



## Big Idea

- How to choose between associativity, block size, replacement & write policy?
- Design against a performance model
  - Minimize: **Average Memory Access Time**
    - = Hit Time
    - + Miss Penalty x Miss Rate
  - influenced by technology & program behavior
- Create the illusion of a memory that is large, cheap, and fast - on average

**How can we improve miss penalty?**

CS61C L24 Cache III, VM (7) Chae, Summer 2008 © UCB

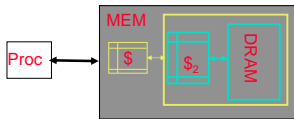
## Ways to reduce miss rate

- Larger cache
  - limited by cost and technology
  - hit time of first level cache < cycle time (bigger caches are slower)
- More places in the cache to put each block of memory – associativity
  - fully-associative
    - any block any line
  - N-way set associated
    - N places for each block
    - direct map: N=1

**Cal** CS61C L24 Cache III, VM (10) Chae, Summer 2008 © UCB

## Improving Miss Penalty

- When caches first became popular, Miss Penalty ~ 10 processor clock cycles
- Today 2400 MHz Processor (0.4 ns per clock cycle) and 80 ns to go to DRAM ⇒ 200 processor clock cycles!



Solution: another cache between memory and the processor cache: **Second Level (L2) Cache**

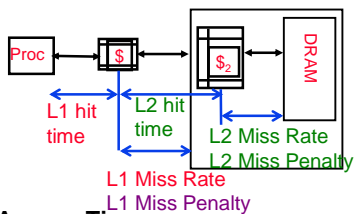
**Cal** CS61C L24 Cache III, VM (8) Chae, Summer 2008 © UCB

## Typical Scale

- L1
  - size: tens of KB
  - hit time: complete in one clock cycle
  - miss rates: 1-5%
- L2:
  - size: hundreds of KB
  - hit time: few clock cycles
  - miss rates: 10-20%
- L2 miss rate is fraction of L1 misses that also miss in L2
  - why so high?

**Cal** CS61C L24 Cache III, VM (11) Chae, Summer 2008 © UCB

## Analyzing Multi-level cache hierarchy



Avg Mem Access Time =  $L1 \text{ Hit Time} + L1 \text{ Miss Rate} * L1 \text{ Miss Penalty}$

L1 Miss Penalty =  $L2 \text{ Hit Time} + L2 \text{ Miss Rate} * L2 \text{ Miss Penalty}$

Avg Mem Access Time =  $L1 \text{ Hit Time} + L1 \text{ Miss Rate} * (L2 \text{ Hit Time} + L2 \text{ Miss Rate} * L2 \text{ Miss Penalty})$

**Cal** CS61C L24 Cache III, VM (9) Chae, Summer 2008 © UCB

## Example: with L2 cache

- Assume
  - L1 Hit Time = 1 cycle
  - L1 Miss rate = 5%
  - L2 Hit Time = 5 cycles
  - L2 Miss rate = 15% (% L1 misses that miss)
  - L2 Miss Penalty = 200 cycles
- L1 miss penalty =  $5 + 0.15 * 200 = 35$
- Avg mem access time =  $1 + 0.05 * 35 = 2.75 \text{ cycles}$

**Cal** CS61C L24 Cache III, VM (12) Chae, Summer 2008 © UCB

## Example: without L2 cache

- Assume
  - L1 Hit Time = 1 cycle
  - L1 Miss rate = 5%
  - L1 Miss Penalty = 200 cycles
- Avg mem access time =  $1 + 0.05 \times 200 = 11$  cycles
- **4x faster with L2 cache! (2.75 vs. 11)**



CS61C L24 Cache III, VM I(13)

Chae, Summer 2008 © UCB

## Summary of Cache Design

- We've discussed memory caching in detail. Caching in general shows up over and over in computer systems
  - Filesystem cache
  - Web page cache
  - Game databases / tablebases
  - Software memoization
  - Others?
- Big idea: if something is expensive but we want to do it repeatedly, do it once and cache the result.
- Cache design choices:
  - Write through v. write back
  - size of cache: speed v. capacity
  - direct-mapped v. associative
  - for N-way set assoc: choice of N
  - block replacement policy
  - 2<sup>nd</sup> level cache?
  - 3<sup>rd</sup> level cache?

