Server: dependability, scalability, efficient throughput
Embedded: Strict resource constraints
Average Selling Price (ASP):
Component cost + direct cost + indirect cost.
List price:
Not ASP. Stores add to the ASP to get their cut. Want 50% to 75% of list price.
Classes of computers
SISD：每个时钟周期一条数据流和一条指令流
MISD：每个时钟周期多条指令流，单个数据集
SIMD：每个时钟周期在多个数据流上执行单个指令流
ISA: Instruction Set Architecture 指令集架构 the interface between hardware and software
Interface Design: 1. Provides convenient functionality to higher levels 2. Permits an efficient implementation at lower levels
Computer Architecture is the science and art of selecting and inter-connecting hardware components to create computers that meet functional, performance, cost and power goals.
1. Instruction Set design 2.Organization 3. Hardware
MTTF 平均故障时间，MTTR 平均修复时间，FIT=1/MTTF 故障率， MTBF = MTTF + MTTR 平均无故障时间，模块可用性=MTTF/MTBF
Measuring and Reporting Performance
Execution time (latency) Throughput MIPS
Criteria of performance evaluation differs among users and designers
response time = elapsed time
CPU time - Measures designer perception of the CPU speed. Further divided into: User CPU time and System CPU time
Throughput - Measure administrator perception of the system performance.
If you improve response time, you usually improve throughput. You can also improve throughput without improving response time
MIPS comparing **the same** Instruction Set
SPEC - The System Performance Evaluation Cooperative
Total Execution Time: A Consistent Summary Measure

$$\sum_{i=1}^{n} Weight_i \times Time_i \qquad \frac{1}{\sum_{i=1}^{n} \frac{Weight_i}{Rate_i}}$$

Fraction enhanced: Fraction of computation time in original machine that can be converted to take advantage of the enhancement. **Speedup = [1- ΣFEi + ΣFEi/Si]-1**

$$Speedup_{overall} = \frac{ExecTime_{old}}{ExecTime_{new}} = \frac{1}{\left((1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}\right)}$$

Instruction Set Design
The type of internal storage in CPU: stack, accumulator, GPR architecture
Three general types of GPR (General Purpose Registers)
1. Register-Register ------ Load/Store
ALU operations use register operands only
Alpha, ARM, MIPS, PowerPC, SPARC, superH, TM
2. Register-Memory
IBM360/370, Inter80x86, Ti TM Motorola 6800
3. Memory-Memory Vas

Metrics	Reg-Reg	Reg-mem	Mem-Mem
Code density	Lowest	Higher	Highest
Instr. count	Largest	Large	Small
Instr. Comp.	Simplest	Complex	Most Comp
Instr. length	Fixed	Variable	Large vari-
Encoding com	Fixed	Hybrid	Variable
CPI	Small	Middle	Large vari-

Memory address bit byte word
1. Support at least 3 addressing mode Register indirect, displacement, immediate 2. The size of the address for displacement mode to be at least 12-16 bits 3.The size of the immediate field to be at least 8-16 bits
Little Endian: Intel **Big Endian**: IBM, Motorola
MAKE THE COMMON CASE FAST
Instructions for Control flow
1. Common used instructions shall be considered firstly: Load, store, add, sub, move R-R, and, shift, =, ≠, branch and etc. 2. Conditional branch: displacement 100 <=2^7
PC-relative branch: displacement > 8 bits. 3. PC-relative conditional branches dominate the control instructions. 4. Jump and link instruction for procedure call. 5. Register indirect jump for procedure return
Three areas in which current high-level languages allocate the data:
1. Stack: local variables; scalars (single variables)
2. Global data area: global variables, constants; arrays
3. Heap: dynamic objects; accessed with pointers
At least 16 GPRs + separate floating-point registers
Caller-saving: Caller saves any registers that it wants to use after the call, then invoke.
Callee-saving: First invoke, then callee saves the registers.
RISC (Reduced Instruction Set Computer)
CISC (Complex Instruction Set Computer)
The MIPS Architecture
1. A simple load-store instruction set
2. Design for pipelining efficiency, including a fixed instruction set encoding
3. Efficiency as a complier target

