CS 5234 –Spring 2013
Advanced Parallel Computing

Architecture

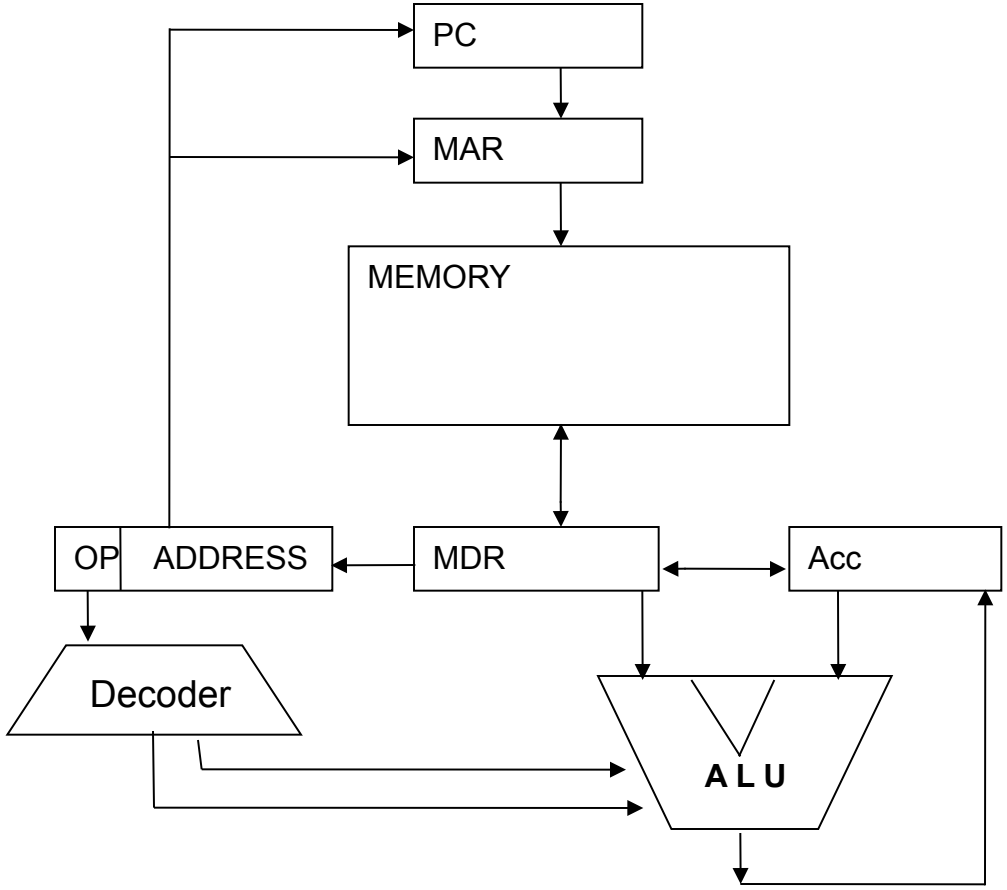
Yong Cao

Goals

- Sequential Machine and Von-Neumann Model
- Parallel Hardware
  - Distributed vs Shared Memory
- Architecture Classes
  - Multiple-core
  - Many-core (massive parallel)
- NVIDIA GPU Architecture

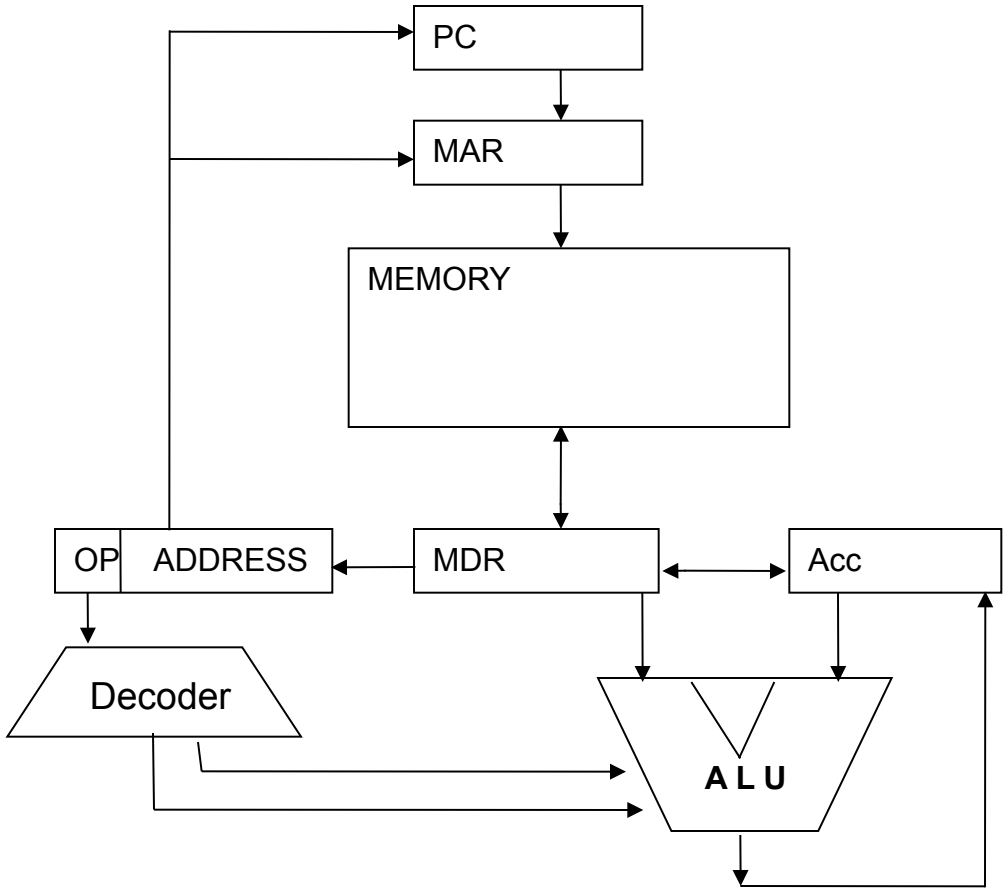
Von-Neumann Machine (VN)

- PC: Program counter
- MAR: Memory address register
- MDR: Memory data register
- IR: Instruction register
- ALU: Arithmetic Logic Unit
- Acc: Accumulator



