

CS 152 Computer Architecture and Engineering

CS252 Graduate Computer Architecture

Lecture 9 – Midterm 1 Review

Chris Fletcher

Electrical Engineering and Computer Sciences

University of California at Berkeley

`https://cwfletcher.github.io/`

`http://inst.eecs.berkeley.edu/~cs152`

Today

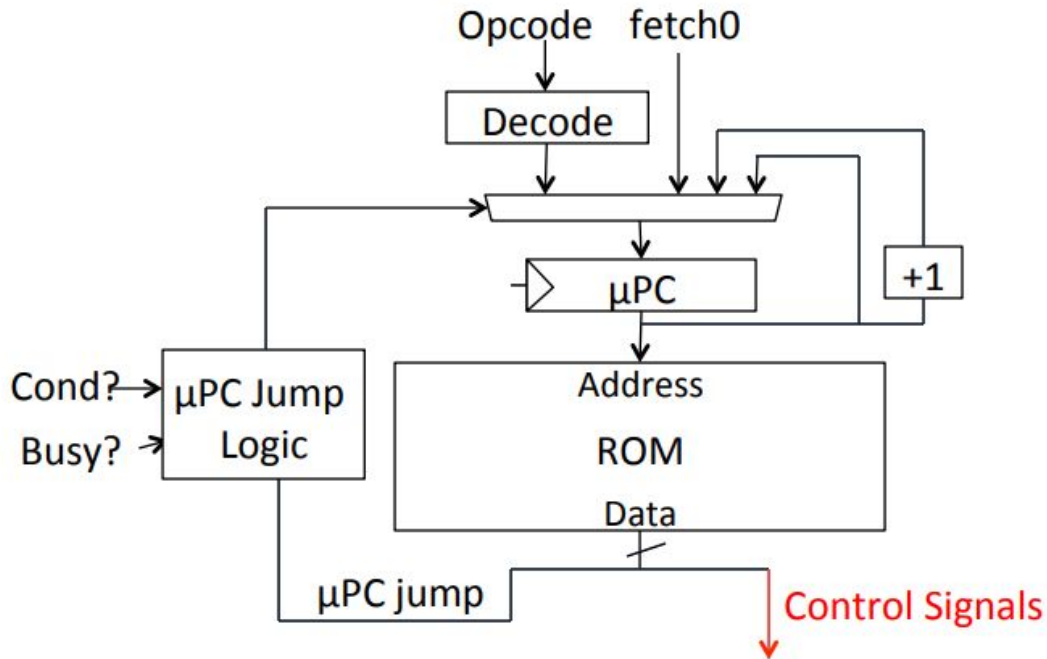
- Microcoding
- Iron Law
- Pipelining
- Hazards
- Exceptions
- Memory Hierarchy
- Basic Caches
- Virtual Memory
- Advanced Cache Optimizations

Warning: This list is non-exhaustive; it is just to seed discussion. Everything covered in the course up to now is fair game.

Microcoding

- **Microcode:** Complex instructions are implemented as “tiny programs” called **microcode** to **control** the **datapath**.
- Separation of **datapath** (arithmetic, data movement) from **control** (sequencing of operations on datapath)
- **Single bus datapath** means all functional units (register file, ALU, memory, etc) share a single bus (group of wires) instead of being connected sequentially after each other.
 - Only one value can be on the single bus at a time
- Historical context: why did microcoding make sense historically?
 - ROM was cheaper and faster than logic
 - Simplified control complexity (helps verification)

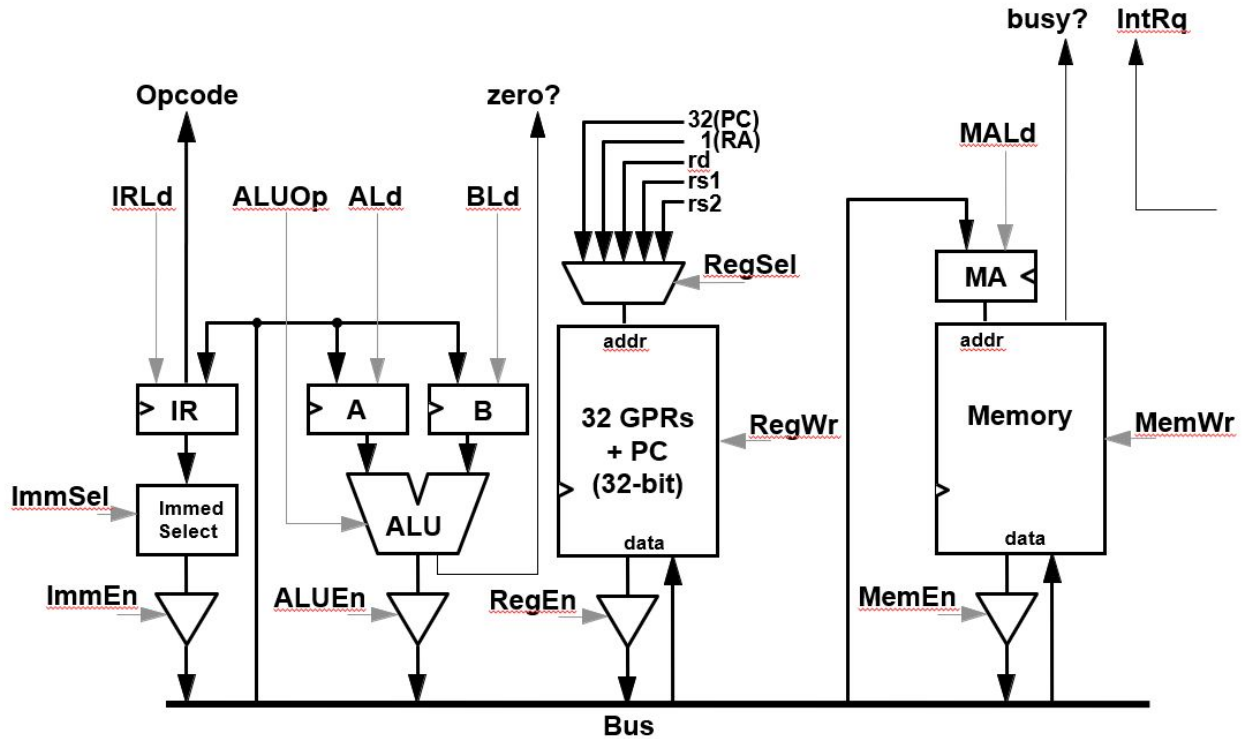
Microcoding – Control Logic



$\mu\text{PC jump} = \text{next} \mid \text{spin} \mid \text{fetch} \mid \text{dispatch} \mid \text{ftrue} \mid \text{ffalse}$

