# 体系结构复习课

Great Architecture Ideas

15 单选 -- 30 points
6 道简答 -- 70 points

**指令集、处理器设计**

指令集并行、乱序 两道大题
memory hir cache 一道大题
cache 的计算和理解，优化策略

## Great Architecture Ideas

# Great Architecture Ideas

- Design for Moore's law.
  - The number of transistors on a chip doubles every 18-24 months.
  - Architects have to anticipate where technology will be when the design of a system is completed.
- Use abstraction to simplify design.
  - Abstraction is used to represent the design at different levels of representation.
  - Lower-level details can be hidden to provide simpler models at higher levels.
- Make the common case fast.
  - Identify the common case and try to improve it.
  - Most cost efficient method to obtain improvements.
- Improve performance via parallelism.
  - Improve performance by performing operations in parallel.
  - There are many levels of parallelism – instruction-level, process-level, etc.

# CPU Performance

In order to determine the effect of a design change on the performance experienced by the user, we can use the following relation:

$$CPU\ Execution\ Time = CPU\ Clock\ Cycles \times Clock\ Period$$

Alternatively,

$$CPU\ Execution\ Time = \frac{CPU\ Clock\ Cycles}{Clock\ Rate}$$

Clearly, we can reduce the execution time of a program by either reducing the number of clock cycles required or the length of each clock cycle.

# Instruction Count and CPI

- Instruction Count for a program
  - Determined by program, ISA and compiler
- Average cycles per instruction（CPI）
  - Determined by CPU hardware
  - If different instructions have different CPI
    - Average CPI affected by instruction mix

$$CPI = \frac{CPU\ Clock\ Cycles}{Instruction\ Count}$$

$$CPU\ Clock\ Cycles = Instructions\ for\ a\ Program \times Average\ Clock\ Cycles\ Per\ Instruction$$

$$CPU\ Time = Instruction\ Count \times CPI \times Clock\ Period$$

$$CPU\ Time = \frac{Instruction\ Count \times CPI}{Clock\ Rate}$$

# Amdahl's Law

Amdahl's Law states that the performance improvement to be gained from using some faster mode of execution is limited by the fraction of the time the faster mode can be used.

Amdahl's Law depends on two factors:

- The fraction of the time the enhancement can be exploited.
- The improvement gained by the enhancement while it is exploited.

$$Improved\ Execution\ Time = \frac{Affected\ Execution\ Time}{Amount\ of\ Improvement} + Unaffected\ Execution\ Time$$

Make the common case fast!

- Example: multiply accounts for 80s/100s
  - How much improvement in multiply performance to get 5× overall?

$$20 = \frac{80}{n} + 20$$

Can't be done!

# Amdahl's Law

- Improved ratio: In the system before the improvement, the ratio of the execution time of the improvement part to the total execution time.
  - It is always less than or equal to 1.
  - For example: a program that needs to run for 60 seconds has 20 seconds of calculation that can be accelerated,
    Then the ratio is 20/60.
- Component speedup ratio: The multiple that can be improved after some improvements. It is the ratio of the execution time before the improvement to the execution time after the improvement.
  - Under normal circumstances, the component acceleration ratio is greater than 1.
  - For example: if the system is improved, the execution time of the improved part is 2 seconds, Before the improvement, its execution time was 5 seconds, and the component acceleration ratio was 5/2.





# Amdahl's Law

- Example 1.1 Increasing the processing speed of a certain function in the computer system to *20 times* the original, but the processing time of this function only accounts for *40%* of the running time of the entire system. After adopting this method to improve performance, how much can the performance of the entire system improve?

Answer:

- Fraction $_{enhanced}$ = 40%
- Speedup $_{enhanced}$ = 20

$$\text{Speedup} = \frac{1}{0.6 + \frac{0.4}{20}} = 1.613$$





# Amdahl's Law

- Example 1.2 After a computer system adopts floating-point arithmetic components, the floating-point arithmetic speed is increased by *20 times*, and the overall performance of a certain program of the system is increased by *5 times*. Try to calculate the proportion of the floating-point operations in this program.

