CS 152: Computer Architecture and Engineering Lecture Notes

Week 1: Lecture 1 Computer Architecture History (1/17)

Abstraction Layers in Modern Systems

- ISA & Microarchitecture

Single-Thread processor performance

- Early 2000s: limited by power -> multi core processors
- Early 2010s: limited by parallelism

Today's Dominant Target Systems

- Mobile (smartphone/tablet)
    - >1 billion /year
    - Dominated by ARM-ISA compatible general-purpose processor in System on a chip (SoC)
- Warehouse-Scale COmputers (WSCs)
    - 100,000's cores per warehouse
    - Market dominated by x86 compatible server chips
    - Dedicated apps, cloud hosting of virtual machines,
    - Increasing use of GPUs, FPGAs, custom hardware
- Embedded computing
    - Wired/wireless network infrastructure, printers
    - Consumer TV/Music?Games
    - Internet of things

New Vertical Semiconductor Business Model

- Chip customers build own differentiated designs instead of buying chip company's standard product
-

Week 1: Lecture 2 Simple Machine Implementations (1/19)

Instruction Set Architecture (ISA)

- The contract between software and hardware
- Giving all programmer-visible state plus semantics of instructions that operate on that state
- IBM first line of machines to separate ISA from implementation
- Many implementations possible for a given ISA

ISA to Microarchitecture Mapping

- Accumulator -> hardwired, unpiplined
- CISC: microcoded
- RISC : hardwired, piplined

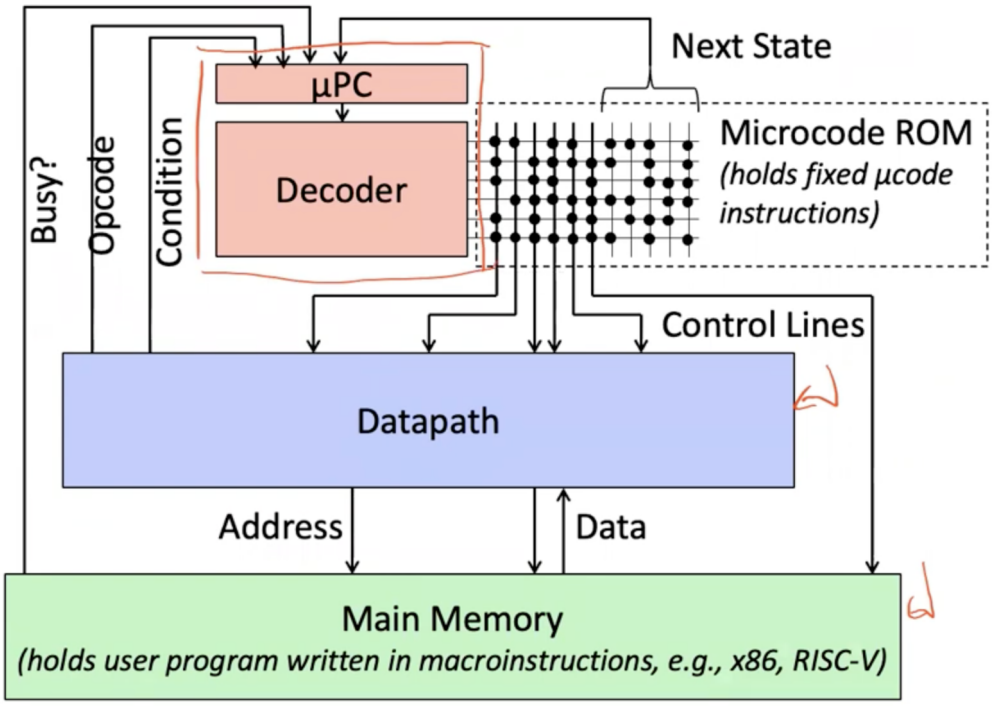
- VLIW : fixed-latency in order parallel pipelines
- This lecture: microded RISC-V machine

Why Learn microprogramming

Processor designs split between datapath

- **Datapath**: Numbers are stored and arithmetic operations computed and control,
- **Control**: sequences operations on datapath

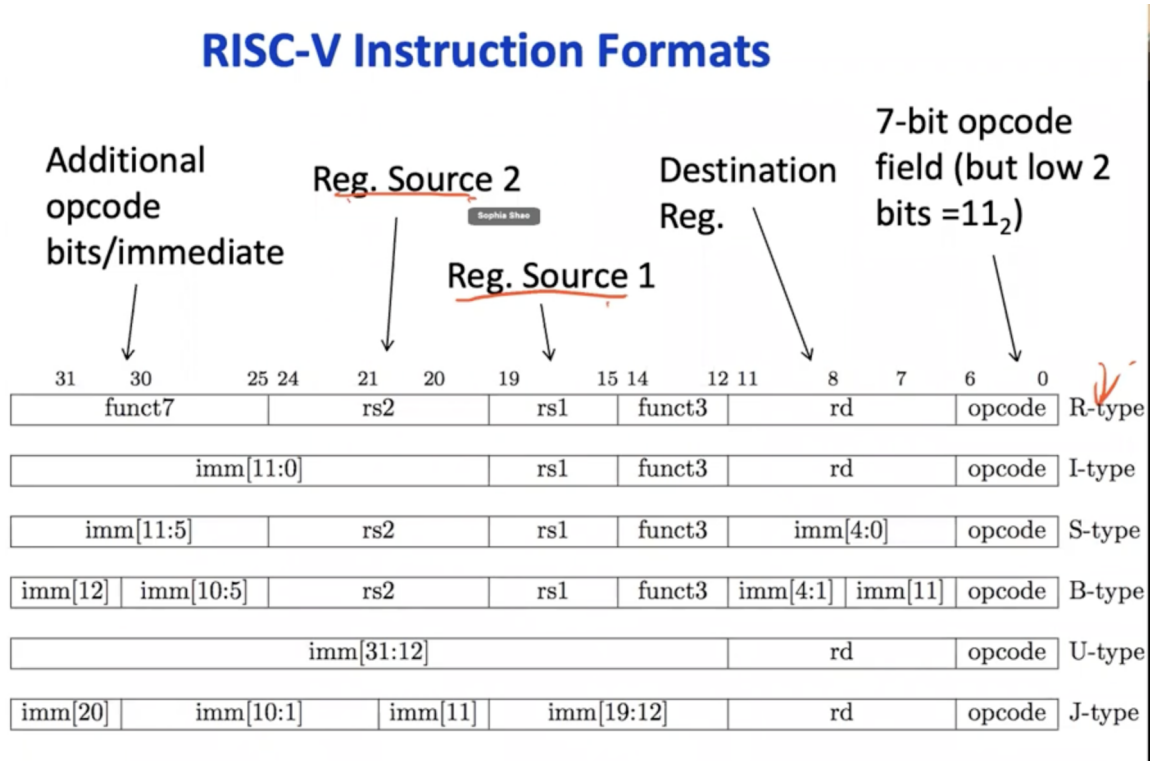
Micocoded CPU



-
- Breaking down the instructions into multiple steps
- Read only memory (ROM)

RISC V Processor state

- Program counter (PC)
- 32 bit integer resgiers (0x0 - x31)
- Instruction encoding



-
- Reigseter source 1 source 2, destination registers
- Microinstructionis written as register transfers
  - MA:=PC means RegSel=PC; RegW=0; RegEn=1; MALd=1
  - B:=Reg[rs2] means RegSel=rs2; RegW=0; RegEn=1; BLd=1
  - Reg[rd]:=A+B means ALUop=Add; ALUEn=1; RegSel=rd; RegW=1

RISC V instruction Execution phases

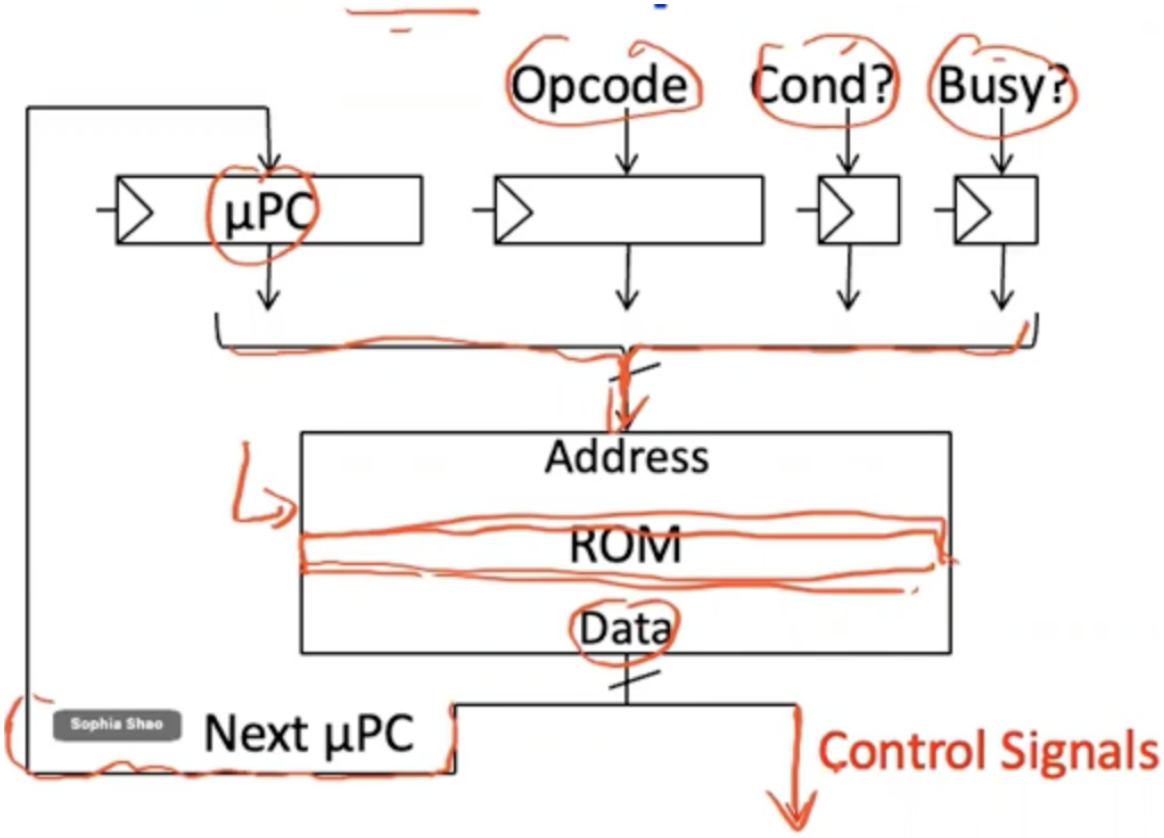
- Instruction Fetch
- Instruction Decode
- Register Fetch
- ALU oeprations
- *Optional* emmory operations
- *Optional* register writeback
- Calculate next instruction

Microcode sketches

- Instruction Fetch
  - MA = PC
  - PC = A + 4
  - Wait for memory
  - IR = Mem
  - Dispatch on opcode

- ALU
    - A = Reg[rs1]
    - B = Reg[rs2]
    - Reg[rd] = ALUOp(A, B)

Week 2: Lecture 3 Pipelining (1/24)



ROM implementation

Single-Bus Microcode RISCV ROM size

- Instruction fetch sequence 3 common steps
- Each group takes ~5 steps
- Total steps needs 6 bits for microPC
- Total size 25KiB

Reducing Control Store Size

- Reduce ROM height (#address bits)
- Reduce ROM width (#data bits)