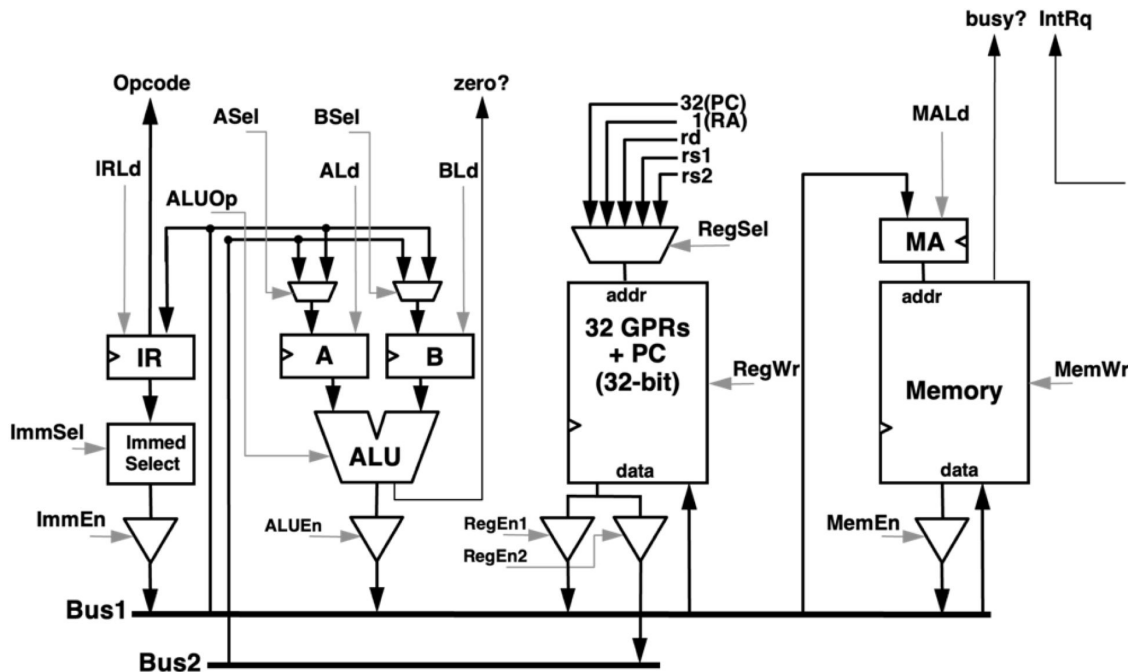
$\mu\text{Br}: \text{N} \mid \text{S} \mid \text{D} \mid \text{J} \mid \text{EZ} \mid \text{NZ}$

Microcoding – Datapath



Microcoding Practice Question [Sp24' MT1]

- A second bus is added, but we only allow regfile to write to this bus.
- Your task is to implement this new instruction:
 - `SetEqualMem rd, rs1, imm(rs2)`
 - This instruction checks if the value in register `rs1` is equal to the value stored in memory address `rs2+imm`. If they are equal, `rd` is set to 1. Otherwise, `rd` is set to 0. This instruction uses a new immediate type called X



[illegible]

[illegible]

Iron Law

- Terms: instructions/program, cycles/instruction, time/cycle
- Know which terms are influenced by:
 - Software
 - Architecture
 - Microarchitecture
 - Process technology
- Evaluate trade-offs between different design decisions
- State all assumptions (be reasonable)

$$- \quad \frac{\text{TIME}}{\text{PROGRAM}} = \frac{\text{INSTRUCTIONS}}{\text{PROGRAM}} \times \frac{\text{CYCLES}}{\text{INSTRUCTION}} \times \frac{\text{TIME}}{\text{CYCLE}}$$

Iron Law Practice Question (Sp25' MT1)

Proposed Change	Instructions per Program	Cycles per Instruction	Time per Cycle
We add a data cache that only stalls the processor on a miss. Assume programs run have high locality. Assume the cache logic is not on the critical path.			

Iron Law Practice Question (Sp25' MT1)

Proposed Change	Instructions per Program	Cycles per Instruction	Time per Cycle
We add a data cache that only stalls the processor on a miss. Assume programs run have high locality. Assume the cache logic is not on the critical path.	Solution: No change, there is no change the ISA or additional delay slots	Solution: Decrease, there will be fewer stalls during load and store operations due to the cache hits	Solution: No change, because from the proposed change description, we assume that the cache logic is not on the critical path.

Pipelining

- What are the bypass paths in a fully-bypassed 5-stage datapath?
 - Does it ever make sense to bypass from the end of a stage to the start of a stage?
 - When can you “remove” a bypass path? What can make a bypass path unnecessary?
- Latency vs throughput/bandwidth vs occupancy
- How to calculate CPI
 - Measure from when the first instruction finishes to when last instruction finishes
 - CPI of 5-stage pipeline
- What is a critical path in synchronous digital logic?