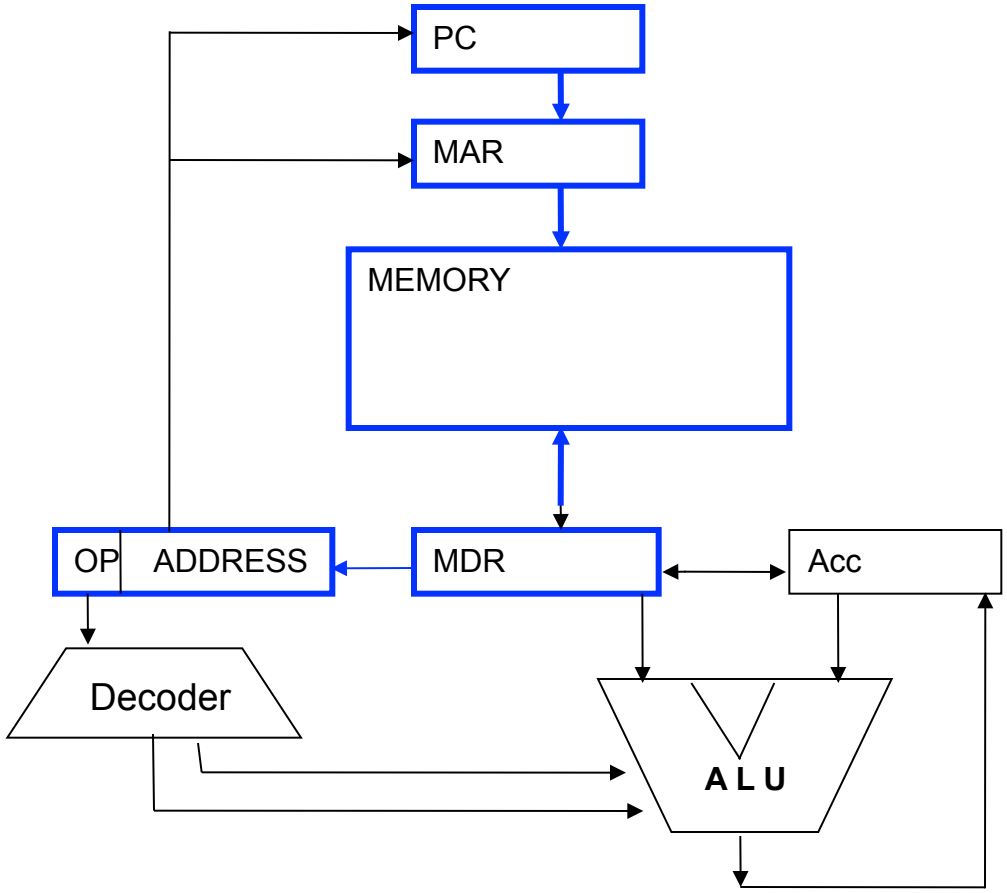
Sequential Execution and Instruction Cycle

- The six phases of the instruction cycle:
  - Fetch
  - Decode
  - Evaluate Address
  - Fetch Operands
  - Execute
  - Store Result



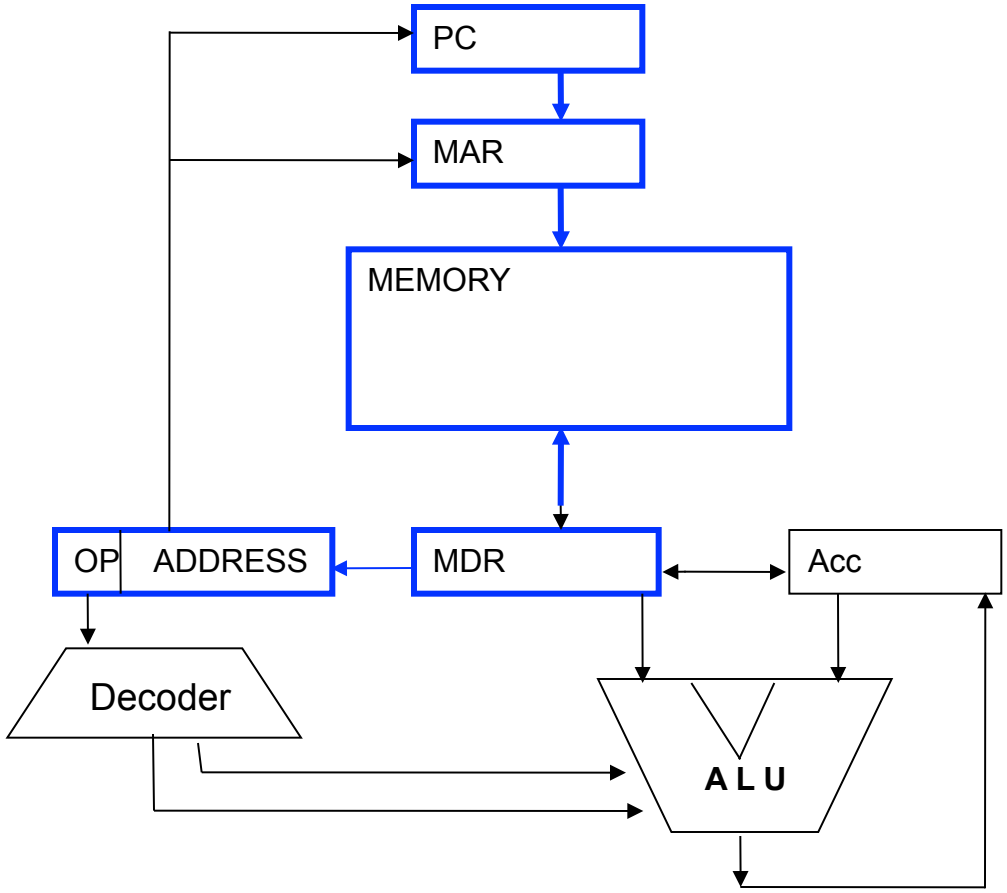
Sequential Execution and Instruction Cycle

- Fetch
  - MAR←PC
  - MDR←MEM[MAR]
  - IR←MDR



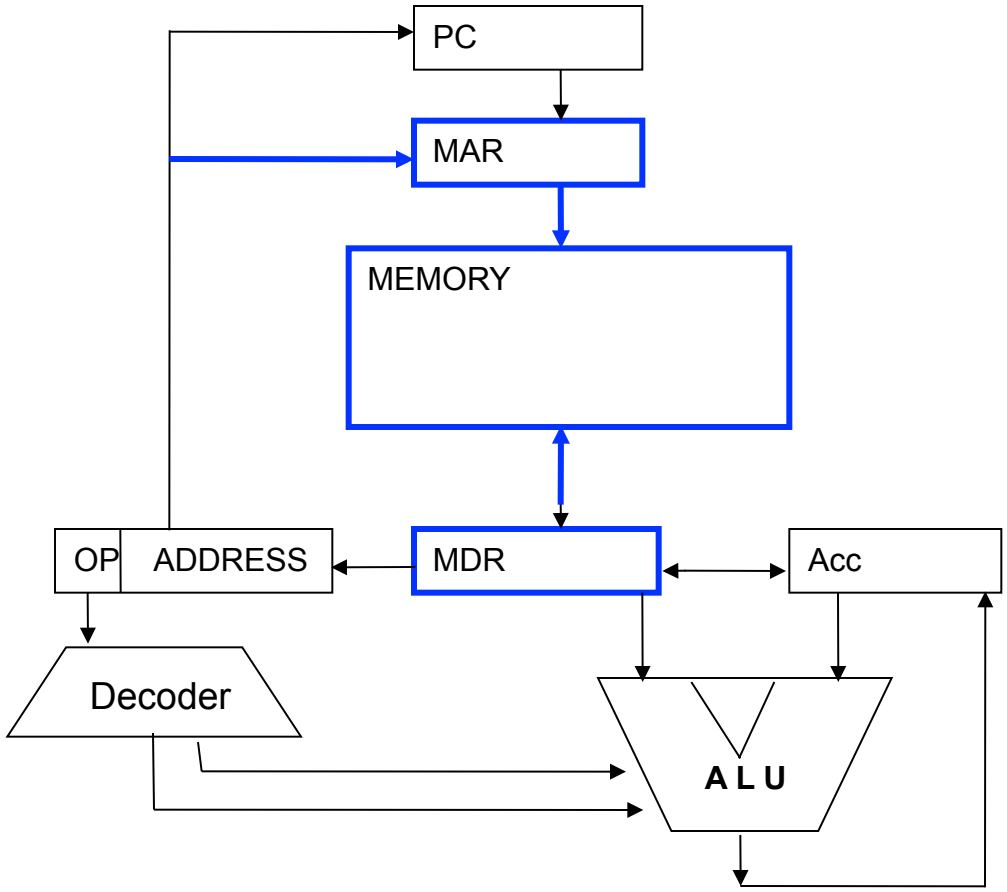
Sequential Execution and Instruction Cycle