Encoded ROM contents

- Spin, don't do any work

Horizontal vs Vertical uCode

- Wider microinstruction or more instructions
    - Multiple parallel operations per uinstruction

- Vertical microcode has narrower uinstructions
- Nanocoding

Nanocoding

- 17 bit ucode containing either 10 bit microjump or 9 bit nanoinstruction pointer
    - Nanoinstructions were 68 bits wide, decoded to give 196 control signals

Microprogramming in IBM 360

Microprogramming in IBM 360

	M30	M40	M50	M65
Datapath width (bits)	8	16	32	64
µinst width (bits)	50	52	85	87
µcode size (K µinsts)	4	4	2.75	2.75
µstore technology	CCROS	TCROS	BCROS	BCROS
µstore cycle (ns)	750	625	500	200
memory cycle (ns)	1500	2500	2000	750
Rental fee ($K/month)	4	7	15	35

-
- Writable Control Store (WCS)
    - Implement control store in RAM not ROM
    - User-WCS failed
        - Difficult, protection was hard

Microprogramming is far from extinct

- Played a crucial role in micros of the eighties

Analyzing Microcoded Machines

- Simple pipelined processor 801, advanced compilers inside IBM
- Easier to use simple compilers and RISC
- 60% of microde for 0.2% of execution time

From CISC to RISC

- Use fast RAM to build instruction cache of user visible instructions not fixed hardware microroutines
- Simple ISA to enable hardwired pipelined implementation
    - Most compiled code only used few CISC instructions
    - RISC ISA comparable to vertical microcode
- Further benefit with integration

Week 2: Lecture 4 Pipelining (1/26)

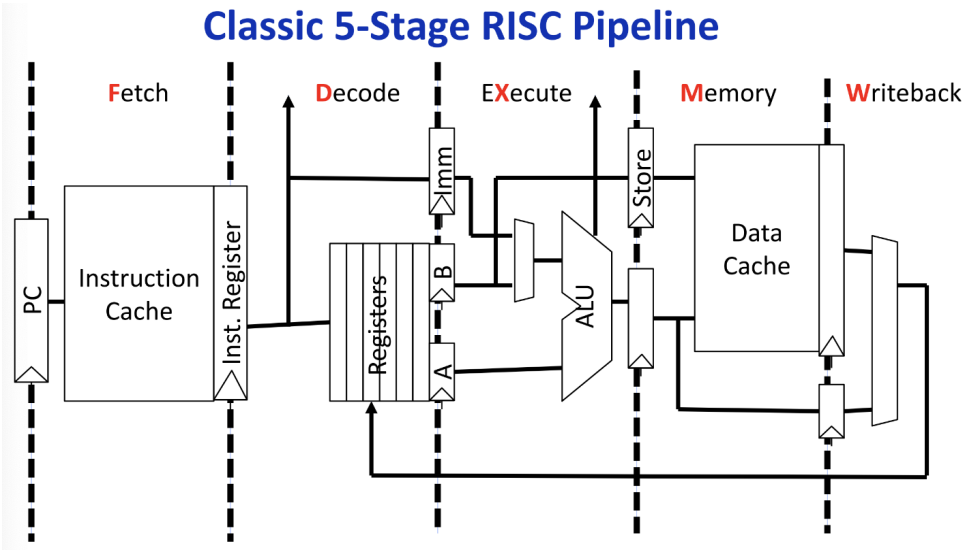
Lecture 2

"Iron Law" of Processor Performance

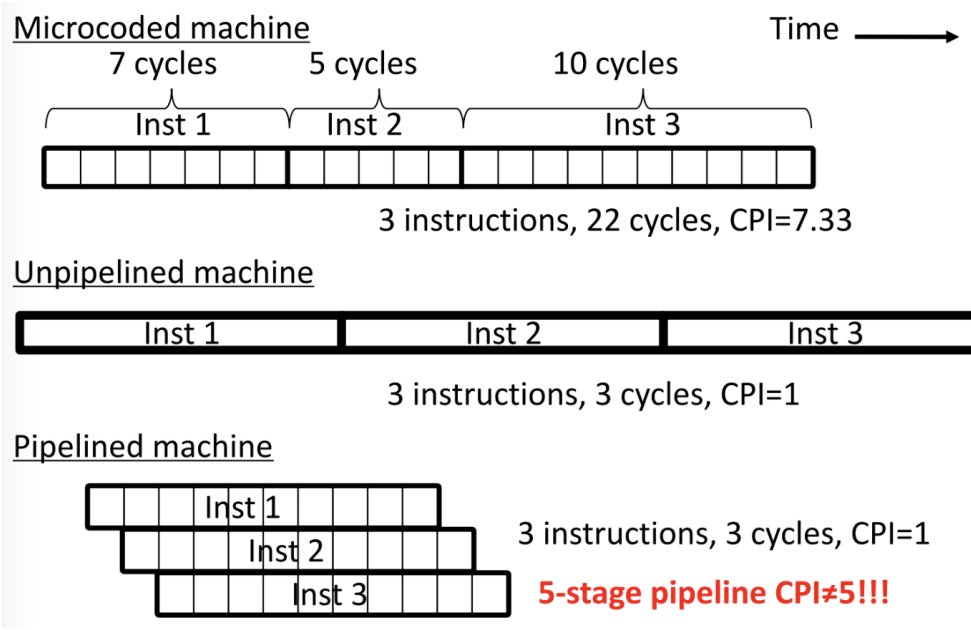
- Time/Program = Instructions/Program * Cycles/Instruction * Time/Cycle
- Instructions per program depends on source code, compiler tech, and ISA
- Cycles per instructions (CPI) depends on ISA and microarchitecture
- Time per cycle depends on micro architecture and base technology

Microarchitecture	CPI	cycle time
Microcoded	>1	short
Single-cycle unpipelined	1	long
Pipelined	1	short

5 stage RISC Pipeline



CPI Examples

- 

Instructions interact with each other in pipeline

- **Structural hazard:** Instruction in the pipeline may need a resource being used by another instruction ni the pipeline
- **Data Hazard:** Instruction depends on something produced by earlier instruction, data hazard
- **Control Hazard:** Dependence may be the next instructions

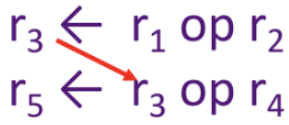
Pipeline CPI Examples

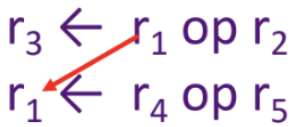
- Measure from when first instruction finishes to when last instruction in sequence finishes

Resolving Structural Hazards

- adding more hardware to design

Types of Data Hazards

- **Data dependence:** Read after Write (RAW) hazard
  - 

- **Anti-dependence:** Write after Read (WAR) hazard
  - 

- **Output dependence:** Write after Write (WAW) hazard

- $r_3 \leftarrow r_1$ op r_2
- $r_3 \leftarrow r_6$ op r_7

Strategies for Data Hazards

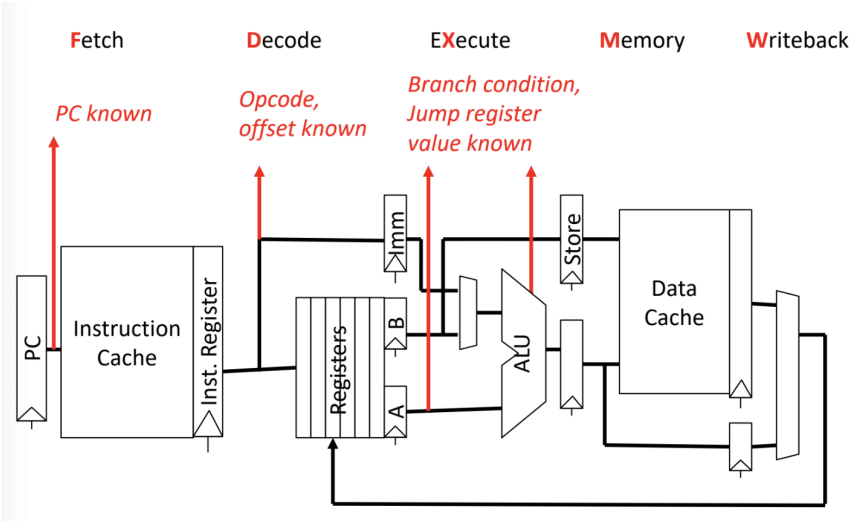
- Interlock/stall
- Bypass
- Speculate

Interlocking Versus Bypassing

- Interlock is 2 bubbles
- Bypass we use the result immediately

Control Hazards

- What do we need to calculate the next PC
- For Unconditional Jumps
  - Opcode, PC, and offset
- For Jump Register
  - Opcode, register value, and offset
- For conditional brnaches
  - Opcode, register, PC and offset
- For all other instructions
  - Opcode and PC



Branch Delay Slots

- Changed ISA semantics so instruction after branch/jump is always executed before the control flow change occurs
- Software has to fill delay slot with useful work, or fill with explicit NOP instruction
- Don't use delay slots anymore
- Performance issues, complicates more advanced microarchitectures

- Better branch prediction reduced need

Why instruction may not be dispatched every cycle in classic 5 stage pipeline (CPI > 1)

- Loads have two cycle latency
- MIPS: Microprocessor with Interlocked pipeline stages

Traps and Interrupts

- **Exception**: an usual internal event caused by program during execution
- **Interrupt**: An external event outside of running program
- **Trap:** Forced transfer of control to supervisor cuased by execption or interrupt

Trap Handler

- Saves EPC before enabling interrupts to allow nested interrupts

Exception handling

Week 3: Lecture 5 Pipelining 2 (1/31)

Trap altering the normal flow

Exception handler

- Where a particular exception occurs
    - PC address, overflow, illegal opcode
- Keep track of differerent exceptions

Commit point

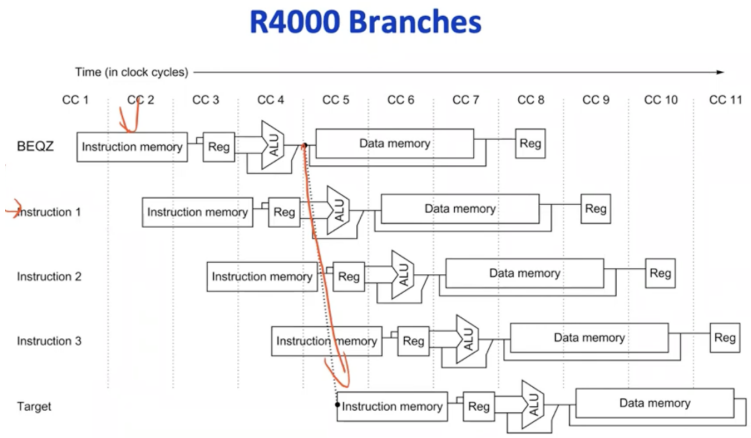
- Don't let the instruction before the commit, finish line
- Anything that happens between has still finished, prograting where the exception occurred and propagate until the commit point
- If trap at commit, update the cause and EPC registers, kill all stages, inject handler PC into fetch stage

Speculating on Exceptions

- Predictioin mechanism
- Check prediction mechanism
- Recovery Mechanism
    - Only state at commit point
- Bypassing allows use of uncommitted instruction results by following instructions

