Midterm II Review

Yizhou Wang

<wangyzh2024>

Digital Circuit & Simple Datapath

- Digital design starting from scratch
- Design the state machine (Mealy/Moore)
- Write down the truth table
- Simplify Boolean formulas (Boolean algebra laws)
- Implement with basic gates (AND, OR, NOT, NAND, NOR, XOR, …)
- Exercise: modulo-3 machine based on 2-input NAND
  - 1-bit stream input, 2-bit remainder output

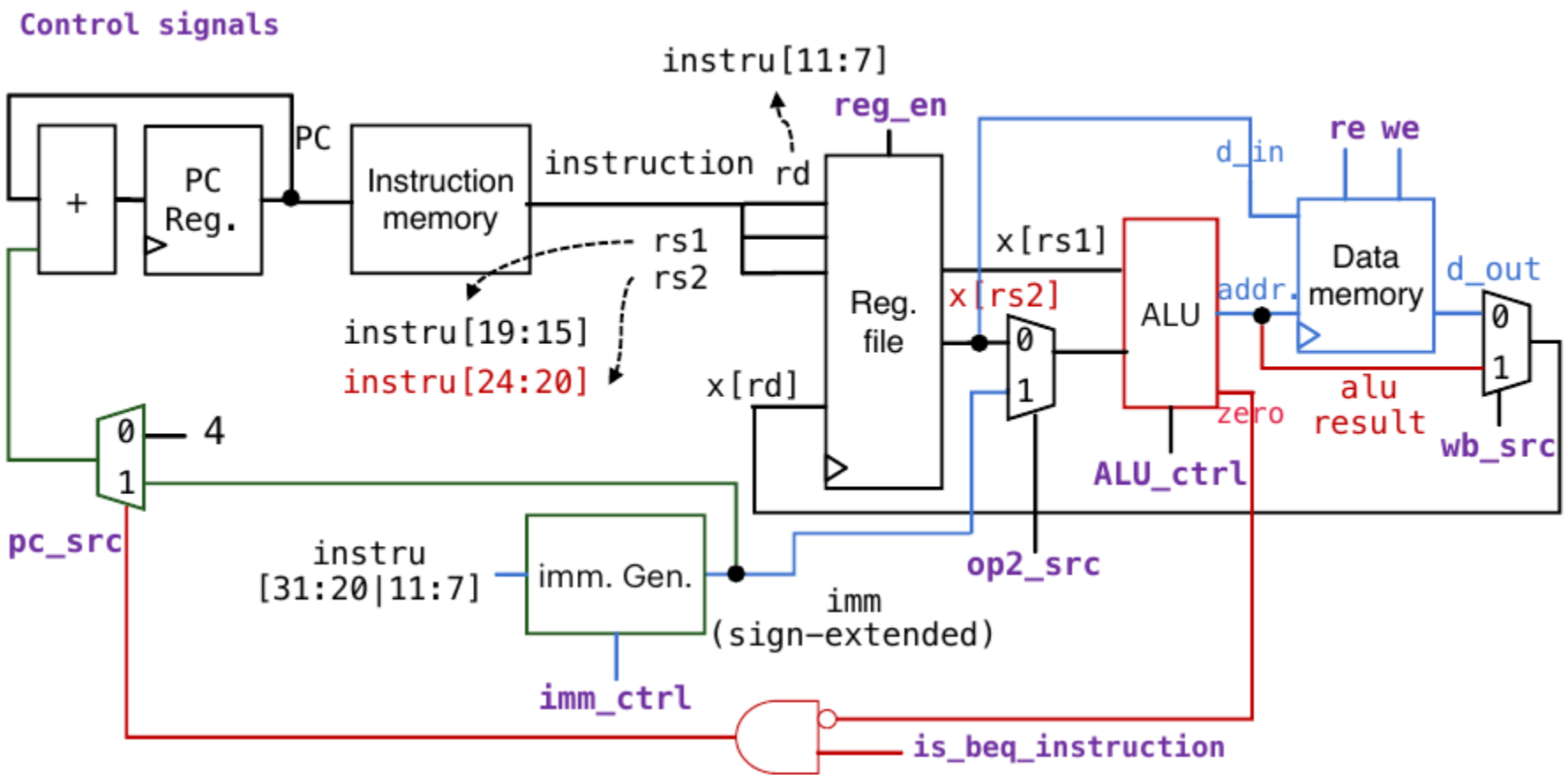
Timing (Setup)