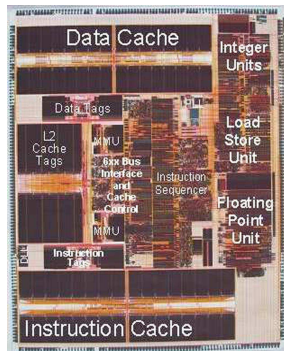


CS61C L24 Cache III, VM I(16)

Chae, Summer 2008 © UCB

## An actual CPU – Early PowerPC

- Cache
  - 32 KByte Instructions and 32 KByte Data L1 caches
  - External L2 Cache interface with integrated controller and cache tags, supports up to 1 MByte external L2 cache
  - Dual Memory Management Units (MMU) with Translation Lookaside Buffers (TLB)
- Pipelining
  - Superscalar (3 inst/cycle)
  - 6 execution units (2 integer and 1 double precision IEEE floating point)



CS61C L24 Cache III, VM I(14)

Chae, Summer 2008 © UCB

## Peer Instruction

1. All caches take advantage of spatial locality.
2. All caches take advantage of temporal locality.
3. On a read, the return value will depend on what is in the cache.

	ABC
0:	FFF
1:	FFT
2:	FTF
3:	FTT
4:	TFF
5:	TFT
6:	FTF
7:	TTT



CS61C L24 Cache III, VM I(17)

Chae, Summer 2008 © UCB

## An Actual CPU – Pentium M

**Intel® Pentium® M Processor**  
New Micro Architecture  
77 Million Transistors

- Micro-Ops Fusion** – fuses operations together to enable faster execution of instructions at lower power
- Advanced Branch Prediction** – fewer re-dos for increased performance
- 32KB IS**
- 32KB DS**
- 4MB Power Optimized L2 Cache** – enables higher CPU performance
- Streaming SIMD Extensions II** – compatible with Pentium® 4 Processor optimized software
- Dedicated Stack Management** – faster instruction at lower power levels
- Enhanced Intel® SpeedStep® Technology** – Multiple voltages & frequency operating points
- 400 MHz Power Optimized System Bus** – faster system bus to enhance performance at lower power levels



CS61C L24 Cache III, VM I(15)

Chae, Summer 2008 © UCB

## Peer Instruction Answer

1. All caches take advantage of spatial locality. **FALSE**
2. All caches take advantage of temporal locality. **TRUE**
3. On a read, the return value will depend on what is in the cache. **FALSE**

1. Block size = 1, no spatial!
2. That's the idea of caches; We'll need it again soon.
3. It better not! If it's there, use it. Oth, get from mem

	ABC
0:	FFF
1:	FFT
2:	FTF
3:	FTT
4:	TFF
5:	TFT
6:	FTF
7:	TTT



CS61C L24 Cache III, VM I(18)

Chae, Summer 2008 © UCB

## Peer Instruction

- In the last 10 years, the gap between the access time of DRAMs & the cycle time of processors has decreased. (i.e., is closing)
- A 2-way set-associative cache can be outperformed by a direct-mapped cache.
- Larger block size  $\Rightarrow$  lower miss rate

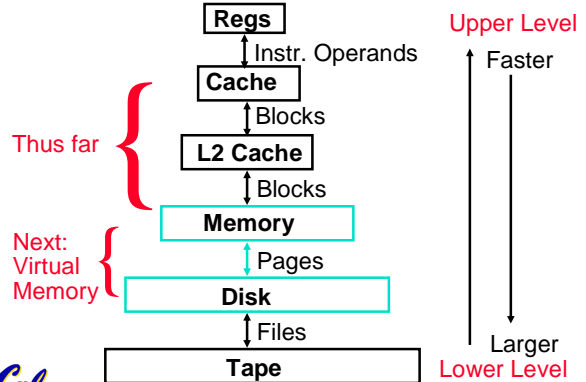
	ABC
0:	FFF
1:	FFT
2:	FTF
3:	FTT
4:	TFF
5:	TFT
6:	TFF
7:	TTT



CS61C L24 Cache III, VM II(19)

Chae, Summer 2008 © UCB

## Another View of the Memory Hierarchy



CS61C L24 Cache III, VM II(22)

