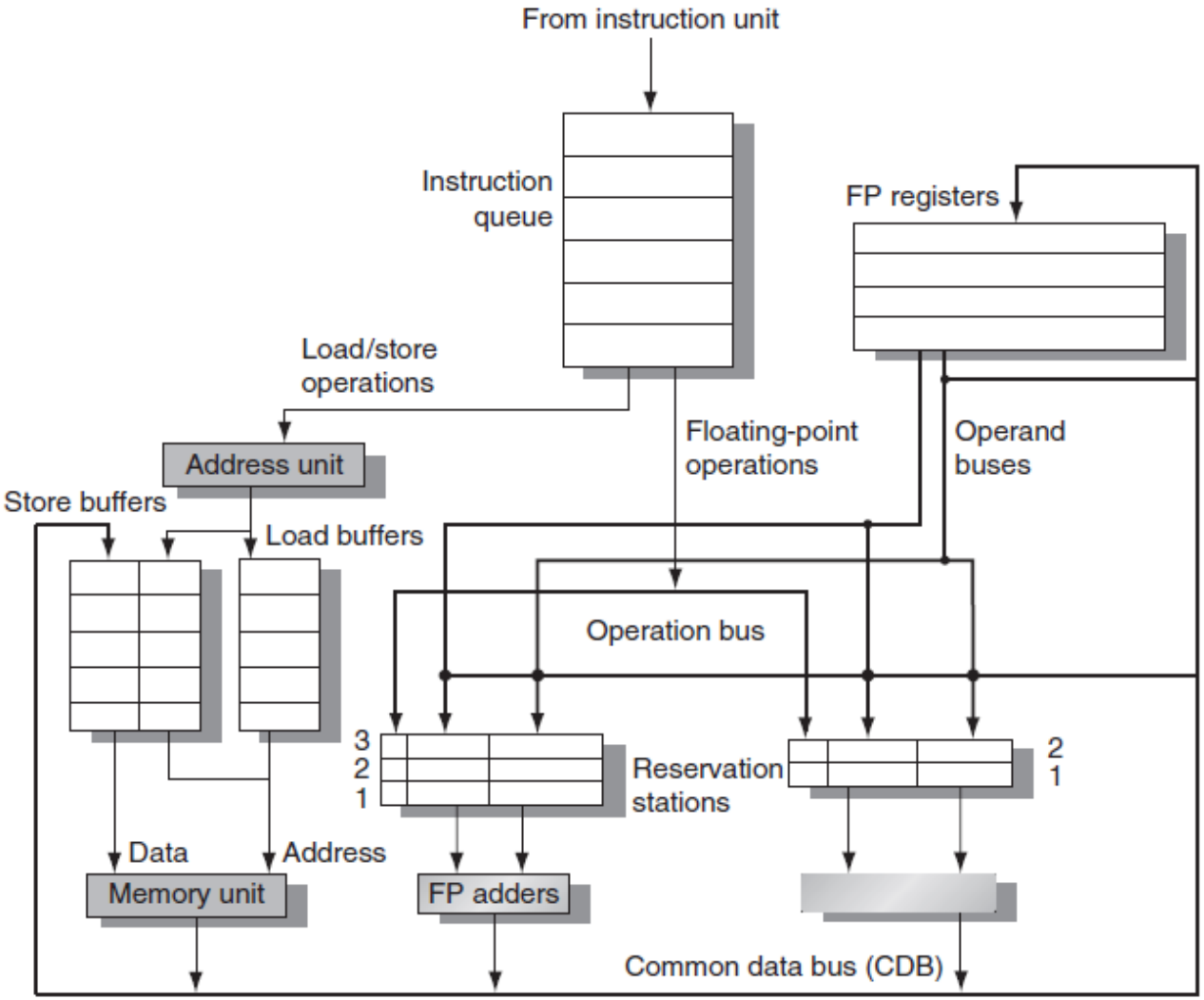
- Split ID
  - Dispatch in-order
  - Schedule out-of-order