Answer:

- Speedup $_{overall}$ = 5
- Speedup $_{enhanced}$ = 20

$$\frac{1}{(1 - \text{Fraction}) + \frac{\text{Fraction}}{20}} = 5$$

Fraction = 84.2%

# The Classic Five-Stage Pipeline for a RISC Processor

- A Simple Implementation of RISC-V
- Instructions Dependences
- Pipeline Hazards

**Dependences are a property of *programs*.**

- Data Dependences
- Name Dependences
  - Anti-dependence
  - Output-dependence
- Control Dependences

- Data Hazards
  - RAW
  - WAR
  - WAW
- Branch Hazards
- Structural Hazards

**Hazard are properties of the *pipeline organization*.**

# Data Hazards

- Read after write: RAW

```
FADD.D   F6, F0, F12
FSUB.D   F8, F6, F14
```

- Write after read: WAR

```
FDIV.D   F2, F6, F4
FADD.D   F6, F0, F12
```

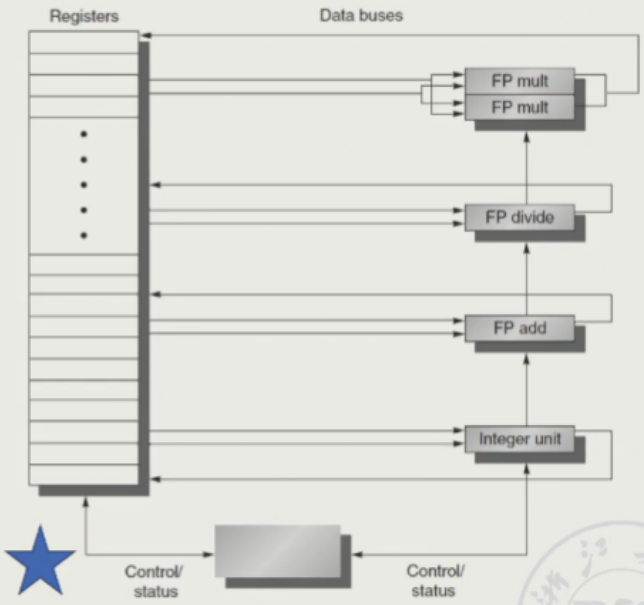
- Write after write: WAW

```
FDIV.D   F2, F0, F4
FSUB.D   F2, F6, F14
```

# Dynamic Scheduling

Idea: Dynamic Scheduling

Method: out-of-order execution




*Dynamic Scheduling with a Scoreboard*

> Scoreboard 未说明就按经典结构

# Dynamic Scheduling: Scoreboard algorithm

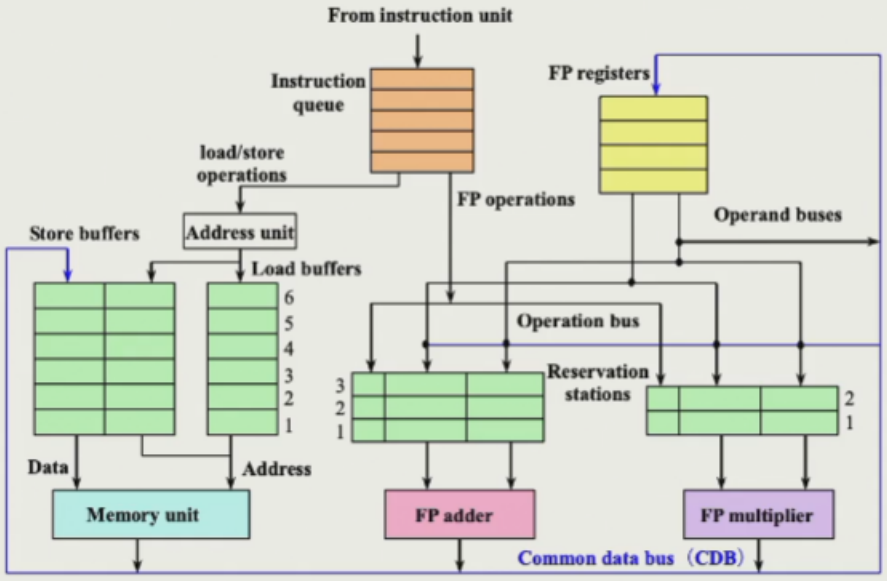
| Instruction | | Instruction Status | | | |
|---|---|---|---|---|---|
| | | IS | RO | EX | WB |
| FLD | F6, 34(R2) | √ | √ | √ | √ |
| FLD | F2, 45(R3) | √ | √ | √ | |
| FMUL.D | F0, F2, F4 | √ | | | |
| FSUB.D | F8, F6, F2 | √ | | | |
| FDIV.D | F10, F0, F6 | √ | | | |
| FADD.D | F6, F8, F2 | | | | |

