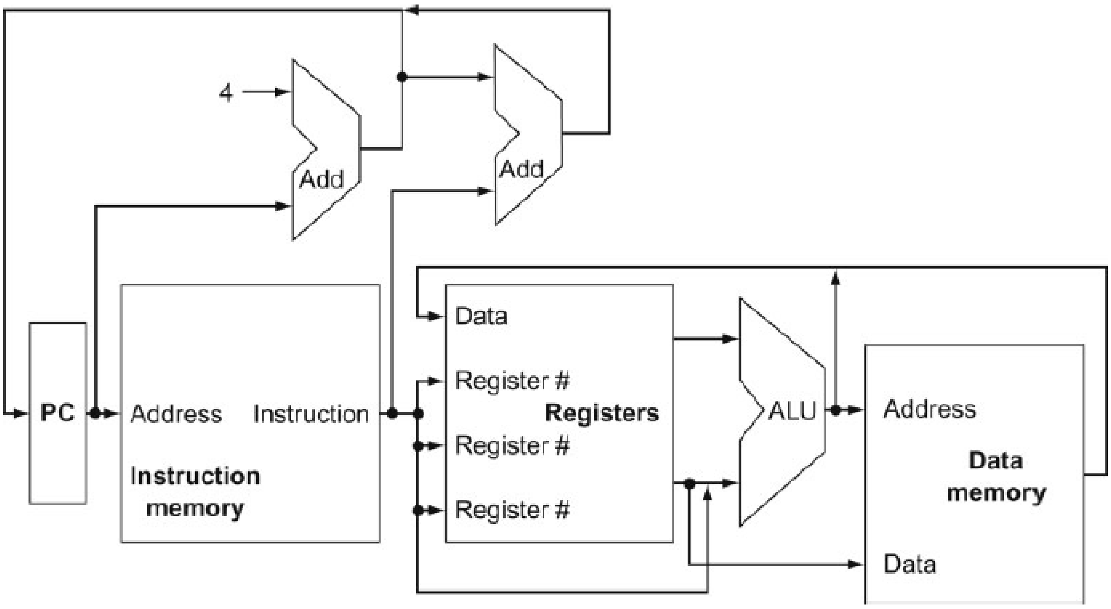
Review8. The Processor

2019年6月2日 10:07

Hint: Bold - formula, underline - important, italic - not so important, grey - not so sure.

Implementation overview

- ***CPU time = instruction count ✕ CPI ✕ clock cycle time***
    - Instruction count: ISA (Instruction set architecture) and compiler.
    - CPI and cycle time: CPU hardware.
- Basic MIPS architecture: math, memory access, branch and jump.
    - Implementation: memory, registers, ALU, and control logics, CPU operations.
- Overall implementation design:



-
    - *Two add units: first add unit increases PC value by 4; second add unit compute J-type instructions' address (offset * 4 + PC).*
    - *Into data memory: memory instructions address after offset, or arithmetic result.*
    - *Into ALU: registers $rs, $rt, $rd from R-type instruction or immediate value from I-type instruction; data from arithmetic computation.*
    - *Into register unit: new data or address from ALU.*

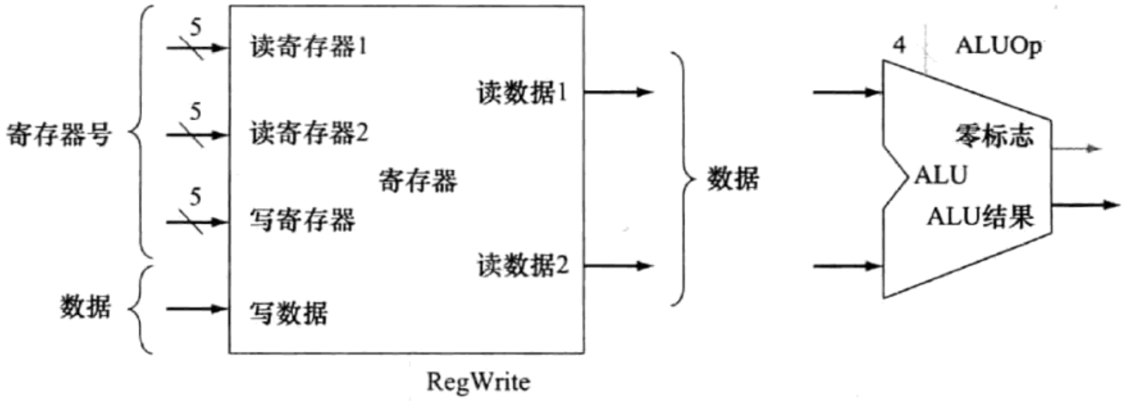
Logic design basics

- Combinational elements: operate on data and output is a function of inputs.
    - MUX: $\boldsymbol{Y = S ? 1 : 0}$, ALU: $\boldsymbol{Y = F(A, B)}$
- Sequential elements (state elements): store information.
    - Sequential without write control: PC, determined by posedge of clk.