Pipeline

Goal: make fast CPU
How: takes advantage of parallelism
Decompose by steps
Pipeline - Implementation technique whereby different instructions are **overlapped** in execution at the same time.
Make fast CPU→decrease CPU time→improve throughput; improve efficiency for resources
Pipeline: 1. Many stages 2. Each stage carries out a different part 3. Cooperate at a synchronized clock. 4. Exploit parallelism
Latency: It's the amount of time between when the instruction is issued and when it completes.
Throughput: The throughput of a CPU pipeline is the number of instructions completed per second
Ideal speedup equal to Number of pipeline stages
Single clock - pipelining decreases cycle time.
Multiple clock - pipelining decreases CPI
Performance issues: 1. Latency. 2. Imbalance among stages. 3. Overhead. 4. Pipeline hazards. 5. Time to "fill" and "drain".
MIPS 5 stage pipeline

Hazard

A hazard is a condition that prevents an instruction in the pipeline from executing its next scheduled pipeline stage.
Hazards can $\boxed{\text{always}}$ be resolved by **Stall**
1. The stall delays all instructions issued after the instruction that was stalled, while other instructions in the pipeline go on proceeding.
2. No new instructions are fetched during a stall
Case of multi-cycle implementation

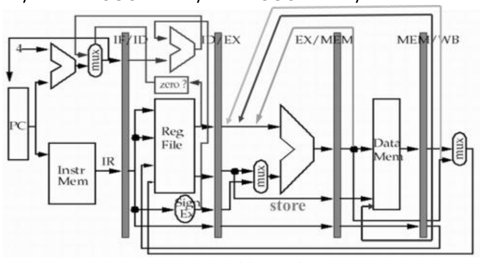
$$\text{CPI pipelined} = \text{Ideal CPI} + \text{Pipeline stall clk cycles per instuction}$$
$$= 1 + \text{Pipeline stall clk cycles per instruction}$$
$$\text{Speedup} = \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall cycles per instruction}}$$

Structural hazard

Occurs when two or more instructions want to use the same hardware resource in same cycle
Overcome by replicating hardware resources
1. Multiple accesses to the register file -- double bump 双端口. 2. Multiple accesses to memory -- split IM and DM / multiple memory port / instr. buffer. 3. Some functional unit is not fully pipelined. 4. Not pipelined functional units
Allow: to reduce cost and latency of the unit

Data hazard

Occur when the pipeline changes the order of read/write accesses to operands comparing with that in sequential executing
Dependencies between "nearby" instructions
1. Insert NOP by compiler
2. Add hardware Interlock
Add extra hardware to detect stall situations
Add extra hardware to push bubbles thru pipe
3. $\boxed{\textbf{Forwarding}}$ (bypass, short-circuiting)
"point backwards"
Load stall – avoid: reordering by compiler
EX/MEM.ALUOUT MEM/WB.ALUOUT MEM/WB.LMD

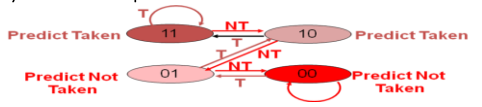


Control hazard

Cause: branch condition and the branch PC are not available in time to fetch an instruction on the next clock.
The next PC takes time to compute
1. Freeze or flush the pipeline
Holding or deleting any instr. after branch until the branch destination is known. Penalty is fixed.
2. Predict-not-taken (Predict-untaken)
3. Predict-taken
Only useful when the target is known before the branch outcome.
No advantage at all for MIPS 5-stage pipeline
4. Delayed branch
Instruction always executes no matter whether the branch taken or not taken.
RAW (Read after write) true depend. B read b4 A write
WAW (Write after write) output depend. B write b4 A write
WAR (Write after read) anti-depend. B write b4 A read