| Name | Function Component Status | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Busy | Op | Fi | Fj | Fk | Qj | Qk | Rj | Rk |
| Integer | yes | Load | F2 | R3 | | | | no | |
| Mult1 | yes | MUL | F0 | F2 | F4 | Integer | | no | yes |
| Mult2 | no | | | | | | | | |
| Add | yes | SUB | F8 | F6 | F2 | | Integer | yes | no |
| Divide | yes | DIV | F10 | F0 | F6 | Mult1 | | no | yes |

Rj, Rk : “ yes” ——operand is ready but not read;
“no” & “Qj = null” ——operand is read ;
“no” & “Qj != null” ——operand is not ready.

| | Register Status | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | F0 | F2 | F4 | F6 | F8 | F10 | ... | F30 |
| Qi | Mult1 | Integer | | | Add | Divide | | |

# Tomasulo’s Approach ★




*The basic structure of a floating-point unit using Tomasulo’s algorithm*

默认三个加法保留栈，两个乘法保留栈

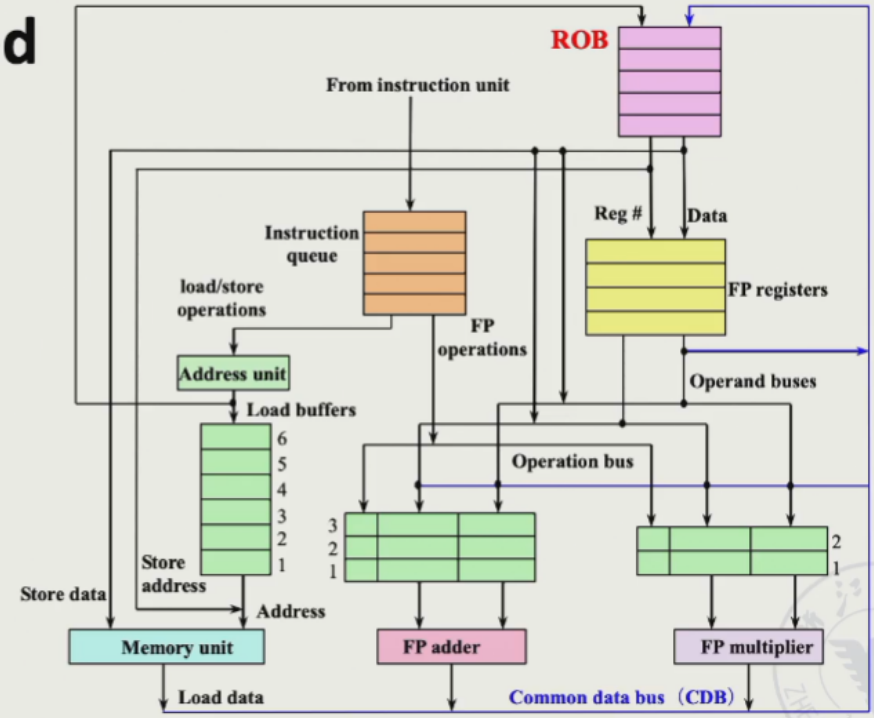
# Tomasulo with Reorder Buffer - Summary ★