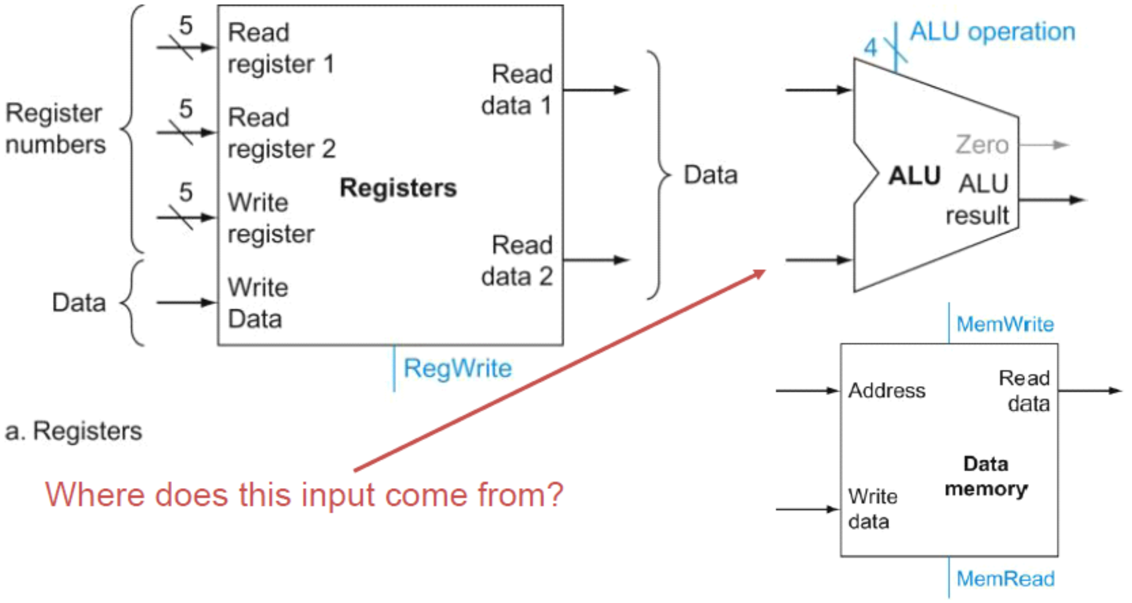
  - Sequential with write control: data memory / register, updates on posedge of clk when write control is 1.
    - Data memory: <input> 32 bit address, 32 bit write data, 1 bit MemWrite, 1 bit MemRead; <output> 32 bit read data. - Read and Write.
    - Instruction memory: <input> 32 bit instruction address, 1 bit clk; <output> instruction. - Read only.
    - Registers: <input> 3 registers of 5 bit, 32 bit write data, 1 bit RegWrite; <output> 2 read data of 32 bit. - Read and Write.
- Units need clock: PC, instruction memory, registers, and data memory.
- Saved data in posedge of clk: PC (address of instruction), instruction, register, data, data memory address.

Detailed implementation for every instruction

- R-type instructions: read data from two registers, if RegWrite valid, then write data back to the third register.



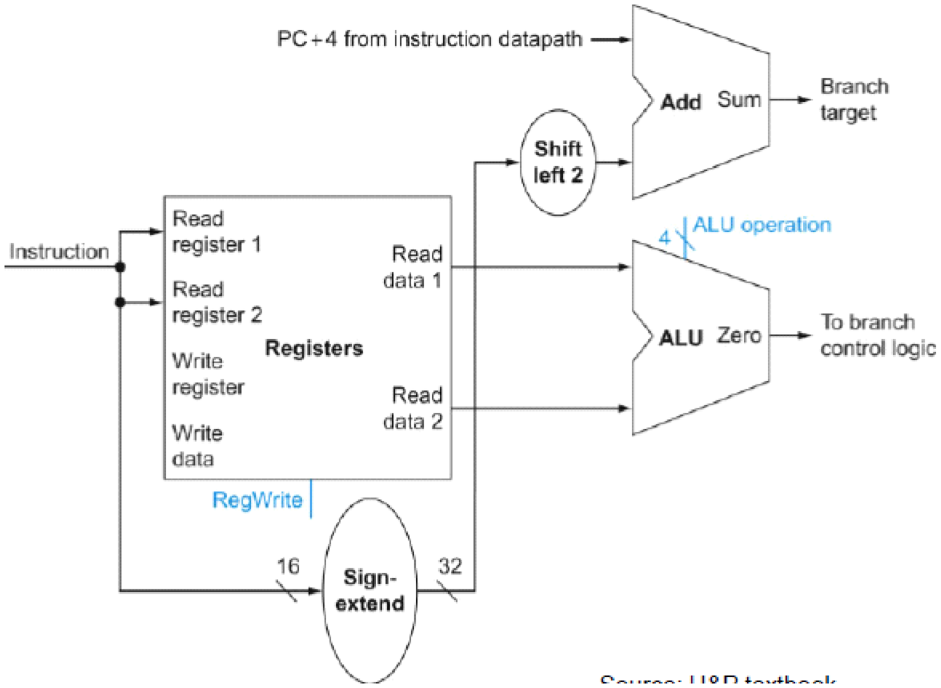
  - *Input two register number of 5 bit and output 2 read data from registers of 32 bit, then input into ALU. Get result and back into write data. If RegWrite valid, write data into write register.*
- Loads / stores instructions: lw and sw, two read registers (two read data), one write register and one write data.