Branch prediction

Static branch prediction
Dynamic branch prediction



Exceptions and Interrupts

Exceptions are *exceptional* events that *disrupt* the normal flow of a program
Pipeline must be safely shutdown when exception occurs and then **restarted at the offending instruction**
Precise Exceptions vs. Imprecise Exceptions
Latency----the number of intervening cycles between an instruction that produces a result and an instruction that uses the result.
Initiation interval----the number of cycles that must elapse between instructions issue to the same unit.
Supports multiple outstanding FP operations
Structural hazards can occur.
Solve the write port conflict
Detect and insert stalls by serializing the writes
1. Track the use of the write port in the ID stage and to stall an instruction before it issues
2. To stall a conflicting instruction when it tries to enter the MEM or WB stage.
Example: Interrupts-User hitting the keyboard, Disk drive asking for attention, Arrival of a network packet
Exceptions- Divide by zero, Overflow, Page fault

Handling exceptions

1. Ignore the problem (imprecise exceptions)
2. Buffer the results and delay commitment
History file - saves the original values of the registers.
Future file - stores the newer values for registers.
3. Keep enough information for the trap handler to create a precise exception for the exception
4. Allow instruction issue only if it is known that all previous instructions will complete without causing an exception.
Designing instruction sets for pipelining
1. Avoid variable instruction lengths and running times whenever possible.
2. Avoid sophisticated addressing modes.
3. Don't allow self-modifying code.
4. Avoid implicitly setting CCs in instructions

Memory - Hierarchy Design

Taking advantage of the principle of **locality** 局部性原则
Cache -- Small, fast storage used to improve average access time to slow memory
Block placement
Fully Associative, Set Associative, Direct Mapped
Sets with n blocks -- n-way set associative
Block identification Address Tag / Block

TAG	Index	offset

Block replacement -- Random, LRU, FIFO
Write strategy
When data is written into the cache,
1. **Write-through cache**: also written to memory.
Can always discard cached data - most up-to-date data is in memory
Cache control bit: only a *valid* bit
Memory always has latest data
2. **Write-back cache**: NOT written to memory
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 advantages: Read misses don't result in writes, memory hierarchy is consistent and it is simple to implement.
Write back advantages: Writes occur at speed of cache and main memory bandwidth is smaller when multiple writes occur to the same block.

Write stall ---When the CPU must wait for writes to complete during write through
Write buffers -- A small cache that can hold a few values waiting to go to main memory.
To avoid stalling on writes, many CPUs use a write buffer. This buffer helps when writes are clustered. It does not entirely eliminate stalls since it is possible for the buffer to fill if the burst is larger than the buffer.
Write misses -- If a miss occurs on a write (the block is not present), there are two options:
1. **Write allocate -** The block is loaded into the cache on a miss before anything else occurs.
2. **Write around -** The block is only written to main memory. It is not stored in the cache.
In general, write-back caches use write-allocate, and write-through caches use write-around.

Cache performance
CPI_{Exec} includes ALU and Memory instructions
CPU Execution time = (CPU clock cycles + Memory stall cycles) × Clock cycle time

$$\text{Memory stall cycles} = \text{IC} \times \text{Mem refs per instruction} \times \text{Miss rate} \times \text{Miss penalty}$$

Average Memory Access Time
AMAT = Hit time + (Miss Rate × Miss Penalty)

$$=(HitTime_{Inst}+MissRate_{Inst}\times MissPenalty_{Inst})\times Inst\%$$
$$(HitTime_{Data}+MissRate_{Data}\times MissPenalty_{Data})\times Data\%$$
$$CPUtime = IC * (\frac{AluOps}{Inst} * CPI_{AluOps} + \frac{MemAcc}{Inst} * AMAT) * CC$$
$$CPUtime = IC \times \left(\frac{AluOps}{Inst} \times CPI_{AluOps} + \frac{MemAccess}{Inst} \times AMAT\right) \times CycleTime$$

Compulsory—cold start/ first reference misses
Capacity—can't contain all the blocks needed
Conflict—in set associative or direct mapped, a too many blocks map to a set. Collision / interference misses.