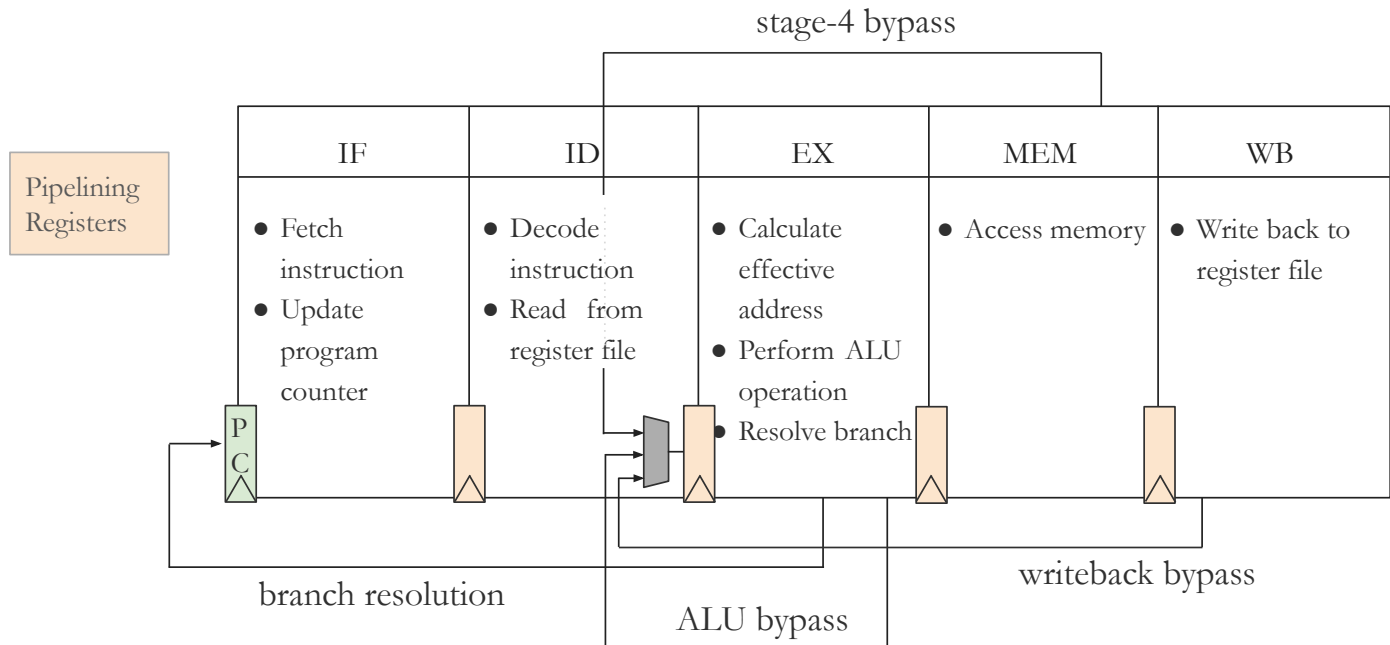
$$\text{CPI} = (\text{last commit cycle} - \text{first commit cycle} + 1) / (\# \text{ committed instructions}).$$

Hazards

- Structural hazards
 - Shared bus, register file ports, unpipelined functional units, etc.
- Data hazards: RAW, WAR, WAW
 - How can WAW hazards arise in an in-order pipeline?
- Control hazards
 - Where in a pipeline are unconditional direct jumps (JAL), unconditional indirect jumps (JALR), and conditional branches resolved?
 - Know what the jump/branch penalties are for a given pipeline

2025 MT1 Q2: Pipelining

Assume memory accesses take one cycle and the branch predictor **predicts not taken**.



Take the following RISC-V snippet, you may assume that accesses to memory addresses in x4, x5 will not cause an exception for the duration of the loop:

```
      1: mv x1, x0
      2: mv x2, x0
loop: 3: lw x6, 0(x4)
      4: lw x7, 0(x5)
      5: mul x6, x6, x7
      6: add x1, x1, x6
      7: addi x4, x4, 4
      8: addi x5, x5, 4
      9: addi x2, x2, 1
     10: bne x2, x8, loop
end:
```

Identify the RAW Hazards in the above piece of code that increase CPI

Take the following RISC-V snippet, you may assume that accesses to memory addresses in x4, x5 will not cause an exception for the duration of the loop:

```
      1: mv x1, x0
      2: mv x2, x0
loop: 3: lw x6, 0(x4)
      4: lw x7, 0(x5)
      5: mul x6, x6, x7
      6: add x1, x1, x6
      7: addi x4, x4, 4
      8: addi x5, x5, 4
      9: addi x2, x2, 1
     10: bne x2, x8, loop
end:
```

Identify the RAW Hazards in the above piece of code

Solution:

4 → 5

Find the CPI of this piece of Code. Note: the instructions below are slightly simplified from the code earlier