| Instruction | Issue | Exec Comp | Writeback | Commit |
|---|---|---|---|---|
| FLD F6, 34(R2) | 1 | 3 | 4 | 5 |
| FLD F2, 45(R3) | 2 | 4 | 5 | 6 |
| FMUL.D F0, F2, F4 | 3 | 6-15 | 16 | 17 |
| FSUB.D F8, F6, F2 | 4 | 6-7 | 8 | 18 |
| FDIV.D F10, F0, F6 | 5 | 17-56 | 57 | 58 |
| FADD.D F6, F8, F2 | 6 | 9-10 | 11 | 59 |

- In-order Issue/Commit, Out-of-Order Execution/Writeback

# Hardware-Based Speculation ★

The basic structure of a FP unit using Tomasulo's algorithm and extended to handle speculation.



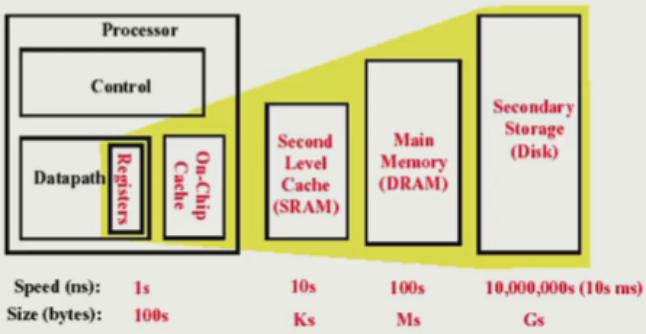

第三章

大题基本都出在这里 Memory hirerarchy

# Memory Hierarchy of a Modern Computer System

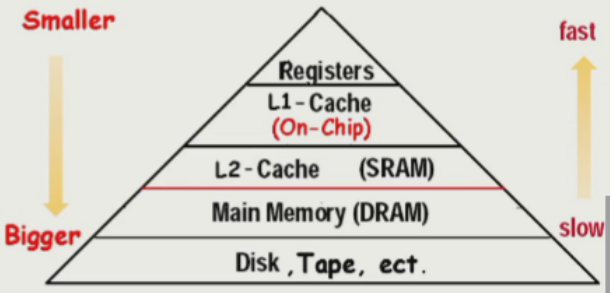
By taking advantage of the principle of locality:

- Present the user with as much memory as is available in the cheapest technology.
- Provide access at the speed offered by the fastest technology.




# What is a cache?

- Small, fast storage used to improve average access time to slow memory.
- In computer architecture, almost everything is a cache!
  - Registers "a cache" on variables – software managed
  - First-level cache a cache on second-level cache
  - Second-level cache a cache on memory
  - Memory a cache on disk (virtual memory)
  - TLB a cache on page table
  - Branch-prediction a cache on prediction information?



# Four Questions for Memory Hierarchy Designers

**Caching** is a general concept used in processors, operating systems, file systems, and applications.

There are **Four Questions** for Memory Hierarchy Designers

- Q1: Where can a block be placed in the upper level/main memory?
  *(Block placement)*
  - Fully Associative, Set Associative, Direct Mapped
- Q2: How is a block found if it is in the upper level/main memory?
  *(Block identification)*
  - Tag/Block
- Q3: Which block should be replaced on a (Virtual Memory) miss?
  *(Block replacement)*
  - Random, LRU,FIFO
- Q4: What happens on a write?
  *(Write strategy)*
  - Write Back or Write Through (with Write Buffer)

# Q4: Write Strategy

- When data is written into the cache (on a store), is the data also written to main memory?
  - If the data is written to memory, the cache is called a write-through cache
    - Can always discard cached data - most up-to-date data is in memory
    - Cache control bit: only a valid bit
    - memory (or other processors) always have latest data
  - If the data is NOT written to memory, the cache is called a write-back cache
    - Can't just discard cached data - may have to write it back to memory
    - Cache control bits: both valid and dirty bits
    - much lower bandwidth, since data often overwritten multiple times

- Write-through adv: Read misses don't result in writes, memory hierarchy is consistent and it is simple to implement.
- Write back adv: Writes occur at speed of cache and main memory bandwidth is smaller when multiple writes occur to the same block.