- Reg’s Q -> Combinational -> Reg’s D
- Critical path sets a lower bound of clock period
- $t_{clk\text{-}to\text{-}q} + t_{comb} <= T_{period} - t_{setup}$

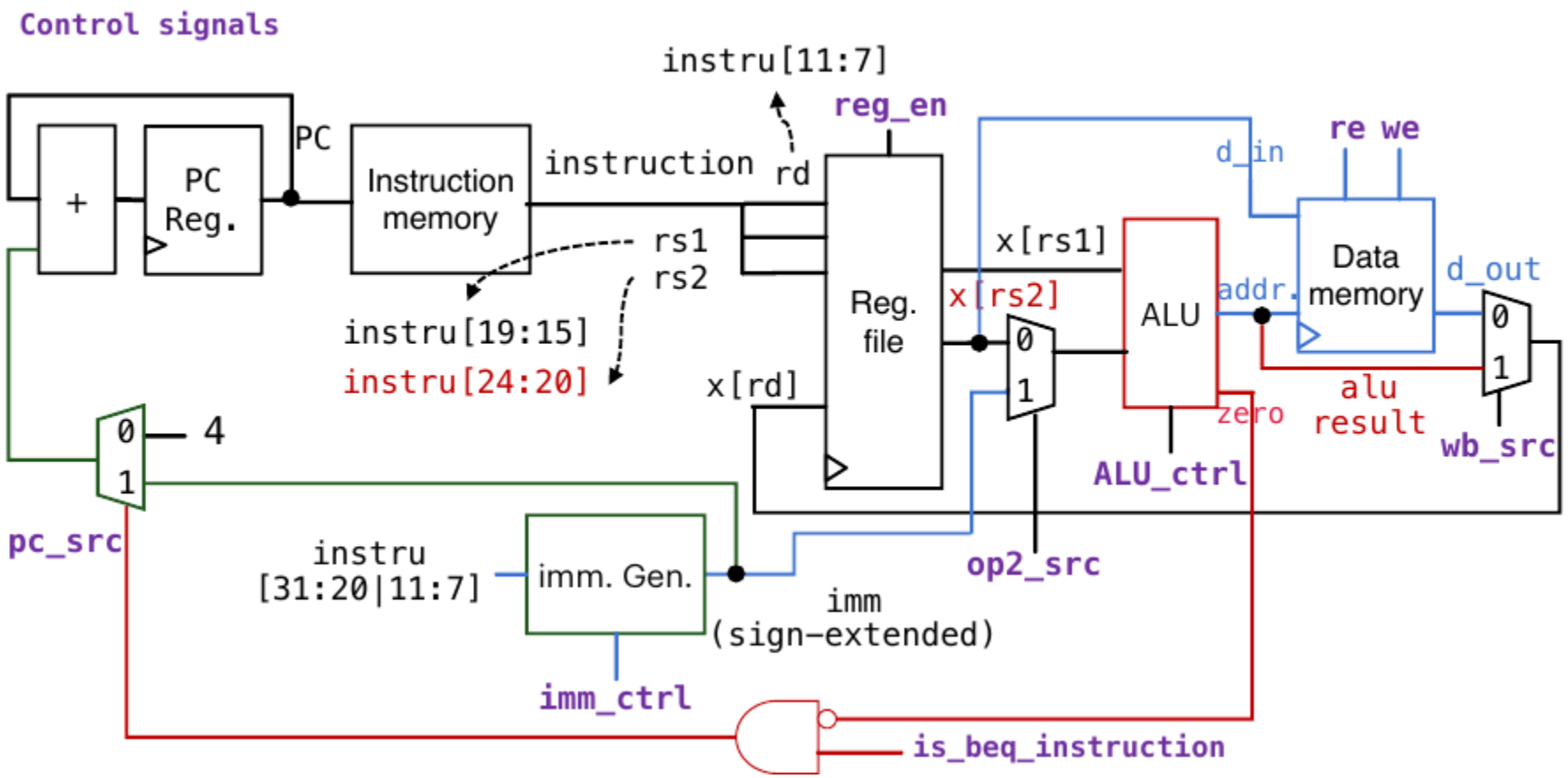
Datapath & Controller

- Datapath serves as “railway tracks” of data
  - Regfile, ALU, Memory, Imm
- Controller controls “turnouts”
  - Should data be written back? Which data should be written to Rd?
  - Which data sources should be selected for ALU operands?
  - Which format should the immediate value be extracted from?
  - And so many other decisions/switches