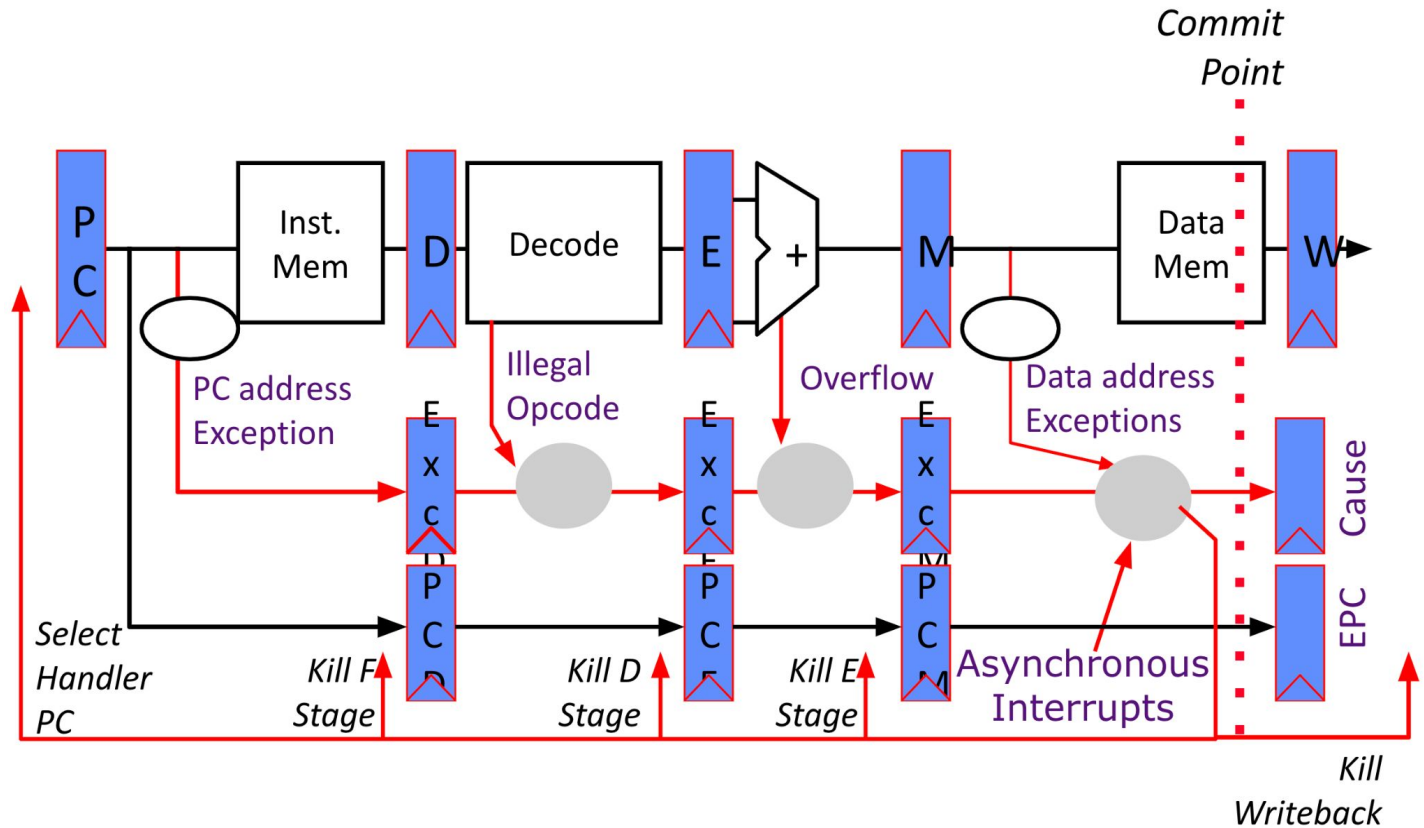
		1	2	3	4	5	6	7	8	9	10	11	12	13
	mv x1, x0	F	D	X	M	W								
	mv x2, x0		F	D	X	M	W							
loop1														
	lw x6, 0(x4)			F	D	X	M	W						
	lw x7 , 0(x5)				F	D	X	M	W					
	mul x6, x6, x7					F	D	D	X	M	W			
	add x1, x1, x6						F	F	D	X	M	W		
	addi x2, x2, 1								F	D	X	M	W	
	bne x2, x8, loop									F	D	X	M	W

$$\begin{aligned}
 \text{CPI} &= (\text{last instruction commit cycle} - \text{first instruction commit cycle} + 1) / (\# \text{ instr}) \\
 &= (13 - 5 + 1) / 8
 \end{aligned}$$

Exceptions

- **Exception:** An unusual internal event caused by program during execution
 - E.g., page fault, arithmetic underflow
- **Interrupt:** An external event outside of running program
 - E.g., a keypress
- **Trap:** Forced transfer of control to supervisor caused by exception or interrupt
- **Precise Exception:** All instructions before the faulting one have completed/committed and nothing after
- Why are precise exceptions useful?

Exceptions



Memory Hierarchy

Trade: storage redundancy and hardware complexity
For: memory-access latency ↓ and bandwidth ↑

Under what assumption?

Trade: storage redundancy and hardware complexity
For: memory-access latency ↓ and bandwidth ↑

Under what assumption? **Locality.**

Locality

If an address is accessed...

Temporal: it is likely to be accessed again in the future

Spatial: adjacent addresses are likely to be accessed in the future

Locality

If an address is accessed...

Temporal: it is likely to be accessed again in the future

Spatial: adjacent addresses are likely to be accessed in the future

Mechanism: memory hierarchy, arranged small/fast → big/slow, with policies to exploit locality

Access Algorithm

level.Read(Address) → Cache Line

1. Break **Address** into **Tag**, **Index**, **Offset**
2. Select set **Index** from cache
3. Search set **Index** for cache line tagged with **Tag**
 - 3.1. If not found, nextLevel.Read(Address)
 - 3.1.1. If set **Index** is full, policy.Evict()
 - 3.1.2. Write (**Tag**, data) into cache
 - 3.2. Return data

Write-allocate? Write-through? Inclusive?

Access Algorithm

level.Read(**Address**)

1. Break **Address** into **Tag**, **Index**, **Offset**
2. Select set **Index** from cache
3. Search set **Index** for cache line tagged with **Tag**
 - 3.1. If not found, nextLevel.Read(**Address**)
 - 3.1.1. If set **Index** is full, policy.Evict()
 - 3.1.2. Write (**Tag**, data) into cache
 - 3.2. Return data

Write-allocate? **True** Write-through? **Property of level.Write** Inclusive? **Property of LLC**

Access Policies

Write-through

Writes to LX
~immediately
propagate to L(X+1)

Write-back

Writes to LX do not
immediately
propagate to L(X+1)

Write-allocate

A miss to address X
returns the data for X
and places X in the
cache

No-write-allocation

A miss to address X
returns the data for X
and does NOT place
X in the cache

Inclusive

Address cached in
LX implies address
cached in L(X+1),
L(X+2), ...