- Decode
  - DECODER←IR.OP



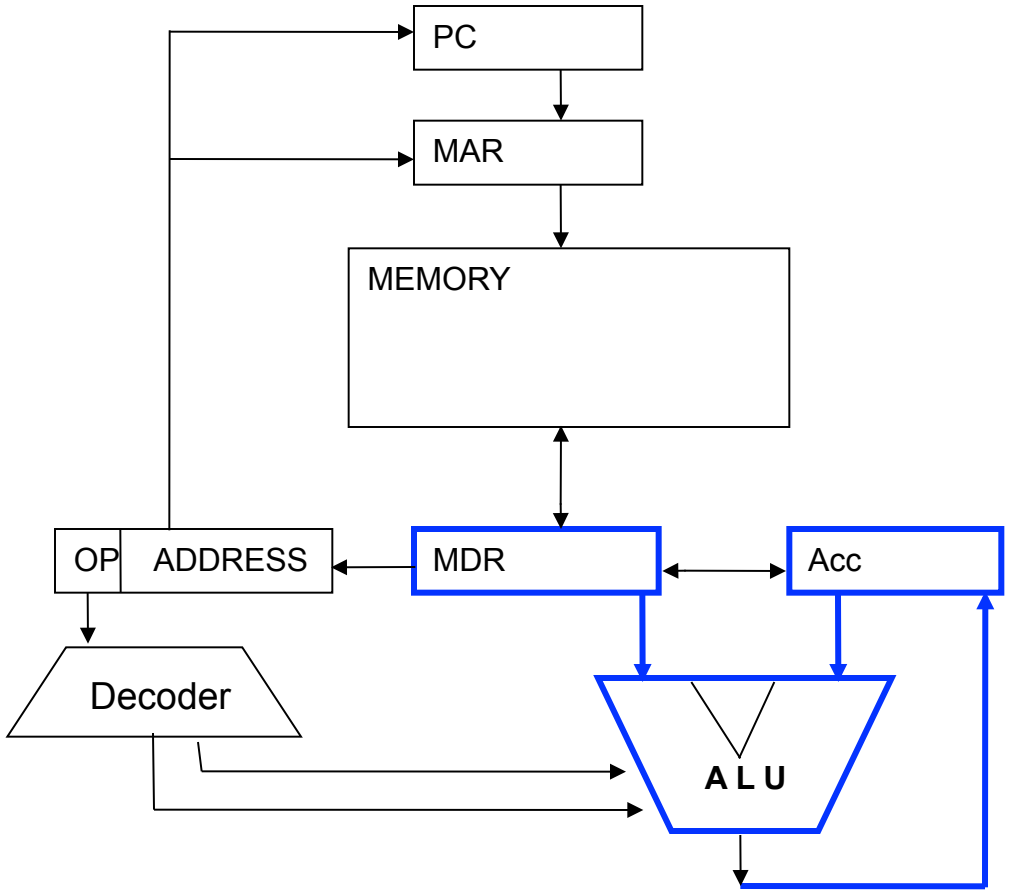
Sequential Execution and Instruction Cycle

- Evaluate Address
  - MAR←IR.ADDR
  - MDR←MEM[MAR]



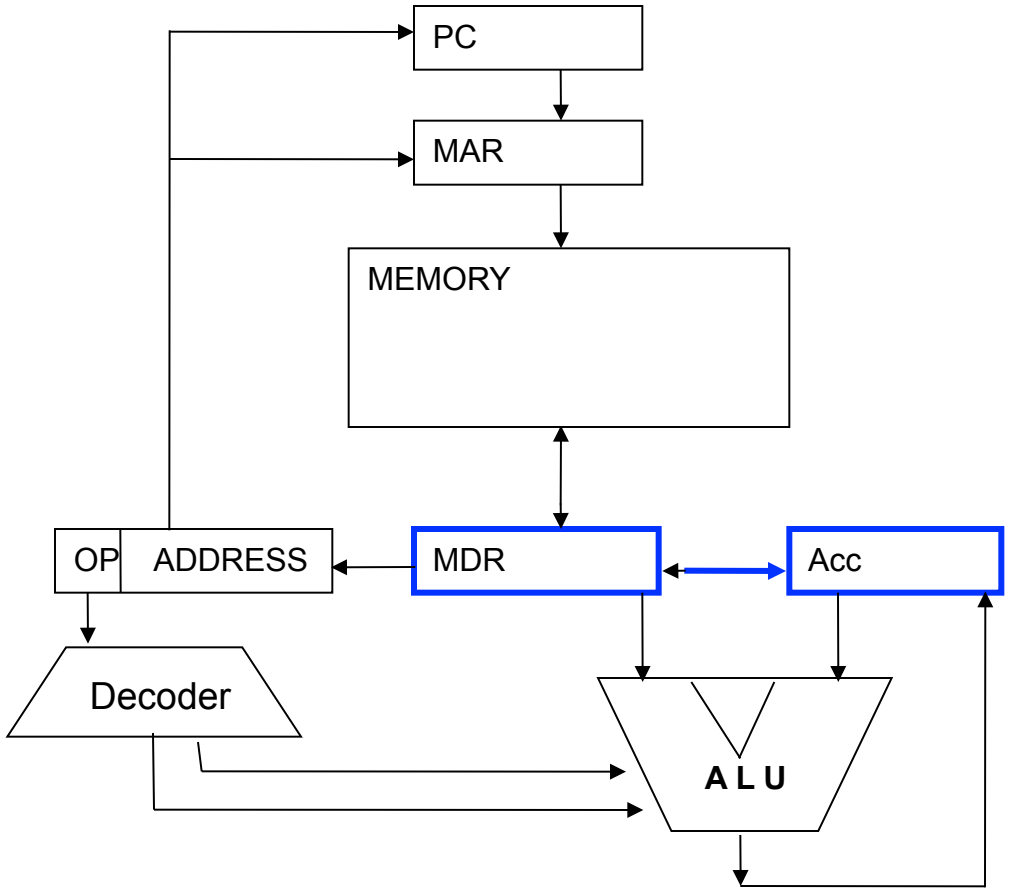
Sequential Execution and Instruction Cycle