8 stage: IF IS ,

- Instruction memory and data memory split into different pipelines



-

Simple Pipeline Scheduling

- Reschedule code to try to reduce pipeline hazards

To Reduce Hazards: Loop Unrolling

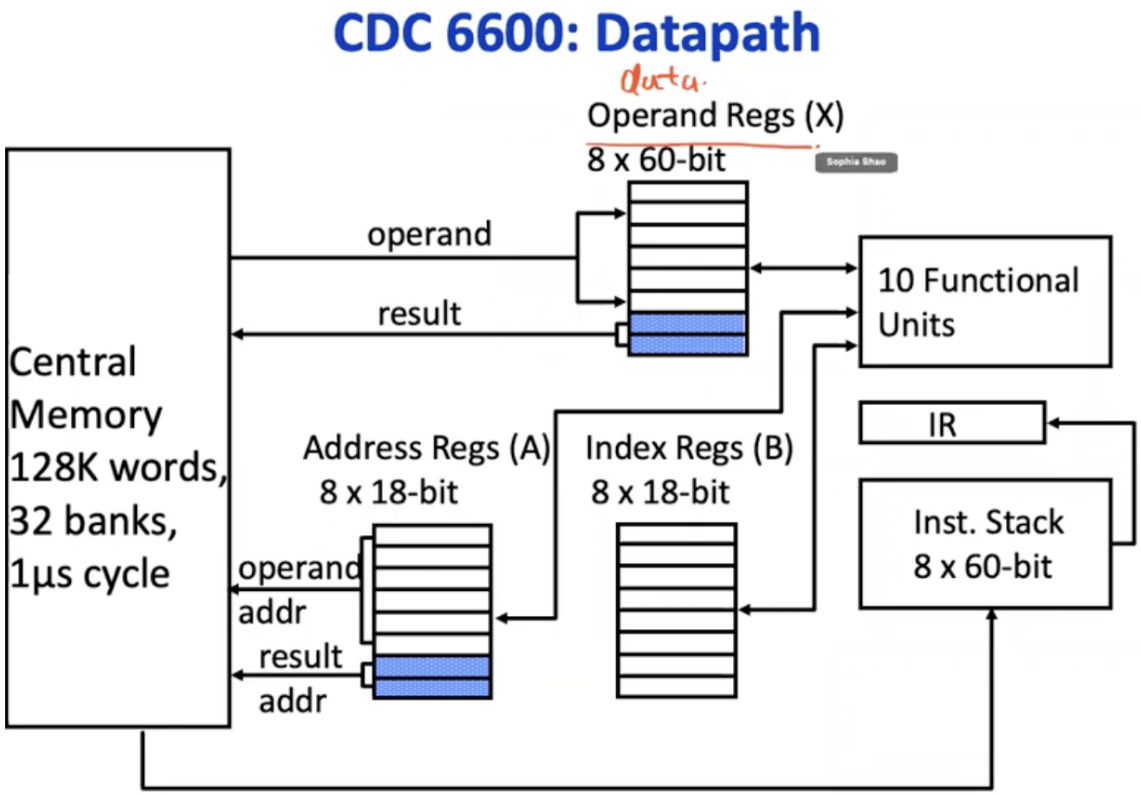
- Expose more parallelism, dynamic instruction count
- If instructions can go earlier, they go earlier

Super Computers

- Definition: fastest machine in world at given task
- A device to turn a compute bound problem into an I/O bound problem
- Costing $30M+

CDC 6600 (1964) First supercomputer

- Fast pipelined machine with 60 bit words 10 MHz very fast
- load/store architecture: separate instructions to manipulate
- three types of reg
    - Data, index, address reg



-

Issues in Complex Pipeline control

- Structual conflicts at execution stage if some FPU or memory unit is not pipelined and takes more than one cycle
- Structual conflicts at the write back stage
- Out of order write hazards due to variable latencies

CDC6600 Scoreboard

- Instructiosn dispacted in order to functional units provided no strucaral hazard or WAW
- Instructions wait for input operands before execution
- Insturctions wait for output register to be read by preceding instructions (WAR)

Week 3: Lecture 6 Memory I: Memory Hiearchy (2/2)

More Complex In-order pipeline

- Deploy writeback so all operations have same latency to W stage

In order Superscalar Pipeline

- Fetch two instructions per cycle
- Both simultaneously if one is integer/memory and other is floating point

Core Memory

- First large scale reliable main memory
- Using magnetization polarity on robust, non-volatile storage

DRAM Operation

- 3 steps in read.write access to a given bank

- Row access (RAS)
    - Decode row address, small change in ovlrage detected by sense amplifiers
- Column access (CAS)
    - Decode column address to select small number of sense amplifier latches, on read, send latched bits out to chip pins
- Precharge
    - Charges bit lines to known value

Computer Architecture Terminology

- **Latency** (in seconds or cycles): Time taken for a single operation from start to finish
- **Bandwidth :** rate of which operations can be performed
- **Occupancy :** time during which the unit is blacked on an operation (structural hazard)

Physical Size Affects Latency

- Big memory, signals have further to travel, fan out to more locations

Capacity, latency, bandwidth: Register << SRAM << DRAM

Management of Memory Hiearchy

- Small/fast storage, registers
- Larger slower storage

Memory References

- Temporal Locality
    - If a location is referenced it is likely to be referenced again in the near future
- Spatial Locality
    - If a location is referenced it is likely that locations near it will be referenced in the near future

Cache Algorithm

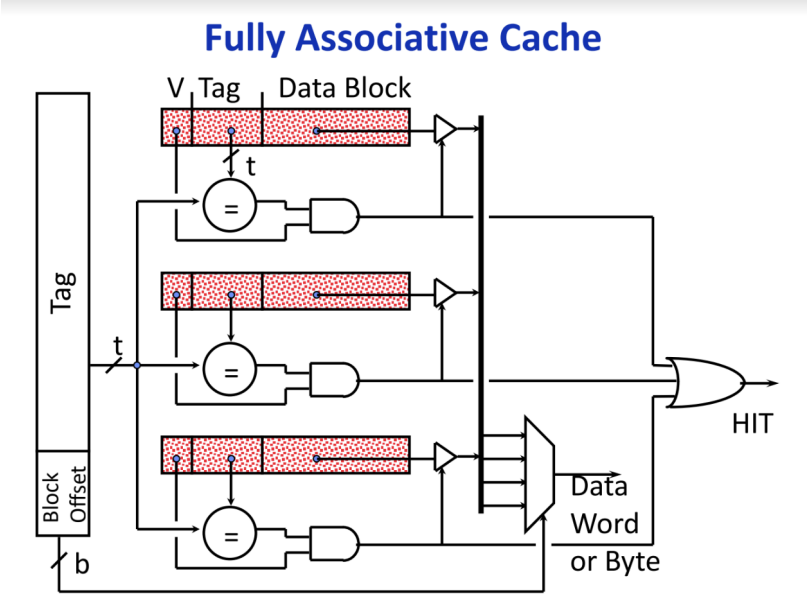
- Look at processor address, search cache tags to find match, found in cache
    - HIT, return copy of data from cache
    - not in cache MISS, read block of data from main memory

Direct Map Address Selection

- Might always evict the same row

2 way set associative Cache

Fully associative cache



Week 4: Lecture 7 Memory II (2/7)

Replacement policy

- Random
- Least Recently Used (LRU)
- First In First Out (FIFO): Round Robin
- Not most recently used (NMRU)

CPU Cache me

Improving Cache Performance

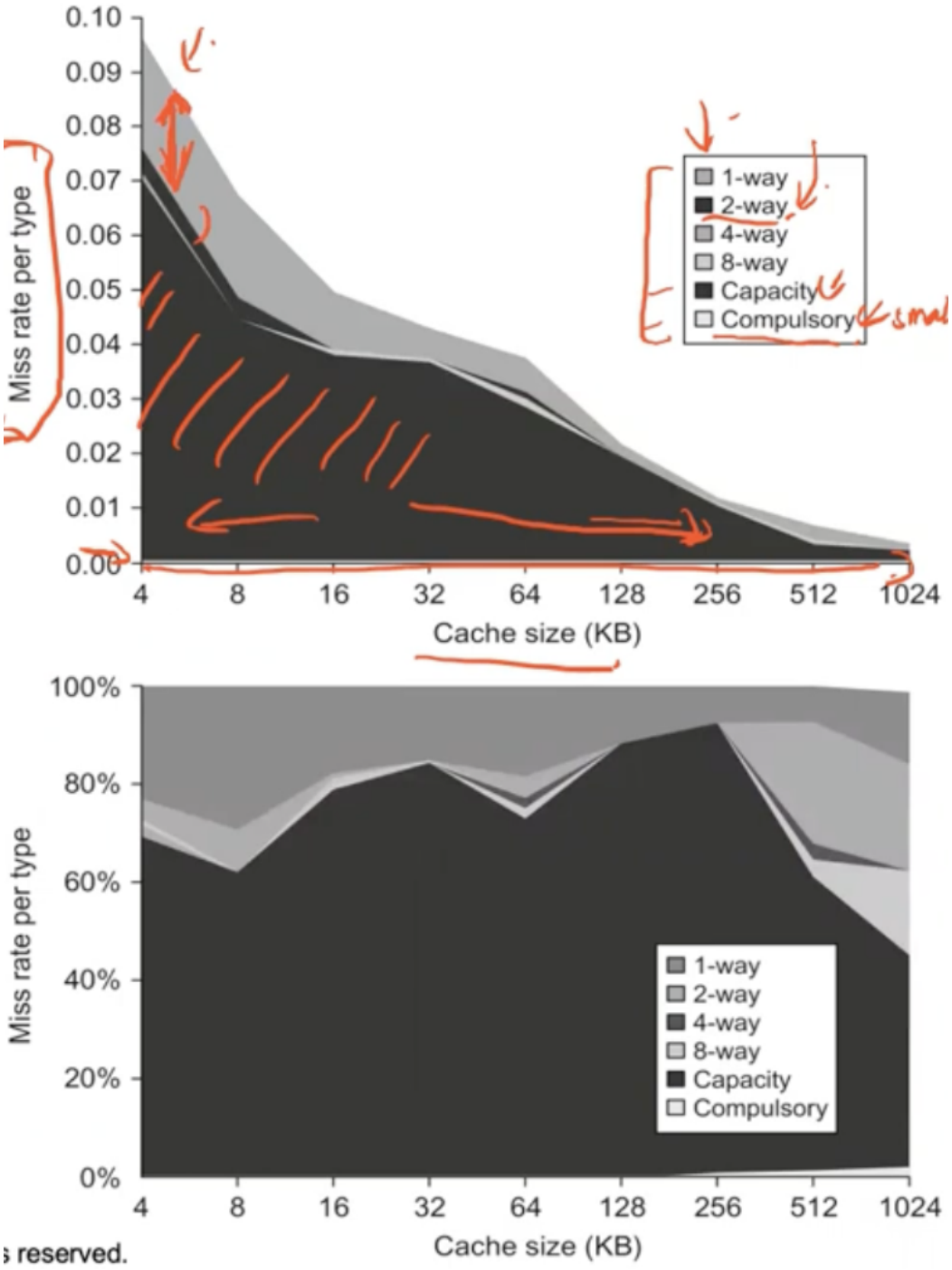
- Average Memory Access time (AMAT) = Hit time + Miss Rate x miss penalty
- To improve performance
    - Reduce the hit time
    - Reduce the miss rate
    - Reduce the miss penalty

Causes of Cache misses

- Compulsory: First reference to a line (cold start misses)
- Capacity: cache is too small to hold all data needed by program
- Conflict: misses that occur because of strategy

Effect of Cache Parameters on Perforamance

- Larger cache size
- Hihger associativity
- Larger line size



- s reserved.