Non-inclusive

Address may exist in
LX but not L(X+1)

Semantics of a write-through, write-allocate cache hierarchy?
What about ^, 2-layer inclusive?

Miss Types

These definitions are **not mutually exclusive**. Evaluate them **in this order**.

#	Type	How you'd know	Logical definition <small>(If the trace was replayed under these circumstances, this access would be a miss)</small>
1	Compulsory	Line has never been accessed before	Misses that would occur in an infinite-size fully-associative cache
2	Capacity	Miss also occurs under fully-associative cache	Misses that would occur in full-associative cache with same size and replacement policy
3	Conflict	Miss does not also occur under fully-associative cache	Misses that occur when a set is fully occupied

Miss Types: Practice

loop:

```
lw t0, 0(s0)
lw t1, 64(s0)
lw t2, 16(s0)
lw t3, 32(s0)
lw t4, 0(s0)
lw t5, 48(s0)
```

```
addi s1, s1, -1
bnez s1, loop
```

done:

```
lw t0, 48(s0)
lw t1, 64(s0)
lw t2, 0(s0)
```

Cache Properties

- 64B
- 16B/line
- Direct-Mapped

Capacity misses defined
against 64B Fully-Associative
with LRU

64B Direct-Mapped: 4 Sets, 16B/Line. Capacity misses defined against 64B Fully-Associative (LRU).

loop:

```
lw t0, 0(s0)
lw t1, 64(s0)
lw t2, 16(s0)
lw t3, 32(s0)
lw t4, 48(s0)
lw t5, 48(s0)
```

```
addi s1, s1,
-1
bnez s1, loop
```

done:

```
lw t0, 48(s0)
lw t1, 64(s0)
lw t2, 0(s0)
```

s0=0, s1=2

	Iteration 1						Iteration 2						Tail		
Tag (FA)	0	4	1	2	0	3	0	4	1	2	0	3	3	4	0
Set (DM)	0	0	1	2	0	3	0	0	1	2	0	3	3	0	0
Type	Cap	Cap	Cap	Cap	Cap	Cap	Cap	Cap							

0 4 0
16
32
48

Direct-Mapped

0 64 48
16
32

Fully-Associative

Capacity: miss also occurs under fully-associative cache
 Conflict: Miss does not also occur under fully-associative cache

64B Direct-Mapped: 4 Sets, 16B/Line. Capacity misses defined against 64B Fully-Associative LRU.

```

loop:
  lw  t0, 0(s0)
  lw  t1, 64(s0)
  lw  t2, 16(s0)
  lw  t3, 32(s0)
  lw  t4, 0(s0)
  lw  t5, 48(s0)

```

```

  addi s1, s1, -1
  bnez s1, loop

```

```

done:
  lw  t0, 48(s0)
  lw  t1, 64(s0)
  lw  t2, 0(s0)

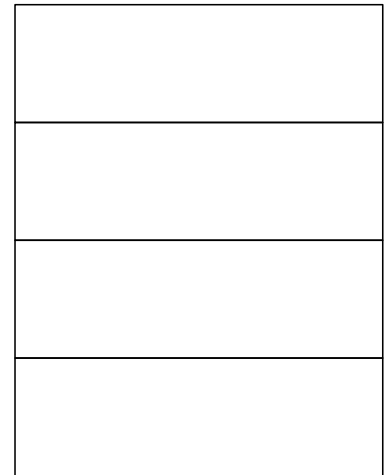
```

s0=0, s1=2

	Iteration 1						Iteration 2						Tail		
Tag (FA)	0	4	1	2	0	3	0	4	1	2	0	3	3	4	0
Set (DM)	0	0	1	2	0	3	0	0	1	2	0	3	3	0	0
Type	Cmp	Cmp	Cmp	Cmp	Cnf	Cmp	Hit	Cnf	Hit	Hit	Cnf	Hit	Hit	Cap	Cnf



Direct-Mapped



Fully-Associative

Capacity: Miss also occurs under fully-associative cache
 Conflict: Miss does not also occur under fully-associative cache

Optimization

Programs contain rich, high-level structure, but caches only see the sequence of memory addresses produced by that structure.

Locality is useful when structure remains visible in the resulting access pattern.

Optimization

Programs contain rich, high-level structure, but caches only see the sequence of memory addresses produced by that structure.

Locality is useful when structure remains visible in the resulting access pattern.

Programming for cache locality =

arranging computation and data

so that semantic relationships in the program

manifest as spatial and temporal relationships in the access pattern.

Optimization: Fusion

```
for (int i = 0; i < 16384; i++) {  
    tmp[i] = A[i] * 3 + 1;  
}
```

```
for (int i = 0; i < 16384; i++) {  
    B[i] = tmp[i] * tmp[i];  
}
```

```
for (int i = 0; i < 16384; i++) {  
    tmp[i] = A[i] * 3 + 1;  
    B[i] = tmp[i] * tmp[i];  
}
```

Length-16384 arrays of 4-byte integers.