- Execute
  - Acc ←Acc + MDR



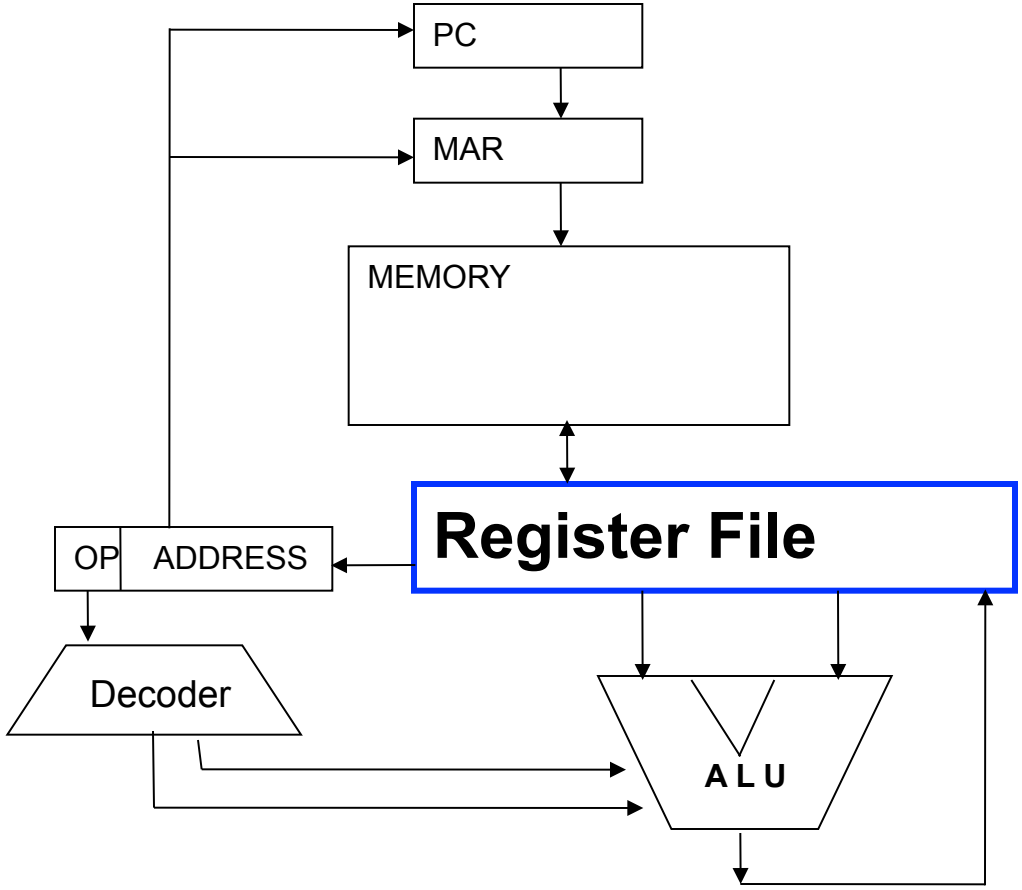
Sequential Execution and Instruction Cycle

