Birla Institute of Technology & Science – Pilani

Hyderabad Campus

1st Semester 2019-2020

Computer Architecture (CS F342) – Comprehensive Examination (Regular) - KEY

Date: 07.12.2019 Weightage: 40% Duration: 3 hrs. Type: Closed Book

---

**Instructions:** Answer all questions; All parts of a question ***should*** be answered consecutively; No of pages in the question paper: **4;** show the steps clearly wherever applicable.

---

**Q1.**

(a)

Let $Perf_{\_max}$ be the maximum possible performance improvement.

(i) if the part that can be improved is 25% of the overall system and its performance can be doubled, then the Perf_max for a given system = 1/[(1- 0.25) + 0.25/2] = **1.14**

(ii) the part that can be improved is 75% of the overall system and its performance can be doubled, then the Perf_max for a given system = 1/[(1- 0.75) + 0.75/2] = **1.60**

(iii) From (i), Perf_max = 1.14 and the fraction that cannot be improved is 0.75 where as from (ii), Perf_max = 1.60 and the fraction that cannot be improved is 0.25. Thus it holds that "the more the part that cannot be improved, the less the benefit of improvement":

(b) The average CPI = [45000 x 1 + 32000 x 2 + 15000 x 2 + 8000 x 2]/100000 = **1.55**
Effective processor performance as MIPS rate = [clock frequency] / [CPI x 1/1 Million]
= 400000000/(1.55 x 1000000) = **258 MIPS**
Execution time = CPI x IC x CCT = CPI x IC/frequency = [1.55x100000] / [400 x 1000000]
= 0.0003875 sec = **0.3875 ms.**

(c)

(i) It is given that we are designing an ISA for 16 registers, using a fixed-length 16-bit encoding. Since it requires 4 bits to encode each register operand, encoding three such registers requires 3 x 4 = 12 bits. So, 4 bits are left which can support up to $2^4 = 16$ distinct values. Thus in this case, we can have maximum of **16 ALU operations**.

(ii) In this case, encoding two such registers requires 2 x 4 = 8 bits. So, 8 bits left which can support up to $2^8 = 256$ distinct values. Thus in this scenario, upto **256 ALU operations** can be supported.

(d) Given 8-stage pipelined CPU with a 2 GHz clock. The time it takes to complete each instruction once it enters the pipeline will be
= (8/cycles per instruction) x (seconds / 2 x $10^9$ cycles) x ($10^9$ nanoseconds/seconds) = **4 nanoseconds.**

**Q2.** (a)

Given a machine for which the cache contains 65,536 (64K) bytes of data in 64-byte blocks with two-way set-associative placement. The address coming into the cache is divided into two fields: 34-bit block address and 6-bit block offset.

(i) width of the tag field

$2^{index}$ = cache size/(block size x set associativity) = 64K/(64 x 2) = $2^6$ x $2^{10}/2^6$ x 2 = $2^9$

Thus the index field is 9 bits. So, the tag field = 34 – 9 = **25 bits**.

(ii) Since it is given that block address is 34 bits and offset is 6 bits, the size of the address for the CPU to reach a word will be 34 + 6 = **40 bits**.

(b)

| Block j of main memory can map to line number ( **j** mod **number of lines** ) only of the cache |
|---|
| Number of multiplexers required = Number of bits in the **tag** field |
| Size of each multiplexer = Number of lines in cache x **1** |
| Number of comparators required = **1** |
| Size of comparator = Number of bits in the **tag** field |
| Hit latency = **multiplexer** latency + **comparator** latency |

(c) Given the TLB hit rate is 0.6; the time to search TLB is 10ms and the physical memory access time is 80ms.

**Effective Access Time** = hit ratio x (TLB access time + main memory access time) +
(1 – hit ratio) x (TLB access time + 2 x main memory time)

i.e.,

= 0.6 x(10+80) + (1-0.6)x(10+2x80)
= 0.6 x (90) + 0.4 x (170)
**Effective Access Time** = **122 ms**

(d) It is given that the binary and the stack each fit in one page, thus each takes one entry in the TLB. When the function is running, it will access the binary page and the stack page all the time. So the two TLB entries for these two pages would reside in the TLB all the time and the data can only take the remaining 6 TLB entries. Since we assume the two entries are already in TLB when the function begins to run, we need only consider those data pages.