32 KiB, empty, fully associative, 64-byte cache lines and LRU replacement.

Arrays:
64K

Optimization: Fusion

```
for (int i = 0; i < 16384; i++) {
```

```
    tmp[i] = A[i] * 3 + 1;
```

```
}
```

tmp[0]

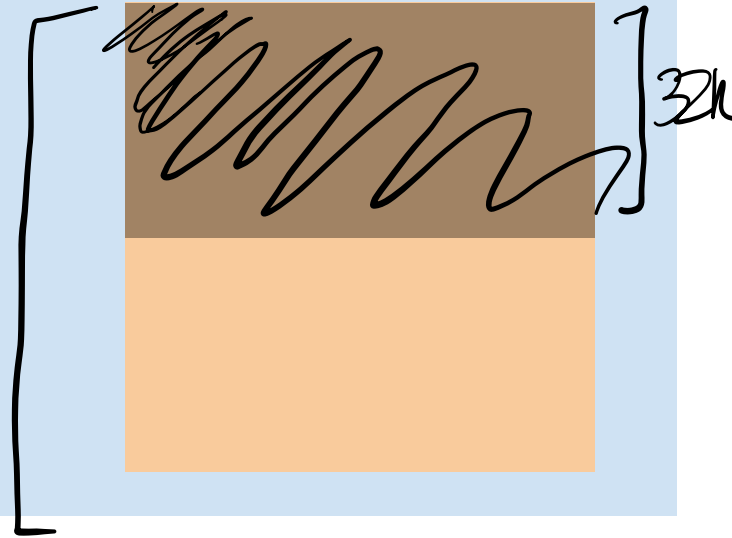
16K

```
for (int i = 0; i < 16384; i++) {
```

```
    B[i] = tmp[i] * tmp[i];
```

```
}
```

tmp[0]



How many iterations between writing `tmp[i]` and reading it again?
Will `tmp[i]` be in cache when the write happens?

Optimization: Fusion

```
for (int i = 0; i < 16384; i++) {  
    tmp[i] = A[i] * 3 + 1;  
    B[i] = tmp[i] * tmp[i];  
}
```

Improve **temporal locality**

By bringing a producer and consumer of the same data
Closer together in **time**.

Optimization: Interchange

```
// i outer, j inner
long sum = 0;
for (int i = 0; i < 512; i++) {
    for (int j = 0; j < 512; j++) {
        sum += A[i][j] + B[i][j] + C[j][i];
    }
}
```

```
// j outer, i inner
long sum = 0;
for (int j = 0; j < 512; j++) {
    for (int i = 0; i < 512; i++) {
        sum += A[i][j] + B[i][j] + C[j][i];
    }
}
```

512 x 512 row-major arrays of 4-byte integers.

32 KiB, empty, fully associative, 64-byte cache lines and LRU replacement.

Assume sum remains in a register.

Optimization: Interchange

```
// j outer, i inner  
long sum = 0;  
for (int j = 0; j < 512; j++) {  
    for (int i = 0; i < 512; i++) {  
        sum += A[i][j] + B[i][j] + C[j][i];  
    }  
}
```

Loading `A[0][0]` brings in `A[0][0..15]`, but the next access is `A[1][0]`.



In Cache



Accessed
Explicitly

A

Optimization: Interchange

```
// j outer, i inner  
  
long sum = 0;  
  
for (int j = 0; j < 512; j++) {  
    for (int i = 0; i < 512; i++) {  
        sum += A[i][j] + B[i][j] + C[j][i];  
    }  
}
```

Loading $A[0][0]$ brings in $A[0][0..15]$, but the next access is $A[1][0]$.
By the time we reach $A[0][1]$, we've traversed all 512 rows of both A and B.
That's at least 1024 lines. Our cache has $32\text{KiB}/64\text{B} = 512$.

Optimization: Interchange

```
// j outer, i inner  
long sum = 0;  
for (int j = 0; j < 512; j++) {  
    for (int i = 0; i < 512; i++) {  
        sum += A[i][j] + B[i][j] + C[j][i];  
    }  
}
```

C[j][i]?

Loading A[0][0] brings in A[0][0..15], but the next access is A[1][0].
By the time we reach A[0][1], we've traversed all 512 rows of both A and B.
That's at least 1024 lines. Our cache has 32KiB/64B = 512.

Optimization: Tiling

```
for (int i = 0; i < 512; i++) {  
    for (int j = 0; j < 512; j++) {  
        for (int k = 0; k < 512; k++) {  
            C[i][j] += A[i][k] * B[k][j];  
        }  
    }  
}
```

512 x 512 row-major arrays of 4-byte integers.

32 KiB, empty, fully associative, 64-byte cache lines and LRU replacement.

How much of A, B, and C is accessed before the program returns to previously used data?

Optimization: Tiling

```
for (int i = 0; i < 512; i++) {  
    for (int j = 0; j < 512; j++) {  
        for (int k = 0; k < 512; k++) {  
            C[i][j] += A[i][k] * B[k][j];  
        }  
    }  
}
```

512 x 512 row-major arrays of
4-byte integers.

32 KiB, empty, fully associative,
64-byte cache lines and LRU
replacement.

Index order != loop order.

Working set for 32x32-tile?

Is it small enough to fit in the 32 KiB L1 cache?

Optimization: Tiling

```
for (int ii = 0; ii < 512; ii += 32)
    for (int jj = 0; jj < 512; jj += 32)
        for (int kk = 0; kk < 512; kk += 32)
            for (int i = ii; i < ii + 32; i++)
                for (int j = jj; j < jj + 32; j++)
                    for (int k = kk; k < kk + 32; k++)
                        C[i][j] += A[i][k] * B[k][j];
```

3 matrices x 32^2 integers/matrix x 4 bytes/integer = 12 KiB tile working set $\leq$ 32 KiB cache.

Visualizer: <https://tile-cache-matrix-lab.safinsingh.chatgpt.site/>

Optimization: Tiling

Why not small tiles? They'll always fit?

Optimization: Tiling

Why not small tiles? They'll always fit?

For a $b \times b$ tile:

- $3b^2$ data
- b^3 operations

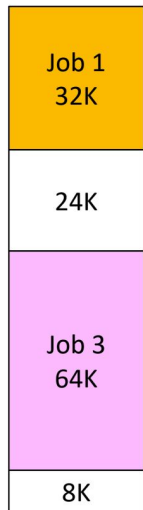
Operations/data (reuse) $\sim b$

Virtual Memory

- Why virtual memory?
 - Process Isolation
 - Programs don't need to know where they are physically loaded
 - We can give processes more virtual memory than physical RAM
 - Sharing memory efficiently

Virtual Memory

- Segment vs Page Table
 - Segment: suffer from *external fragmentation*
 - Page Table: suffer from *internal fragmentation*



Can't run Job 4, as not enough contiguous space. Must compact.

