MAJOR WITH ANSWERS

MAJOR
(ELL305) Computer Architecture
2015

Time: 1 Hour

Max. Marks: 40

NAME:

Entry No.: -

Group: -

Serial Number as in Attendance Sheet: -

N. B.: Do the calculations on this sheet last pages / continuation sheet. This is **open book / notes examination**, but **transfer** of notes to each other is strictly **prohibited**.

Q1 (a) : - (8086 based) What will be the value in **AX** after executing the following instructions? Give the answer in both hexadecimal and binary. (Numbers are **decimal** if they are <u>not</u> indicated with '**H**') **(2)**

```
MOV     AL, 15
MOV     AH, 15
XOR     AL, AL
MOV     CL, 03
SHR     AX, CL
ADD     AL, 90 H
ADC     AH, 0
```

ANSWER: ~~Final AX in decimal~~ =

Final AX in Hexadecimal =

(b): - (8086 based) Show how to use **XLAT** instruction to access **9th** element in the table called '**MARKS**' that is located in the stack segment. **(1)**

Answer:

Q2 (a): - One day famed student Joe Surfer had an inspiration while hanging ten at Mission Beach. He observes that most programs have most of their data at the beginning of the address space. For his home-grown Salt Water **OS**, he decides that he is going to implement his page tables similar to the way **UNIX** implements inodes. He calls this page table design ***Inode Page Tables***. Inode Page Tables are essentially two-level page tables with the following twist: The first half of the page table entries in the master page table directly map physical pages, and the second half of the entries map to secondary page tables as normal. Call the first half the entries ***fast***, and the second half ***normal***.

For the following questions, assume that addresses are 32 bits, the page size is 4 KB, and that the master and secondary page tables fit into a single page.

(a) How many virtual pages are ***fast*** pages?

(b) How many virtual pages are ***normal*** pages?

(c) What is the maximum size of an address space in bytes (use exponential notation for convenience, e.g., $2^{3□}$)?

(Hint: - Inode – *A UNIX file descriptor for the layout of file blocks on the disk; it is index-based and hierarchical*)

Calculation & Answer:

(a):

(b):

(c):

(b):- Memory management with paging is under consideration. A virtual memory address space of **64** Kbyte is split into **16** pages of **4** Kbyte each. A physical address space respectively divided into **4** page frames of **4** Kbyte each. The page table is such that the pages **2, 4, 6** and **8** go to the page frames **0, 1, 2** and **3**, respectively. What physical address will be chosen for virtual address 9000_{10}? **(4 + 2)**

Calculation & Answer:

Q 3: - For a memory system following features are there:

- **90 %** of all memory accesses are found in the cache.
- Each cache block is two words, and the whole block is read on any miss.

• Processor sends references to its cache at the rate of 10^9 words / sec.
• **25 %** of those references are writes.
• Assume that the memory system can support 10^9 words / sec, read or writes.
• The bus reads or writes a single word at a time (the memory system cannot read or write two words at once.)
• Assume at any one time, **30 %** of blocks in cache have been modified.
• The cache uses write allocate on write miss. One peripheral is to be added. How much memory system bandwidth is in use? Calculate the **%** of memory system bandwidth for **read miss** and **write miss** when: **(4)**
• **Cache is write-back.**
ANSWER:
Q4 (a): - Assume the following register and memory contents in an **ARM** computer:
Register R_0 contains **1000**.
Register R_1 contains **2000**.
Register R_2 contains **1016**.
Register R_6 contains **20**.
Register R_7 contains **30**.
The numbers **1, 2, 3, 4, 5, and 6**, are stored in successive word locations starting at memory address **1000**. What is the effect of executing each of the following three short instruction blocks, starting each time from the given initial values? **(4 ½)**

```
(I)     LDR R8, [R0]
        LDR R9, [R0, #4]
        ADD R10, R8, R9
(II)    STR R6, [R1, #-4]!
        STR R7, [R1, #-4]!
        LDR R8, [R1], #4
        LDR R9, [R1], #4
        SUB R10, R8, R9
(III)   LDMIA R2! , {R4, R5}
        ADD R4, R4, R5
```

ANSWERS:
(I):
(II):
(III):
(b): - For the **ARM** instructions given below write on **right** hand **side** of each instructions what is happening (in mathematical notation)? **(2 ½)**
RSB Rd, Rm, Rm, LSL #3; =>
ADD Rd, Rd, Rm, LSL #4; =>
RSB Rd, Rd, Rm, LSL #7; =>
RSB Rt, Rm, Rm, LSL #4; =>
RSB Rd, Rt, Rt, LSL #3; =>
Q5 (a): - The following 32-bit word is a floating point number in the IEEE 754 format (using excess 127 code): **(2 + 2)**