- Store Result
  - MDR←Acc

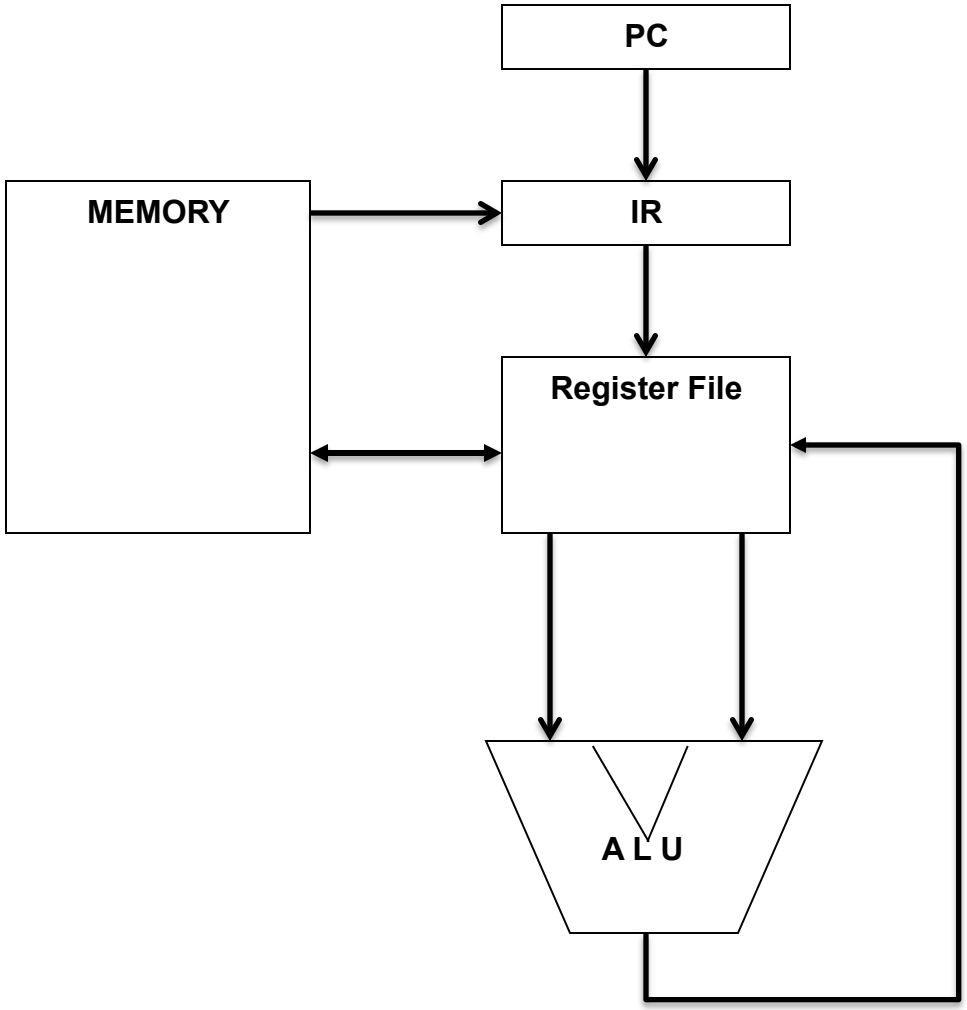


Sequential Execution and Instruction Cycle

- Register File

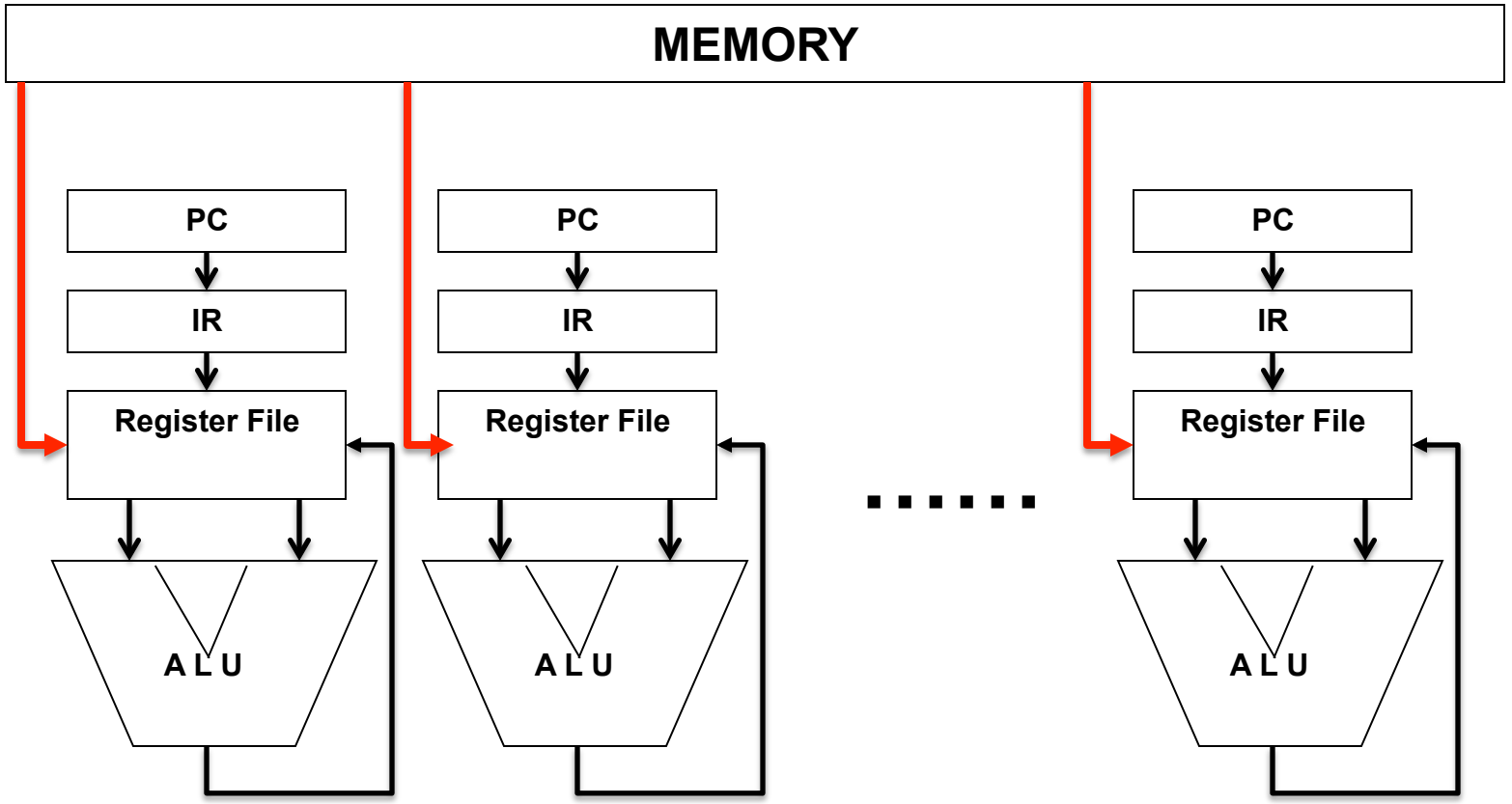


Sequential Execution and Instruction Cycle



Parallel Hardware

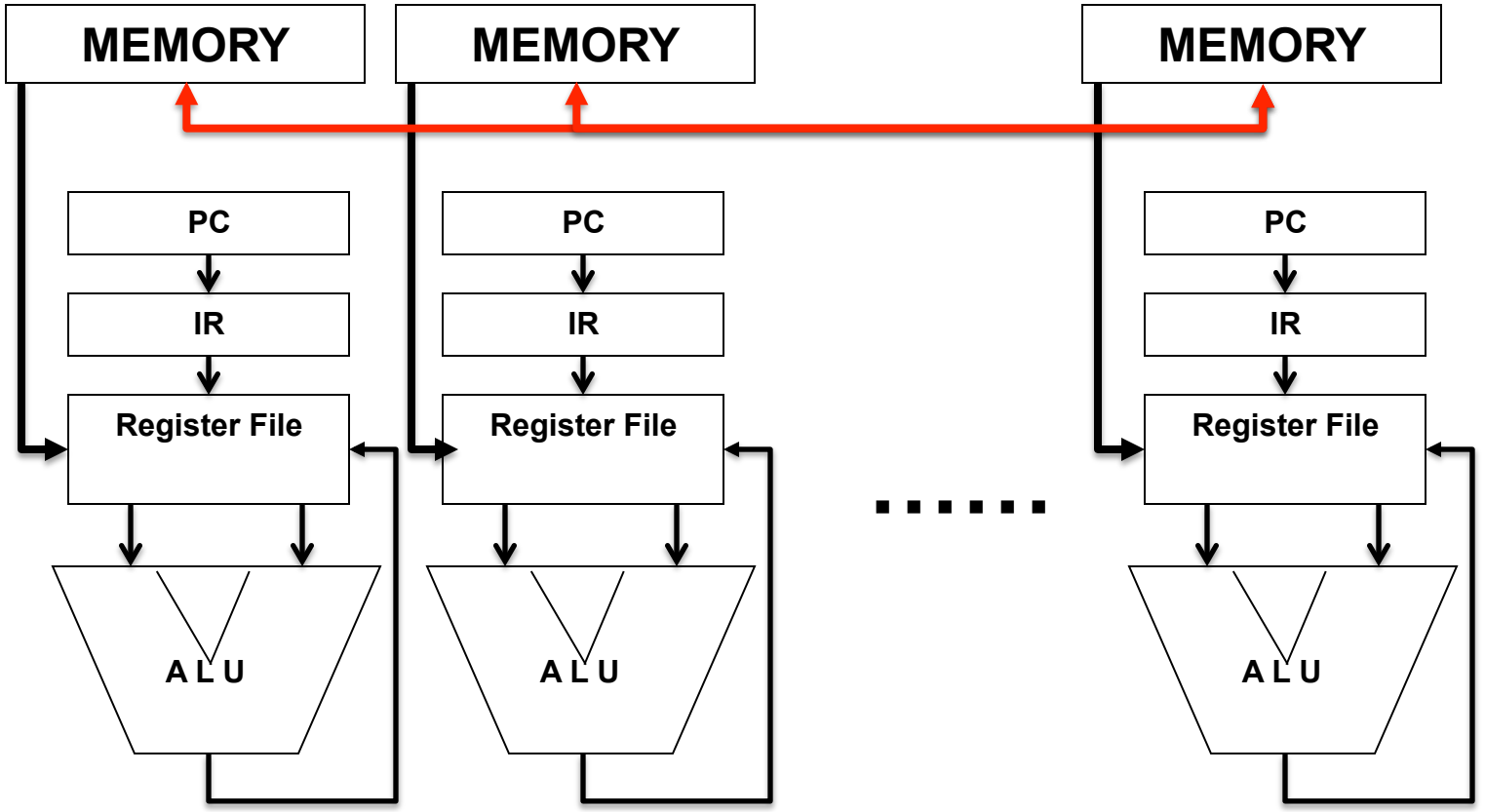
- Shared vs Distributed Memory



- Multi-Core and Many-Core Architecture

Parallel Hardware

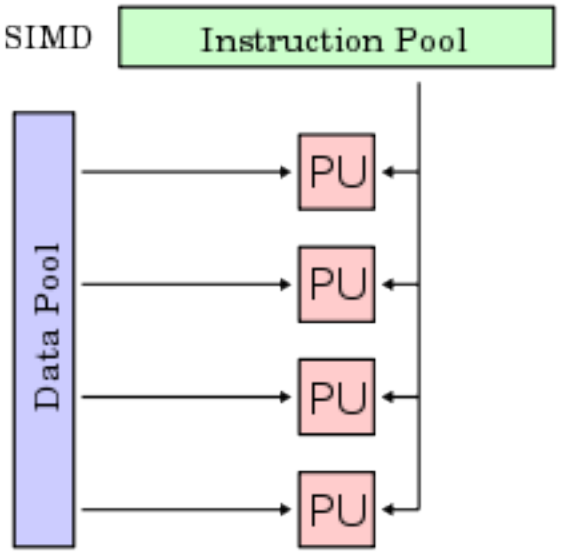
- Shared vs Distributed Memory