Line Size and Spatial Locality

- A line is a unit of transfer between the cache and memory

Larger LIne size has distinct hardware advantages

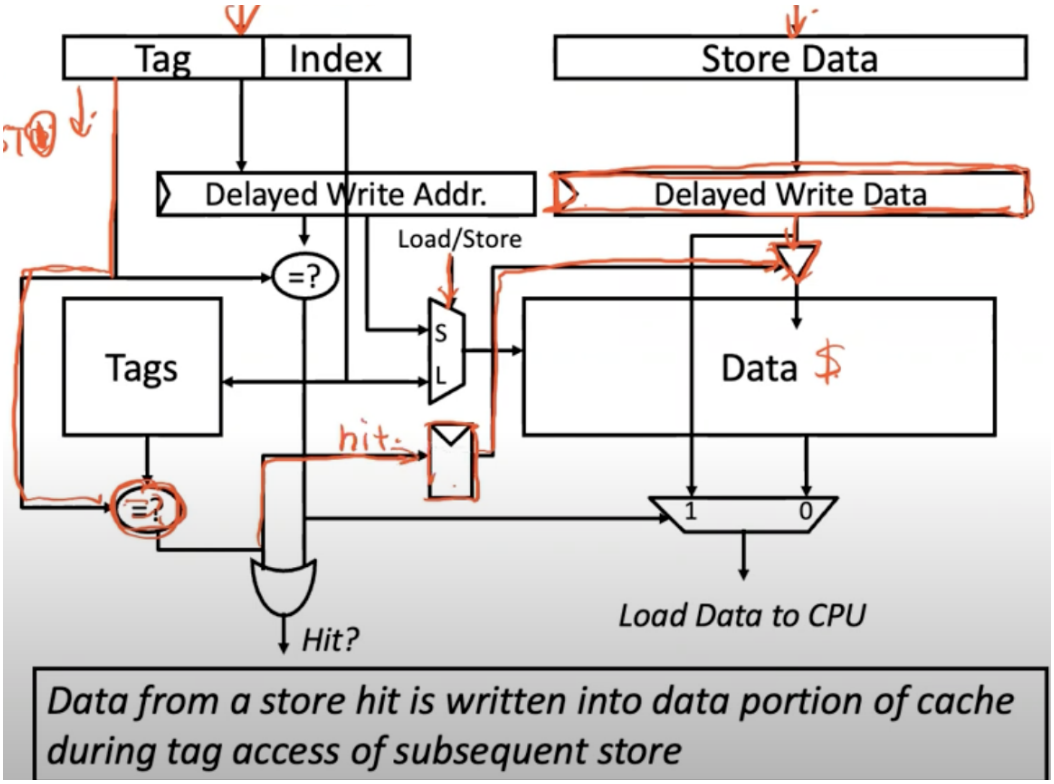
- 64k bytes is pretty good size for block size

Write Policy Choices

- Cache Hit
  - Write Through: Write both cache & memory

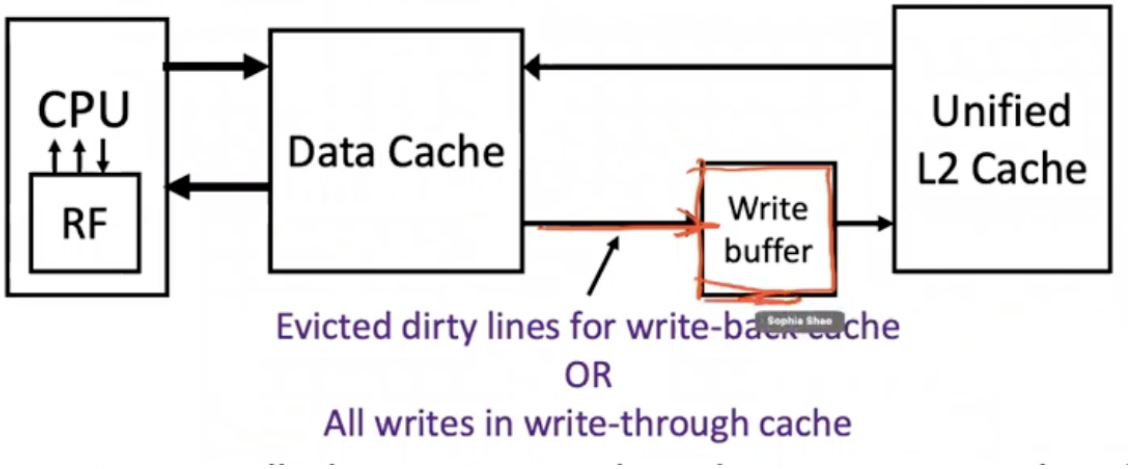
        - Higher traffic but simpler pipeline and cache design
    - Write back: write cache only memory is written only when the entry is evicted
- Cache Miss
    - No write-allocate: only write to main memory
    - Write allocate : (fetch on write) fetch into cache
- Common combinations
    - Write through and no write allocate
    - Write back with write allocate

Reducing Write Hit TIme

- Problem: Writes take two cycles in memory stage, one cycle for tag check plus one cycle for data write if hit
- Solutions
    - Design data Ram that can perform read and write in one cycle, restore old value after tag miss
    - Pipelined writes: hold write data for store in single buffer ahead of cache
    - Fully associative cache: word line only enabled if hit
    - 

Write Buffer to Reduce Read Miss Penalty

- Processor is not stalled on writes, and read misses can go ahead of write to main memory
- Write buffer may hold updates value of locatio needed



-
- On a read miss, wait for the write buffer to go empty
- Check write buffer addresses against read miss addresses, if no match, allow read miss to go ahread of writes, else, return value in write buffer

Reducing Tag Overhead with Subblocks

- Problem: Tags are too large, too much overlhead
- Solution: Sub block placement (sector cache)
    - Valid bit added to units smaller than full line, called sub blocks
    - Only read a sub block on a miss
    - Other blocks not in cache
    - Finer granuarlity of cache

Multilevel Caches

- A memory cannot be large and fast
- Sol: increasing sizes of cache at each level
- Local miss rate = misses in cache / accesses to cache
- Global miss rate = misses in cache / CPU memory accesses
- Misses per instruction (MPI) = misses in cache/ number of instructions

Presense of L2 influences L1 design

- Use smaller L1 if there is also L2
- Use simpler write through L1 with on-chip l2
    - Write back l2 cache absorbs write traffic, doesn't go off chip

Inclusion Policy

- Inclusive multilieve cache
    - Inner cache can only hold lines also present in outer cache
- Exclusive multileve caches
    - Inner cache may not hold lines in outer cache

Week 4: Lecture 8 Prefetching (2/9)

Definitions

- **Memory Hierarchy:** way memory is organized, serices of smaller cache memories

- **Memory Latency:** time to access memory
- Read/Write Latency: time to read data
- Memory Bandwidth (BW):
- Memory Footprint:

Memory Hiearchy

- Ideal: upper layers are lowlatency and high bandwidth
- Most requests in L1 and L2 caches
- Slow DRAM sees little bandwidth

How to hide latency?

1. **Caches** hide latency much of the time (via hits)
    a. Work well when the working set fits and spatial and temporal locality are favorable
    b. 64B fetch granular hides some latency to rest of the line
2. **Reordering code** to hoist loads and block operations to fit in caches
3. **Software prefetching** to get the memory beforehand
4. **Speculative execution** of loads and sotres along with multiple outstanding misses helps as well (non-blocking caches), multiple core
5. **Hardware Prefetching**

Large out of order machine

- How about large out-of-order architecture with many outstanding misses
- Limited to hiding L1 and L2 cache
- Size of instruction window:
    - Little's law (occupancy = latency * throughput)
    - Throughput = window size / latency
    - High latency really hurts, high latency reduces throughput (IPC)

Adding hardware

- 2 load pipes,
- Computer a delta between these two addresses
- Add delta to current miss address
- If it is to a new cachleline, issue a prefetch fill request
- Miss buffer would only have demand misses if ideal

Toy hardware prefetcher

- Prior Miss PA, Prior Prior Miss PA
    - Subtractor to get the delta and add it to the prior miss PA
- Accuracy = useful prefetches / total prefetches
- Coverage = userful prefetches / total unique accesses
- Timeliness = number of prefetches arriving on time / total prefetches

What to prefetch?

- Large instructions will not even fit in instruction

Basic Schemes Heuristic-Based Next-N-Line Prefetching

- Prefetch next N sequential cachelines from current cacheline
- History-Based Target Line prefetching
    - Maintain table of {current line, next line} pairs in hardware
    - Prefetch for non-sequential program flow
    - Lacks compulsory misses
    - Assosumes historic conditional branch direction
- Heuristic based wrong path prefetching
    - Prefetch cacheline containing target of conditional branch once decoded (even if not taken)
- Hybrid Prefetching
    - Combine next-n-line prefetching with targetprefetching and wrong path prefetching
        - Broader

Data Prefetching

- Next Line prefetchers
- always prefetch next n cache lines

Stride Prefertchers

- Instruction program counter (PC) based
- Cache block paddress based
- Instruction based
- Cache block address based stride prefetching
    - Can detect A, A + N, A + 2N...

Address correlation based prefetching

- Create a markov model based on previous scheme for cache
- Cover arbitrary access patterns
- Coorelation tables needs to be very large for high coverage
- Low timlines
- Memory bandwidth

Execution-based prefetchers

- Idea: pre execute a peice of the pruned program solely for prefetchign data

How to construct the speculative thread

- Run ahead prefetching

Prefetching in Multi core

- Prefetching shared data
    - Coherence misses
- Prefetch efficiency is alot more important
- One cores' prefetches interfere with other cores prefetches

Week 5: Lecture 9 Address Translation (2/14)

Victim Caches

- Small associative backup cache, added to a direct mapped cache, which holds recently evicted lines

Way-Predicting Cache

- Return copy of data from cache

Reduce Miss Penalty of Long Blocks: Early Restart and Critical Word first

- **Early Restart:** as soon as the requested word of the block arrives, send it to the CPU and let the CPU continue execution
- **Critical word first:** Request teh missed word first from memory and send it to the CPU as soon as it arrives: let the CPU continue execution while filling the rest of the words in the block

Address Translation

Bare Machine

- Only kind of address is a physical address, corresponding to address lines of a actual hardware memory

Managing Memory in Bare machines

- Early machines only ran one program at a time,
- Use linker or loader program to relocate library modules to actual locations in physical memory

Dynamic Address Translation

- Motivation
    - In early machines, I/O was slow and each I/O transfer involved the CPU
- Location Independent programs
    - Programming and storage management ease
- Protection
    - Need for a bound register

Simple Base and Bound Translation

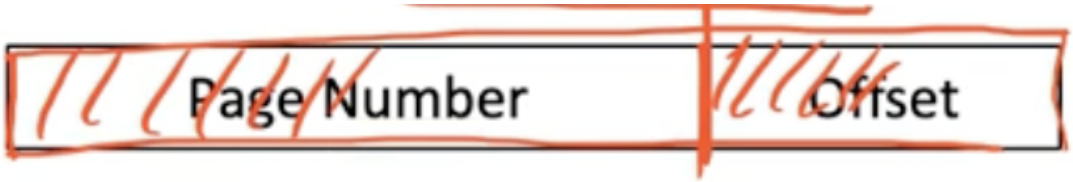
- Base and Bounds registers are visible/accessible only when processor is running in the supervisor mode

Separate Areas for Program and Data

- Data segment and program segment base and bound

Paged Memory Systems

- Program generated virtual or logical address split into



-
- Page Table contains physical address of start of each fixed-sized page in virtual address space
- Paging makes it possible to store large continuous memory using non contiguous physical memory pages

