CS6560: Parallel Computer Architecture

Recap: Basics of Computer Architecture

Madhu Mutyam
PACE Laboratory
Department of Computer Science and Engineering
Indian Institute of Technology Madras

Jan 15-30, 2018

Course Objectives

- Understanding the process of parallel programming and exploiting underlying architecture to improve performance
- Understanding the fundamentals of architecture and design of parallel computers

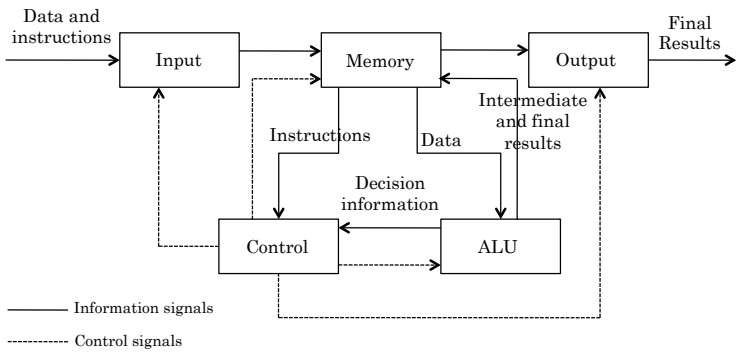
Course Syllabus

- Recap: Basics of Computer Architecture
- Introduction to Parallel Computer Architecture
- Basics of Parallel Programming
- Performance Issues in Parallel Programs
- Workload-Driven Performance Evaluation
- Shared Memory Multiprocessors
- Memory Consistency
- Snoop-based Cache Coherence
- Synchronization
- Snoop-based Multiprocessor Design
- Scalable Multiprocessors
- Directory-based Cache Coherence
- Interconnection Networks

Resources and Grading Policy

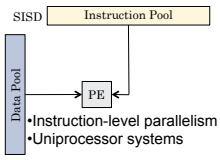
- Reference Books/Material:
  - D.E. Culler, et. al. Parallel Computer Architecture – A Hardware/Software Approach. Morgan Kaufmann, 2010.
  - D.J. Sorin, et. al. A Primer on Memory Consistency and Cache Coherence. Morgan & Claypool, 2011.
  - N.E. Jerger and L.S. Peh. On-Chip Networks. Morgan & Claypool, 2009.
- Evaluation Mechanism:
  - Assignments: 25%
  - Mid Semester Exam (6/3/18): 25%
  - End Semester Exam (2/5/18): 50%

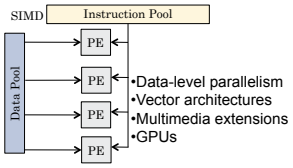
Von Neumann Model of a Computer

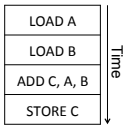


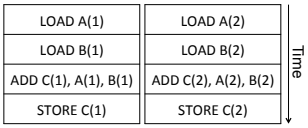
Computers and Processors

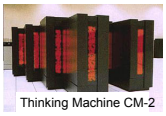
Flynn's Classification



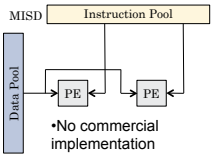


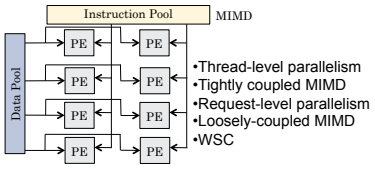


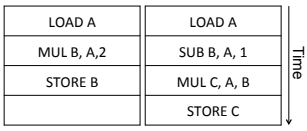


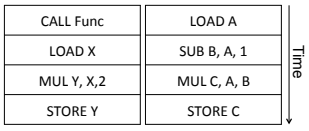


Flynn's Classification (Contd)