- Cluster Computing, Grid Computing, Cloud Computing

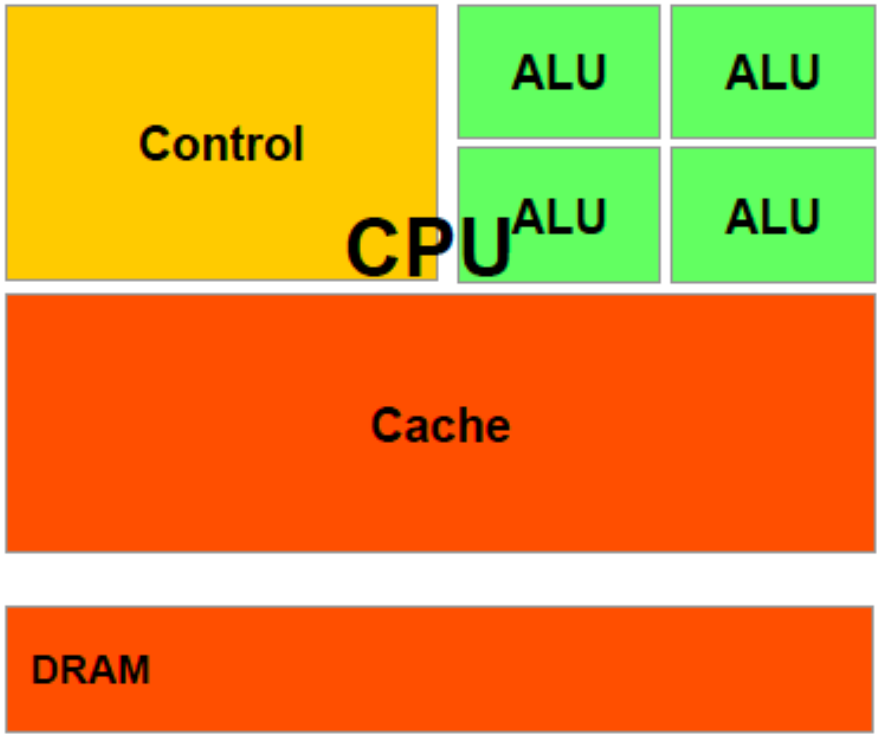
Multi-Core vs Many-Core

- Definition of Core – Independent ALU
- How about a vector processor?
  - SIMD: E.g. Intel's SSE.
- How many is "many"?
  - What if there are too "many" cores in the Multi-core design?
    Shared control logic (PC, IR, Schedule)



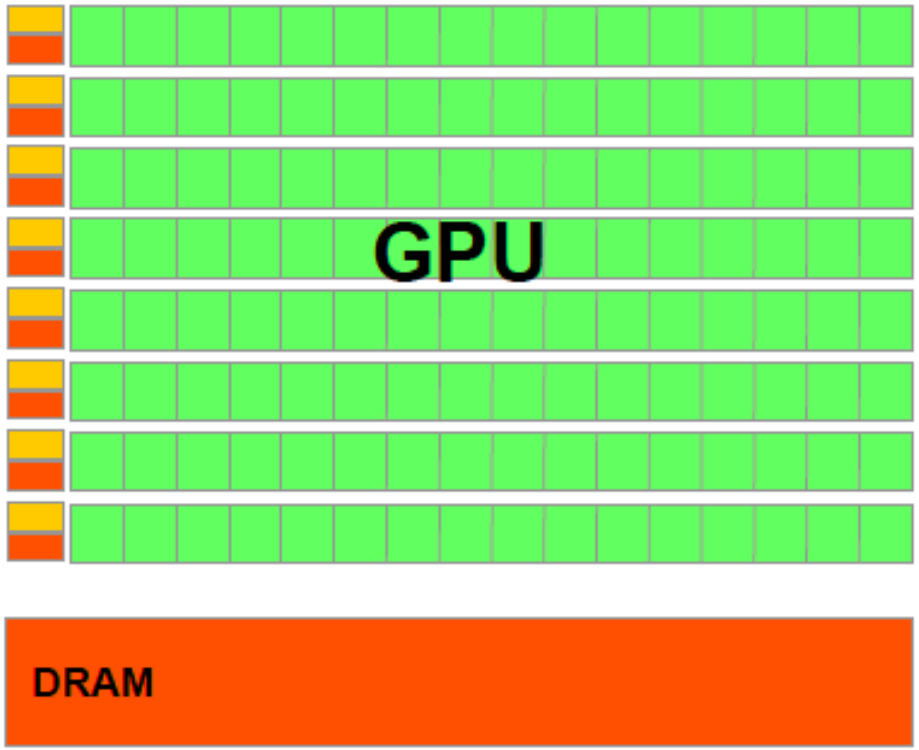
Multi-Core

- Each core has its own control (PC and IR)

