CS 3410 Lab 10.5 (Review)

Fall 2025

Cornell Bowers C·IS
Computer Science

Agenda

1. RISC-V Calling Convention (Review)
2. Caches (Review)
3. Virtual Memory (Review)

RISC-V Calling Convention (Review)

RISC-V Calling Convention Review

- Arguments go in registers `a0-a7`
- Return values go in `a0-a1`
- Return *address* goes in `ra`
- Callee-saved registers are: `s0-s11`
- Stack pointer is saved in `sp`

Prologue:	`Epilogue:`
1. Adjust stack pointer 2. Save return address (`ra`) 3. Save callee-saved registers (i.e. save whichever of `s0-s11` are used)	1. Restore saved registers 2. Restore return address (`ra`) 3. Restore stack pointer 4. Return to the caller using `ret`

Example: leaf function (no function calls in function body)

Objective: Add 1 to the argument and return the result

```
int addOne(int i) {
  return i + 1;
}
```

Prologue:

1. Adjust stack pointer
2. Save return address (`ra`)
3. Save callee-saved registers

`Epilogue:`

1. Restore saved registers
2. Restore return address (`ra`)
3. Restore stack pointer
4. Return to the caller using `ret`

Add One Example (Solution)

```
int addOne(int i) {
  return i + 1;
}
```

```
addOne:
    # Prologue.
    addi sp, sp, -8  # Push the stack frame.
    sd   ra, 0(sp)   # Save return address.

    # Body.
    addi a0, a0, 1

    # Epilogue.
    ld   ra, 0(sp)   # Restore return address.
    addi sp, sp, 8   # Pop the stack frame.
    ret
```

Caches (Review)

Types of caches

- **Direct-mapped**
- Fully-associative
- Set-associative

Today we're only reviewing direct-mapped caches!

Cache Parameters

- Tag: High-order bits used to compare addresses of the same cache mapping
- Index: Bits that determine where in the cache an address can go
- Offset: Low-order bits to select a byte within a cache block

Address breakdown:

Direct Mapped Cache

Each address maps to exactly 1 cache block

ex) 4 byte direct mapped cache

- 4 blocks, 1 byte each
- Need 4 = 2^2 indices, so 2 index bits

CACHE

index	Tag	Data
11	00	D
10	00	C
01	00	B
00	00	A

tag|index
1101

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

The access algorithm

CACHE

index	V	Tag	Data
11	0	00	D
10	0	00	C
01	0	00	B
00	0	00	A

Data in the cache already

Data you're trying to load in the register

tag | index

1101

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

1. Split the address between **tag** `t` and **index** `i`
2. Check the entry `i`
3. Is it valid? If no, cache miss!
4. If the bit is valid, then is it the tag `t`? If no, cache miss!
5. Otherwise, cache hit!

x

Increasing the block size for a Direct-Mapped cache

- Each cache block now stores multiple bytes
- Use offset to access each byte

tag | index | offset

XXXX

1. Check the index in the $ (cache)
2. Check the tag
3. Check the valid bit

CACHE

index	V	Tag	Data
11	0	x	X | X
10	0	x	X | X
01	0	x	X | X
00	0	x	X | X

Fully Associative Cache

Each address can map to any cache block

ex) 8 byte fully associative cache

- 4 blocks, 2-bytes each (use offset to access each byte)
- 4-bit addresses, no need for index bit

CACHE

V	Tag	Data
0	010	E | F
0	000	A | B
0	001	C | D
0	011	G | H

tag|offset
1101

No more index!

