a. The real-time constraint is one of crucial performance requirements in embedded applications.
b. CISCs can be also suitable for processors of embedded systems.
c. Dynamic power consumption is proportional to the product of clock frequency and voltage.
d. The scheduling of VLIW architectures is performed only by compiler.
e. Reorder buffer cannot be replaced with register renaming.
f. The number of entries in a (2,2) predictor of a total of 8K bits is 2K.
g. Amdahl's Law is not suitable for muticore processors.
h. Increasing issue rate per clock cycle can reduce CPI.

a. T
b. F
c. ~~T~~ F
d. ~~T~~ F
e. F
f. F
g. F
h. T

a. Moore's law said that the clock rate of CPU is doubled in 18 months.
b. RISC processors can be also suitable for processors of embedded systems due to low power.
c. Graphic Processor Units (GPUs) is used to exploit task-level parallelism.
d. Dynamic energy is consumed by switching bits from 0 to 1 or vice versa.
e. Dynamic Voltage-Frequency Scaling (DVFS) is a common technique to save static power.
f. Compared with SRAM, DRAMs must continue to be refreshed occasionally so as to not lose information with a minimal power.
g. DDR and DDR2 are two DRAM standards; each of them has three types. Any type of DDR2 is faster than any type of DDR.
h. For virtual machine (VM), the mapping of virtual resources to physical resources is determined by host OS.
i. Disks partitioned by virtual machine monitor (VMM) are to create virtual disks for guest VMs.
j. There are four protection levels in paravirtualization of Xen.

a. False,	b. True.	c. True.	d. True	e. False	f. True
g. False.	h. False.	i. True.	j. True		

(15%) To evaluate the performance, we must find metrics and approaches to measure different computers.
(a) Why embedded benchmarks are less mature than PC benchmarks?
(b) Explain what the Amdahl's Law is. Notice please give the formal equation and don't just describe it.
(c) For synthetic benchmarks, why and what modifications are needed?

-1

(a) Since embedded system developed later than PC and the hardware constraint on them cause the result.

(b) $\text{Speedup}_{\text{overall}} = \dfrac{1}{(1-\text{Fraction}_{\text{enhanced}}) + \dfrac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$

The improvement of some fast mode will be limited by the fraction that the fast mode can be used.

-5

(c) Since the real-programs may have some constraint so that we might not to evaluate the performance on our own and for the portability of program, we may modify the programs so that it may execute in different platforms.

(18%) Three enhancements with the following speedups are proposed for a new architecture:

Speedup1 = 30

Speedup2 = 20

Speedup3 = 15

Only one enhancement is usable at a time.

a. (6%) If enhancements 1 and 2 are each usable for 25% of the time, what fraction of the time must enhancement 3 be used to achieve an overall speedup of 10?

b.(6%) Assume the enhancements can be used 25%, 35%, and 10% of the time for enhancements 1, 2, and 3, respectively. For what fraction of the reduced execution time is no enhancement in use?

c.(6%) Assume, for some benchmark, the possible fraction of use is 15% for each of enhancements 1 and 2 and 70% for enhancement 3. We want to maximize performance. If only one enhancement can be implemented, which should it be? If two enhancements can be implemented, which should be chosen?

(a) $\frac{1}{(1-0.25-0.25-x)+\frac{0.25}{30}+\frac{0.25}{20}+\frac{x}{15}} = 10 \Rightarrow \frac{28}{3}x = \frac{101}{24} \Rightarrow x \doteq$ ~~0.45~~ $\underline{.45\%}$ -6

(c) Only one: $\frac{1}{(1-0.15)+\frac{0.15}{30}} \doteq \frac{1}{0.855}$

$\frac{1}{(1-0.7)+\frac{0.7}{15}} \doteq \frac{1}{0.347}$ ✓ $\underline{\therefore \text{chose enhancement 3}}$ +6

If two: $\frac{1}{(1-0.15-0.15)+\frac{0.15}{30}+\frac{0.15}{20}} \doteq \frac{1}{0.7125}$

$\frac{1}{(1-0.15-0.7)+\frac{0.15}{30}+\frac{0.7}{15}} \doteq \frac{1}{0.202}$ $\underline{\therefore \text{chose enhancement 1 and 3}}$

(20%) Availability is the most important consideration for designing servers, followed closely by scalability and throughput.

(a) (5%) We have a single processor with a failures in time (FIT) of 200. What is the mean time to failure (MTTF) for this system?

(b) (5%) If you will build a supercomputer out of inexpensive computers. What is the MTTF for a system with 1000 processors? Assume that if one fails, they all fail.

(c) (10%) Assume a disk subsystem with the following components and MTTF: 10 disks, each rated at 1,000,000 hour MTTF; 1 SCSI controller, 500,000 hour MTTF; 1 power supply, 200,000 hour MTTF; 1 fan, 200,000 hour MTTF; 1 SCSI cable, 1,000,000 hour MTTF. Using the simplifying assumption that the components lifetimes are exponentially distributed–which means that the age of the component is not important in probability of failure–and that failures are independent, compute the MTTF of the system as a whole.