# Summary

| Read Cache | Write Cache ★ | |
|---|---|---|
| | hit | miss |
| Read through | ***Write-through***<br>write is done synchronously both to the cache and to the backing store | ***Write allocate***<br>data at the missed-write location is loaded to cache, followed by a write-hit operation. |
| Read allocate | ***Write-back***<br>writing is done only to the cache | ***No-write allocate***<br>data at the missed-write location is not loaded to cache, and is written directly to the backing store |

# How to Improve

Hence, there are more than 20 cache optimizations into four categories:

★ $AMAT = HitTime + MissRate \times MissPenalty$

1.Reduce the miss penalty
——multilevel caches, critical word first, read miss before write miss, merging write buffers, and victim caches
2. Reduce the miss rate
——larger block size, large cache size, higher associativity, way prediction and pseudo-associativity, and compiler optimizations
3. Reduce the miss penalty and miss rate via parallelism
——non-blocking caches, hardware prefetching, and compiler prefetching
4. Reduce the time to hit in the cache
——small and simple caches, avoiding address translation, pipelined cache access, and trace caches

# Average Memory Access Time ★

- Average Memory Access Time

$$\text{Average Memory Access Time} = \frac{\text{Whole accesses time}}{\text{All memory accesses in program}}$$

$$= \frac{\text{Accesses time on hitting+ Accesses time on miss}}{\text{All memory accesses in program}}$$

$$= \text{Hit time} + (\text{Miss Rate} \times \text{Miss Penalty})$$

$$= \left(HitTime_{Inst} + MissRate_{Inst} \times MissPenalty_{Inst}\right) \times Inst\%$$
$$\left(HitTime_{Data} + MissRate_{Data} \times MissPenalty_{Data}\right) \times Data\%$$

$$CPUtime = IC \times \left(\frac{AluOps}{Inst} \times CPI_{AluOps} + \frac{MemAccess}{Inst} \times AMAT\right) \times CycleTime$$

cache 计算部分既有大题也有小题

# Learn from History

- **Translation lookaside buffer (TLB)**

  /translation buffer (TB)

  a special cache!

  that keeps (prev) address translations

- TLB entry

  --tag: portions of the virtual address;

  --data: a physical page frame number, protection field, valid bit, use bit, dirty bit;