Private address space per user

- Separate space

Paging Simpliefes Allocation

- Fixed size pages can be kept on OS free list and allocated as needed to any process
- Process memory usage can easily grow or shrink dynamically
- Paging suffers from internal fragmentation where not all bytes on a page are used

Page Tables live in memory

- Herarchical page tables are used

Coping With limite primary storage

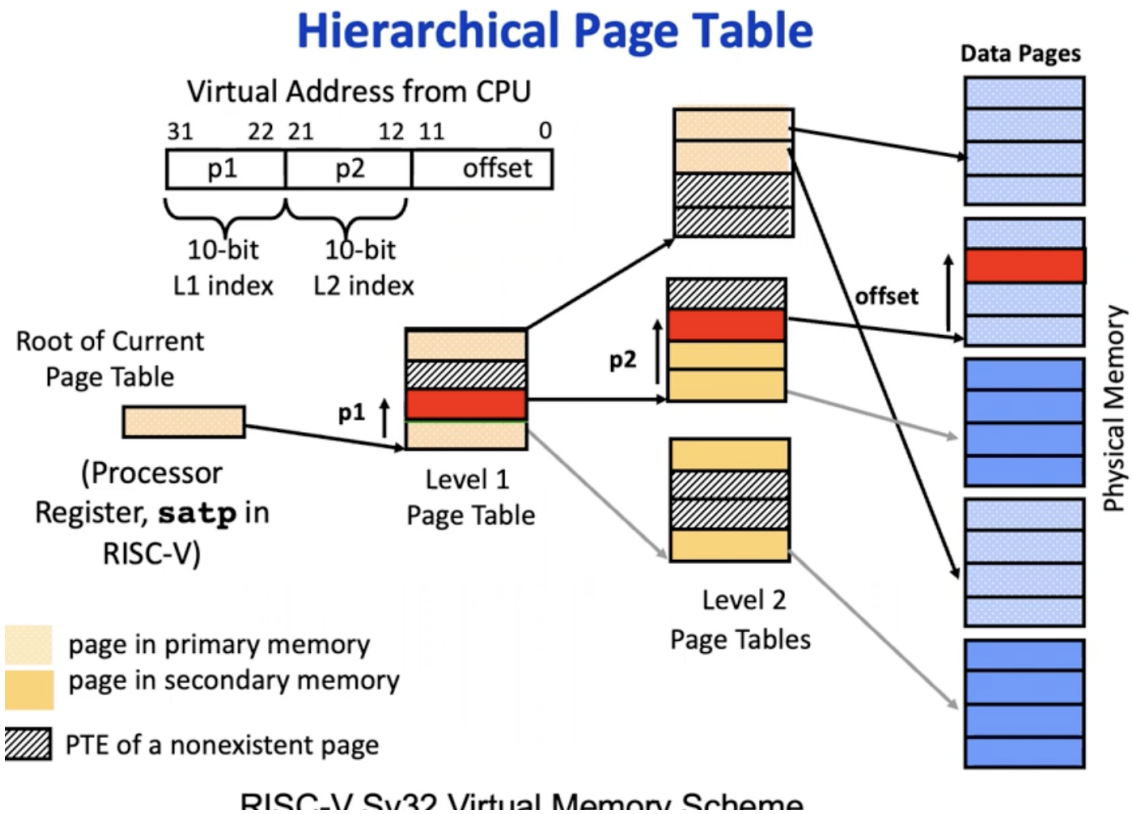
- Paging reduces fragmentation
- Still many problems would not fit into primary memory, have to copy data to and from secondary storage (drum, disk)
    - Manual Overlays
        - Copies code and data ina nd out of primary memory
    - Software interpretive coding
        - Detects variables that are swapped out to drum and brings them back in

Demand Paging in Atlas (1962)

- Page from secondary storage bright into primary storage when demanded by the processor
- Page Address Register (PAR) per page frame
- Compare the effective page address against 32 PAR
    - March or no march -> page fault

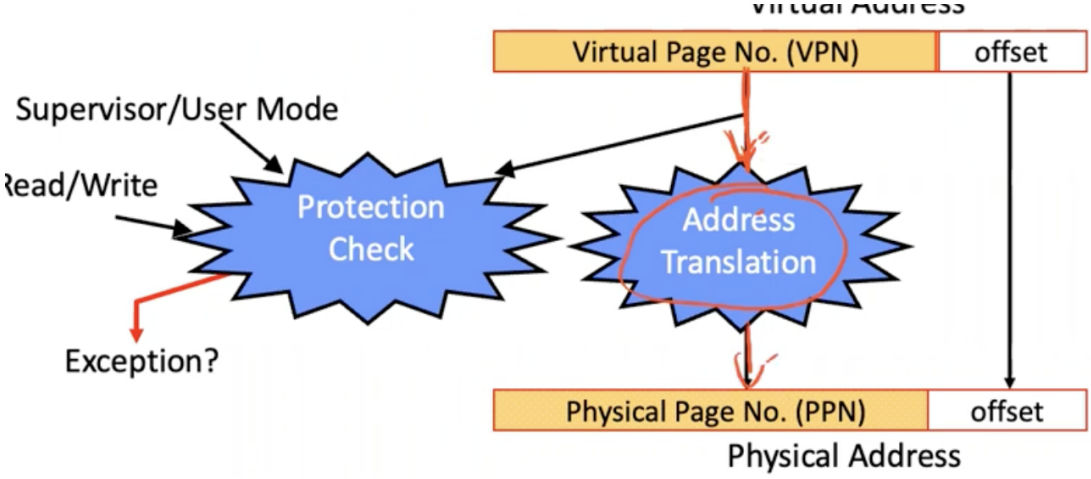
Size of Linear Page Table

- With 32 bit addresses 4 KB pages & 4 Byte PTEs
- 2^20 PTE 4MB page table per user
- 4GB of swap needed ot back up full virtual address space



- RISC-V Sv32 Virtual Memory Scheme
- P1 index, P2 index, offset on the data page

Address Translation & Protection



- Every instruction and data access needs address translation and protection checks

Translation-Lookaside Buffers (TLB)

- Address translation is very expensive
- 2 level page table, each reference becomes several memory accesses
- Cache translations in TLB

TLB Designs

- 32-128 entries, fully associative

- Maps to a large page, less spatial locality across pages
- Random or FIFO replacement policy
- TLB reach: size of largest vietual address psace that can be simultanesiouly mapped by TLB
    - TLB entires * pages * page/entry

TLB Miss

- Software
    - Causes an exception and OS walks the page tables and reloads the TLB
    - Software TLB very expensive on out of order superscalar processor
- Hardware
    - Memory management unit (MMU) walks the page tables and reloads the TLB
    - If missing page is encountered during the TLB reloading, MMU does a walk

Week 5: Lecture 10 Virtual Memory (2/16)

Lecture 8 recap

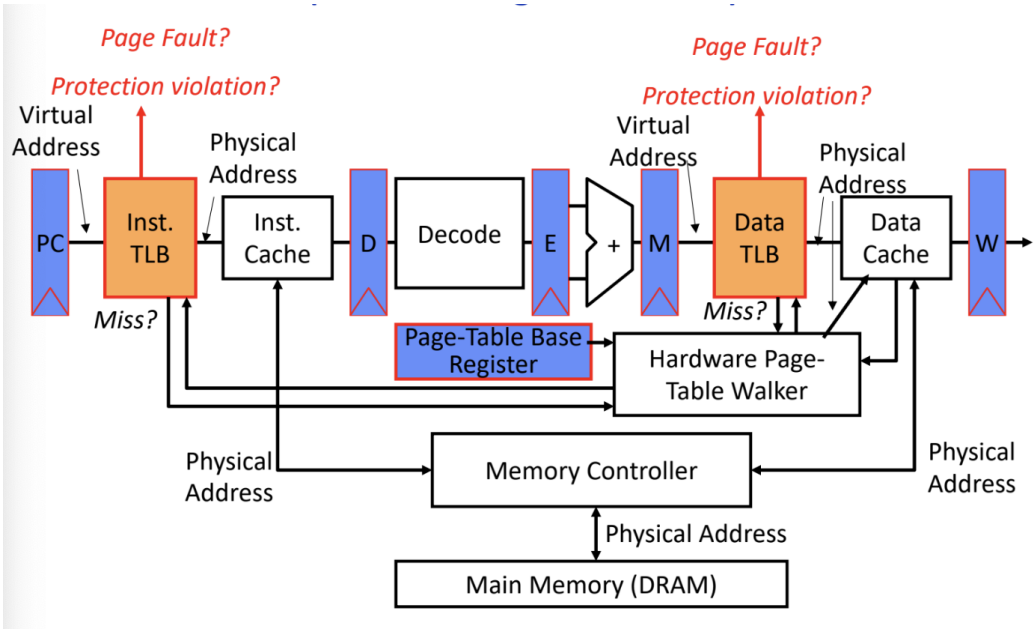
- Protection and translation
- Page based translation and protection avoids need for memory compaction

Modern Vritual memory Systems

- Protection and privacy
    - Own private address space
- Demand paging
    - Run programs larger than primary memory

Page based Virtual memory machine

- Instruction TLB

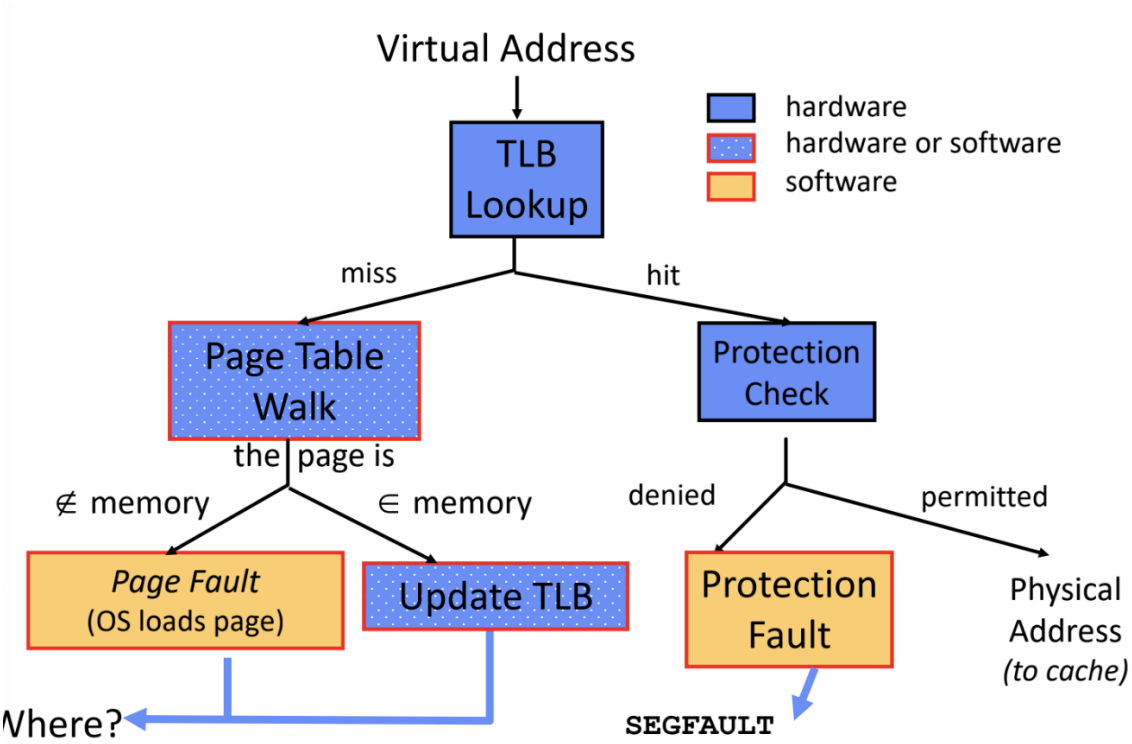


-

- Go through page table walker
- Page fault
    - Not in memory

Address Translation

- Virtual Address
    - TLB lookup



- Page table walk in hardware so we don't interrupt and keep the machine running

Tagged TLB

- When switched to a new process, entires in TLB become invalid
- To reduce this overhead, we allow entires from multiple processes to be stored in TLB