(a) $MTTF = \frac{10^9}{200} = 5 \cdot 10^6$ ✓

(b) $MTTF = \frac{5 \times 10^6}{1000} = 5 \times 10^3$ ✓

(c) ∵ independent. probability of failure

$\Rightarrow \frac{10 \times 1}{10^6} + \frac{1}{500000} + \frac{1}{200000} + \frac{1}{200000} + \frac{1}{1000000}$

$= \frac{11}{10^6} + \frac{1}{5 \times 10^5} + \frac{2}{2 \times 10^5}$

$= 2.3 \times 10^{-5}$

$\therefore MTTF_{system} = \frac{1}{2.3 \times 10^{-5}} \doteq 4.35 \times 10^4$ ✓

-0

(18%) The Whetstone benchmark contains 195,578 basic floatingpoint operations in a single iteration, divided as shown in the following:

Operation	Count
Add	82,014
Subtract	8,229
Multiply	73,220
Divide	21,399
Convert integer to	FP 6,006
Compare	4,710
Total	195,578

MIPS = $\frac{1}{CPI}$ =

Whetstone was run on a Sun 3/75 using the F77 compiler with optimization turned on. The Sun 3/75 is based on a Motorola 68020 running at 16.67 MHz, and it includes a floatingpoint coprocessor. The Sun compiler allows the floating point to be calculated with the coprocessor or using software routines, depending on compiler flags. A single iteration of Whetstone took 1.08 seconds using the coprocessor and 13.6 seconds using software. Assume that the CPI using the coprocessor was measured to be 10, while the CPI using software was measured to be 6.

a. (6%) What is the MIPS rating for both runs?

b.(6%) What is the total number of instructions executed for both runs?

c.(6%) On the average, how many integer instructions does it take to perform a floating-point operation in software?

(a) MIPS = $\frac{1}{CPI} \times$ (clock cycle per second) $\times 10^{-6}$

12✓ using ~~the~~ the processor MIPS = $\frac{1}{10} \times 16.67M \times 10^{-6} = 1.667$ MIPS

using the software. MIPS = $\frac{1}{6} \times 16.67M \times 10^{-6} \doteq 2.778$ MIPS

(b) using the processor: 1.667 MIPS $\times 1.08 = 1.80036 \times 10^6$ 個 instruction

using the software: 2.778 MIPS $\times 13.6 = 37.7808 \times 10^6$

$= 3.77808 \times 10^7$ 個 instruction.

(c) $3.77808 \times 10^7 \times \frac{6006}{195578} \doteq 1.13 \times 10^6$ 個 instruction

(10%) Assume we have a computer where the clocks per instruction (CPI) is 1.0 when all memory accesses hit in the cache. The only data accesses are loads and stores, and these total 50% of the instructions. If the miss penalty is 25 clock cycles and the miss rate is 2%, how much faster would the computer be if all instructions were cache hits?

① 都 cache hit ⇒ 須 100 clock cycle

② 當 miss rate 為 2% 時.

① access time 為: $1 + 2\% \times 25 = 1.5$

$\therefore$ total $= 50 + 50 \times 1.5 = 125$.

⇒ 都 cache hit 時快 $\frac{125}{100} = 1.25$ 倍 ✓

(20%) For cache issues, one is that the write policies often distinguish cache designs. There are two basic options when writing to the cache on a hit:

a. (6%) What are these two options? Please compare them.

b. (4%) In a, which one is easier to be implemented? Why?

c. (4%) Give and explain a technique to reduce write stalls.

d. (6%) Since the data are not needed on a write, there are two are two options on a write miss. Compare these options.

a. write through: 當 cache hit 時. 同時寫入 cache 和 memory (or 下層 cache)

write back: 當 cache hit 時. 只寫入 cache 並標記 dirty,
當 cache block 被置 swap out 時. 才寫入 memory (下層)

write back 因為不需等待寫入 memory (or 下層). ∴ 效果較好, 速度較快

但必須負擔當斷電時的風險 (如尚未寫回下層) 造成資料不同步.

write through 需等待同時寫入下層 memory, 造成 stall, 較慢.

b. write through 比較容易 implement!

因為 write back 至少須要多 dirty bit, ~~cycle~~ replace 的時候, 檢查是否需要寫回下層, 需增加 hardware cost.

c. write buffer.

當需要 write 到下層時, 僅需寫入較快速的 write buffer, 等有空時. write buffer 再寫入下層. 減少 write stall

d. write allocate: 當 write miss 時. 先將資料讀入 cache, 再依 write hit 執行寫入

non-write allocate: 當 write miss 時, 直接寫入 memory (下層), 不讀入 cache.

若 data 已經不會再用到的話. non-write allocate 比較好. 不用浪費寫入 cache 的時間.

但當該 data 不久後又會被用到時. write allocate 已經將 data 放入 cache 中. 之後可以降低 miss rate.

(20%) For the following memory optimizations, explain their purposes and their impact on memory hierarchy.

a. Way prediction
b. Pipelined cache access
c. Critical word first
d. Hardware prefetching

Average access time = hit time + missrate × miss penalty

a. Way prediction：利用一些 bit 來記錄，那些 ~~set~~ cache line 最近曾經被用過。
5 當搜到該 set 時，優先預測那些被標記的 優先檢查是否為所需的資料 可能增快查找速度

可降低 hit time

b. Pipelined cache access：將 cache 設計為 pipeline
2-3 可增加平行度。降低 hit time.

c. Critical word First：當 cache miss 時，到下層抓 data，不必等整 block 載入。
5 先由所抓需要的 word，先送到 CPU 執行，block 其餘資料再慢慢載入

可降低 miss penalty ✗ 增加 performance.

d. Hardware prefetching：抓資料時，由 hardware 也連同下一資料一起抓取，放入 stream buffer
5

~~可降低~~ 可增加平行度 parallelism

降低 miss rate

降低 miss penalty