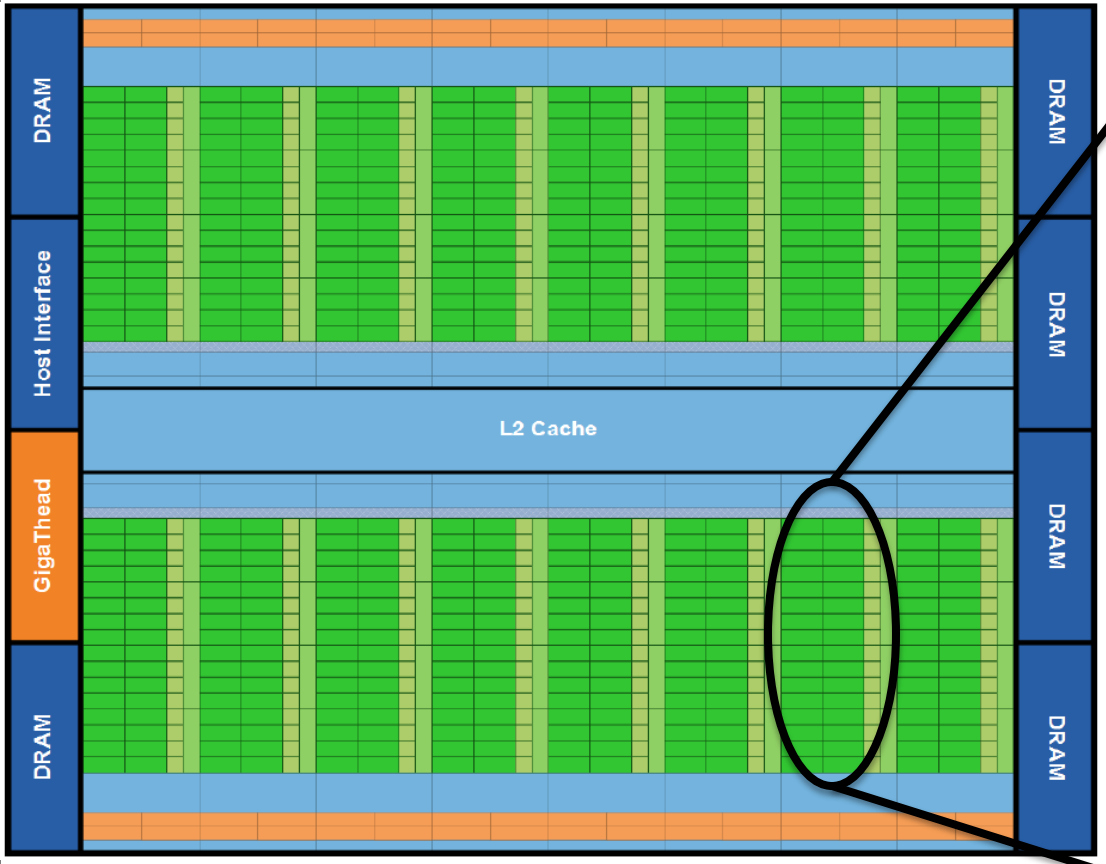


Many-Core

- A group of cores shares the control (PC, IR and Thread Scheduling)