Since an integer requires 4 bytes for storage and the page size is 4096 bytes, each array requires 1024 pages. Let us assume each row of an array is stored in one page. Then these pages can be represented as a[0..1023], b[0..1023], c[0..1023]: Page a[0] contains the elements a[0][0..1023], page a[1] contains the elements a[1][0..1023], etc.

For the reference string (1024 rows in total), a[0], c[0] will contribute to two TLB misses. Since a[0] and b[0] each will be accessed every four memory references, the two pages will not be replaced by the LRU algorithm. For each page in b[0..1023], it will incur one TLB miss every time it is accessed. So the number of TLB misses for the second inner loop is

2+1024*1024 = 1048578

So the total number of TLB misses is = 1024*(2+1024*1024) = 1024*1048578 = **1073743872**

**Q3.** (a)

Step1: Take 2's complement of the given multiplier as: **11000111**
Step 2: Append 0 to LSB: **110001110**

Step 3: The Booth's coding logic is: **00 = 0, 01 = +1, 10 = -1, 11 = 0**

Step 4: Apply the Booth's coding logic as (add 1 bit at a time, from LSB to MSB):

= **11, 10, 00, 00, 01, 11, 11, 10**

Step 5: The multiplier -57 will be recoded as: = **0 -1 0 0 1 0 0 -1**

(b)

From the given values, it is obvious that $A \neq B$; but in finite precision normalized floating point arithmetic, $A - B =$ **0** as $A - B = 0$**.02 x $10^{-306}$;** i.e, $A - B =$ **2.0 x $10^{-308}$** in the normalized form, which is too **small** to be represented as a normalized number. It is therefore rounded to the value of **0**

(c)

Set Register A = **000000**; Set Register Q = Dividend = **101110**; AQ = **000000 101110** , $Q_0$ = **0**; Set M = Divisor = **010111**, M' = 2's complement of M = **101001**; Set Count = **6**, since **6** bit operation is being done here.

(d)

The multiplicand is **subtracted from** the partial product upon encountering the first least significant 1 in a string of 1's in the multiplier; The multiplicand is **added to** the partial product upon encountering the first 0 (provided that there was a previous '1') in a string of 0's in the multiplier; The partial product is **left unchanged if** the multiplier bit is identical to the previous multiplier bit.

Q4.

(a)

| Initial value of x | x = = 0? (Y/N) | b1(Taken/Not Taken) | Value of x before b2 | x = = 1? (Y/N) | b2 (Taken/Not Taken) |
|---|---|---|---|---|---|
| 0 | Y | Not Taken | 1 | Y | Not Taken |
| 1 | N | Taken | 1 | Y | Not Taken |
| 2 | N | Taken | 2 | N | Taken |

(b) Given three 5-stage MIPS machines, each with different branch resolution strategies.

Machine M1 has a branch predictor which always predicts the correct target and hence ideal.

Machine M2 resolves branches in the EX stage using a predict-not-taken scheme.

Machine M3 resolves branches in the ID stage using one branch delay slot.

Assume the following:- all the three have the same frequency; 20% of the instructions are branches; 25% of branches are taken and the stalls are due to branches alone; the compiler is able to fill 30% of the delay slots with useful instructions

Answer:

CPI of M1 = **1.0**, as it is ideal.

CPI of M2: M2 mispredicts 25% of branches which are 20% x 25% = 5% of the total instructions. It is given that the branches are solved in the EX stage, the branch penalty is 2 cycles. CPI is increased since those 5% of mispredicted branches waste 2 cycles each.

Thus the CPI of the second machine M2 = 1 + 0.05 x 2 = **1.1**

CPI of M3: Given that Machine M3 resolves branches in the ID stage using one branch delay slot.

Thus, 70% of branch delay slots are wasted and increase the total number of cycles.