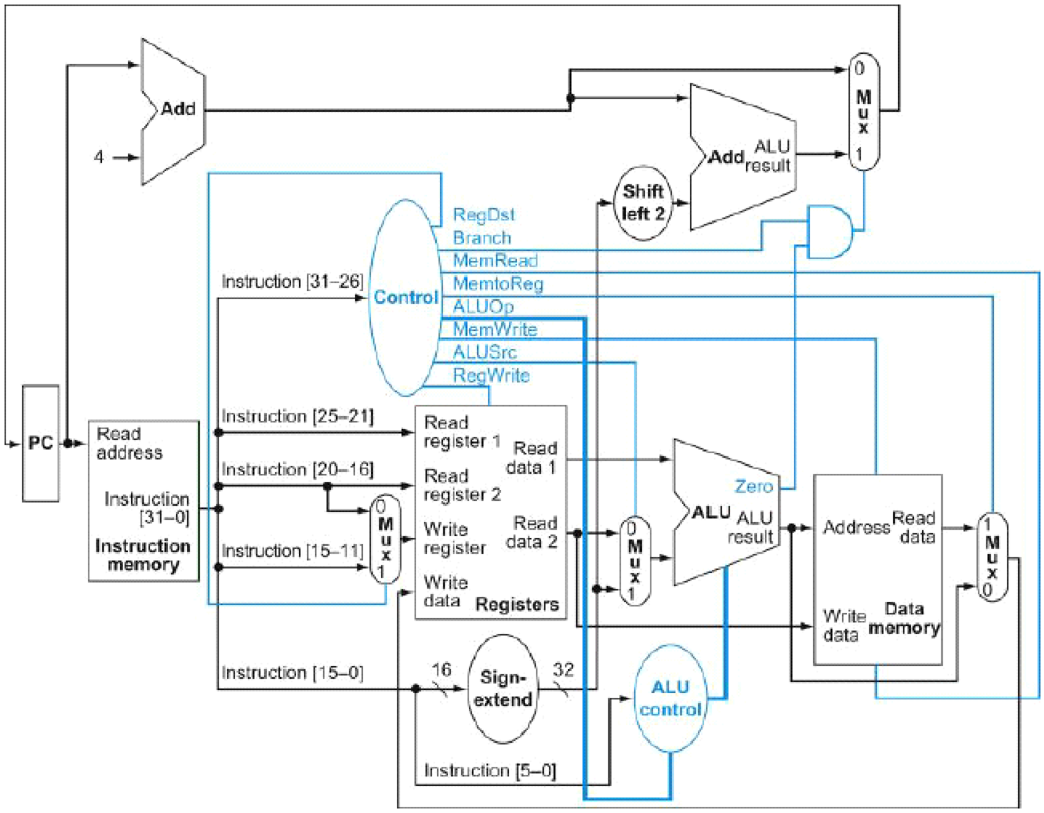
  - Right bottom is data memory, two controls are MemWrite and MemRead. ALU result goes into address, and read data 2 goes into write data.

  - Input of ALU comes from two read data.

- J-type instructions: offset should be left shift 2 bits, so can be store 4 times more offset address. Therefore, before use, offset should be left shift 2 bits.



- Overall control signals:



  - RegDst: assign the write register, $rd (1) or $rt (0).
  - Branch: whether an instruction is a branch instruction.
  - MemRead: control data memory to read data from address (lw).
  - MemtoReg: choose source to write into register, memory-read-data (1) or ALU result (0).
  - ALUOp: from function code to determine operation ALU should operate.

- MemWrite: control data memory to write data to address (sw).
- ALUSrc: determine the input of ALU, sign extend immediate value (1) or $rt (0).
- RegWrite: enable to write to registers.

Review9. The Overview of Pipeline

2019年6月2日 10:07

Pipeline design and performance

- $\boldsymbol{Speed\ up\ of\ pipeline \approx number\ of\ stages}$
- Stages of pipeline: IF (instruction fetch), ID (instruction decode & register read), EX (execute ALU), MEM (memory access), WB (write back).
    - *Left Reg means write, and right Reg means read.*
- Pipeline speedup: $\boldsymbol{time\ between\ pipilined\ instructions = \frac{time\ between\ nonpipelined\ instructions}{number\ of\ stages}}$
    - Latency: time for each instruction.

Hazards

- Structure hazards: required resource is busy.
    - *Hardware problem, use separate memories of instruction and data.*
- Data hazards: wait for data read / write.
    - Forwarding: use result from ALU immediately into register read.
    - Load-use data hazard: use data from lw, solved by reorder.
- Control hazards: decisions depends on previous instruction.
    - Branch prediction: static (typical branch behavior) and dynamic (hardware record recent branch).

Review10. Instruction-Level Parallelism

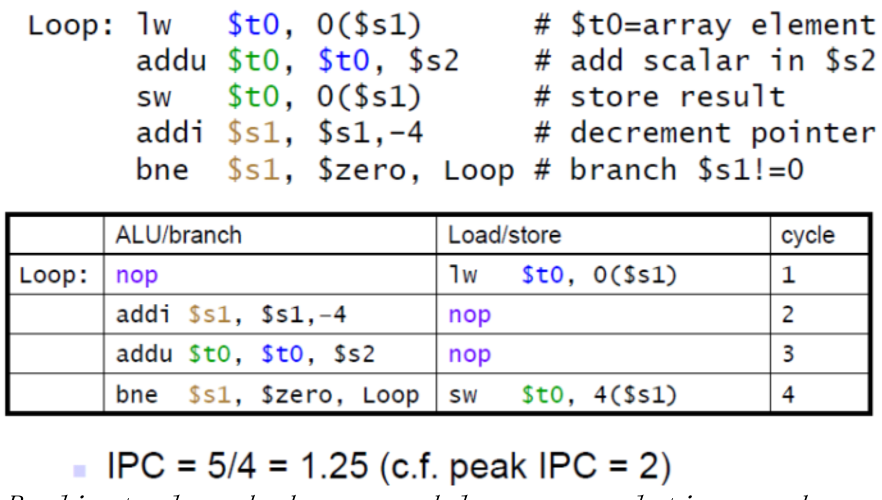
2019年6月2日 10:07

Multiple issue

- Instruction level parallelism (LIP): deeper pipeline, or multiple issue.
- $\boldsymbol{CPI < 1}$ then use $\boldsymbol{IPC = \frac{1}{CPI}}$.
- Static multiple issue by compiler and dynamic multiple issue by processor.
    - Static multiple issue: instructions into issue packets (very long instruction word, VLIW).
        - *Hazards: data hazard split into two packets.*
- Dynamic multiple issue (superscalar processor): CPU makes decision and no need for compiler scheduling.
- Speculation: guess and start operation as soon as possible.
    - Compiler reorder and hardware looks ahead for instructions (buffer results and flush on incorrect speculation).
    - Exceptions: static speculation add ISA support for deferring exceptions, dynamic speculation buffers exceptions until instruction completion.