Page Fault Handler

- Referenced page is not in DRAM

Handling VM related exceptions

- TLB miss in hardware
- Page fault in OS
    - Restartable exception so software handler can resume after retrieving page
- Protection violation may abort

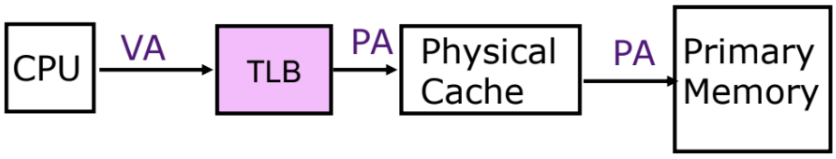
Address Translation in CPU pipeline

- Need to cope with additional latency of TLB
    - Slow down the clock
    - Typical: Pipeline the TLB and cache access
    - Virtual address caches

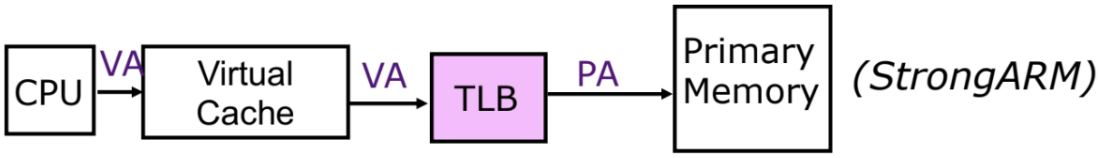
- Parallel TLB/cache access

Virtual Address Caches

- Place the cache before the TLB
- Cut the latency and don't need to do the translation before

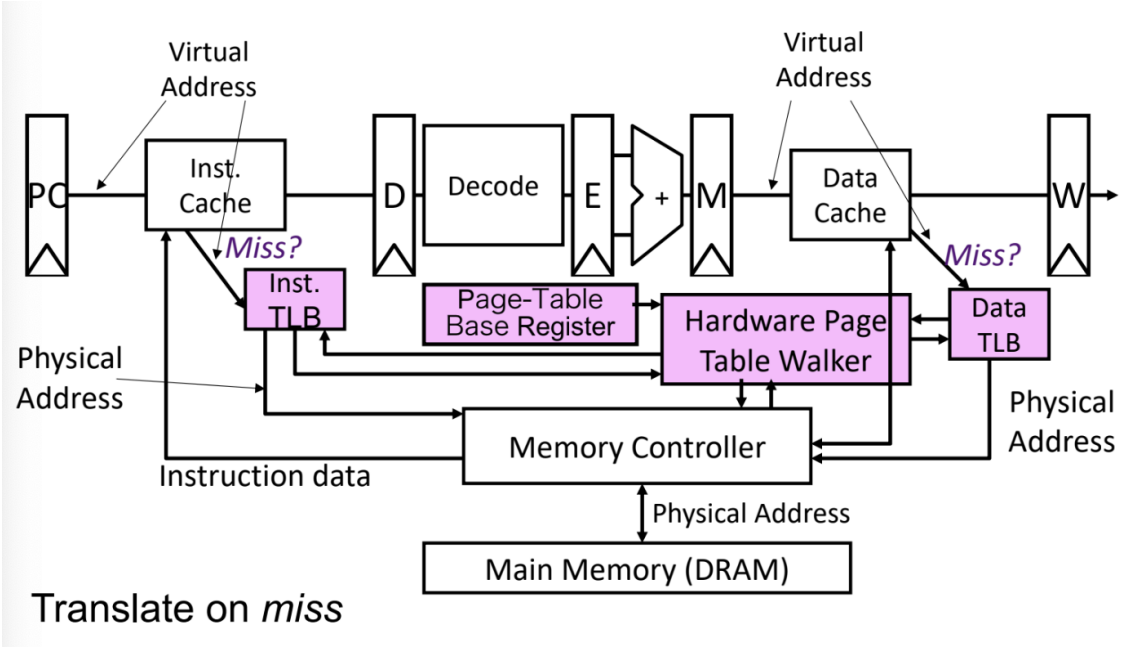


\lternative: place the cache before the TLB



- Flush and repopulate the virtual cache
- Aliasing problems due to sharing of pages
  - Multiple virtual addresses that map to same physical address

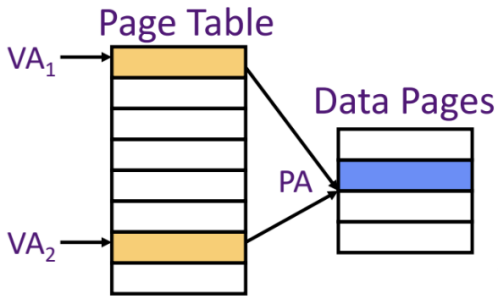
Virtually Addressed Cache



-
- Virtual index/ virtual tag

Aliasing in Virtual Address Cache

- Page Table
- Two virtual addresses map to same physical address



Two virtual pages share one physical page

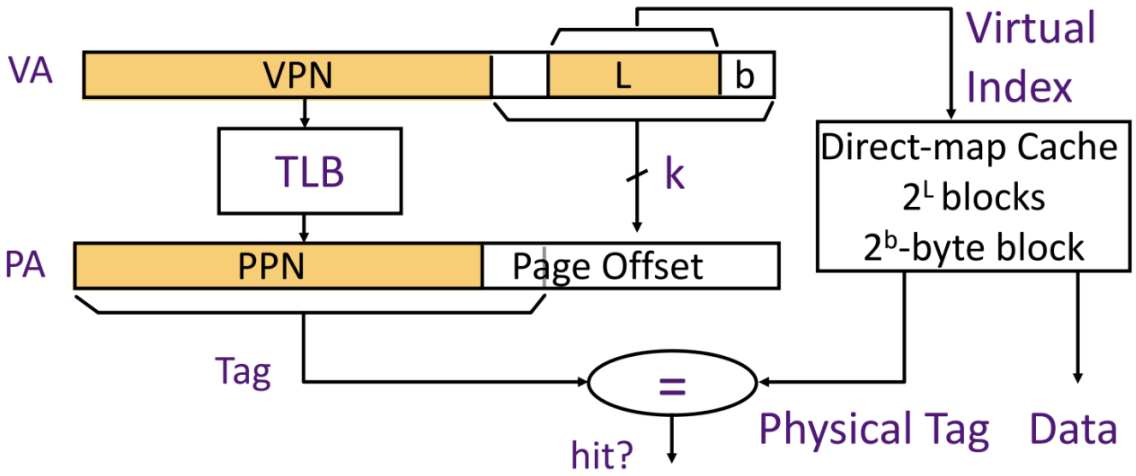
Tag	Data
VA1	1st Copy of Data at PA
VA2	2nd Copy of Data at PA

Virtual cache can have two copies of same physical data. Writes to one copy not visible to reads of other!

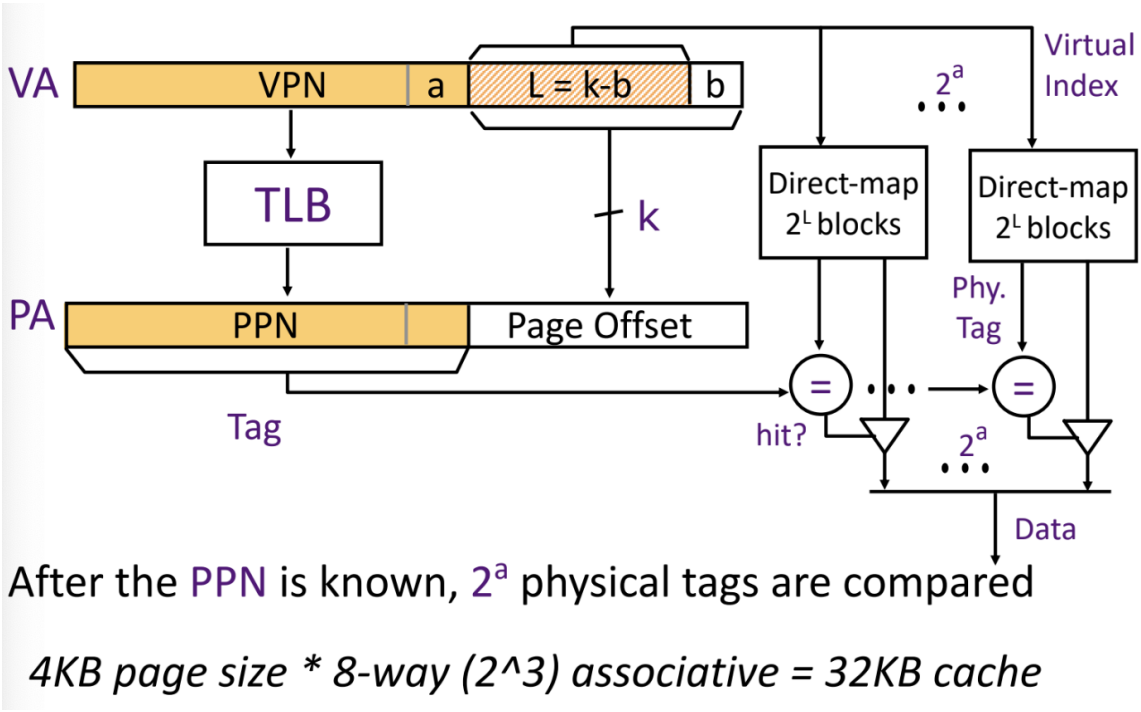
-
- Solution: Direct Mapped Cache
    - Same PA will conflict in direct mapped cache

Concurrent Access to TLB & Cache

- Virtual Index/ PHysical Tag
- Index L is available without consulting the TLB
- Tag comparison is made after both accesses are completed
- Only works when L + b <= k



-
- Virtual index physical tag caches: use associativity



-
- a is bigger, 3 bit
- B is used to map into the byte inside the cache

A solution via second level cache

- Usually a common L2 cache bauskcs up both instruction and data L1 caches
- L2 is inclusive of both instruction and data caches
- Inclusive means L2 has copy of any line in either L1

Anti aliasing using l2

- Make sure there are no collisions in L1 and L2
- Vitually tagged L1

VM Features track historical uses

- Bare machine only physical addresses
- Tim esharing
- Servers/esktops/laptops/smartphones
- Most embedded processors and DSPs provide physical addressing only

Week 6: Lecture 11 Complex Pipelines (2/21)

Improve the pipeline by issuing

Register vs. Memory Dependence

- Can only be determined only after computing the effective address

Week 6: Lecture 12 Out of Order Execution (2/23)