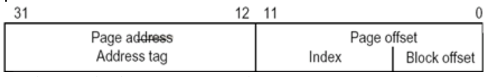
How to improve CPU time

1. Reduce the miss penalty--5
a. Multilevel caches
Add a second-level cache between main memory and a small, fast first-level cache.
AMAT = Hit $Time_{L1}$ + Miss $Rate_{L1}$ x (Hit $Time_{L2}$ + Miss $Rate_{L2}$(local) x Miss $Penalty_{L2}$)
L1 cache: Misses: 40 misses/1000 memory; Hit time: 1 CC; MPI: 1.5 ;
L2 cache: 20 misses ; Miss penalty: 100 CC; Hit time: 10 CC
AMAT and average stall cycles per instruction ?
Answer: Calculating Miss rate for local and global.
MRL1=40/1000=4%; MRL2=20/40=50%; MRL2g=20/1000=2%
AMAT=1+4% ×(10+50% ×100)=1+4% ×6=3.4 clock cycle
L1(global): 1.5 × 4% ×1000=60 ; L2(g):1.5 × 2% ×1000=30
Average memory stalls per instruction
=Misses per $instruction_{L1(global)}$ ×Hit $time_{L2}$+Misses per $instruction_{L2(global)}$ ×Miss $penalty_{L2}$
=(60/1000) ×10+(30/1000) ×100=3.6 clock cycles
→Reduce miss rate of the second-level cache
b. 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.
c. Read miss before write miss
With a write buffer, writes can be delayed to come after reads. Write-through -- Check write buffer contents before read; Write-back -- copy the dirty block to a write buffer, then do the read
d. Merging write buffers
Merge multiword writes to one. Also reduces stalls due to the write buffer being full.
e. victim caches
Small fully-associative cache. Hold a few of most recently replaced blocks. Check on miss.

2. Reduce the miss rate--5
Assume total cache size not changed
a. larger block size -- Decrease compulsory miss rate (patial locality). May increase the miss penalty. Certainly increase conflict misses.
b. large cache size – Longer hit time & Higher cost
Old rule of thumb: 2 x size => 25% cut in miss rate
c. higher associativity -decreasing Conflict misses to improve miss rate. **2:1 rule** direct-mapped cache of size N has the same miss rate as a 2-way set-associative cache of size N/2. Eight-way set associative ≈ Fully associative
d. way prediction and pseudo-associativity
Reduce conflict misses and maintains hit speed of direct-mapped cache. (Column associative)
e. compiler optimizations
Reorder instr. Data: 1) Merging Arrays, 2) Loop Interchange, 3) Loop Fusion, 4) Blocking

3. Reduce the MP and MR via parallelism-- 3
a. non-blocking caches (reduce MP)
Continue to supply hits while processing read misses. Conjunction with out-of-order execution.
b. hardware prefetching (reduce MR)
Get data from memory before actually needed.
Rely on having extra memory bandwidth.
c. compiler prefetching (reduce MR)
Binding prefetch: Requests load directly into register.
Non-Binding prefetch: Load into cache.

4. Reduce the time to hit in the cache.--4
a. small and simple caches
Using small and Direct-mapped cache
b. avoiding address translation
Translation Lookaside Buffer (TLB)
Index with Physical Portion of Address -- Start tag access in parallel with translation



Dealing with aliases -- page coloring
c. pipelined cache access
d. trace caches
Find a dynamic sequence of instructions to load a cache block. Cache blocks contains dynamic traces of the executed instructions