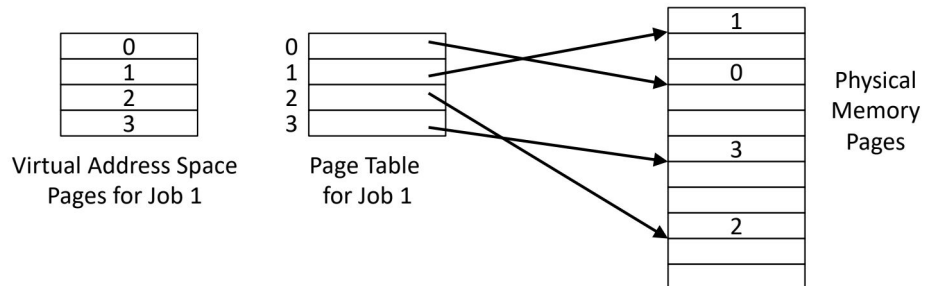


Segment Approach & External Fragmentation

- Program-generated (*virtual or logical*) address split into:

Page Number	Offset
-------------	--------

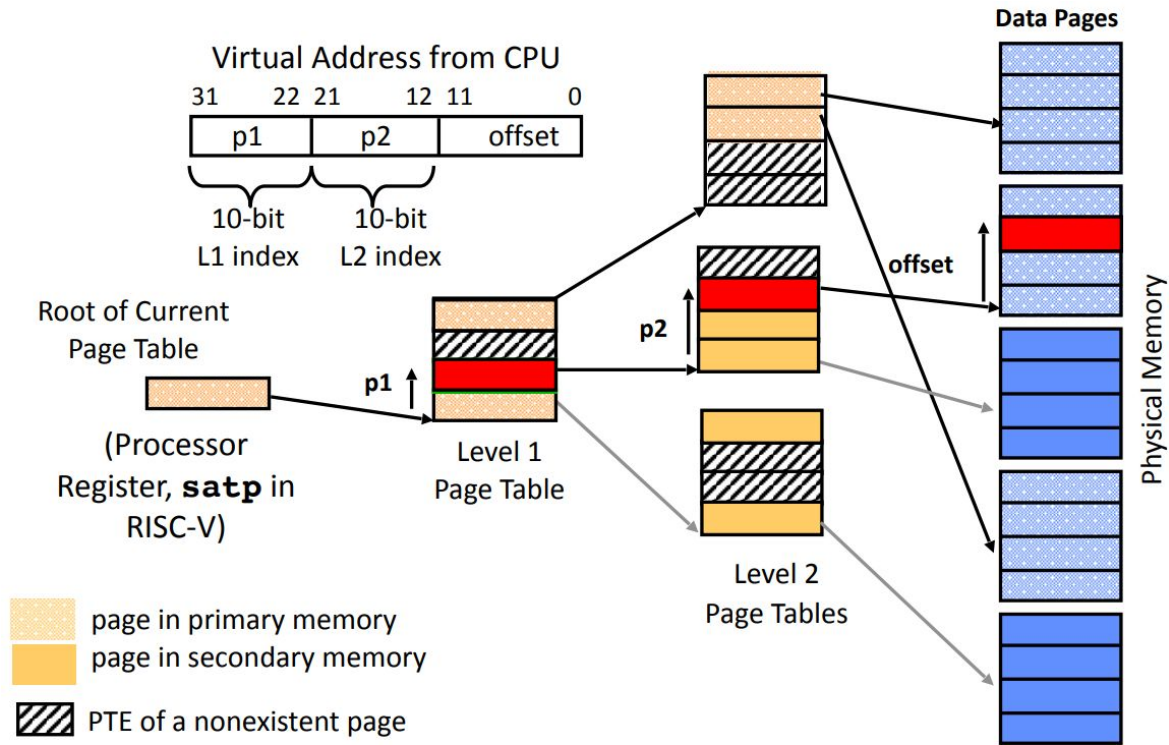
- Page Table contains physical address of start of each fixed-sized page in virtual address space



- Paging makes it possible to store a large contiguous virtual memory space using non-contiguous physical memory pages

Page Table Approach & Internal Fragmentation

Hierarchical Page Table



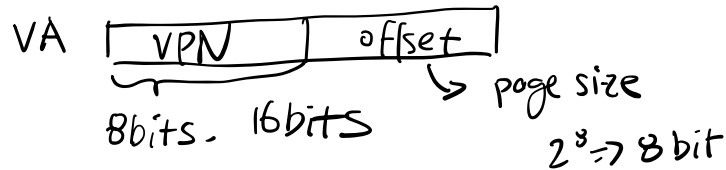
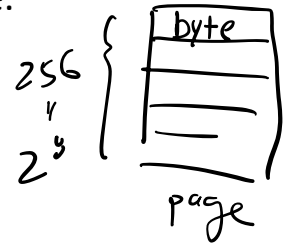
RISC-V Sv32 Virtual Memory Scheme

Virtual Memory Practice Question [Sp22' MT1]

Consider a computer with 256-byte pages, 16-bit virtual addresses, and 16-bit PTEs (4 bits of which are used for valid and protection). The computer uses two-level hierarchical page tables. Assume the machine is byte-addressable.

(i) How many virtual pages can be addressed by this computer?

$$2^8 = 256 \text{ pages}$$



$$2^{10} \text{ B} = 1 \text{ MB}$$

(ii) What is the maximum size of the physical memory that can be supported by this computer? $2^{30} \text{ B} = 1 \text{ GB}$

PTE: $16 \text{ bit} - 4 = 12 \text{ bits}$.