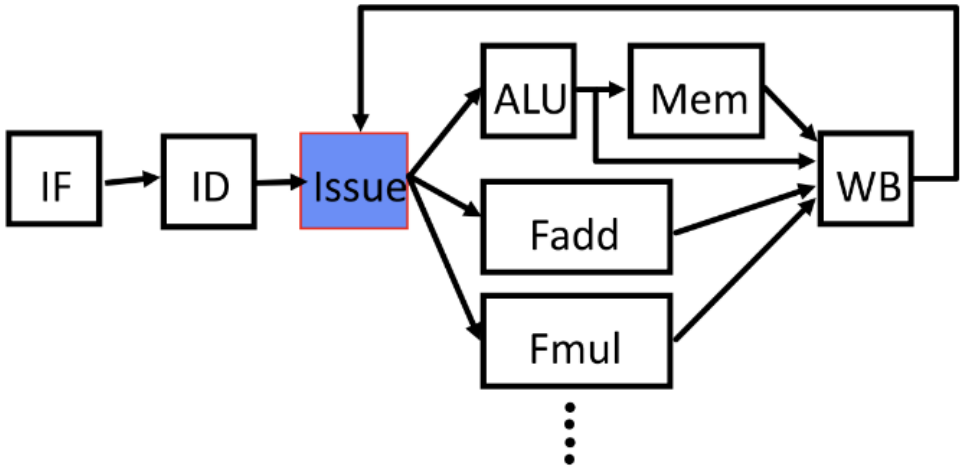
Last time

- Pipelining is complicated by multiple variables, latency functional units

- Out of order and pipliened execution requires tracking depndenceis
- OoO limited by WAR and WAW

Register Renaming

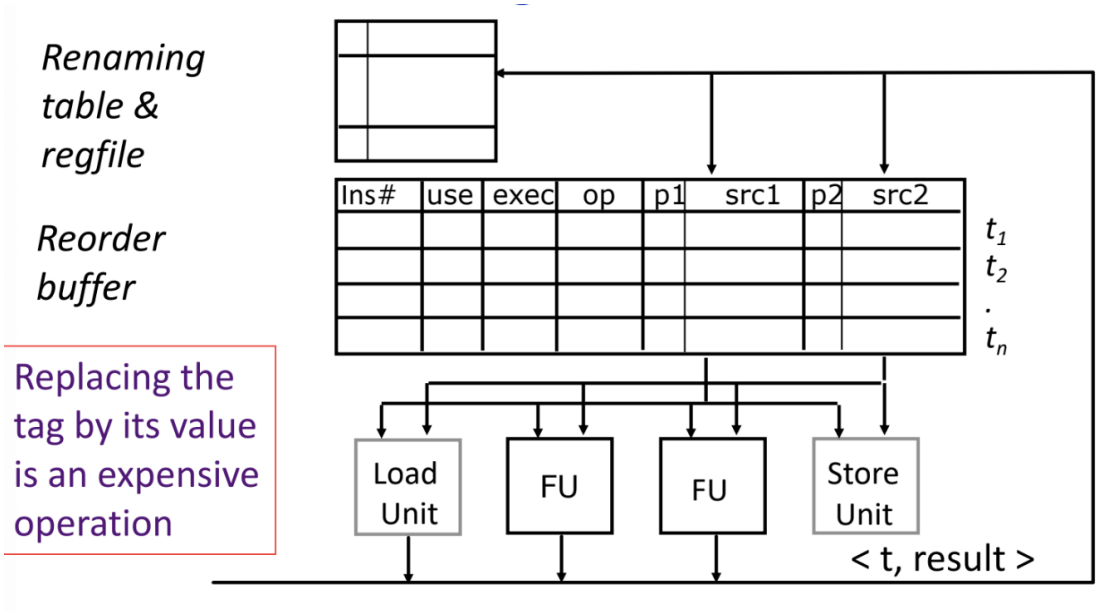
- Decode does register renaming and adds instructions to instructin reorder buffer (ROB)



-

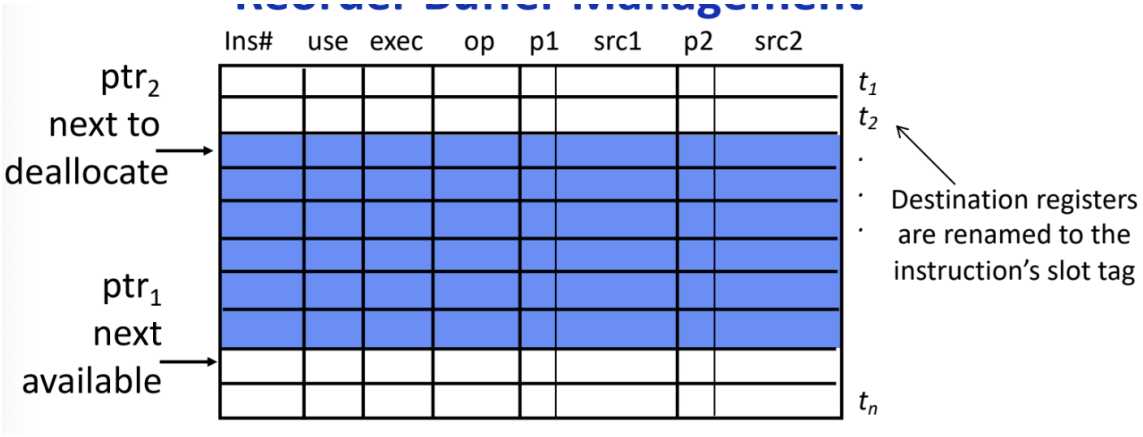
Renaming Structures

- Instruction template is allocated by the Decode which associates tag with register in regfile
- When instruction completes its tag is deallocated
- Reorder buffer



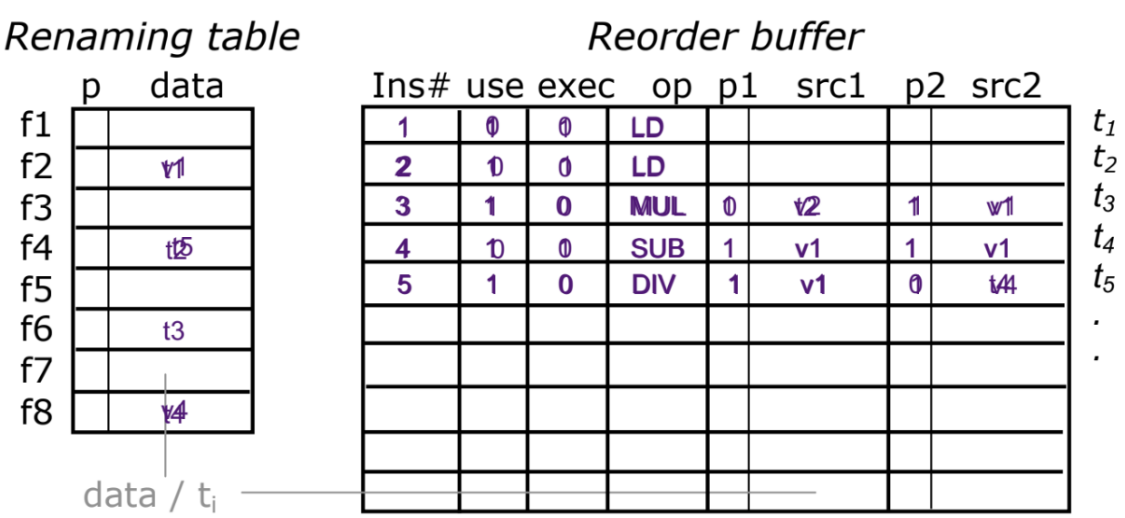
-

Reorder Buffer Management



-
- ROB manager circularly
- Instruction slot is candidate for execution when
  - Holds a valid instruction "use"
  - Not already started , exec bit is clear
  - Both operands are available, p1 and p2

Renaming & Out of order issue



-
- Use t1 as output instead of the previous register (f2)

```
1 FLD     f2,   34(x2)
2 FLD     f4,   45(x3)
3 FMULT.D f6,   f4,   f2
4 FSUB.D  f8,   f2,   f2
5 FDIV.D  f4,   f2,   f8
6 FADD.D  f10,  f6,   f4
```

-
- Start executing whenever both p1 p2 values are available
- Reuse name whenever an instruction completes

IBM 360/91 floating point unit

Out of order fades into background

- Implemented in 1960s, disappeared in 1990s until precise traps, branch prediction

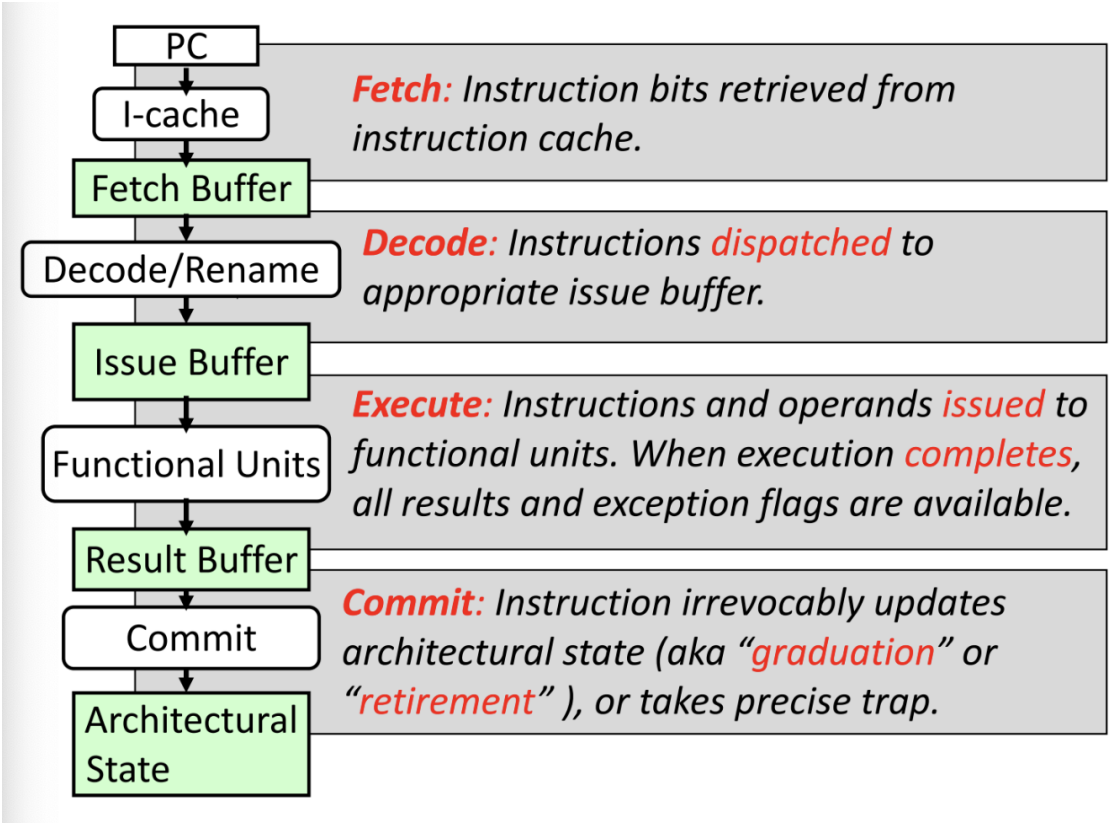
In order commit for precise Traps

- In order instruction fetch and decode and dispatch to reservation stations inside reorder buffer