Multiple issue scheduling

- Static multiple issue scheduling: reorder, dependencies between packets, pad with nop.
- An ALU / branch, load / store two packages multiple issue example:

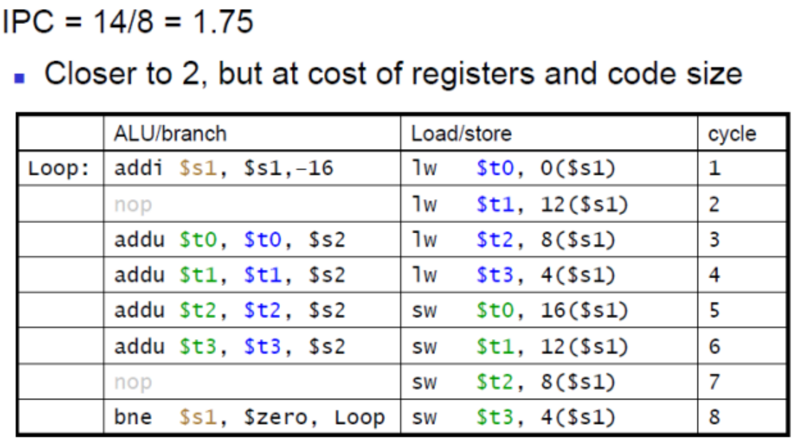


```
Loop: lw   $t0, 0($s1)       # $t0=array element
      addu $t0, $t0, $s2     # add scalar in $s2
      sw   $t0, 0($s1)       # store result
      addi $s1, $s1,-4       # decrement pointer
      bne  $s1, $zero, Loop  # branch $s1!=0
```

	ALU/branch	Load/store	cycle
Loop:	nop	lw $t0, 0($s1)	1
	addi $s1, $s1,-4	nop	2
	addu $t0, $t0, $s2	nop	3
	bne $s1, $zero, Loop	sw $t0, 4($s1)	4

- IPC = 5/4 = 1.25 (c.f. peak IPC = 2)

    - *Replicate loop body: expand loop several times and use different registers (register renaming).*
        - *Anti-dependencies (name dependence): no data flow between two instructions, only because of the name of register.*

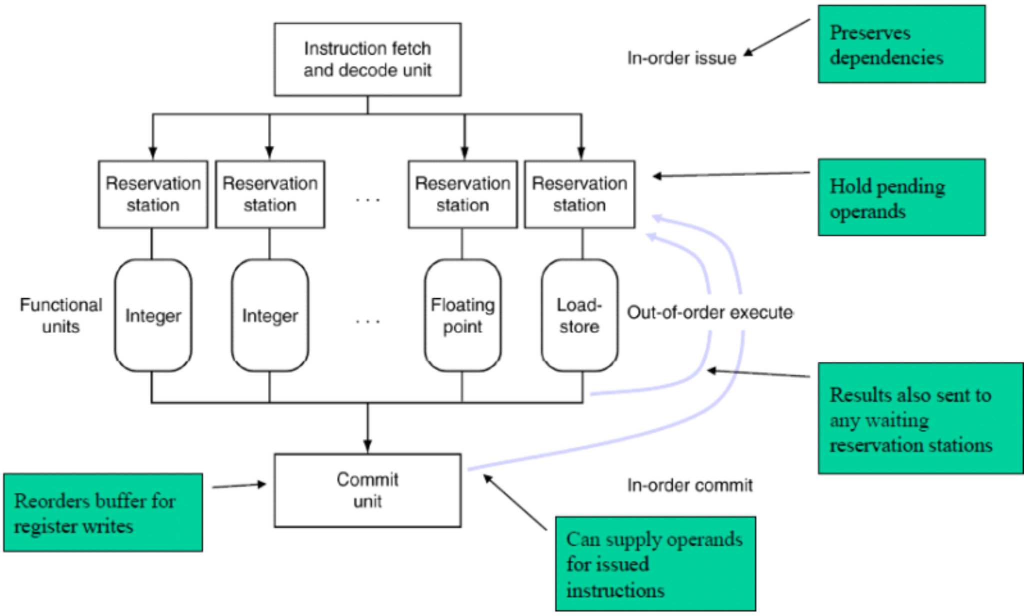


IPC = 14/8 = 1.75

- Closer to 2, but at cost of registers and code size

	ALU/branch	Load/store	cycle
Loop:	addi $s1, $s1,-16	lw $t0, 0($s1)	1
	nop	lw $t1, 12($s1)	2
	addu $t0, $t0, $s2	lw $t2, 8($s1)	3
	addu $t1, $t1, $s2	lw $t3, 4($s1)	4
	addu $t2, $t2, $s2	sw $t0, 16($s1)	5
	addu $t3, $t3, $s2	sw $t1, 12($s1)	6
	nop	sw $t2, 8($s1)	7
	bne $s1, $zero, Loop	sw $t3, 4($s1)	8

- Hardware support, CPU executes out of order (commit result to registers in order).



  - One IF / ID unit, several functional units, and one commit unit.