Summary

	Technique	MP	MR	HT	Complexity
miss penalty	1.Multilevel caches	+			2
	2.Early Restart & Critical Word 1st	+			2
	3.Priority to Read Misses	+			1
	4.Merging write buffer	+			1
	5.Victim caches	+	+		2
miss rate	6.Larger block size	–	+		0
	7.Larger cache size		+	-	1
	8.Higher Associativity		+	-	1
	9.Way-predicting cache and Pseudoassociative aches		+		2
	10.Compiler techniques reduce cache misses		+		0
parallelism	11.Non-Blocking Caches	+			3
	12.HW Prefetching of Instr/Data	+	+		2instr./3data
	13.Compiler Controlled Prefetching	+	+		3
hit time	14.Small & Simple Caches		–	+	0
	15.Avoiding Address Translation			+	2
	16.Pipelined Cache Access			+	1
	17.Trace cache			+	3

Main memory

Higher Bandwidth
4 send address, 56 access/word, 4 send a word
1. Wider Main Memory
Doubling or quadrupling the width of the cache and the memory. 2*(4+56+4)
2. Simple Interleaved Memory
Memory chips are organized in banks to read or write multiple words at a time. 4+56+4*4
# of banks ≥ # of CC to access word in bank
3. Independent Memory Banks
Independent memory controller was present for every bank. Avoiding Memory Bank Conflicts -- use a prime number of memory banks 2^n-1

Memory Technology
Access time ----- time between read is requested and desired word arrives. **Cycle time** ----- minimum time between requests to memory.
Random Access Memory
1. SRAM – Cache, no refresh
2. DRAM– Mainstream Main Memory, refreshed periodically, divided into 2 halves RAS (Access Strobe) & CAS
Embedded: Read-Only Memory and Flash Memory.

Virtual Memory
Provides illusion of very large memory
Virtual memory space - what the program "sees"
Physical memory space - what the program runs in
Pages-fixed-size blocks, **segments**-variable-size
1. Place block: fully associative strategy, anywhere
2. Find block: segmentation-the offset is added to the segment's PA; paging-the offset is simply concatenated to this page's PA
3. Replace block: LRU block-minimize page faults
4. Write strategy: always write back, dirty bit

Pseudo-associative 伪相联
AMATx = HTx + MRx * MPx
HTx = HTy + HRx * C1
HRx = HRz - HRy =（1 – Hry） - (1 - HRz) = MRy - MRz
HTx = HTy + (MRy - MRz) * C1
MRx = MRz
MPx = MPy = MPz
AMATx = HTy + (MRy - MRz) * C1 + MRz * MPy
x 表示伪相联，y 表示一路组相联，z 表示二路组相联
C1 是伪关联命中时用的时钟周期
· Suppose a processor executes at
- Clock Rate = 200 MHz (5 ns per cycle), Ideal (no misses) CPI = 1.1
- 50% arith/logic, 30% ld/st, 20% control
· 10% of memory operations get 50 cycle miss penalty
· 1% of instructions get same miss penalty
· What is the CPUtime and the AMAT ?
·Answer:CPI = ideal CPI + average stalls per instruction
= 1.1(cycles/ins) + [0.30 x 0.10 x 50] + [1 x 0.01 x 50] = (1.1 + 1.5 + .5) cycle/ins = 3.1
·AMAT=(1/1.3)x[1+0.01x50]+(0.3/1.3)x[1+0.1x50]=2.54

Ideal CPI=1 (no misses) ;50% of instructions are data accesses ; Miss penalty is 25 clock cycles ;Miss rate is 2% ; How faster would the computer be if all instructions were cache hits?
• Answer: first compute the performance for always hits:
$CPU_{execution\ time}$ =(CPU clock cycles+memory stall cycles)×clock cycle
=(IC ×CPI+0) ×Clock cycle
=IC ×1.0 ×clock cycle
Now for real cache, first compute memory stall cycles:

$$Memory\ stall\ cycles = IC \times \frac{Memory\ accesses}{Instruction} \times Missrate \times Miss\ penalty$$
$$= IC \times (1+0.5) \times 0.02 \times 25 = IC \times 0.75$$