So, CPI of the third machine M3 = 1 + 0.2 * 0.7 = **1.14**

**Q5.**

(a)

| RAID level | Data storage capacity | No.of disk failures we can afford |
|---|---|---|
| 0 | 2TB | 0 |
| 5 | 3TB | 1 |
| 6 | 3TB | 2 |

(b)

MTBF = 30 days

MTTR = 12 hours = ½ day

MTTF = 29 ½ days

Availability is 29.5/30 = **98.3 %**

(c) Given a system environment comprising 1000 disks with MTTF = 100,000 hrs and MTTR = 100 hrs.

(i)

MTBF = MTTR + MTTF = 100,100 hr

Therefore, Availability = MTTF/MTBF = 0.9990 = **99.9%**

(ii) Given that Annualized Failure Rate (AFR) is the average rate of failures per year

AFR = (365 x 24)/ MTTF = 8760/MTTF = 0.0876 = **8.76%**

(d)

| Operation | (i)stripe data across all disks | (ii) logical write amounts to two physical writes | (iii) logical write means minimum 2 to maximum N physical reads and writes. |
|---|---|---|---|
| RAID level | RAID 0 | RAID 1 | RAID 4 |

**Q6.**

(a)

| CPU activity | Bus activity | Contents of A's cache | Contents of B' cache | Contents of X |
|---|---|---|---|---|
| | | | | 10 |
| CPU A reads X | Cache miss for X | 10 | | 10 |
| CPU B reads X | Cache miss for X | 10 | 10 | 10 |
| CPU A writes 20 to X | Broadcasts write | 20 | 20 | 20 |
| CPU B reads X | Cache hit for X | 20 | 20 | 20 |

(b) search times for each of the three cache mapping schemes:

direct-mapped -> O(1)

fully associative-> O(n)

k-way set associative -> O(k)

Q7.

(a) Represent the number $-176.375_{10}$ in IEEE 32 bit format. The sign is negative; so left most bit of the pattern has to be 1.

The number in binary : 10110000.011

Normalizing: $10110000.011 = 1.0110000011 * 2^7$

Mantissa: 01100000110000000000000

The exponent is represented by 8 bits (256 states) and is shifted by 127. In our example ( $1.0110000011 * 2^7$ ) the true exponent is 7. The biased exponent is 134 (7+127)

The biased exponent in binary : 10000110

Thus the number in 32-bit format is **1 10000110 01100000110000000000000**

and the hexa equivalent is **0xc3306000**

(b)

$a = -2.7 \times 10^{23}$, $b = 2.7 \times 10^{23}$, and $c = 1.0$

**a + (b + c)** $= -2.7 \times 10^{23} + ( 2.7 \times 10^{23} + 1.0 ) = -2.7 \times 10^{23} + 2.7 \times 10^{23} =$ **0.0**

**(a + b) + c** $= ( -2.7 \times 10^{23} + 2.7 \times 10^{23} ) + 1.0 = 0.0 + 1.0 =$ **1.0**

Thus it is proven that FP arithmetic is not associative.

(c) Add the floating point numbers $9.76 \times 10^{25}$ and $2.59 \times 10^{24}$ assuming 3 digit mantissa

**(i) with no extra digits for the internal registers and no round digits**

Steps:

1. Shift mantissa of the smaller number to the right: $0.25 \times 10^{25}$
2. Add mantissas: $10.01 \times 10^{25}$
3. Check and normalize mantissa if necessary: $1.00 \times 10^{26}$

**Answer: $1.00 \times 10^{26}$**

**(ii) the internal registers have two extra digits and rounding is possible**

Steps:

1. Since internal registers have extra two digits: $9.7600 \times 10^{25}$ and $2.5900 \times 10^{24}$
2. Shift mantissa of the smaller number to the right: $0.2590 \times 10^{25}$
3. Add mantissas: $10.0190 \times 10^{25}$
4. Check and normalize mantissa if necessary: $1.0019 \times 10^{26}$
5. Round the result: $1.00 \times 10^{26}$

**Answer: $1.00 \times 10^{26}$**

(d)

MIPS instructions that cause overflow (or some other violation) lead to an **exception**, which sets the **exception** code. It then switches to the **kernel** mode (designated by a bit in the **status** register of coprocessor no: **C0**, register no: **12**) and transfers control to a predefined address **0x80000180** to invoke a routine (**exception handler**) for handling the exception.