Separating Completion from Commit

- Re-order buffer holds register results from completion until commit

Phases of instruction execution

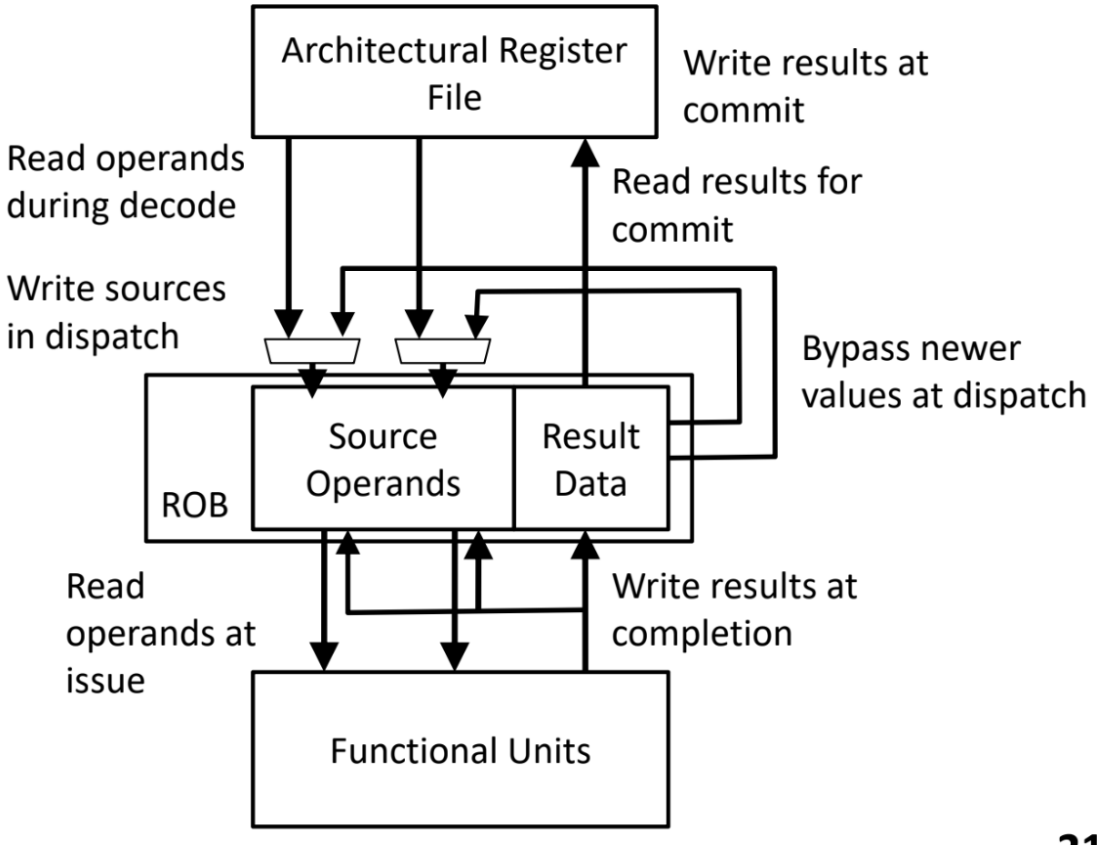


-

In order vs out of order phases

- Some use dispatch to mean issue, not in these lectures

Data Movement in Data-in-ROB Design



-

Lifetime of physical registers

- Old committed and speculative values
- Decouple from ROB entires
- Need to store previous writer because need to be able to go back

Week 7: Lecture 14 VLIW (3/2)

Last time in Lecture

- Phases of Instruction execution
    - Fetch/decode
- Data in ROB design versus unifed physical register design
- Superscalar register renaming
- Instruction level parallelism (ILP)
    - Instruction schedule to facilitate parallel processing
    - Precise Exception: Necessity after virtual pages
- Check instruction dependencies
- Schedule execution

Very Long Instruction Word

- Multiple operations packed into one instructioon
- Each operation slot is for a fixed function

- Constant operation latencies are specified
- Architecture requires guarantee of
    - Parallelism within an instruction -> no cross operation RAW check
    - No data use before data ready -> no data interlocks

VLIW Compiler Responsibilities

- Schedule operations to maximize parallel execution
- Guarantees intra instruction parallelism
- Schedule to avoid data hazards (no interlocks)

Scheduling loop unrolled code

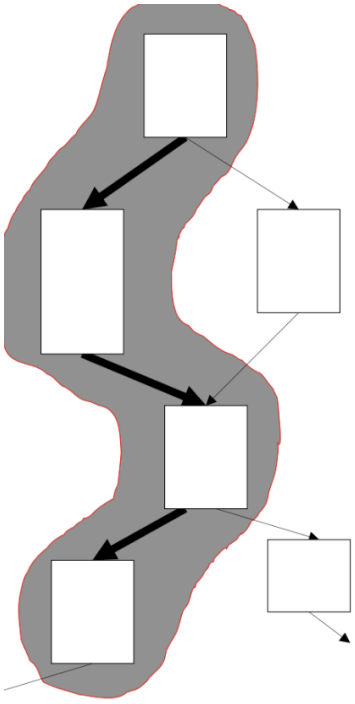
- Overlap different parts of the piepline

What if there are no loops

- Branches limit basic block size in control flow intensive irregular code

Trace Scheduling

- Pick string of basic blocks, a trace, that reporesents most frequent branch path
- Use profiling feedback or compiler heuristics to find common branch paths
- Schedule whole trace at once
- Add fixup code to cope with branches jumping out of trace



-
- Can also be profiled by branch predictor

Problems with Classic VLIW

- Object doe compatibility
    - Recomplile all code for every machine, even for two machines in same generation
- Object code size

        - Instruction padding wastes instruction memory/cache
        - Loop unrolling/software pipelining replicates code
- Scheduling variable latency memory operations
    - Caches and memory bank conflicts impose statically unpredictable variablility
- Knowing branch probabilities
    - Profiling requires significant extra step in build process
- Scheduling for statically unpredictable branches
    - Optimal schedule varies with branch path

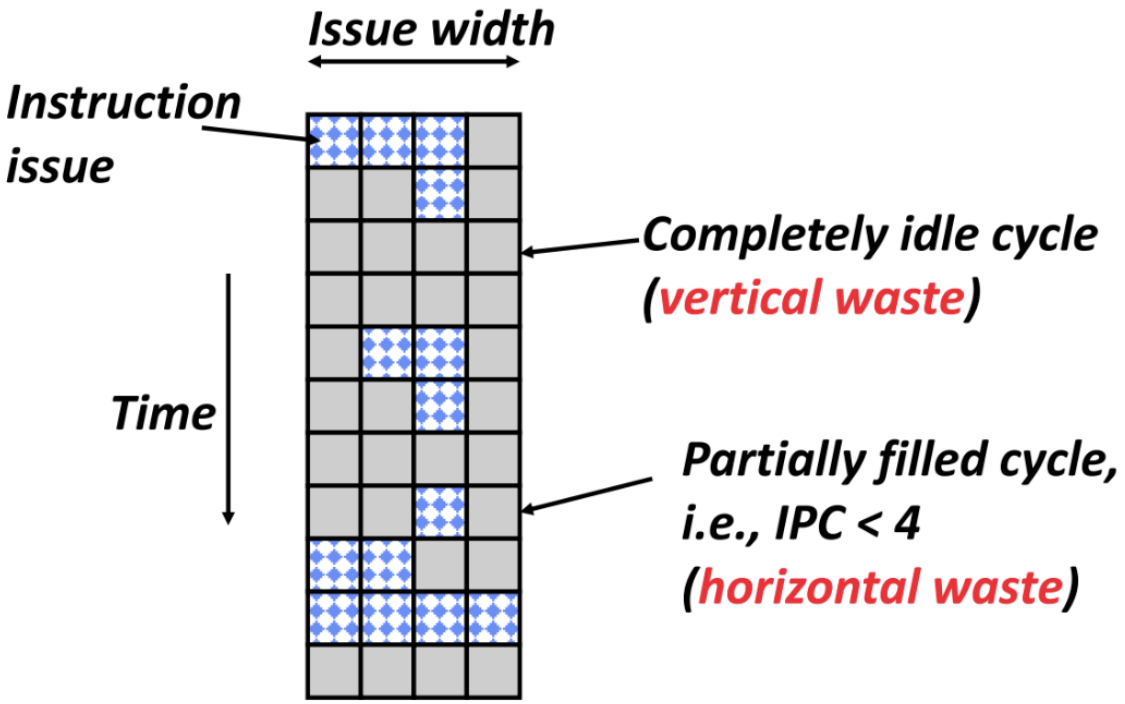
VLIW Instruction Encoding

- Schemes to reduce effect of unused fields
    - Mark parallel groups, compressed format in memory, expand on i cache refill
    - Single op VLIW instruction

Limits of Static Schedculing

- Statictially unpredictable branche s
- Variable memory latency
- Code size explosion
- Compiler complexity
- VLIW has fialed in general-purpose computing arena, close to in order superscalar in complexity
- Successful in embedded DSP
    - More constrained environment, friendlier code
    - Came back for TPU

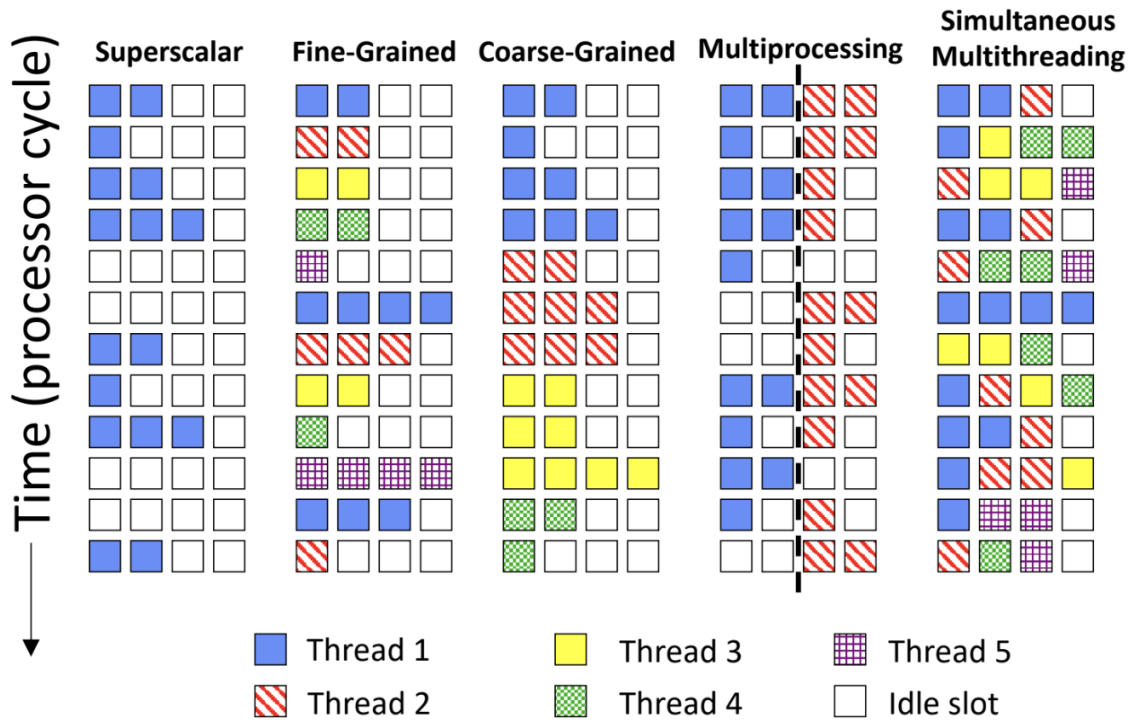
Week 8: Lecture 16 Multithreading (3/9)



- Simultaneously partially filled cycle
- Multi core reduces the vertical waste

SMT Adaptation to parallelism type

- Entire machiine width is shared by all threads
- Entire machine width is available for instruction level parallelism


Superscalar
Fine-Grained
Coarse-Grained
Multiprocessing
Simultaneous Multithreading
Time (processor cycle)
Thread 1
Thread 2
Thread 3
Thread 4
Thread 5
Idle slot

-

-