1 10000010 11000000000000000000000

What are the sign, significand and unbiased exponent of this number? Using the floating point multiplication procedure calculate the square of this number. Express the result in IEEE 754 format.
Answer: The given number is (in decimal with sign):
Final Answer: Square of the number in IEEE 754 format =
(b): - In ARM single instruction like: **ADD R_0, R_0, R_0, LSL #2** will multiply the contents of R_0 by '5'. By using other single instructions (such as **MOV, MVN, SUB,** or **RSB**) determine if the following numbers can be used as multiplier in this fashion. If yes, write down the single ARM instruction that will multiply R_0 in-place by the specified number. If no, explain why not.
Numbers are: (1): **15,** (2): **16,** (3): **17,** (4): **- 1**

Answer:
(1):
(2):
(3):
(4):

Q 6 (a): - For the collision vector in $C_6\ C_5\ C_4 \ldots C_1$ format as given below. Draw the <u>**neat**</u> reduced state diagram for this pipeline. Also list the **greedy** cycles along with latency sequence that can achieve the MAL (Write MAL value). **(2 + 2)**

Given collision vector in $C_6\ C_5\ C_4 \ldots C_1 = 101010$

Answer:

(b): Given collision vector in $C_7\ C_6\ C_5\ C_4 \ldots C_1 = 1110011$

Answer:

Q7: - A **NUMA** parallel computer has **64 CEs**. Each CE has **32 MB** memory. The word length is **64** bit. In a set of programs (which has **500,000** instructions) **20 %** of instructions are loads and **15 %** are stores. The memory access time for local load / store is **2** clock cycles. An overhead of **10** clock cycle is needed to initiate transmission of a request to a remote CE. The bandwidth of interconnection network is **50 MB / sec** and **30 %** of the accesses are to remote computers. **(5)**

(Clock cycle time = 1 Nano second)

(a): What is the total load / store?

(b): How much will it be if all accesses are to local memory?

Calculation & Answer:

(a):

(b):

Q 8(a):- What is the main problem that the Thumb Instruction Set addresses and why is it a useful extension to the standard **ARM** instruction set? Suppose you are designing a new processor for ***servers***. Would you include a mode similar to the Thumb mode in your processor? Why? (**Answer in less than 30 words)** **(2 + 2 + 3)**

Answer:

(b): - Distinguish between **UMA, NUMA and CC-NUMA** parallel computer architecture in **total less than 40 words. Use** diagrams wherever needed.

Answer:

(c): - When performing signed division, the sign of the remainder should be the same as the sign of the dividend. Why? Your answer should be in <u>less</u> than **60 words** and consider all possible cases. Use the terms of equation:

D = (Q X V) + R, where **D** is dividend, **V** is divisor, **Q** is quotient and **R** is remainder.

Answer:

4. (12 pts) One day famed student Joe Surfer had an inspiration while hanging ten at Mission Beach. He observes that most programs have most of their data at the beginning of the address space. For his homegrown SaltWater OS, he decides that he is going to implement his page tables similar to the way Unix implements inodes. He calls this page table design *Inode Page Tables*. Inode Page Tables are essentially two-level page tables with the following twist: The first half of the page table entries in the master page table directly map physical pages, and the second half of the entries map to secondary page tables as normal. Call the first half the entries *fast*, and the second half *normal*.

For the following questions, assume that addresses are 32 bits, the page size is 4 KB, and that the master and secondary page tables fit into a single page.

(a) How many virtual pages are *fast* pages?

$4KB/4 = 1024$ *PTEs*

$1024/2 = 512$ *PTEs* $\Rightarrow 512$ *or* (2^9) *pages are fast*

(b) How many virtual pages are *normal* pages?

The remaining 512 PTEs refer to second-level page tables. Each second-level page table has 1024 PTEs, so:

$2^9 * 2^{10} = 2^{19}$ *pages are normal*

(c) What is the maximum size of an address space in bytes (use exponential notation for convenience, e.g., 2^3)?

The size of the address space is the total number of pages times the size of each page:

$(2^9 + 2^{19}) * 2^{12} = 2^{21} + 2^{31}$ *bytes*

(d) Inode Page Tables reduce the lookup time for fast pages by one memory read operation. Do you think that this is an effective optimization? Briefly explain.

Not really. This optimization saves one memory access whenever there is a miss in the TLB. Since TLB misses are relatively infrequent, saving one memory access in the miss handler is not going to improve performance significantly. It also makes the PTE lookup a bit awkward since you have to check to see what kind of a page needs to be loaded into the TLB.

A number of people answered that this shrank the address space too much to be useful. The address space is still > 2 GB, which is pretty large. Most programs will still easily fit inside that address space.