Chae, Summer 2008 © UCB

## Peer Instruction Answer

- That was one of the motivation for caches in the first place -- that the memory gap is big and widening.
- Sure, consider the caches from the previous slides with the following workload: 0, 2, 0, 4, 2  
2-way: 0m, 2m, 0h, 4m, 2m; DM: 0m, 2m, 0h, 4m, 2h
- Larger block size  $\Rightarrow$  lower miss rate, true until a certain point, and then the ping-pong effect takes over

- In the last 10 years, the gap between the access time of DRAMs & the cycle time of processors has decreased. (i.e., is closing)
- A 2-way set-associative cache can be outperformed by a direct-mapped cache.
- Larger block size  $\Rightarrow$  lower miss rate

	ABC
0:	FFF
1:	FFT
2:	FTF
3:	FTT
4:	TFF
5:	TFT
6:	TFF
7:	TTT



CS61C L24 Cache III, VM II(20)

Chae, Summer 2008 © UCB

## Memory Hierarchy Requirements

- If Principle of Locality allows caches to offer (close to) speed of cache memory with size of DRAM memory, then recursively why not use at next level to give speed of DRAM memory, size of Disk memory?
- While we're at it, what other things do we need from our memory system?



CS61C L24 Cache III, VM II(23)

Chae, Summer 2008 © UCB

## Administrivia

- Quiz 12 due Friday 8/1
- HW6 due Friday 8/1
- Proj3 out now, due next Wednesday 8/6
  - Will be hand graded in person, signups will be posted soon
- Drop or grading option deadline
  - August 1
  - summer.berkeley.edu for more details
- Final 8/14 – 9:30-12:30pm in 105 North Gate
- 61A/B conflicts? Talk to me at end of class.



CS61C L24 Cache III, VM II(21)

Chae, Summer 2008 © UCB

## Memory Hierarchy Requirements

- Allow multiple processes to simultaneously occupy memory and provide protection – don't let one program read/write memory from another
- Address space – give each program the illusion that it has its own private memory
  - Suppose code starts at address 0x40000000. But different processes have different code, both residing at the same address. So each program has a different view of memory.



CS61C L24 Cache III, VM II(24)

Chae, Summer 2008 © UCB

## Virtual Memory

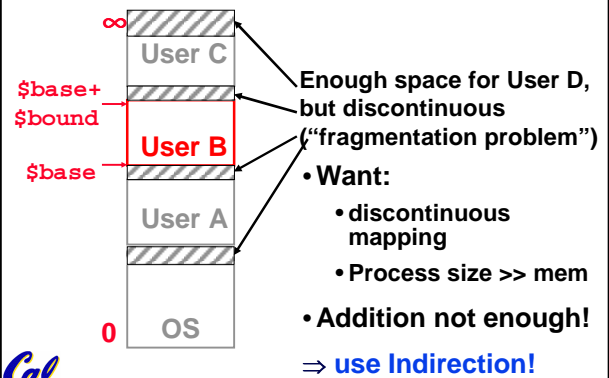
- Called "**Virtual Memory**"
- Next level in the memory hierarchy:
  - Provides program with illusion of a very large main memory:
  - Working set of "pages" reside in main memory - others reside on disk.
- Also allows OS to share memory, protect programs from each other
- Today, more important for **protection** vs. just another level of memory hierarchy
- Each process thinks it has all the memory to itself

**Cal** (Historically, it predates caches)

CS61C L24 Cache III, VM I(25)

Chae, Summer 2008 ©UCB

## Simple Example: Base and Bound Reg

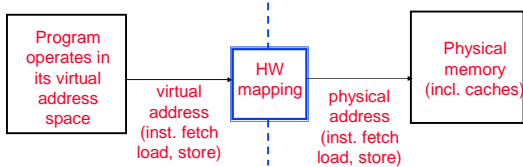


**Cal**

CS61C L24 Cache III, VM I(26)

Chae, Summer 2008 ©UCB

## Virtual to Physical Address Translation



- Each program operates in its own virtual address space; ~only program running
- Each is protected from the other
- OS can decide where each goes in memory
- Hardware (HW) provides virtual  $\Rightarrow$  physical mapping

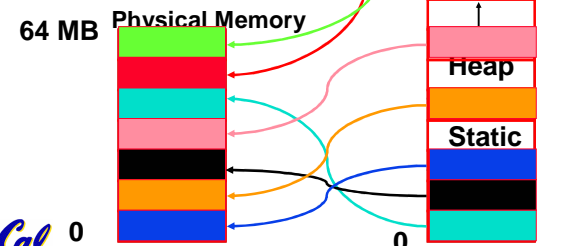
**Cal**

CS61C L24 Cache III, VM I(26)