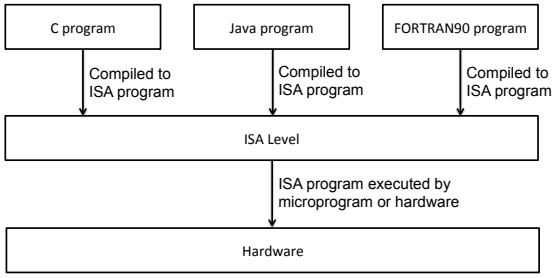
- Suitable for fault-tolerant real-time computers

Computer Architecture

- Instruction set architecture (ISA)
- Microarchitecture
- Implementation

The Role of ISA

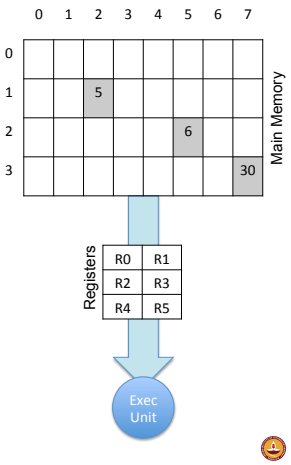
- Specifies the functionality of a processor
- Provides the interface between the compilers and hardware



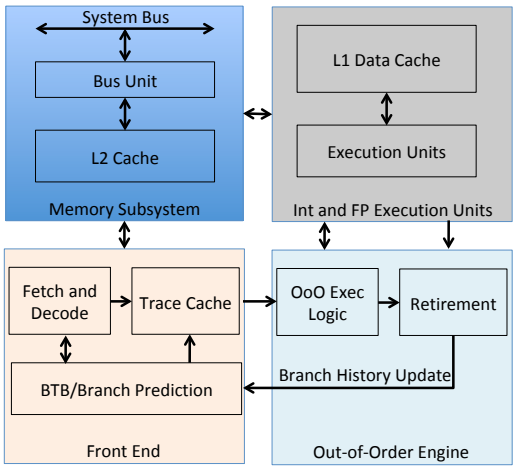
CISC Vs RISC

- CISC: Complex Instruction Set Computer
  - Uses multi-word instructions
  - Supports the operations and data structures used by high-level language
  - Supports memory/register addressing
  - Example ISA: x86
  - `MULT [3,7], [1,2], [2,5]`
- RISC: Reduced Instruction Set Computer
  - Uses one-word instructions
  - Uses processor registers extensively and operands are from registers only
  - Uses register-based addressing
  - Example ISA: ARM

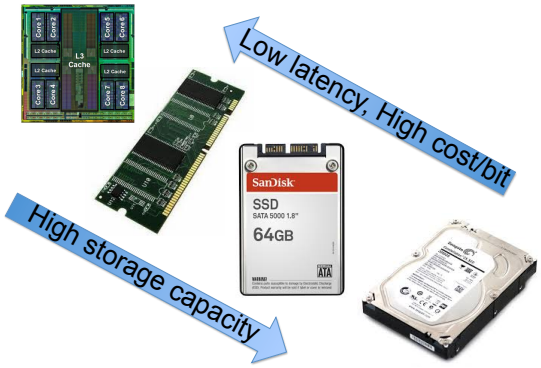
```
LOAD R1, [1,2]
LOAD R2, [2,5]
MULT R3, R1, R2
STORE [3,7], R3
```



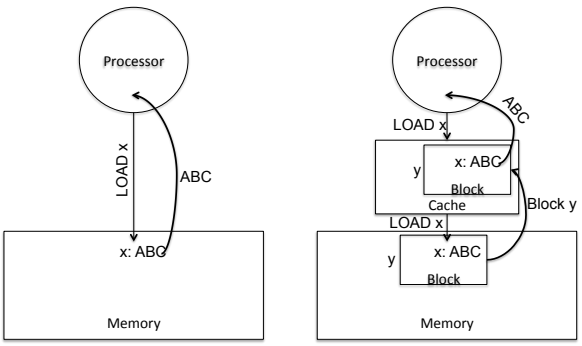
Microarchitecture



Memory Hierarchy

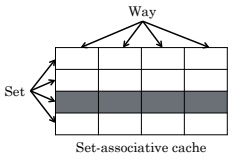


Cache Memory

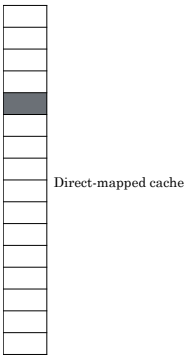


- Works on principle of locality
- Designed using SRAM-based technology
- Cache block is typically 32B or 64B