Review11. Large and Fast: Exploiting Memory Hierarchy

2019年6月2日 10:07

Memory

- DRAM (high bit density but poor latency) store data and instructions, SRAM designs cache.
  - Registers > L1 cache > L2 cache > memory > disk.
  - Memory in unit of byte or in unit of word.
- Cache: block is unit of copying.
  - *Miss, miss penalty, and miss ratio - hit.*
  - Temporal locality and spatial locality.
  - $\boldsymbol{Access\ time = hit\ time + miss\ rate \times miss\ penalty}$
- DRAM: refresh to keep data. Store in bank, with several row, and each row has several column.
  - *Pre to open / close a bank and act to access one or several (burst mode) rows (into buffer).*
  - *SDRAM (synchronous DRAM) adds a clock to synchronize with processor. Burst mode won't send multiple addresses.*
    - *DDR (double data rate) DRAM: transfer on both sides of clock.*
- Flash: nonvolatile semiconductor storage.
  - *Flash bits wears out, and use wear leveling to remap data to less used blocks.*
- Disk: nonvolatile rotating magnetic storage.
  - *Track, sector, queuing delay, seek (find track), rotational latency, data transfer, controller overhead.*

Cache map

- Direct mapped cache: index, valid bit, tag, and data.
  - Index: from 0 to n - 1 of cache size, which is lower bits of memory address.
  - Tag: high order bits of memory address.
  - Valid bit: whether there is data in a location.
  - Data: data from memory.
- For unit of word memory, two bits of byte offset determines which byte to access.
- Larger block size: can be considered as more offset bits.
  - For 32 bits address, block size m word (m+2 bits), n bits of index:
    - $\boldsymbol{cache\ blocks = 2^n}$
    - $\boldsymbol{tag\ bits = 32 - n - m - 2}$
    - $\boldsymbol{size\ of\ cache = 2^n_{blocks} \times \left(2^{m+5}_{block\ size\ in\ byte} + (32 - n - m - 2)_{tag\ size} + 1_{valid\ bit}\right)}$
- Cache miss: stall CPU pipeline and fetch block (instruction cache miss and data cache miss).
- Write through (update memory on cache change, use write buffer) and

write back (update memory when leave cache, and log dirty bit).

- *Write allocate: on write miss,*
    - *Write through (alternatives): allocate and fetch the block, or write around and don't fetch the block.*
    - *Write back: fetch the block, in fact much complex.*

Review12. Memory Performance and Dependable Memory Hierarchy

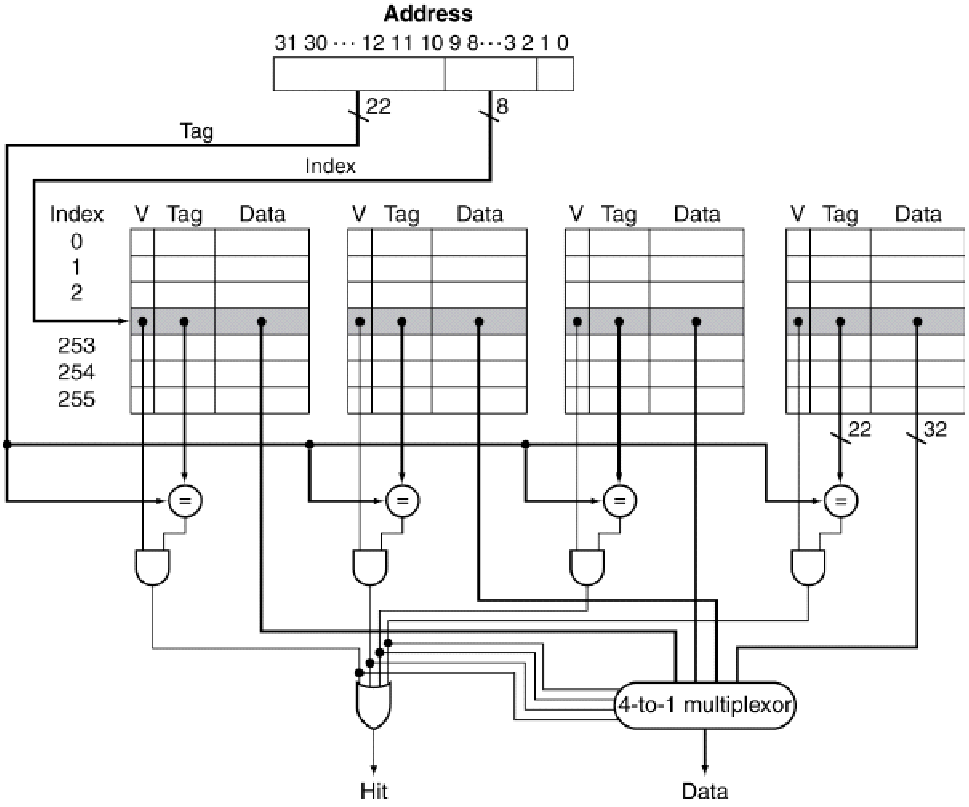
2019年6月2日 10:07

Cache performance

- CPU time: program execution cycles and memory stall cycles.
    - $\boldsymbol{Memory\ stall\ cycles}$ $\boldsymbol{= Instructions\ per\ program \times miss\ rate \times miss\ penalty}$
    - I-cache (instruction cache) miss and D-cache (data cache) miss: all instructions can cause I-cache miss, while only lw / sw can cause D-cache miss.
- $\boldsymbol{AMAT(average\ memory\ access\ time)}$ $\boldsymbol{= hit\ time + miss\ rate \times miss\ penalty}$

Associative caches

- Group blocks into set: n-way associative n blocks in one set, fully associative n blocks in one set.



    - N-way associative needs n comparator, and search n times, and n places to put blocks.
- Replacement policy: non-valid, then LRU (least-recently used), or random (approximately LRU for high associativity).