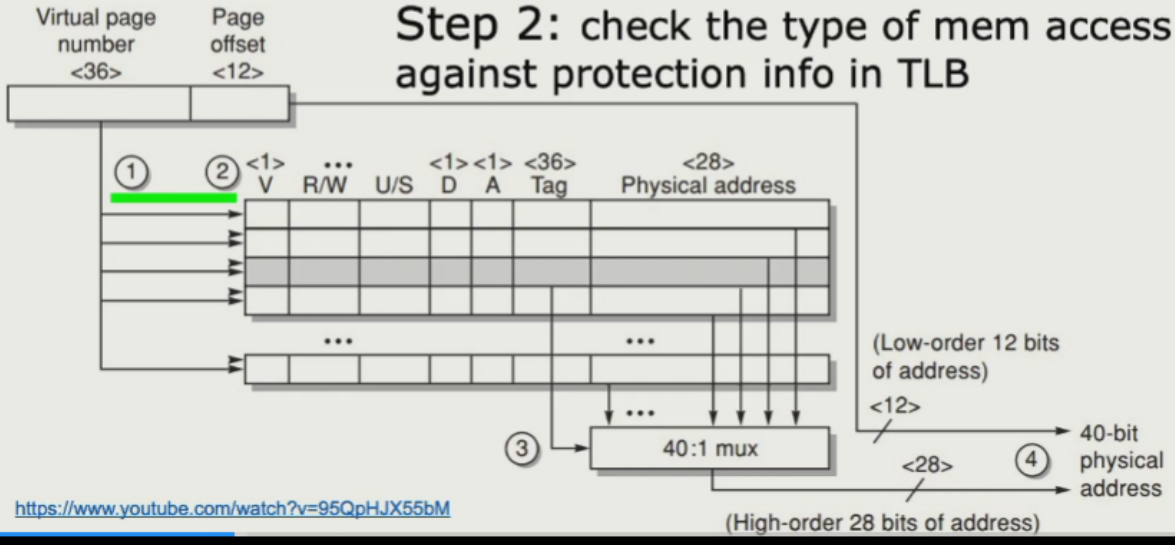
TLB 有大题

# TLB Example ★

- **Opteron data TLB**

Steps 1&2: send the virtual address to all tags

Step 2: check the type of mem access against protection info in TLB

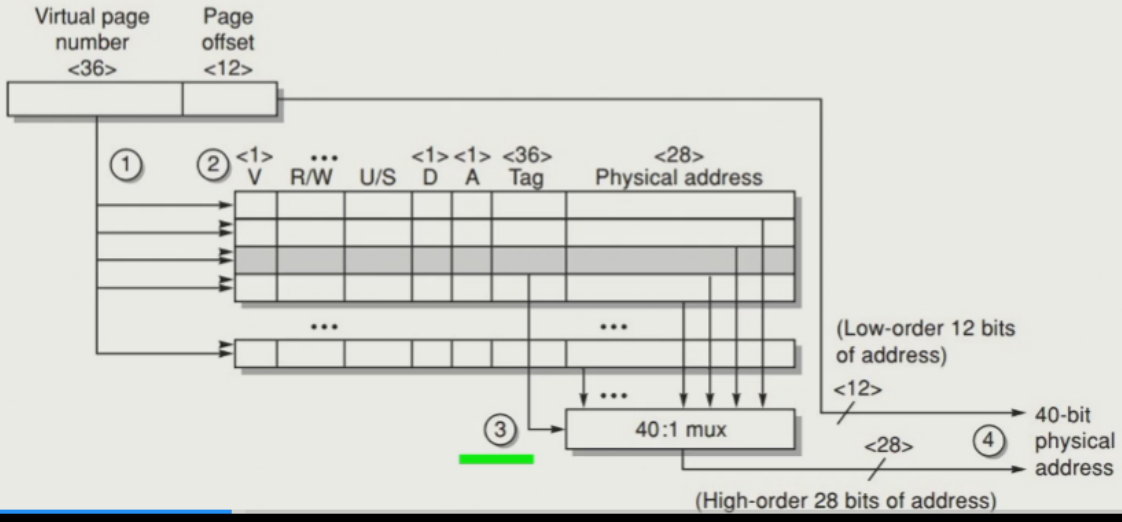



https://www.youtube.com/watch?v=95QpHJX55bM

# TLB Example

- **Opteron data TLB**

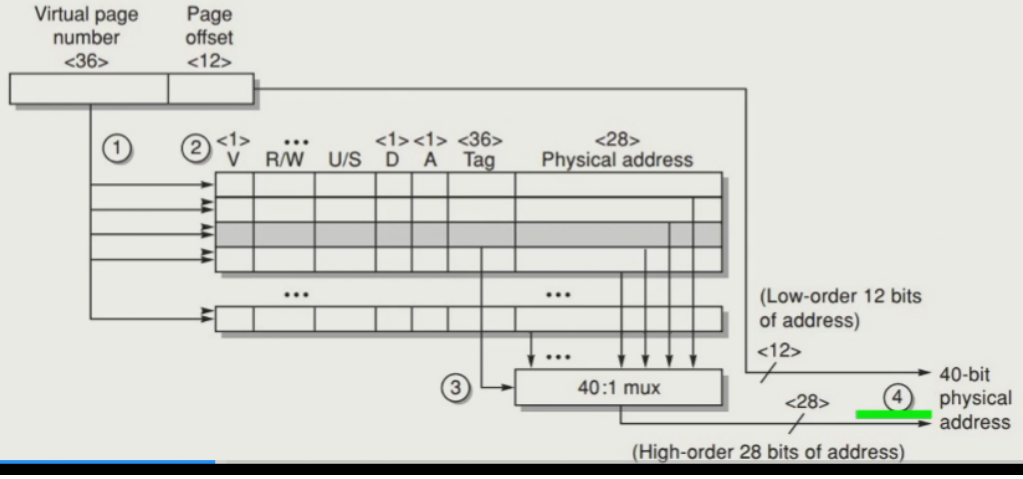
Steps 3: the matching tag sends phy addr through multiplexor