Cache Memory Organization

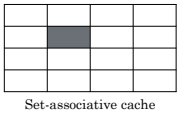


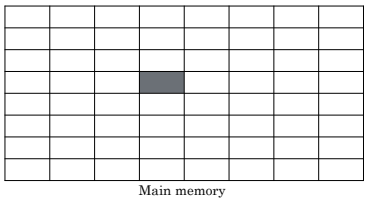
Mapping: (Block address) MOD (# of sets in the cache)



Mapping: (Block address) MOD (# of blocks in the cache)

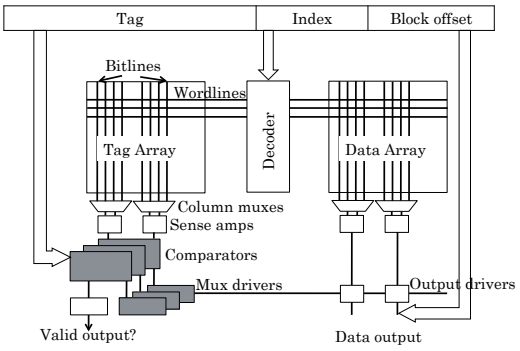
Finding a Block in Cache Memory





Block address		
Tag	Index	Block offset

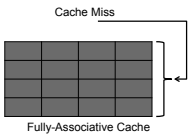
Cache Access

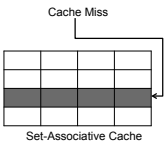


- *Parallel mode of access* – access both tag and data arrays in parallel
- *Serial mode of access* – access the tag array first followed by the data array

Cache Miss Classification

- *Compulsory* misses – the very first access to a block, independent of the cache size
  - Increase the block size to reduce the compulsory miss rate



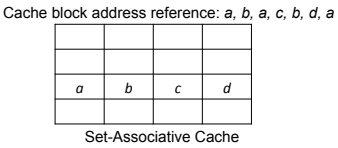


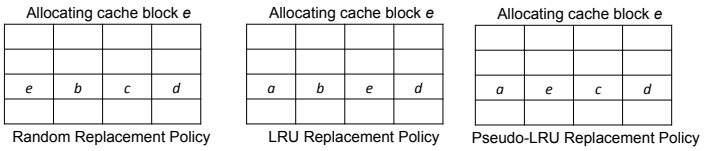
- *Capacity* misses
  - Increase the cache size to reduce the capacity miss rate
- *Conflict* misses
  - Increase the associativity to reduce the conflict miss rate

- Direct mapped cache – search is simple, but high conflict miss rate
- Fully-associative cache – low conflict miss rate, but search time is more
- Set-associative cache – reduces conflict miss rate

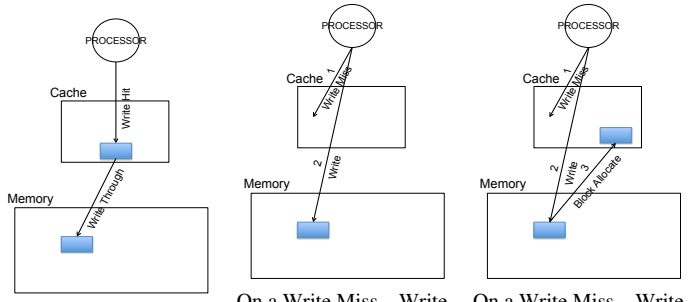
Block Replacement on a Cache Miss

- Direct-mapped caches – Only one block position
- Set-associative caches – Random, LRU, Pseudo-LRU