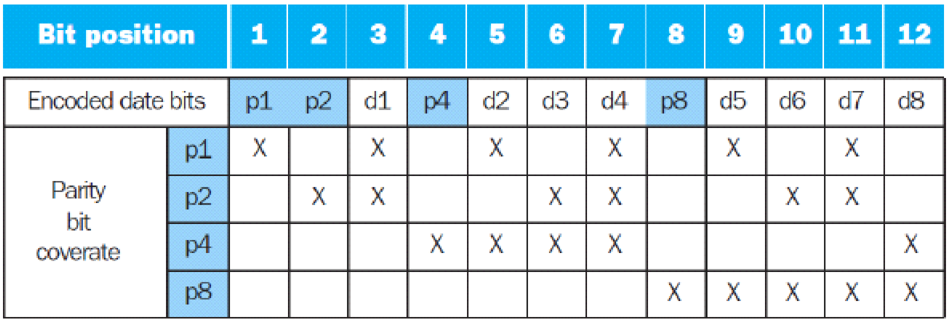
Multilevel caches

- Primary cache minimal hit time, L-2 cache low miss rate.
- *Service accomplishment failure to service interruption, then restore.*
    - *Reliability: MTTF (mean time to failure).*

- *Service interruption: MTTR (mean time to repair).*
- $\boldsymbol{MTBF(mean\ time\ between\ failures) = MTTF + MTTR}$
- $\boldsymbol{Availability = \frac{MTTF}{MTTF + MTTR}}$

The Hamming SEC code

- Hamming distance: number of different bits between two bit pattern.
  - Distance 2 (parity code) SED, distance 3 SEC / DED.
  - Distance n can check $\boldsymbol{n-1}$ errors, correct $\left\lfloor\boldsymbol{\frac{n-1}{2}}\right\rfloor$ errors.
- Encoding SEC: bit position from 1 to n, equals $\boldsymbol{2^i}$ is a parity bit, other are data bits.



Bit position		1	2	3	4	5	6	7	8	9	10	11	12
Encoded date bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8
Parity bit coverate	p1	X		X		X		X		X		X	
	p2		X	X			X	X			X	X	
	p4				X	X	X	X					X
	p8								X	X	X	X	X

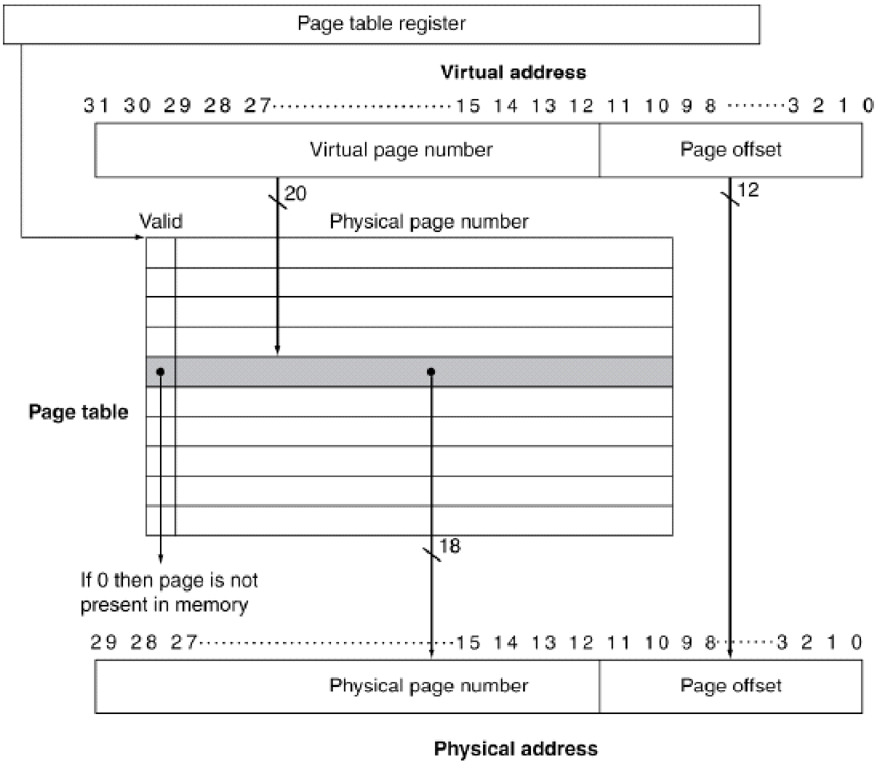
  - Encode and decode, error of parity bits can identify error position.
- SEC / DED code: add an additional parity bit for whole word p_n.
  - Hamming distance becomes 4.
  - H is SEC parity bits: H even p_n even no error, p_n odd p_n error. H odd p_n odd correct one error, p_n even detect double error.
    - p_n is the additional bit, at the end of pattern. H is the calculated SEC parity bits after transfer (includes p_n should be considered).
  - Can detect at most 3 error, but don't know exactly which bit wrong.