R-type, I-type, lw, sw, beq



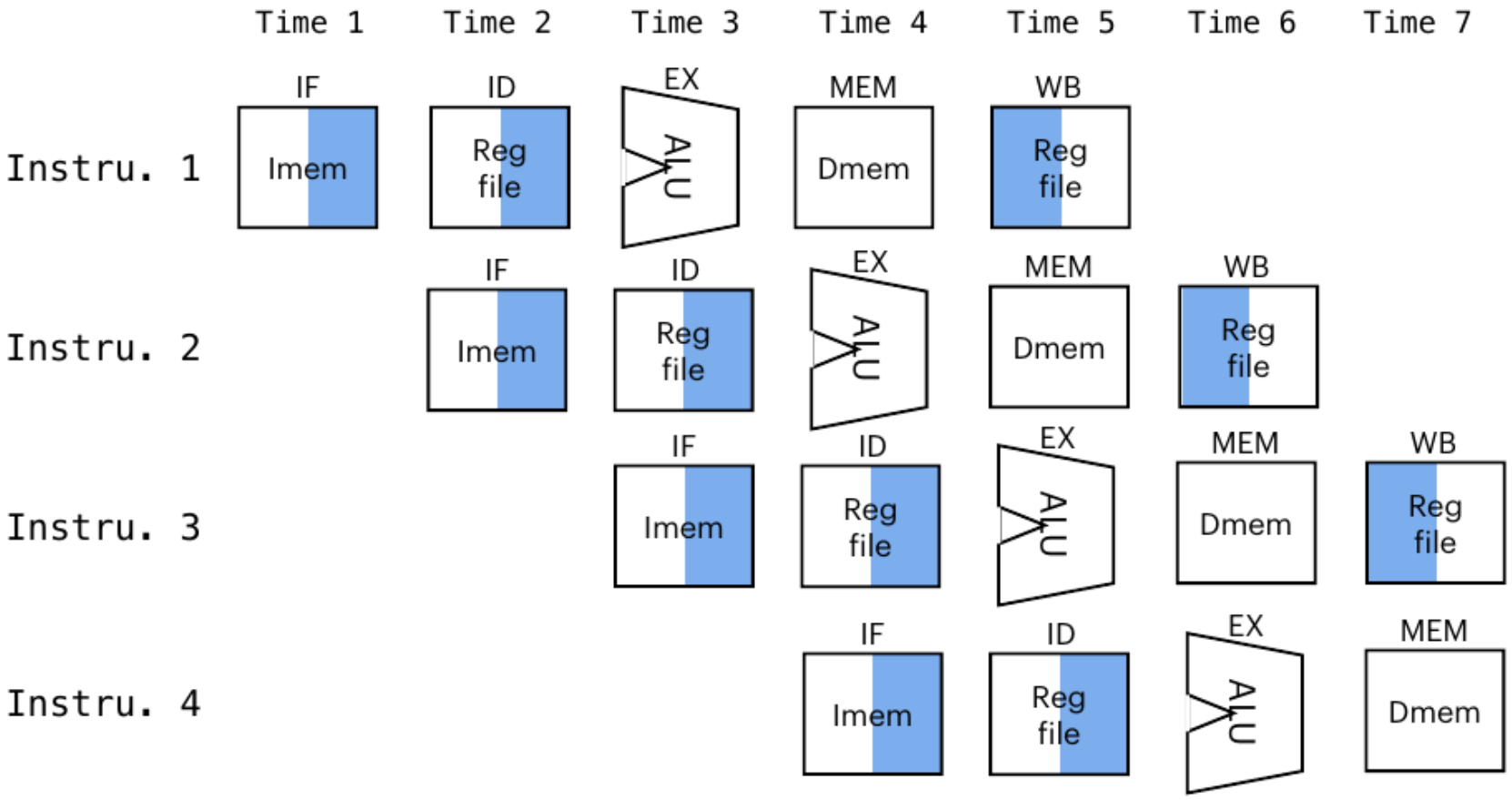
Datapath Timing



Instruction-Level Parallelism

- Iron Law:
- $\frac{Time}{Program} = \frac{Instructions}{Program} \times \frac{Cycles}{Instruction} \times \frac{Time}{Cycle}$
- Instr. per program determined by ISA, compilers and programmers
- Digital circuit design limits min. time per cycle
- Wanna reduce cycles per instruction!
  - Improves throughput, i.e., more instructions executed within some cycles
  - Unchanged or worse latency, i.e., one instruction alone is not made faster.