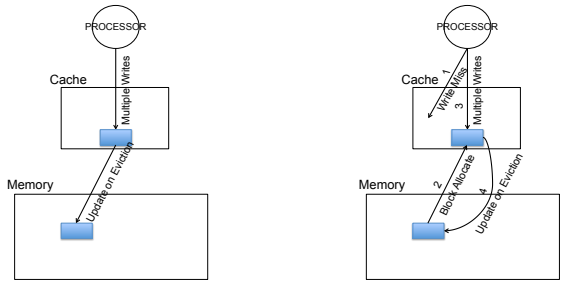


Cache Block Updates: Write-Through Caches



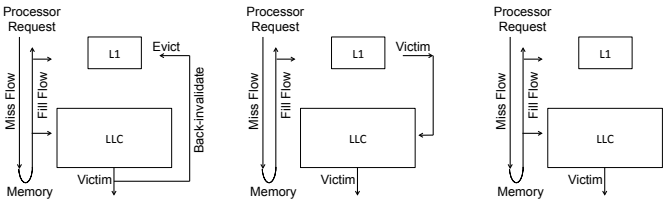
- Requires more bandwidth at lowest levels of caches
- Keeps the cache consistent with lower levels of the memory hierarchy

Cache Block Updates: Write-Back Caches

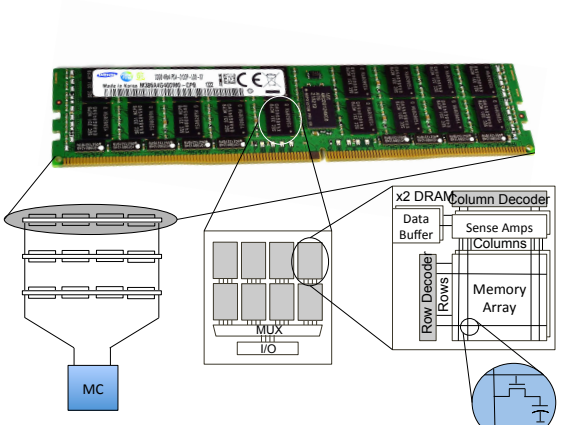


- Requires less write bandwidth
- Attractive for multiprocessor systems
- Power efficient

Different Types of Cache Hierarchies

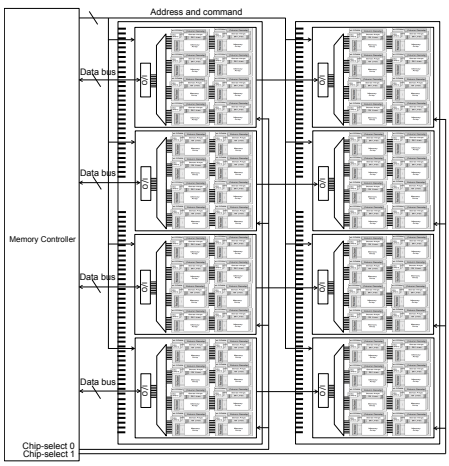


DRAM-based Main Memory Organization

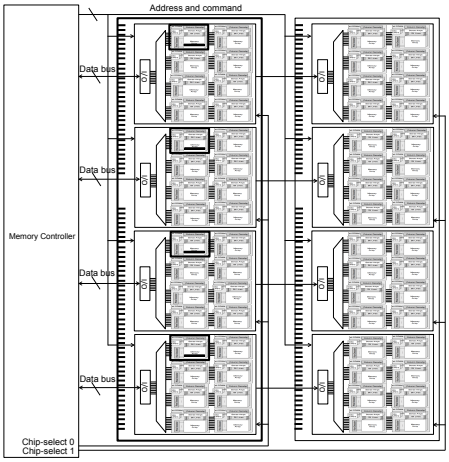


Channels → DIMMs → Ranks → DRAM Devices → Banks → Array of rows and columns

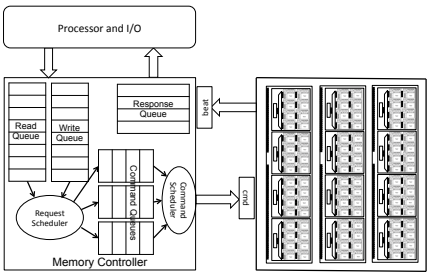
2-Rank 4-Device 8-Bank DIMM-DRAM Organization