CPU execution time cache =(IC ×1.0+IC ×0.75) ×Clock cycle
=1.75 ×IC ×Clock cycle

CPI=2(perfect cache) CC=1.0 ns; MPI(memory reference per instruction)=1.5 ; both caches is 64K; both block is 64 bytes; One cache is direct mapped and other is two-way set associative. the former has miss rate of 1.4%, the latter 1.0%; The selection multiplexor forces CPU clock cycle time to be stretched 1.25 times; Miss penalty is 75ns,and hit time is 1 clock cycle; What is the impact of two diffect cache organizations on performance of CPU?
Average memory access time=Hit time+Miss rate×miss penalty
$AMAT_{1\text{-}way}$=1.0+(0.14 ×75)=2.05 ns ; $AMAT_{2\text{-}way}$=1.0×1.25+(0.01 ×75)=2.00 ns

$$CPUtime = IC \times \left[CPI_{execution} + \left(Miss\ rate \times \frac{Mem\ Acc}{Instr} \times Miss\ penalty\right)\right] \times CC$$

Substituting 75 ns for (miss penalty×Clock cycle time)
$CPU\ time_{1\text{-}way}$=IC×(2×1.0+(1.5 ×0.014 ×75))=3.58 ×IC
$CPU\ time_{2\text{-}way}$=IC×(2×1.0×1.25+(1.5 ×0.010 ×75))=3.63 ×IC

Unified caches: 32K; Split cache: 16K D-cache and 16K I-cache ; 36% of the instructions are data transfer instructions ; A hit takes 1 CC ; Miss penalty 100 CC ;A load/store take 1 extra CC on a unified cache; Write-through with a write-buffer. Miss rate in each case? AMAT each case?
Answer : Convert misses per 1000 instr. into miss rate.
The unified miss rate for instruction and data accesses:

$$Miss\ rate_{32KB\ unified} = \frac{43.3/1000}{1.00+0.36} = 0.0318$$
$$Miss\ rate_{16KB\ instruction} = \frac{3.82/1000}{1.0} = 0.004$$
$$Miss\ rate_{16KB\ data} = \frac{40.9/1000}{0.36} = 0.114$$

Average miss rate for the split cache is:(74%×0.004)+(26% × 0.114)=0.0324
$AMAT_{split}$=74%×(1+0.004×100)+ 26%×(1+0.114×100)=4.24
$AMAT_{unified}$=74%×(1+0.0318×100)+26%×(1+1+0.0318×100)=4.44

4. (50points)You are building a system around a processor with in-order execution that runs at 1.1 GHz an has a CPI of 0.7 excluding memory accesses. The only instructions the read or write data from memory are loads (20% of all instructions) and stores (5% of all instructions).
The memory system for this computer is composed of a split L1 cache that impose no penalty on hits. Both the I-cache and D-cache are direct mapped and hold 32KB each. The I-cache has a 2% miss rate and 32-byte blocks, and the D-cache is write through with a 5% mis rate and 16-byte blocks. There is a write buffer on the D-cache that eliminates stalls for 95% of all writes.
The 512KB write-back, unified L2 cache has 64-byte blocks and an access time of 15ns. It is connected to the L1 cache by a 128-bit data bus that runs at 266MHz and can transfer on 128-bit word per bus cycle. Of all memory references sent to the L2 cache in this system, 80% are satisfied without going to main memory. Also 50% of all blocks replaced are dirty.
The 128-bit-wide main memory has an access latency of 60ns, after which any number of bus words may be transferred at the rate of one per cycle on the 128-bit-wide 133 MHz main memory bus.
a) What is the average memory access time for instruction accesses ?
b) What is the average memory access time for data reads ?
c) What is the average memory access time for data writes ?
d) What is the overall CPI, including memory accesses ?
e) You are considering replacing the 1.1GHz CPU with one that runs at 2.1GHz, but is otherwise identical. How much faster doe the system run with a faster processor ? Assume the L1 cache still has no hit penalty, and that the speed of the L2 cache, main memory, and buses remains the same in absolute terms (e.g. the L2 cache has a 15n access time and a 266MHz bus connecting it to the CPU and L1 cahce.