Classic 5-stage Pipeline



Hazards

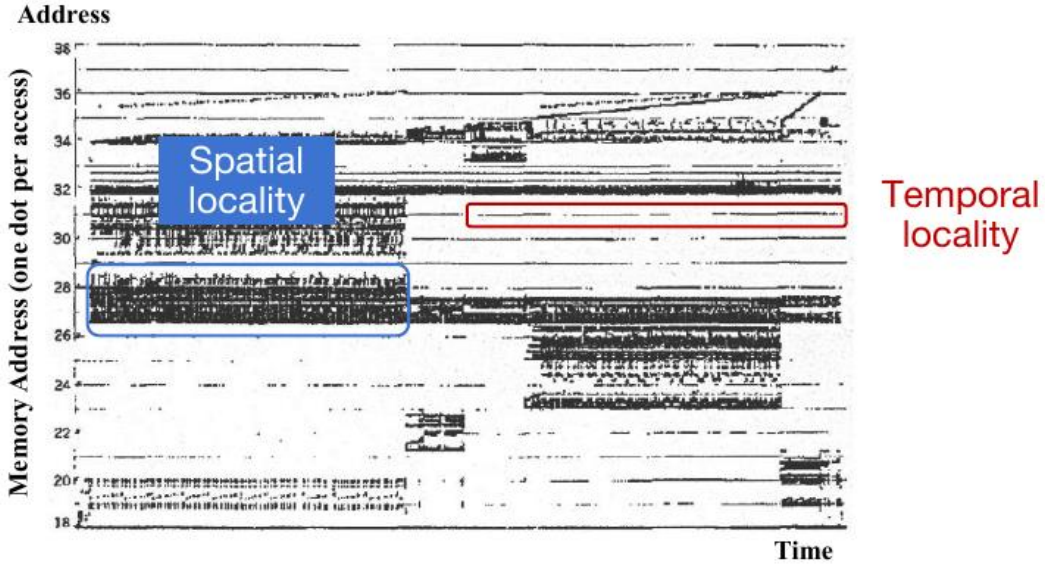
- A situation in which a planned instruction cannot execute in the proper clock cycle (perfect pipelining fails)
- Structural: datapath unit not available
  - Memory unit shared for both data and instruction (IF and MEM)
  - Regfile that does not support simultaneous read and write (ID and WB)
- Data: RAW, WAW, WAR
  - Forwarding may not always solve it (lw then add)
- Control: next PC not fully determined
  - Stalling, speculation, forwarding