Accessing a DRAM Row

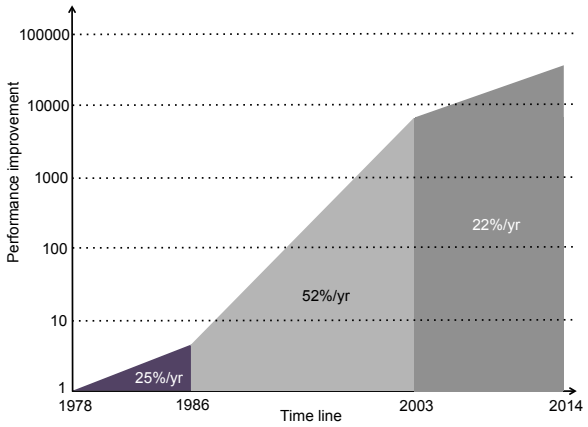


DRAM Memory Controller

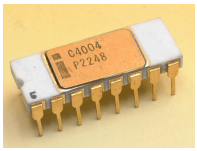


- Functions:
  - Converts memory requests to DRAM commands
  - Enforces timing constraints
  - Row-buffer management policies
  - Address mapping
  - Refresh management
  - Memory scheduling
- Microarchitecture of a DRAM memory controller determines the latency and sustainable bandwidth
- Design goals of a DRAM memory controller are:
  - Maximize system performance
  - Minimize power consumption

Growth in Processor Performance



Bit-Level Parallelism



- 4-bit processor
- 10μm process technology
- 740KHz clock frequency
- Approximately 2300 transistors

- Adding two 8-bit numbers, A[0:7] and B[0:7]:

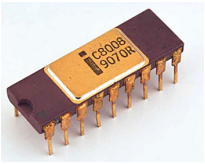
4-bit processor

C[0:3] ← A[0:3]+B[0:3]
C[4:7] ← A[4:7]+B[4:7]+C_{out}[0 : 3]

- 8-bit processor
- 10μm process technology
- 0.5MHz clock frequency
- 3500 transistors

8-bit processor

C[0:7] ← A[0:7]+B[0:7]



Bit-Level Parallelism (Contd)

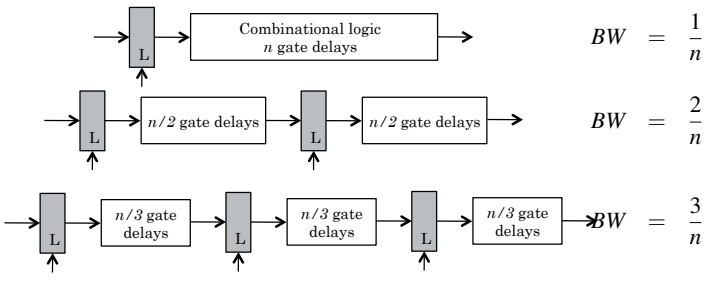
- 16-bit processor
- 3μm process technology
- 5MHz clock frequency
- 29000 transistors
- CISC Architecture

- 32-bit processor
- 1μm process technology
- 20MHz clock frequency
- Approximately 275K transistors

- 64-bit processor
- 90nm process technology
- 3.8GHz clock frequency
- 125million transistors

Pipelining

- Pipelining involves partitioning the system into multiple stages with added buffers between the stages
- Pipelining can increase the throughout of a system