2^{12} physical pages, $1 \text{ page} = 2^8$

Physical mem: $2^{12} \times 2^8$
 $= 2^{20} = 1 \text{ MB}$

(iii) Suppose that a running program is currently using 300 bytes of memory. What is the smallest possible number of PTEs and PTPs that can be valid in the page table(s) of this program?

$\left\lceil \frac{300 \text{ B}}{256 \text{ B}} \right\rceil = 2 \text{ pages}$

at least 2 PTEs in 2nd level PT

at least 1 PTP in 1st level PT





Virtual Memory Practice Question [Sp22' MT1]

Consider a computer with 256-byte pages, 16-bit virtual addresses, and 16-bit PTEs (4 bits of which are used for valid and protection). The computer uses two-level hierarchical page tables. Assume the machine is byte-addressable.

(i) How many virtual pages can be addressed by this computer?

$$2^{16} / 2^8 = 256$$

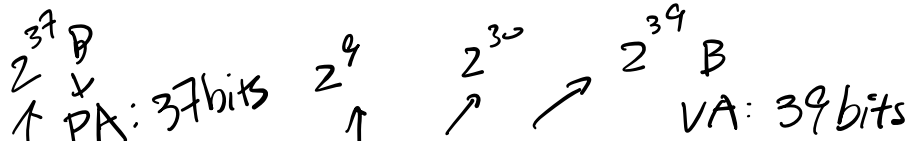
(ii) What is the maximum size of the physical memory that can be supported by this computer?

$$2^{12} * 256 = 1 \text{ MB}$$

(iii) Suppose that a running program is currently using 300 bytes of memory. What is the smallest possible number of PTEs and PTPs that can be valid in the page table(s) of this program?

Since the page size is 256 Bytes, there must be at least two valid pages in memory. This means at least 2 PTEs must be valid, and since we have a 2-level page table, at least 1 PTP must be valid

Virtual Memory Practice Question [Sp26' MT1]



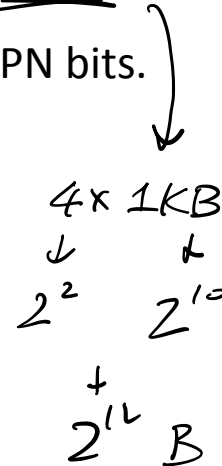
Suppose we have a 512 GiB byte-addressable virtual memory space with a 128 GiB byte-addressable physical memory space, and 4 KiB pages.

Calculate the number of page offset bits, VPN bits, and PPN bits.

VPN bits: $39 \text{ bits} - 12 \text{ bit} = 27 \text{ bits}$

PPN bits: $37 - 12 = 25 \text{ bits}$

Offset bits: 12 bits



Virtual Memory Practice Question [Sp26' MT1]

Suppose we have a 512 GiB byte-addressable virtual memory space with a 128 GiB byte-addressable physical memory space, and 4 KiB pages.

Calculate the number of page offset bits, VPN bits, and PPN bits.

VPN bits: 27 bits

PPN bits: 25 bits

Offset bits: 12 bits

Advanced Cache Optimizations

- Victim Cache
- Way-predicting
- Early restart
- Critical word first
- Non-blocking
- Hardware prefetching
- Software prefetching
- Compiler optimization

Victim Cache

- A small, fully associative cache in addition to the L1 Data Cache to hold lines that were evicted recently

Why? When?

- Access patterns with frequent conflict misses or recently evicted blocks are used soon

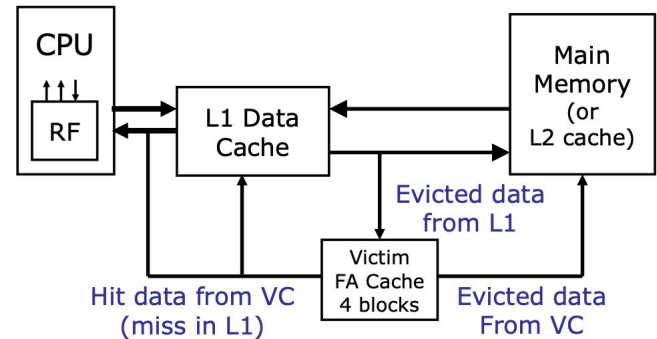


Figure H7-A: A Victim Cache Organization

Early Restart

- As soon as the requested word of the block arrives, we send that data to the CPU and let the CPU continue execution

Why? When?

- Reduces miss penalty
- Best when requested word arrives before the full block
- CPU doesn't wait for the entire cache block

Critical Word First

- Request the 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.

Why? When?

- Reduces miss penalty
- Blocks are large

Difference between Critical Word First and Early Restart:

Early restart: resume when the requested word naturally arrives

Critical word first: request that word first so it arrives sooner

Way Prediction

- Keep extra bits in cache to predict the “way” of the next access
- Access predicted way first
- If miss, access other ways like in set associative caches
- Fast hit when prediction is correct

Why? When?

- Reduce hit time of set-associative caches
- Sometimes a particular way is very predictable

Non-blocking Cache

- Non-blocking cache allow data cache to continue to supply cache hits during a miss

Why? When?

- Can hide some miss latency, especially when memory latency is high
- Multiple misses can be overlapped using a non-blocking cache

SW Prefetching

- Use compiler to fetch data early before it is needed

Why? When?

- Goal: while you are busy computing, you can fetch from memory and hide some memory latency
- Works best for predictable access patterns like arrays/loops

Example from FA26 HW2:

```
long sum = 0;
long matrix[64][64];
for (int i = 0; i < 64; i++) {
    for (int j = 0; j < 64; j++) {
        if (prefetch_cond) prefetch((char *)&matrix[i][j] + OFFSET);
        sum += matrix[i][j];
    }
}
```

HW Prefetching

- Piece of HW recognizes the next few elements that the program might need based on this stream, and prefetches into the processor's cache

Why? When?

- Hides memory latency and you do not need explicit prefetch instructions
- Access patterns are sometimes very predictable