# TLB Example

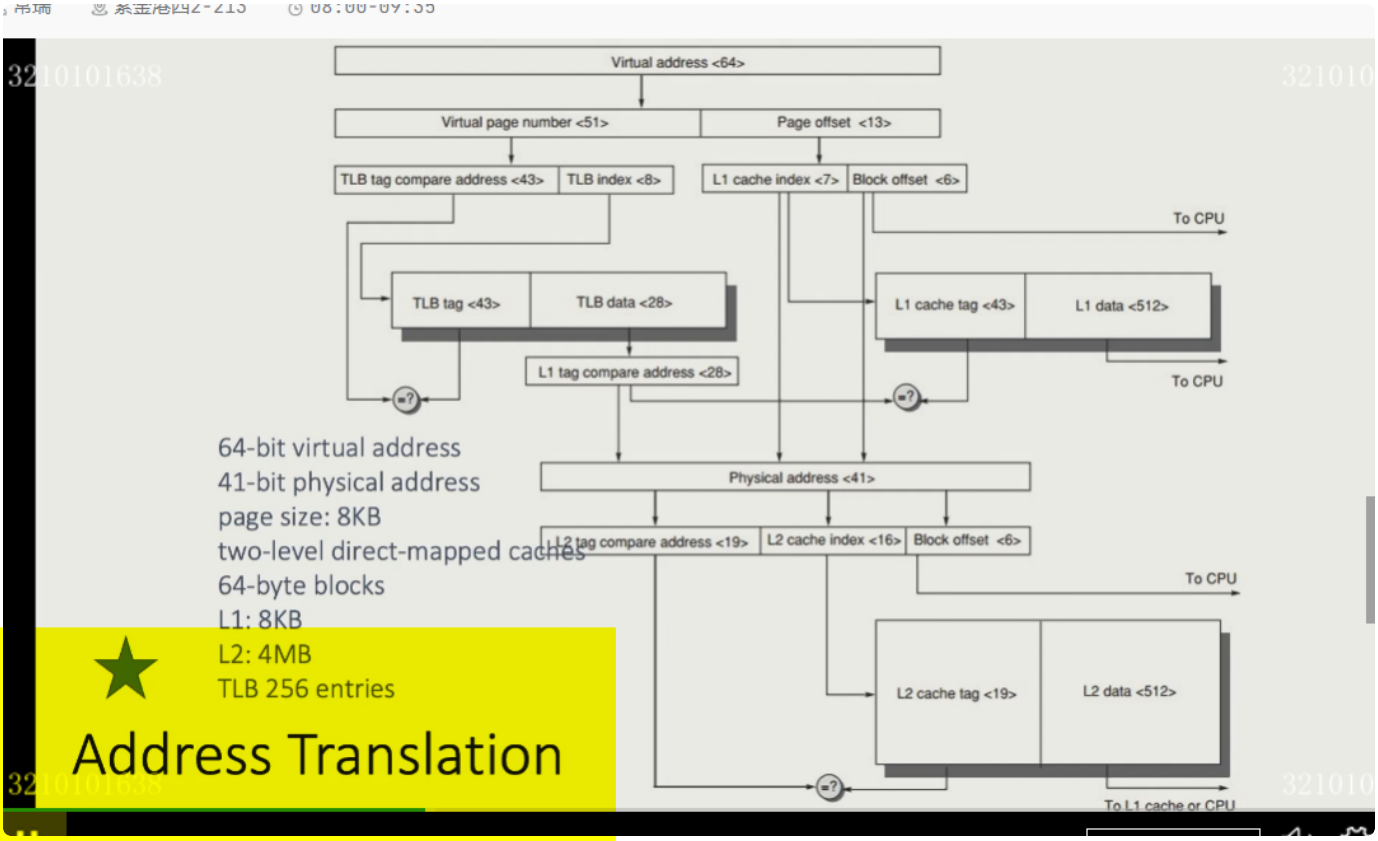
- **Opteron data TLB**

Steps 4: concatenate page offset to phy page frame to form final phy addr




3210101638

# Address Translation: one more time, with cache




未出贯通，分开出，TLB 部分，cache 部分，address Translation

# Summary

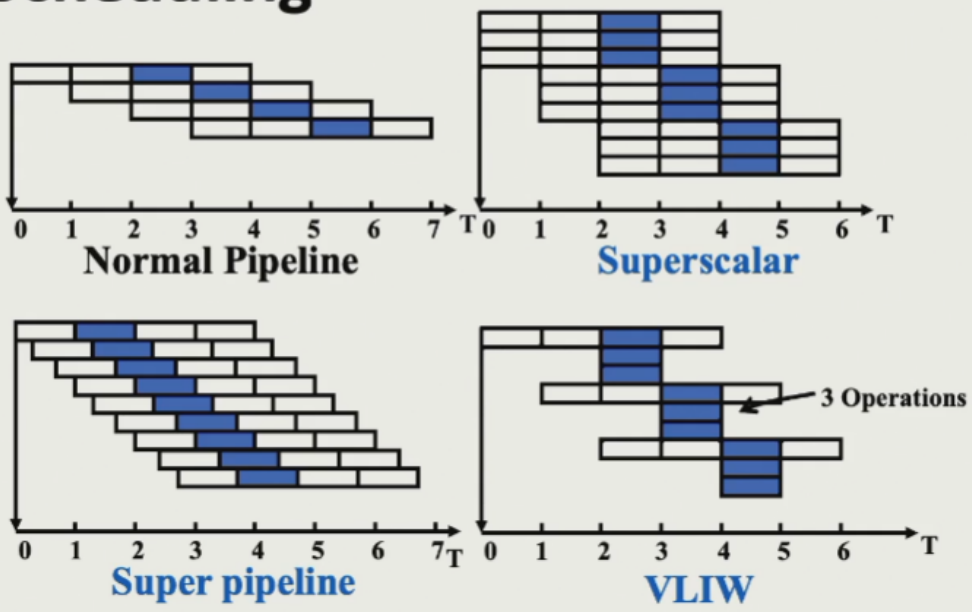
- Memory hierarchy
  - From single level to multi level
  - Evaluate the performance parameters of the storage system (average price per bit C; hit rate H; average memory access time T)
- Cache basic knowledge ★
  - Mapping rules
  - Access method
  - Replacement algorithm
  - Write strategy
  - Cache performance analysis

→ Reduce miss rate
→ Reduce miss penalty ★
→ Reduce hit time

- Virtual Memory & TLB & Cache ★

File System 一道大题

# Exploiting ILP Using Multiple Issue and Static Scheduling




# Classes of Parallel Architectures

according to the parallelism
in the instruction and data streams
called for by the instructions:

SISD, SIMD, MISD, MIMD