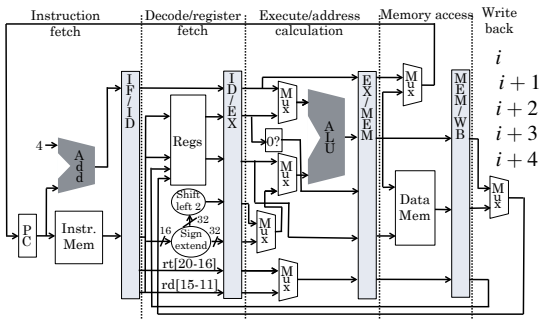


- Potential k-fold increase in throughput with k-stage pipeline

Instruction Pipeline

- Instruction fetch cycle (IF)
  - Based on PC value, fetch the instruction from memory
  - Update PC
- Instruction decode/register fetch cycle (ID)
  - Decode the instruction and register read
  - Do the equality check on the registers for a possible branch
  - Compute branch target address, if needed
- Execution/effective address cycle (EX)
  - Memory reference: Calculate the effective address (EA)
  - Register-register ALU instruction
  - Register-immediate ALU instruction
- Memory access cycle (MEM)
  - Load instruction: Read data from memory using the EA
  - Store instruction: Write data to memory using the EA
- Write-back cycle (WB)
  - ALU or load instruction: Write result into the register file

Scalar Pipelined Processors: In-Order Execution



i : $R1 \leftarrow R2 + R3$
$i+1$: $R4 \leftarrow R5 * R3$
$i+2$: $R6 \leftarrow R1 + R2$
$i+3$: $R3 \leftarrow R4 - R5$
$i+4$: $R7 \leftarrow R8 + R9$

i				
$i+1$	i			
$i+2$	$i+1$	i		
$i+3$	$i+2$	$i+1$	i	
$i+4$	$i+3$	$i+2$	$i+1$	i

Pipeline Hazards

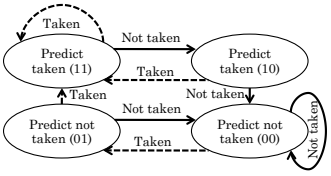
- Consequences of the pipeline organization and inter-instruction dependencies
  - Structural hazards – due to resource conflicts
  - Data hazards – due to instruction dependence
  - Control hazards – due to branches and jumps
- Hazards can stall the pipeline

Conditional Branches Limit Performance

- Make a branch prediction and speculatively execute the instructions in the predicted path
- For each misprediction, recover the state of the processor to the point before the mispredicted branch
- Branch misprediction penalty increases as the pipelines deepen and the number of outstanding instructions increases

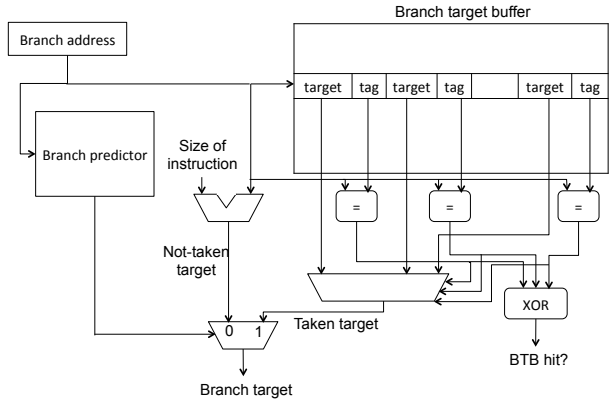
Branch Prediction Mechanisms

- Predict the branch direction – whether a conditional branch is taken or not taken
- Predict the branch target
- Static branch prediction techniques
  - always-not-taken, always-taken
- Dynamic branch prediction techniques
  - 2-bit dynamic branch prediction, correlating branch prediction, ...