Review13. Virtual Memory

2019年6月2日 10:07

Virtual memory

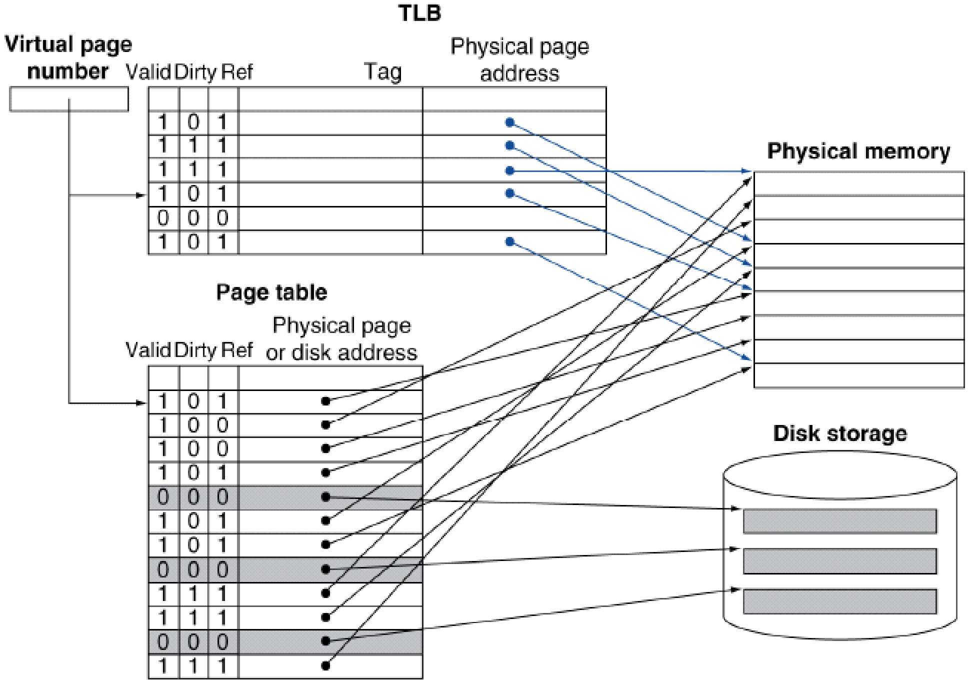
- Programs share main memory and use private virtual memory.
  - VM page - block, VM page fault - miss.
- Page table: index PTE (page table entries) by virtual page number.



  - Page table register in CPU and page table in memory.
  - PTE: stores physical page number, referenced, dirty, valid…
  - Page not present: PTE refer to location in swap space on disk.
- Replacement: LRU replacement using reference bit (use bit).
- Write: use write-back and set dirty bit in PTE.

Fast translation using TLB

- TLB (translation lookaside buffer): store recently used PTEs.



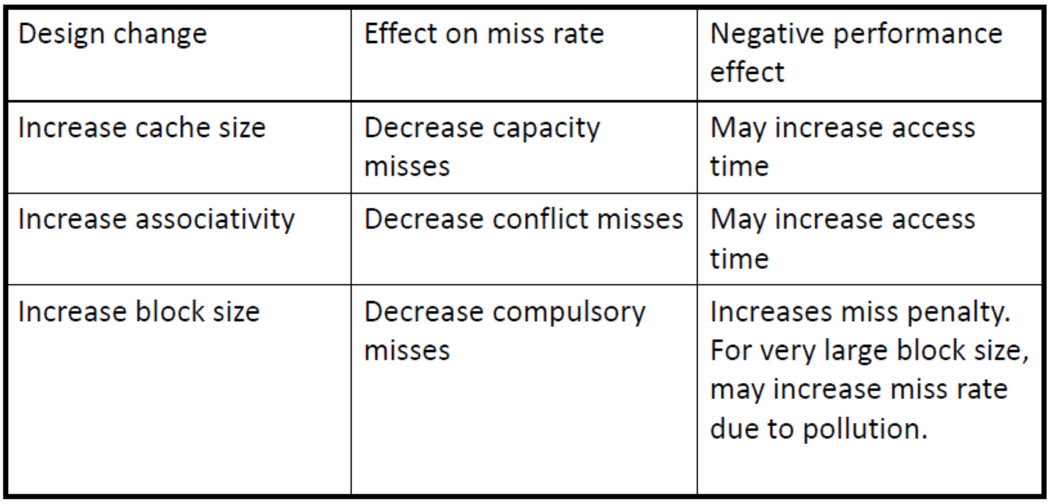
Valid	Dirty	Ref
1	0	1
1	1	1
1	1	1
1	0	1
0	0	0
1	0	1

Valid	Dirty	Ref
1	0	1
1	0	0
1	0	0
1	0	1
0	0	0
1	0	1
1	0	1
0	0	0
1	1	1
1	1	1
0	0	0
1	1	1

    - Tag stores high bits of PTE virtual memory address.
- TLB miss: search PTE in memory.
    - Page in memory: load PTE and retry.
        - *Hardware design more complicated page table, software raises a special exception.*
    - Page not in memory: page fault, *OS handles fetching page and restart the faulting instruction.*
- TLB to find address and cache to find data.

Memory hierarchy

- *Memory protection: tasks share parts of their virtual address spaces.*
    - *Privileged supervisor mode (kernel mode).*
    - *Privileged instructions and read only data (page table, state information).*
    - *Switch between kernel mode and user mode (system call exception).*
- *Block placement: direct map, n-way set associative, and fully associative.*
- Finding a block: full lookup table of fully associative cost 0 comparison.
- *Replacement on a miss: LRU or random.*
    - Virtual memory uses LRU approximation with hardware support while cache uses either LRU or random.
- *Write policy: write-through with write buffer, write-back with more states.*
- Sources of misses: compulsory miss (cold start miss), capacity miss, conflict miss (collision miss).



Design change	Effect on miss rate	Negative performance effect
Increase cache size	Decrease capacity misses	May increase access time
Increase associativity	Decrease conflict misses	May increase access time
Increase block size	Decrease compulsory misses	Increases miss penalty. For very large block size, may increase miss rate due to pollution.