Assumption:
CR = 1.1 GHz cycle = 1/(1.1*10^9) = 0.9 (ns)
CPI excluding memory = 0.7 (cycle)
Load% = 20%
Store% = 5%

L1: HitTime = 0
I-cache: direct mapped, Capacity = 32KB, MR = 2%, 32B/block;
D-cache: direct mapped, wirte through, Capacity=32KB, MR=5%, 16B/block;
Write buffer eliminates 95% stall.

L2: Capacity=512KB, write back 64B/block, Accesstime=15ns, 128bit/data bus,
64*8/128=4(times transformation for one block)
TR= 266MHz, 128bit/transfer cycle,
one transfer cycle time(TT_{L2}) = 1/(266*10^6) = 3.76ns
MR= 20%, DirtyRate = 50%

MM: 128bit wide, access latency=60ns, 128bit/transfer cycle,
one transfer cycle time(TT_{mm}) = 1/(133*10^6) = 7.52ns
load / write block latency = 60ns + 4*7.52 = 90ns

a. AMAT of Instruction Access
AMATinstr = HitTime_I-cache + MRi-cache* (HT-L2 + Time to fetch miss block from L2 + Time to fetch from MM + Time to write back the dirty block when miss)
= 0+ 2% * (15 + (32*8/128) * TT_{L2}+ 20% * ((60 + (64*8/128) * TTmm + 20%*50%* (60 + 64*8/128*TTmm))
= 2% * (15 + 2*3.76 + 20%*(60+4*7.52) + 20%*50%*(60+4*7.52)
= 2% * (15 + 7.52 + 20% * 90.08 + 20% * 50% *90.08) = 0.99 (ns)
b. AMAT of Data Read
AMAT = HitTime_D-cache + MRd-cache* (HT-L2 + Time to fetch miss block from L2 + Time to fetch from MM + Time to write back the dirty block when miss)
= 0 + 5% * (15 + (16*8/128) * TT_{L2} + 20% * ((60 + (64*8 /128) *TTmm + 20%*50%* (60 + 64*8/128*TTmm))
= 5% * (15 + 3.76 + 20% *1.5* (60+4*7.52)) = 5% * 45.78=2.29(ns)
c. AMAT of Data Write (Every write will go to the write buffer. Each time for one word.)
AMAT = WriteTime when hit the write buffer Hit time of L1 + (1-95%)* （100% x (Hit time of L2 + Miss rate of L2 x Miss penalty of L2) ）
= 0 + (1-95%) * (15 + 1 * TT_{L2} + 20% * (60 + 1* TT_{mm}))
= 5% * (15 + 3.76 + 20% * (60 + 7.52 + ~~50%*90~~)) (NO WRITE ALLOCATE)
= 1.61(ns)
* ~~50%*90~~)): No replacement will happen due to NO WRITE ALLOCATE .
= 0+ (1-95%) * (15 + 1 * TT_{L2} + 20% * (90 + 50% * 90))
= 5% * (15 + 3.75 + 20%* 135)
= 2.29
* 50%*90: time latency for write back the dirty block.

d. Over all of CPI
CPI = CPI of org + stalls for instruction reference + stalls of Data read per instruction + stalls of Data write per instruction
= 0.7 + 0.99/0.9 + 20%*2.29/0.9 + 5%*2.29/0.9 = 0.7 + 1.1 +0.51 + 0.13 = 2.44
e.
CR = 2.1 GHz cycle = 1/(2.1*10^9) = 0.48 (ns)
CPI = 0.7 + 0.99/0.48 + 20%*2.29/0.48 + 5%*2.29/0.48 = 0.7 + 2.1 + 1.2 = 4.0

$$\text{Speedup} = \frac{IC * 2.44 * 0.9}{IC * 4.0 * 0.48} = 1.14$$

So the fast processor is 1.14 times faster.