Branch Target Buffers

- Cache-like structure to store the last seen target address for branches
- Accessed in parallel with branch prediction algorithm

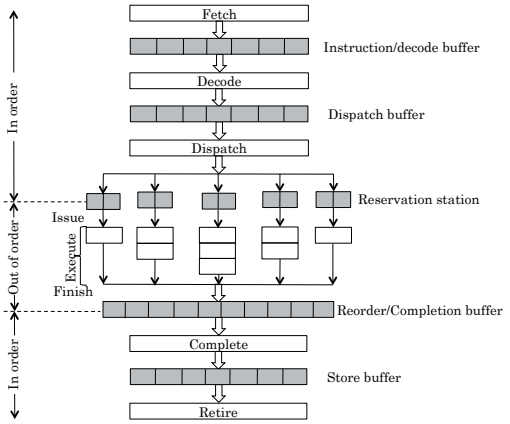


Limitations of Scalar Pipelines

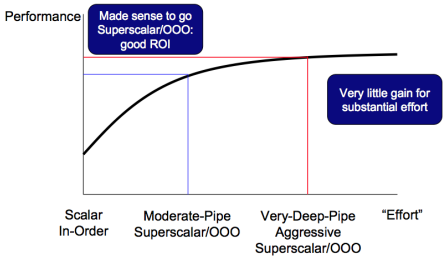
- Single k-stage instruction pipelines
  - All instructions, regardless of type, traverse through the same set of pipeline stages
  - At most one instruction can be resident in each pipeline stage at any time
  - Instructions advance through the pipeline stages in a lockstep fashion
- Fundamental limitations
  - The maximum throughput for a scalar pipeline is bounded by one instruction per cycle
  - The unification of all instruction types into one pipeline can yield an inefficient design
  - The stalling of a lockstep or rigid scalar pipeline induces unnecessary pipeline bubbles

Superscalar Processors: Exploiting Instruction-Level Parallelism

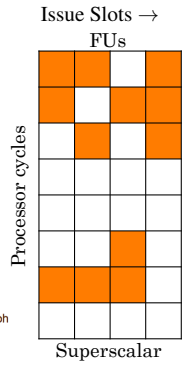
i : $R1 \leftarrow R2+R3$
$i+1$: $R4 \leftarrow R5*R3$
$i+2$: $R6 \leftarrow R1+R2$
$i+3$: $R3 \leftarrow R4-R5$
$i+4$: $R7 \leftarrow R8+R9$



The ILP Wall



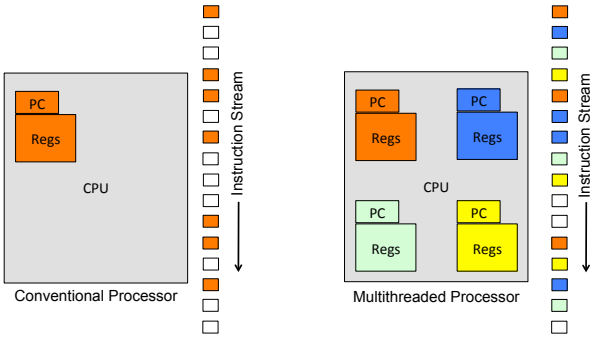
Source: G. Loh



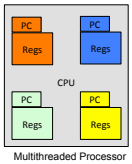
- Utilize the idle functional units when insufficient ILP exists
- Allow multiple threads to share the functional units of a single processor in an overlap fashion

Multithreading

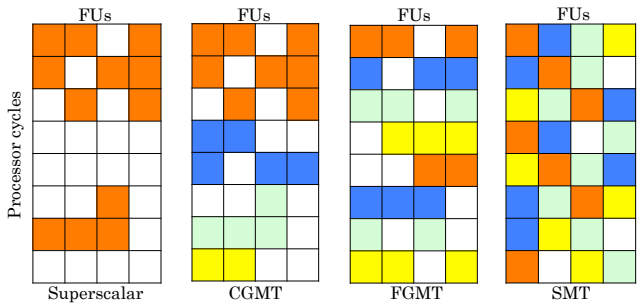
- Thread is a control flow of execution
- *Thread context* – PC + Regs



Various Forms of Multithreading



- Fine-grain multithreading
- Coarse-grain multithreading
- Simultaneous multithreading



Thank You