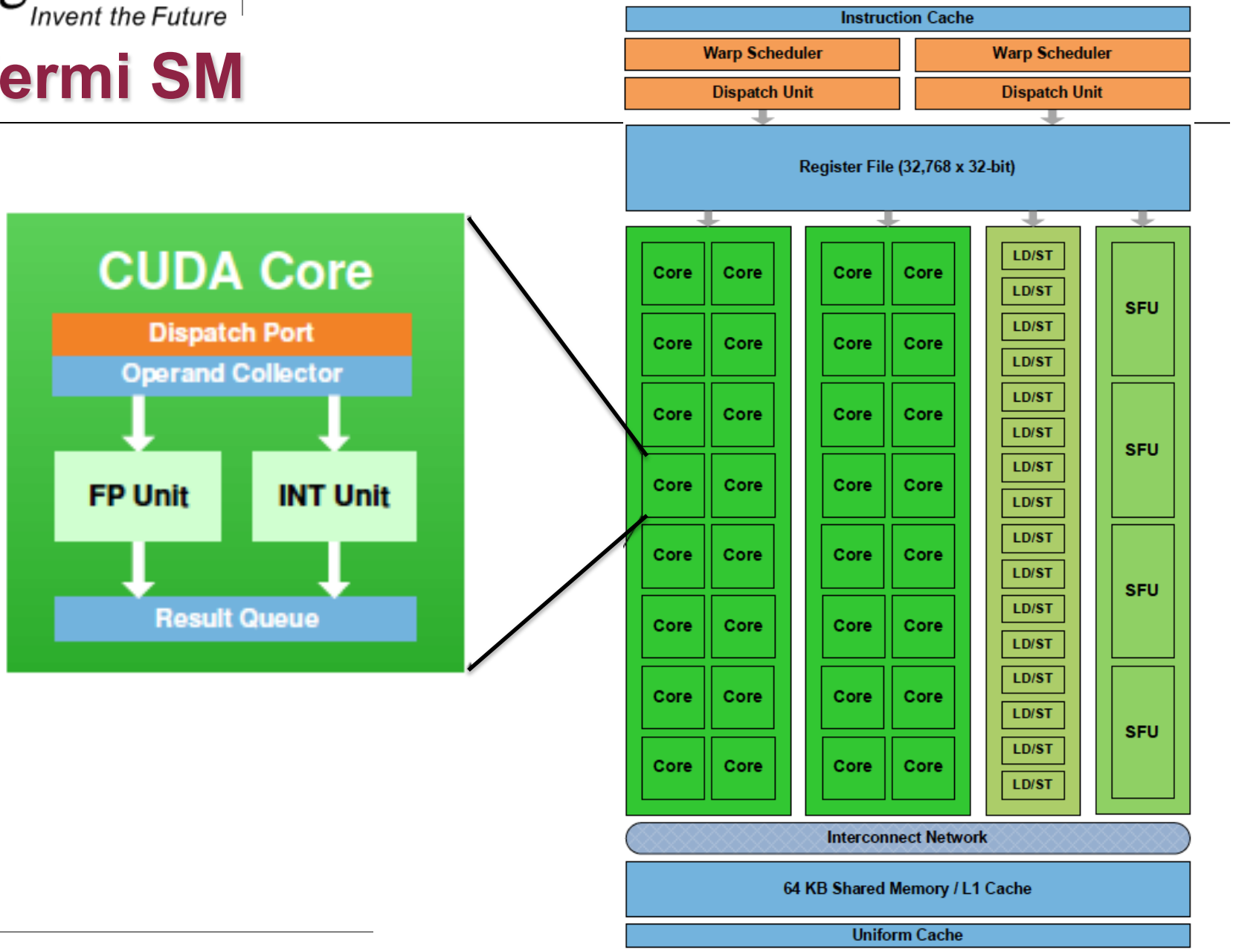


NVIDIA Fermi Architecture

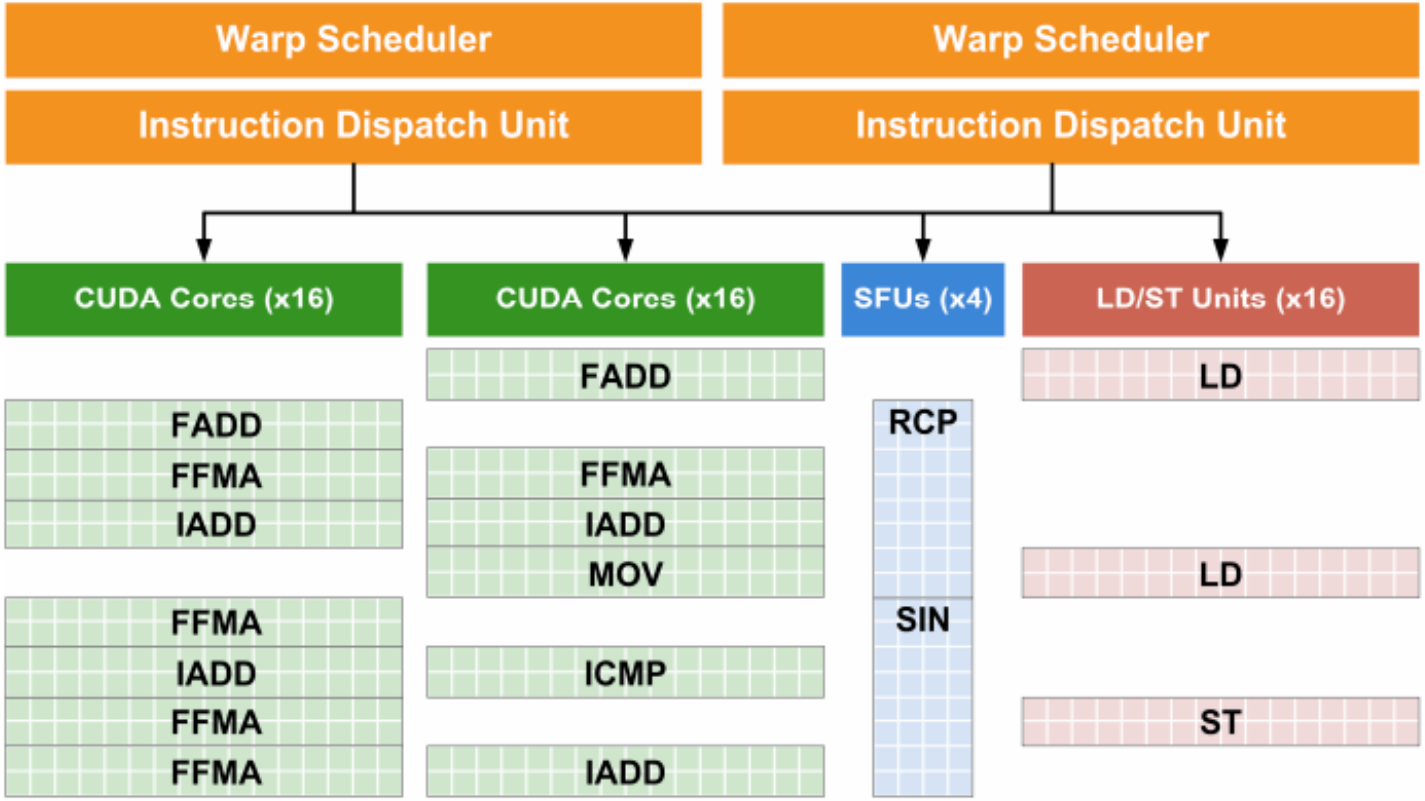


16 Stream Multiprocessor (SM)
32 Core for Each SM

Fermi SM


CUDA Core
Dispatch Port
Operand Collector
FP Unit
INT Unit
Result Queue
Instruction Cache
Warp Scheduler
Warp Scheduler
Dispatch Unit
Dispatch Unit
Register File (32,768 x 32-bit)
Core
LD/ST
SFU
Interconnect Network
64 KB Shared Memory / L1 Cache
Uniform Cache

Execution in a SM



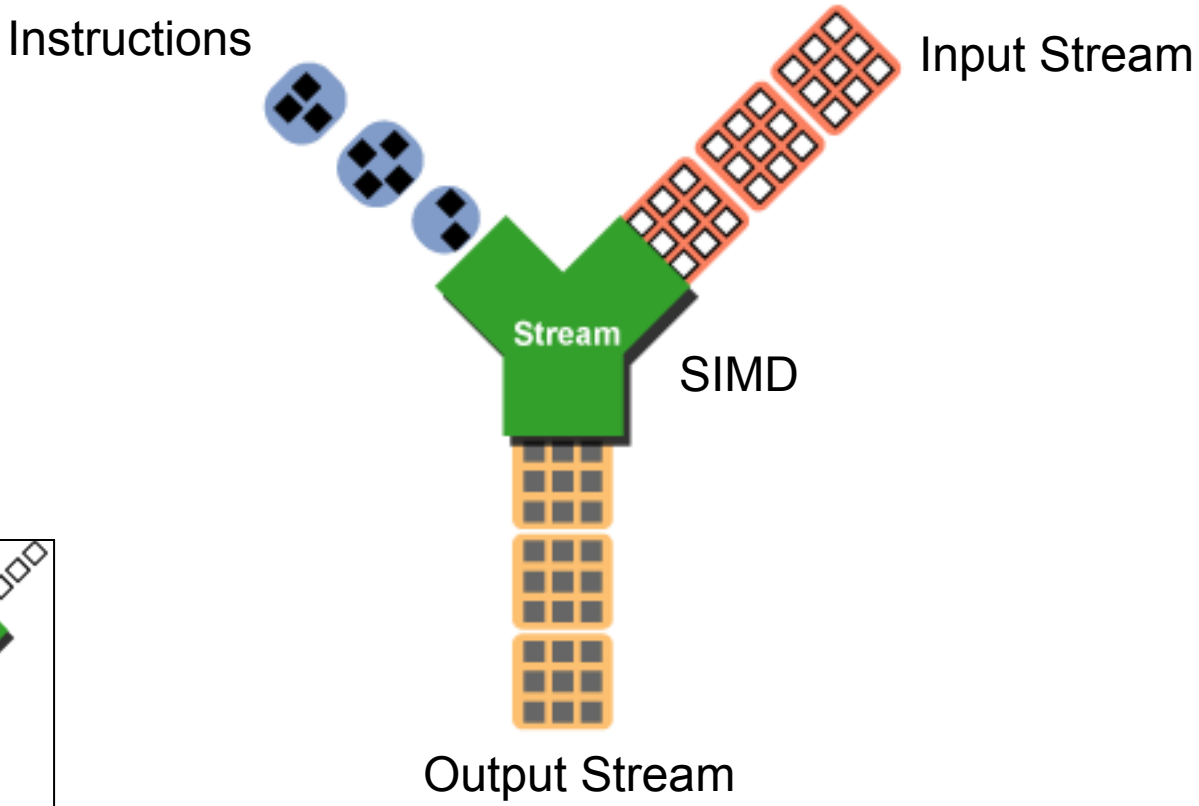
A total of 32 instructions from one or two warps can be dispatched in each cycle to any two of the four execution blocks within a Fermi SM: two blocks of 16 cores each, one block of four Special Function Units, and one block of 16 load/store units. This figure shows how instructions are issued to the execution blocks.

Data Parallel

- Data Parallel vs Task Parallel
  - What to partition? Data or Task?
- Massive Data Parallel
  - Millions (or more) of threads
  - Same instruction, different data elements

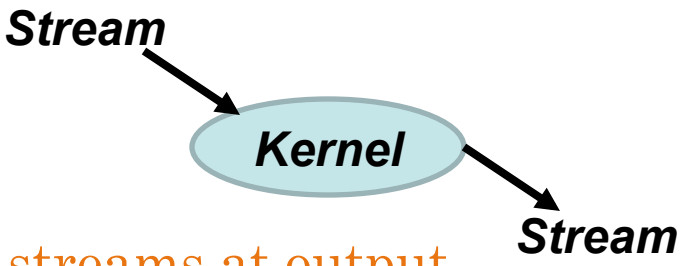
Computing on GPUs

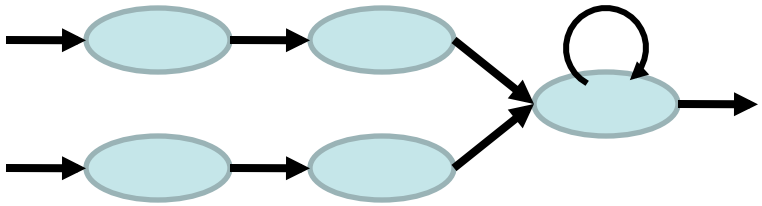
- Steam processing and Vectorization (SIMD)



GPU Programming Model: Stream

- Stream Programming Model
- Streams:
  - An array of data units
- Kernels:
  - Take streams as input, produce streams at output
  - Perform computation on streams
  - Kernels can be linked together





Why Streams?

- Ample computation by exposing parallelism
  - Stream expose data parallelism
    - Multiple stream elements can be processed in parallel
  - Pipeline (task) parallelism
    - Multiple tasks can be processed in parallel
- Efficient communication
  - Producer-consumer locality
  - Predictable memory access pattern
    - Optimize for throughput of all elements, not latency of one
    - Processing many elements at once allows latency hiding