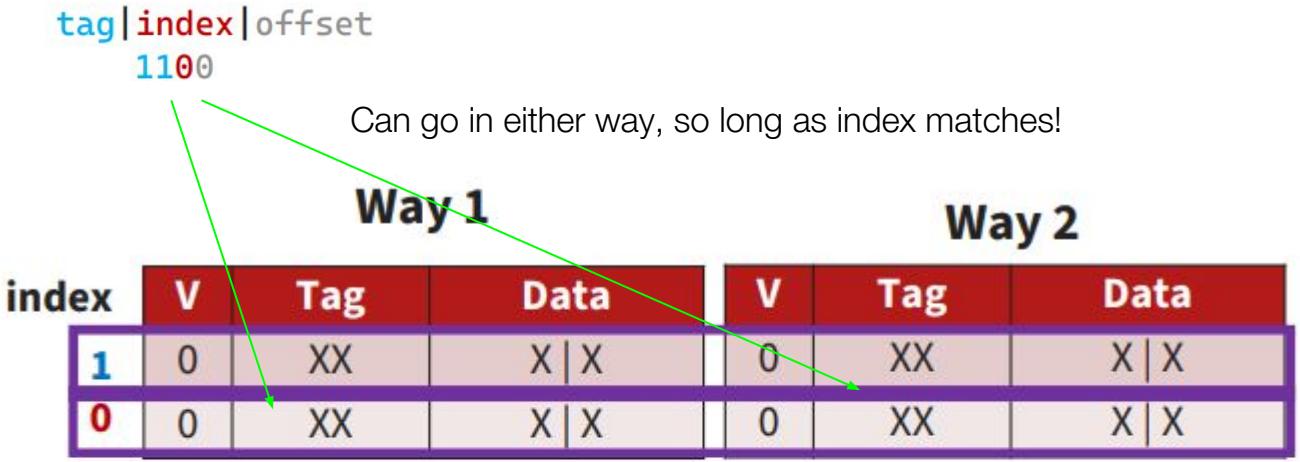
- Any address, any entry
- 4-bit addresses
- 2-byte blocks

MEMORY

addr	Data
1111	P
1110	O
1101	N
1100	M
1011	L
1010	K
1001	J
1000	I
0111	H
0110	G
0101	F
0100	E
0011	D
0010	C
0001	B
0000	A

Set Associative Cache

- Divide cache into **sets**; can store data in any **way** within a set
- Index is used to map addresses to a specific **set**, so not fully associative



Cache Hits/Misses

- Cache Hit: data **is** in cache
- Cache Miss: data is **not** in cache
  - Have to retrieve data from memory -> extra time!
  - **Cold/Compulsory miss**: first ever access to a block
  - **Conflict miss**: two hot address map to the same cache line
    - (esp. for direct-mapped caches)

CACHE

index	V	Tag	Data
11	0	XX	X
10	0	XX	X
01	0	XX	X
00	1	11	M

- For the 4 byte direct mapped cache on the right, would the following loads be a hit or a miss?
  - **Load 1100**?
  - **Load 1101**?

```
tag|index
  1101
```

Cache Hits/Misses

- Cache Hit: data **is** in cache
- Cache Miss: data is **not** in cache
  - Have to retrieve data from memory -> extra time!
  - **Cold/Compulsory miss**: first ever access to a block
  - **Conflict miss**: two hot address map to the same cache line
    - (esp. for direct-mapped caches)

CACHE

index	V	Tag	Data
11	0	XX	X
10	0	XX	X
01	0	XX	X
00	1	11	M

- For the 4 byte direct mapped cache on the right, would the following loads be a hit or a miss?
  - **Load 1100**? **Hit! (Valid bit is 1 and the tag matches)**
  - **Load 1101**? **Miss (Valid bit is 0)**

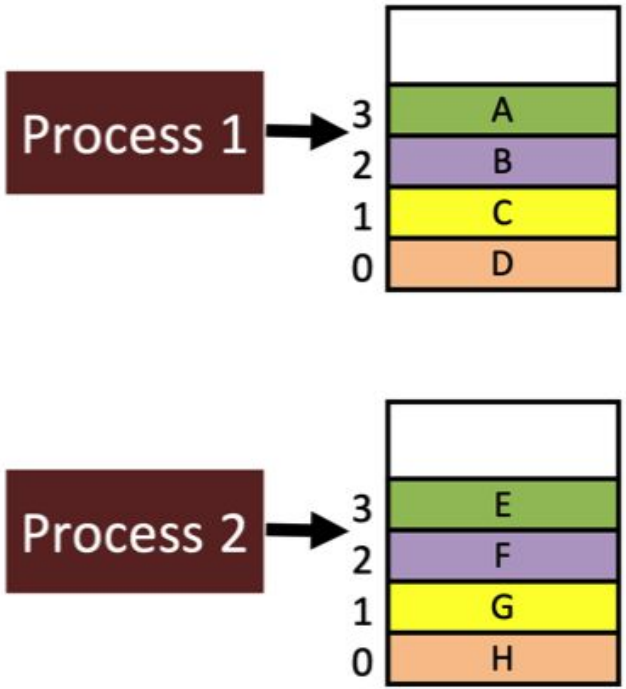
```
tag|index
 1101
```

Computing Average Memory Access Time (AMAT)

- AMAT = access time + miss rate x miss penalty
    - miss rate = # cache misses / # cache accesses
    - miss penalty = extra time it takes to retrieve data from lower memory tiers (e.g. if we miss an L1 cache, we have to go to L2 cache, etc.)