Chae, Summer 2008 ©UCB

## Mapping Virtual Memory to Physical Memory

- Divide into equal sized chunks (about 4 KB - 8 KB)
- Any chunk of Virtual Memory assigned to any chunk of Physical Memory ("**page**")



**Cal**

CS61C L24 Cache III, VM I(27)

Chae, Summer 2008 ©UCB

## Analogy

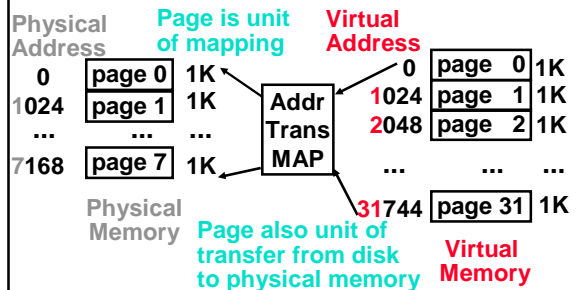
- Book title like **virtual address**
- Library of Congress call number like **physical address**
- Card catalogue like **page table**, mapping from book title to call #
- On card for book, in local library vs. in another branch like **valid bit** indicating in main memory vs. on disk
- On card, available for 2-hour in library use (vs. 2-week checkout) like **access rights**

**Cal**

CS61C L24 Cache III, VM I(27)

Chae, Summer 2008 ©UCB

## Paging Organization (assume 1 KB pages)



**Cal**

CS61C L24 Cache III, VM I(30)

Chae, Summer 2008 ©UCB

## Virtual Memory Mapping Function

- Cannot have simple function to predict arbitrary mapping
- Use table lookup of mappings
 

Page Number	Offset
-------------	--------
- Use table lookup (“**Page Table**”) for mappings: Page number is index
- Virtual Memory Mapping Function
  - Physical Offset = Virtual Offset
  - Physical Page Number = PageTable[Virtual Page Number]



## Requirements revisited

Remember the motivation for VM:

- **Sharing memory with protection**
  - Different physical pages can be allocated to different processes (sharing)
  - A process can only touch pages in its own page table (protection)
- **Separate address spaces**
  - Since programs work only with virtual addresses, different programs can have different data/code at the same address!

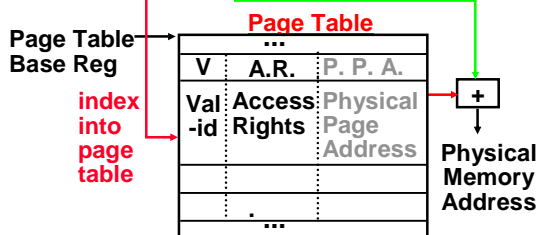


What about the memory hierarchy?

## Address Mapping: Page Table

Virtual Address:

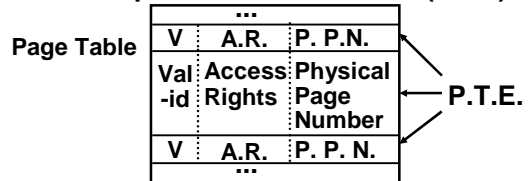
page no. offset



Page Table located in physical memory

## Page Table Entry (PTE) Format

- Contains either Physical Page Number or indication not in Main Memory
- OS maps to disk if Not Valid (V = 0)



- If valid, also check if have permission to use page: **Access Rights** (A.R.) may be Read Only, Read/Write, Executable

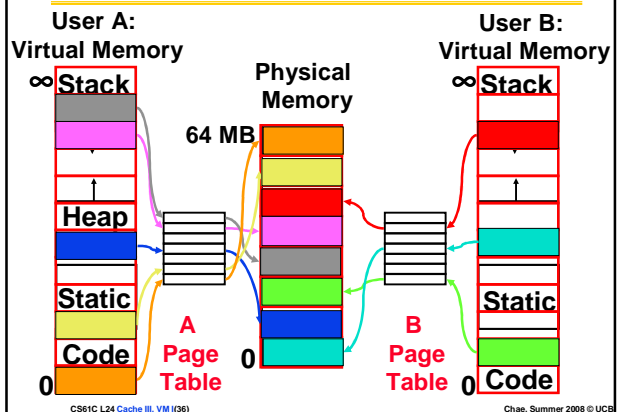


## Page Table

- A page table is an operating system structure which contains the mapping of virtual addresses to physical locations
  - There are several different ways, all up to the operating system, to keep this data around
- Each process running in the operating system has its own page table
  - “**State**” of process is PC, all registers, plus page table
  - OS changes page tables by changing contents of **Page Table Base Register**