Multi-issue

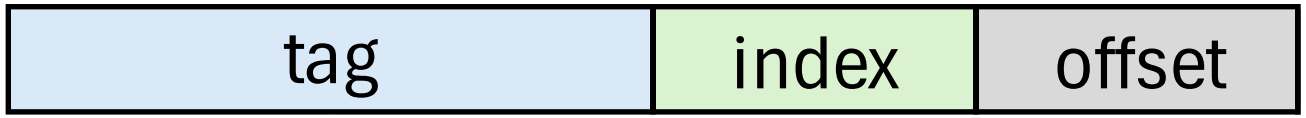
- Static: Very Long Instruction Word
  - Compilers or programmers carefully pack multiple instructions to be executed in the same cycle and avoid hazards.
- Dynamic: Superscalar
  - Hardware packs instructions and avoids hazards.
- E.g., each cycle has an arithmetic slot and a memory slot
- Require more resources to avoid structural hazard
- Orthogonal to multithreading, SIMD and pipelining

Cache

- Make use of locality to provide fast memory access
  - Temporal locality
    - Ref this addr again soon
  - Spatial locality
    - Ref near this addr soon
- If missed, then turn to accessing main memory
- Basic unit to fetch and replace is cache **line/block**

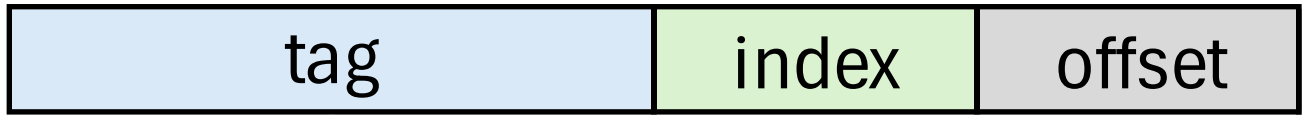


Direct Mapped



- Each memory address is mapped to a fixed cache line
- Index field locates the cache entry
- Offset field locates the byte inside the entry
- Tag used to distinguish addresses mapped to the same line
- On conflict, replacement is involved

Set Associative



- Cache is divided into sets and each sets have multiple slots
- Each address is mapped to a fixed set but may reside in *any* slots.
- Index field locates the set
- Offset field locates the byte inside the entry
- Tag used to distinguish addresses mapped to the same set and determine in which slot the address reside.
- On all slots used, replacement is involved to replace one slot

Fully Associative

- The whole cache is essentially a set of slots
- Each address may reside in *any* slots.
- No need of index field
- Offset field locates the byte inside the entry
- Tag used to distinguish addresses and determine in which slot the address reside.
- On all slots used, replacement is involved to replace one slot

Replacement Policy

- LRU: utilize temporal locality but complex in hardware
- MRU: easy hardware implementation
- FIFO, LIFO: reasonable approximation
- Random

Write Policy

- Write-back vs. write-through
  - Write dirty on eviction vs. write to memory
  - Whether a hit write updates the main memory
  - Averagely shorter but variable latency vs. fixed longer latency
- Write-allocate vs. no-write-allocate (write-around)
  - Whether a write allocates a cache line on miss
  - No-write-allocate implies write-through (must write to memory)

Cache Implementation

- How many bits for one entry?
  - General: data, tag, valid
  - State bit(s) for replacement policy
  - Dirty bit for write-back policy
- How many digital circuit components required?
  - Direct Mapped: MUX for set and one {tag,valid} comparator
  - Fully Associative: A great many {tag,valid} comparator
  - Set Associative: MUX for set and N {tag,valid} comparator for N-way