Virtual machines

- *Host computer emulates guest operating system and machine resources.*
  - *Improve isolation, avoid security and reliability problems, share resources.*
  - *Has some performance impact.*
- *Virtual machine monitor:*
  - *Maps virtual resources to physical resources.*
  - *Guest in user mode while VMM in privileged supervisor mode.*
  - *Guest OS different from host OS.*
  - *VMM handles real I/O devices and emulates for guest.*
- *Instruction set support.*

Review14. Parallel Processors from Client to Cloud

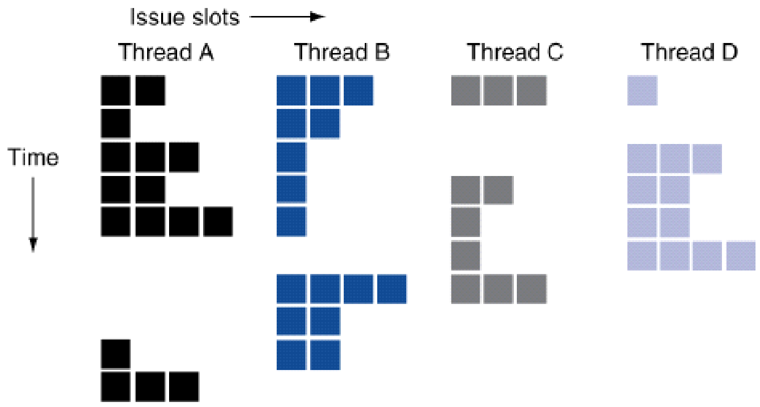
2019年6月2日 10:07

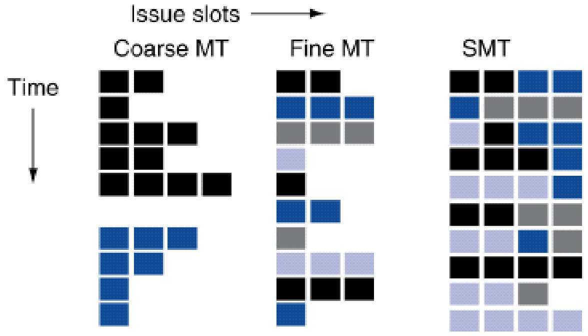
Multiprocessors

- *Parallel processing program: efficiently executed.*
  - Hardware: serial and parallel.
  - Software: sequential and concurrent.
- Parallel and scaling: partition, coordination, and communication overhead.
  - Amdahl's Law: $T_{new} = \frac{T_{parallelizable}}{100} + T_{sequential}.$
  - Strong scaling: problem size fixed.
  - Weak scaling: problem size increases due to number of processors increases.
- *Load balancing: each processor has same number of tasks.*

Level parallelism

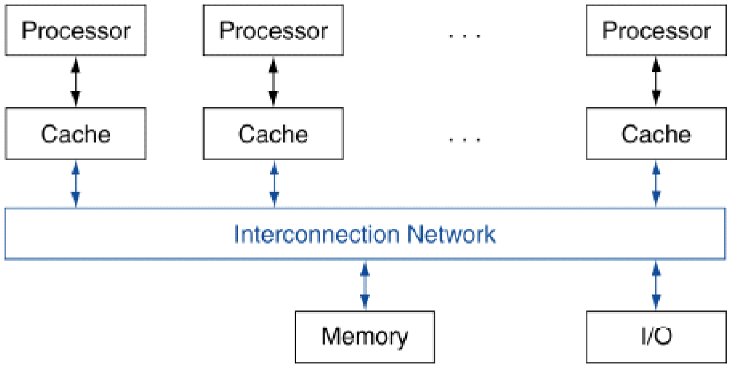
- Parallel processing: SISD (single instruction stream, single data stream), MIMD (multiple instruction multiple data), SPMD (single program multiple data), SIMD (single instruction multiple data), and vector.
  - SPMD: program on a multi-core processor, different processor execute on different sections of code.
  - SIMD: operate on vectors of data (data level parallelism), processors execute same instruction on different data address.
  - Vector processors: designed for vector operation, pipelined execution units.
- Multithreading: related to MIMD, multiple threads share a single processor.
  - *Fine-grain multithreading: switch threads, interleave instruction execution, and execute other threads when one is stall.*
  - SMT (simultaneous multithreading): multiple-issue dynamically scheduled processor.



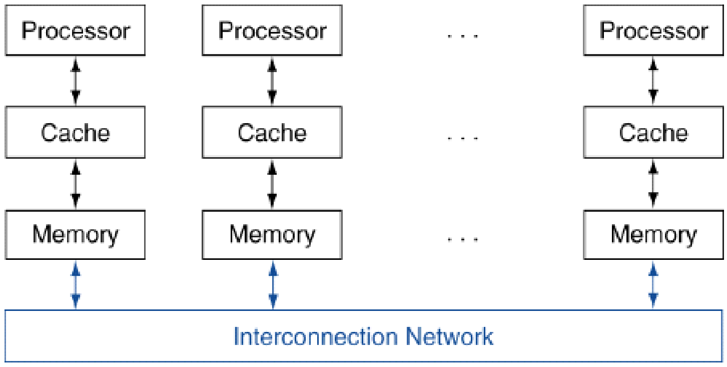


Multicore microprocessors

- *SMP (shared memory multiprocessor): efficiently programming on multiprocessor.*



- Message parsing multiprocessors: each processor has private physical address space, and share messages via hardware.



- *Loosely coupled clusters: network of independent computers.*
- *Cloud computer: WSC (warehouse scale computers), SaaS (software as a service), PMD (personal mobile device) or cloud running software.*
- *Data center: computers connected by off-the-shelf networking devices.*

Other processor units

- *GPU (graphic processing unit): highly data-parallel processing,* *oriented towards bandwidth.*

- *TPU (tensor processing unit): tensorflow platform.*
- *DPU (deep learning processing unit): FPGA-based processing unit.*
- *NPU (neural network processing unit): IBM TrueNorth.*
- *BPU (brain processing unit).*