## Paging/Virtual Memory Multiple Processes



### Comparing the 2 levels of hierarchy

Cache version	Virtual Memory vers.
Block or Line	<b>Page</b>
Miss	<b>Page Fault</b>
Block Size: 32-64B	Page Size: 4K-8KB
Placement: Direct Mapped, N-way Set Associative	Fully Associative
Replacement: LRU or Random	Least Recently Used (LRU)
Write Thru or Back	Write Back



### Virtual Memory Problem #1

- Map every address  $\Rightarrow$  1 indirection via Page Table in memory per virtual address  $\Rightarrow$  1 virtual memory accesses = 2 physical memory accesses  $\Rightarrow$  SLOW!
- Observation: since locality in pages of data, there must be locality in **virtual address translations** of those pages
- Since small is fast, why not use a small cache of virtual to physical address translations to make translation fast?
- For historical reasons, cache is called a **Translation Lookaside Buffer**, or **TLB**



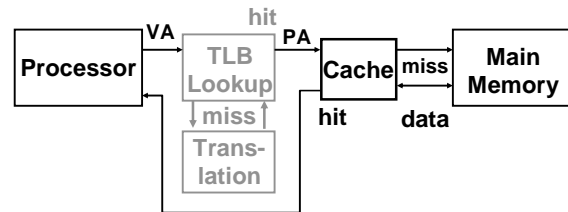
### Notes on Page Table

- Solves Fragmentation problem: all chunks same size, so all holes can be used
- OS must reserve "**Swap Space**" on disk for each process
- To grow a process, ask Operating System
  - If unused pages, OS uses them first
  - If not, OS swaps some old pages to disk
  - (Least Recently Used to pick pages to swap)
- Each process has own Page Table
- Will add details, but Page Table is essence of Virtual Memory



### Translation Look-Aside Buffers (TLBs)

- TLBs usually small, typically 128 - 256 entries
- Like any other cache, the TLB can be direct mapped, set associative, or fully associative



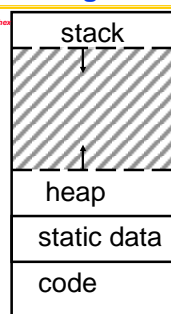
On TLB miss, get page table entry from main memory



### Why would a process need to "grow"?

• A program's **address space** contains 4 regions:

- **stack**: local variables, grows downward
- **heap**: space requested for pointers via `malloc()`; resizes dynamically, grows upward
- **static data**: variables declared outside main, does not grow or shrink
- **code**: loaded when program starts, does not change



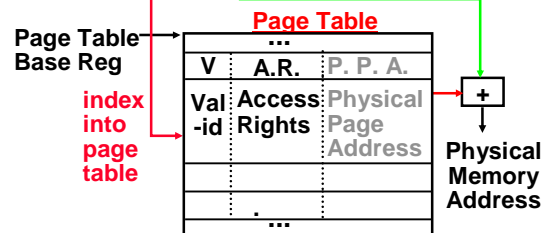
For now, OS somehow prevents accesses between stack and heap (gray hash lines).



### Review Address Mapping: Page Table

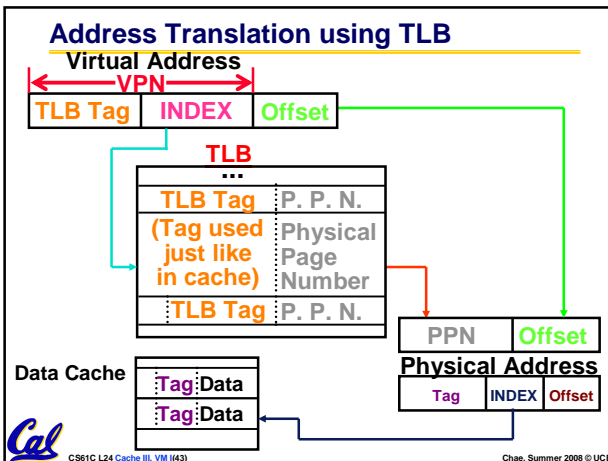
Virtual Address:

page no. offset



Page Table located in physical memory





### What if the data is on disk?

- We load the page off the disk into a free block of memory, using a DMA transfer (Direct Memory Access – special hardware support to avoid processor)
  - Meantime we switch to some other process waiting to be run
- When the DMA is complete, we get an **interrupt** and update the process's page table
  - So when we switch back to the task, the desired data will be in memory