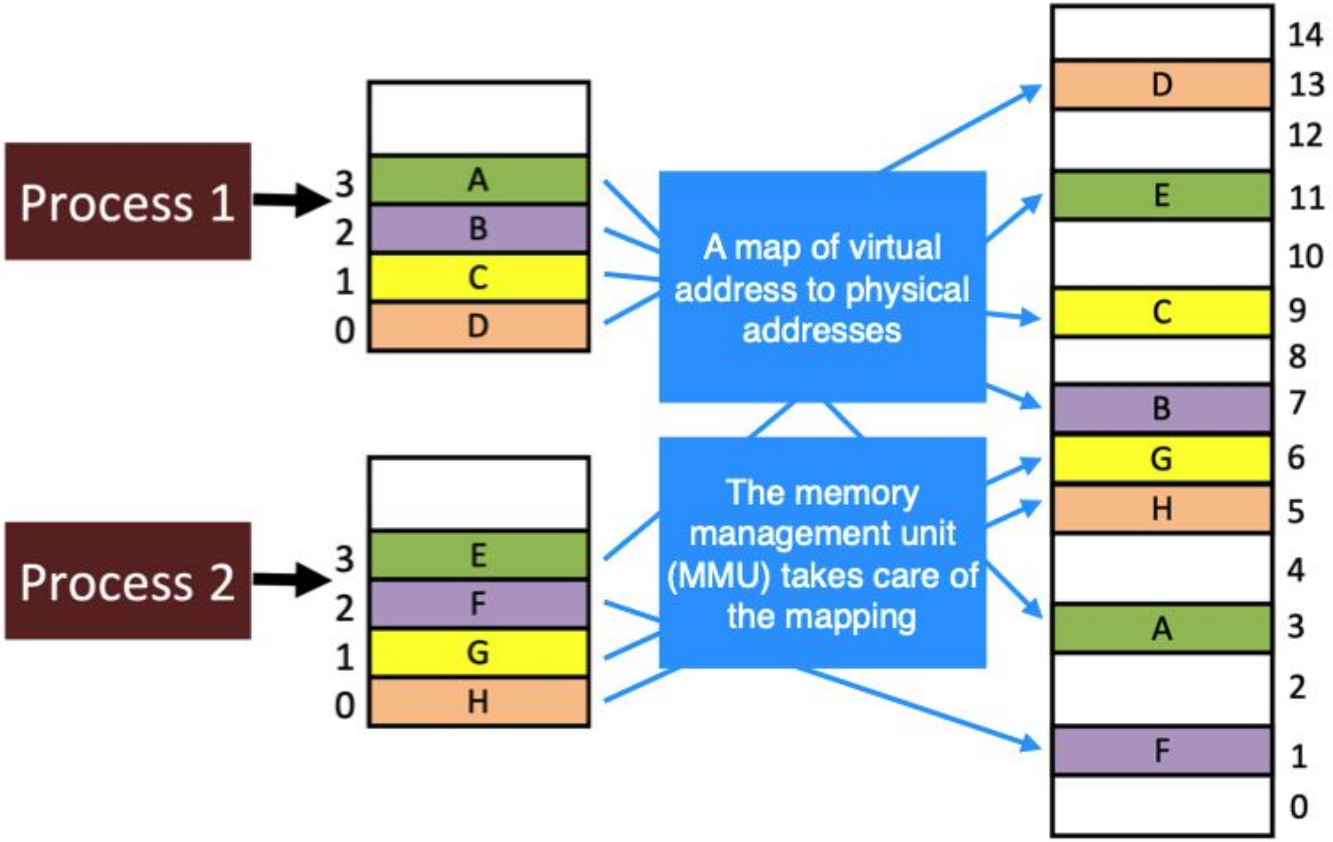
Virtual Memory (Review)