- Common Data Bus (or result bus(es))
  - Broadcast completed results to all waiting instructions

Renaming Registers

- Robert Tomasulo had a problem
  - IBM 360 architecture had 4 FP registers
    - Lots of hazards
    - Not necessarily superscalar (hence one CDB)
  - Everything scheduled gets a tag

Tomasulo Hardware



Tomasulo algorithm, part 1

For each cycle of execution…

```
[Dispatch unit ] for all instructions I in DispatchQueue do:
    if (SchedQueue is not full) then
        (a) Add I to first free slot of SchedQueue  (="RS")      // Call this RS below
        (b) Delete I from DispatchQueue                          // Insttruction I is now "dispatched"
        (c) RS.FU = I.FU
        (d) RS.Dest = I.Dest
        (e) for all source registers, i, of I do
                if (Regs[I.Src[i]].Ready = True) then
                    RS.Src[i].Value = Regs[I.Src[i]].Value
                    RS.Src[i].Ready = True
                else
                    RS.Src[i].Tag = Regs[I.Src[i]].Tag  // Copy the tags from the RF
                    RS.Src[i].Ready = False
        (f) Regs[I.Dest].Tag = unique tag id            // tag (rename) the source
        (g) RS.DestRegTag = Regs[I.Dest].Tag
        (h) Regs[I.Dest].Ready = False                  // value ready only after execution

    else exit loop // stop dispatching if scheduling queue is full
```

Tomasulo algorithm, part 2

[*Scheduling unit*]
for each RS = entry in SchedQueue do:
    (a) for all source registers, i, of RS do:
        if (CDB.Tag = RS.Src[i].Tag) then // *broadcast on CDB matches stored tag*
            RS.Src[i].Ready = True
            RS.Src[i].Value = CDB.Value
    (b) if ((for all i: RS.Src[i].Ready = True) // WAKEUP: *If all registers are ready...*
        and Scoreboard[RS.FU].Busy = False) then
            Scoreboard[RS.FU].Busy = True // *...reserve the FU and issue*
            Issue the instruction at RS *on* FU

step (b) is called "wake-up"
when more than one instruction is ready, picking between them is called "select" (here we're using FIFO as a select algorithm)

Tomasulo Algorithm, part 3

[*Execution unit, for function unit* FU]
    if (FU *pipeline advances and now first stage is free*) then Scoreboard[FU].Busy = False
[*Execution unit, at completion of instruction* I]
    if (CDB.Busy = False) then // *Broadcast the instruction results*
        (a) CDB.Busy = True
        (b) CDB.Tag = I.Tag
        (c) CDB.Value = I.Value
        (d) CDB.Reg = I.Dest
        (e) if (I.FU *is not pipelined*) then
            Scoreboard[I.FU].Busy = False
        (f) *Delete* I *from* Sched Queue

[*State update unit*]
    if (CDB.Busy = True)
        (a) CDB.Busy = False
        if (CDB.Tag = Regs[CDB.Reg].Tag) then
            (a) Regs[CDB.Reg].Ready = True // *Update the RF contents*
            (b) Regs[CDB.Reg].Value = CDB.Value

Tomasulo and Speculation

- This approach does not handle speculation
  - I.e. branch prediction

- Add a reorder buffer
  - CDB still updates reservation stations, but not RF
  - New stage "**commit**"
  - Oldest instruction(s) update RF

ROB Designs

- Future file (Smith / Pleszkun '88)
  - "messy" register file updated by CDB
  - Architectural register file updated by ROB

- Checkpoint Repair (Patt / Hwu '88)
  - Pick one or more instruction points / barriers (IB)
  - "messy" updated by CDB
  - Other RF are updated by CDB if older than IB
  - On exception, copy oldest RF to architectural file
  - If everything older than IB successful, then copy to older RF