CS61C L24 Cache III, VM II(45) Chae, Summer 2008 ©UCB

### Typical TLB Format

Tag	Physical Page #	Dirty	Ref	Valid	Access Rights

- TLB just a cache on the page table mappings
- TLB access time comparable to cache (much less than main memory access time)
- **Dirty**: since use write back, need to know whether or not to write page to disk when replaced
- **Ref**: Used to help calculate LRU on replacement
  - Cleared by OS periodically, then checked to see if page was referenced

CS61C L24 Cache III, VM II(44) Chae, Summer 2008 ©UCB

### What if we don't have enough memory?

- We chose some other page belonging to a program and transfer it onto the disk if it is dirty
  - If clean (disk copy is up-to-date), just overwrite that data in memory
  - We chose the page to evict based on replacement policy (e.g., LRU)
- And update that program's page table to reflect the fact that its memory moved somewhere else
- If continuously swap between disk and memory, called **Thrashing**

CS61C L24 Cache III, VM II(47) Chae, Summer 2008 ©UCB

### What if not in TLB?

- Option 1: Hardware checks page table and loads new Page Table Entry into TLB
- Option 2: Hardware **traps** to OS, up to OS to decide what to do
  - MIPS follows Option 2: Hardware knows nothing about page table

CS61C L24 Cache III, VM II(46) Chae, Summer 2008 ©UCB

### Three Advantages of Virtual Memory

1) **Translation:**

- Program can be given consistent view of memory, even though physical memory is scrambled
- Makes multiple processes reasonable
- Only the most important part of program ("**Working Set**") must be in physical memory
- Contiguous structures (like stacks) use only as much physical memory as necessary yet still grow later

CS61C L24 Cache III, VM II(48) Chae, Summer 2008 ©UCB

### Three Advantages of Virtual Memory

#### 2) Protection:

- Different processes protected from each other
- Different pages can be given special behavior
  - (Read Only, Invisible to user programs, etc).
- Kernel data protected from User programs
- Very important for protection from malicious programs ⇒ Far more “viruses” under Microsoft Windows
- Special Mode in processor (“Kernel mode”) allows processor to change page table/TLB

#### 3) Sharing:

- Can map same physical page to multiple users (“Shared memory”)



### And in conclusion...

- Manage memory to disk? Treat as cache
  - Included protection as bonus, now critical
  - Use Page Table of mappings **for each user** vs. tag/data in cache
  - TLB is **cache** of Virtual ⇒ Physical addr trans
- Virtual Memory allows protected sharing of memory between processes
- Spatial Locality means Working Set of Pages is all that must be in memory for process to run fairly well



### Peer Instruction

- A. Locality is important yet different for cache and virtual memory (VM): temporal locality for caches but spatial locality for VM
- B. Cache management is done by hardware (HW), page table management by the operating system (OS), but TLB management is either by HW or OS
- C. VM helps both with security and cost

- ABC
- 0: **FFF**
- 1: **FFT**
- 2: **FTF**
- 3: **FTT**
- 4: **TFF**
- 5: **TFT**
- 6: **TTF**
- 7: **TTT**



### Peer Instruction Answer

- A. Locality is important yet different for cache and virtual memory (VM): temporal locality for caches but spatial locality for VM
- B. Cache management is done by hardware (HW), page table management by the operating system (OS), but TLB management is either by HW or OS
- C. VM helps both with security and cost

- A. No. Both for VM and cache**
- B. Yes. TLB SW (MIPS) or HW (\$ HW, Page table OS)**
- C. Yes. Protection and a bit smaller memory**

- ABC
- 0: **FFF**
- 1: **FFT**
- 2: **FTF**
- 3: **FTT**
- 4: **TFF**
- 5: **TFT**
- 6: **TTF**
- 7: **TTT**