Big Picture: Virtual Memory

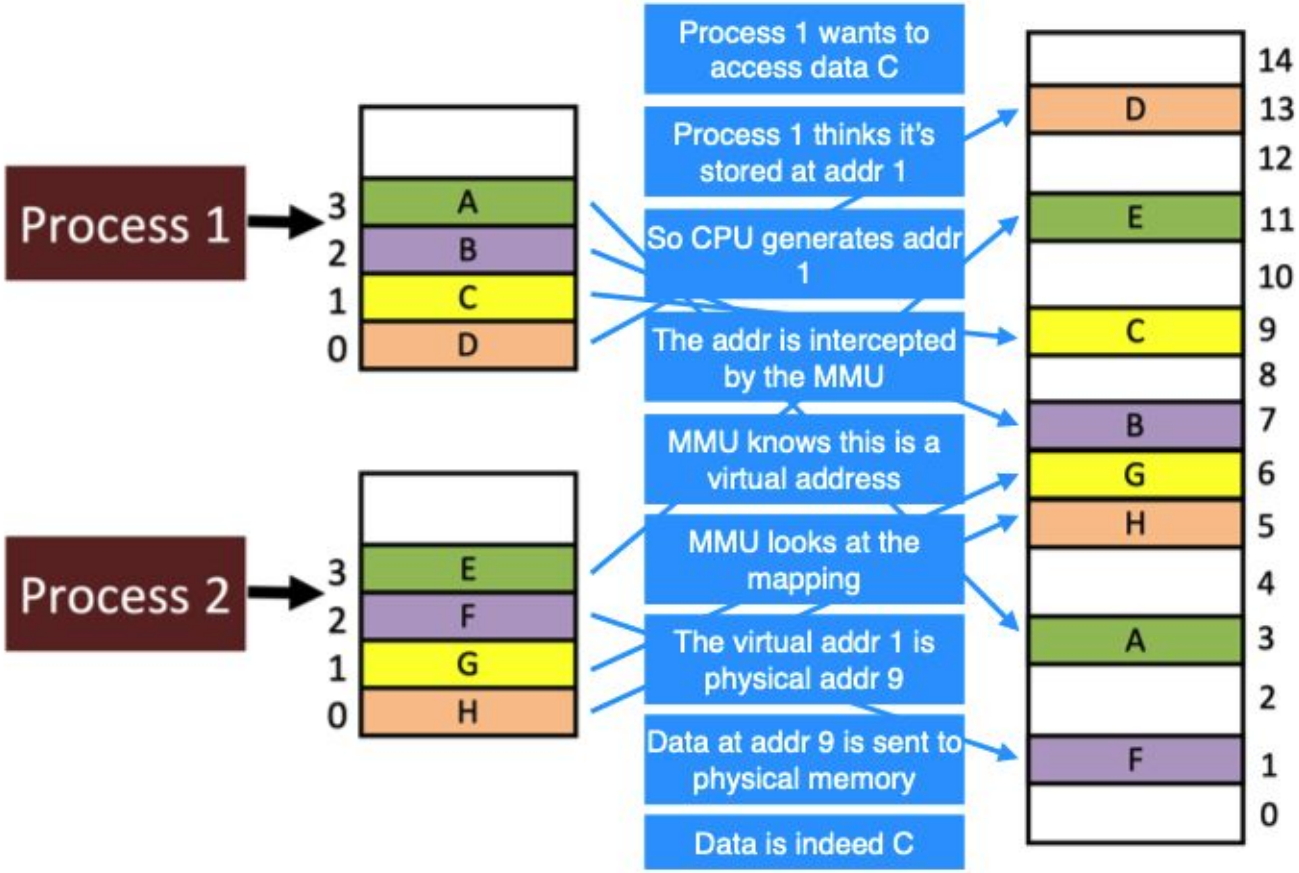


Give each process an **illusion** that it has **exclusive** access to entire main memory

How Do We Create the Illusion?



How Do We Create the Illusion?



The virtual-to-physical address mapping

- The **CPU generates a virtual address** when running a program
- The OS wants to give each process the illusion of a contiguous, linear memory space
- To do this, the CPU uses a **page table**, which is a “map” maintained by the OS that links virtual pages to physical frames

Good Luck!