

CSE 220

Midterm #1 Practice Exam

P1: Convert A7 in hex to a number in base 8.

Answer:

A7 in hex is 10100111 in binary. Add one zero to left to make this 9 bits long and group them in 3 bits. 010 100 111 in binary means 247 in octal.

P2: Convert 4.9 in decimal to a number in base 3. The algorithm for converting fractions to base 3, is similar to what we have studied. Apply it carefully with minor changes. Stop after pattern starts repeating.

Answer:

First convert integer part. Decimal 4 is 11 in base 3. To convert fractional part, multiply it by 3 and collect overflow.

0.9 0.22002200 Answer is 11.22002200...

x 3

2.7 --> Collect 2

0.7

x 3

2.1 --> Collect 2

0.1

x 3

0.3 --> Collect 0

0.3

x 3

0.9 --> Collect 0, Since we get 9 again, after this pattern will repeat.

P3: Normalize this binary number (-0.0001101). You must follow normalization of single precision IEEE format. State the sign bit, exponent, and significand in binary form for the above normalized number.

Answer:

-1.101 x 2⁻⁴

Sign bit is 1, exponent is 127-4=123, which is 01111011 in binary. Significand is 101000....00.

P4: How many single precision floating point registers are there in MIPS?

How many double precision registers are there in MIPS?

Answer:

32 single precision registers.

16 double precision registers.

P5: Perform subtraction (1.01 x 2¹ - 1.11 x 2⁰) by aligning exponents. Numbers are already in IEEE normal form. You may do it without 2's complement if you wish. Normalize the result and state the answer using power of 2.

(i) Align exponents and subtract.

(ii) Normalize the result and state it using power of 2.

CSE 220

Midterm #1 Practice Exam

Answer:

Shift the smaller exponent.

1.010×2^1 *Comments: There was no need to use 2's complement*
 $- 0.111 \times 2^1$ *system to perform subtraction. If you did use it,*
----- *then ignore carry. Many students use that carry*
 0.011×2^1 *bit in final answer. Big mistake.*

State in power of 2 means do not convert to decimal or other IEEE binary format.

1.1×2^{-1} after normalizing.

P6: Consider 6-bit long (including sign) binary numbers.

(a) Consider a binary number (110010) stored in signed magnitude form. What is this number in decimal?

(b) How is -5 stored in 1's complement form?

(c) Perform (001001 + 001101). Answer:

(d) What is the smallest positive number that we need to add to (011010) so that overflow is generated? Decimal answer is expected; remember numbers are 6-bit long. Here negative numbers are stored in 2's complement form. Show all work.

Answer:

(a) -18,

(b) 111010,

(c) 010110

(d) Given number is 011010. If we add 000101, we would get 011111. This means adding 5 will not generate overflow, but if we add 6 we will have an overflow. Therefore smallest number is 6.

P7: (a) Suppose we want to code letters A through Z (only upper case) using binary numbers, starting with 0. What is the minimum number of bits that we need?

(b) Suppose we want to code letters A through Z (only upper case) using base 3 numbers, starting with 0. What is the minimum number of base 3 digits that we need?

(c) What would be the base-3 code for letter F in the part (b) above.

Answer:

(a) We need 5 bits to code A through Z. 26 options)

(b) We need 3 base-3 digits for such a code.

(c) 012

P8: Consider a machine with 32-bit word (4 bytes in a word). Bytes are numbered 0, 1, 2, 3, and words are numbered 0, 1, 2, etc. Word-0 contains byte-0, byte-1, byte-2 and byte-3. Word-1 contains byte-4, byte-5, byte-6, and byte-7 etc.

(a) A byte numbered 406 would belong to what word? _____ (word-number)

(b) Assume, this memory is organized using big endian byte order, what would be the least significant byte (one that holds the Least Significant Bit) of Word-20? _____ (byte-number)

Answer:

(a) A byte numbered 406 would be in a word numbered $(406)/4 = 101$ (integer division).

(b) The number of the LSByte of word 20 is byte-83. $(20*4 + 3)$

CSE 220

Midterm #1 Practice Exam

P9: Consider the fetch-decode-execute cycle for the John Von Neumann machine that we studied in class. Note that it does not have a cache memory.

(a) Give an example of an instruction that requires accessing memory during execution of an instruction.

(b) Do we always have to access memory in fetch part of the cycle? Why or why not?

Answer:

(a) *LOAD or ADD instruction or something similar. There is no READ or MOVE instruction for John Von Neumann machine.*

(b) *Yes. We always access memory in fetch part of the cycle. Fetch means reading the next instruction, which is stored in memory. For John von Neumann machine, one word contains two instructions. So a fetch operation gets both left and right instructions. There is no fetch for right instruction. So if someone claims that there is no memory access for right instruction, then it is not valid argument, as there is no fetch either.*

P10: Consider a JVN like processor whose instructions have one memory address and an opcode in each of them. But the size or length of each memory word is 3 times the size of an instruction. (Each memory word holds three instructions.) Each instruction has a 7-bit opcode. The computer has 8192 (or 8K) words of memory. Assume that memory reads or writes one word at a time.

(a) What is the size of MAR? _____

(b) What is the size of instruction? _____

(c) What is the size of MBR/MDR? _____

(d) Assume that memory is connected to the processor using a memory bus. This bus uses five control lines (5 bits) and some lines for address and/or data. It is just wide enough to permit transfer of either data or an address, but not both simultaneously. Including control lines, what is the width (number of line/bits) of this memory bus?

Answer:

(a) $\log(8192) = 13$ bits.

(b) $13+7 = 20$ bits.

(c) $20+20+20 = 60$.

(d) 65. (60 bits are enough for Data or Address. Plus 5 bits for control.)

P11: Consider an array A of integers in memory of MIPS. The array is indexed by 0, 1, 2, ... etc. Also assume that address of A is in register \$s1. Suppose we want to copy and store the third element of this array (indexed by 2) to variable i. Address of variable i is in register \$s2.

Write the exact sequence of instructions MIPS that will accomplish this. You may use a temporary register. (In other words, write MIPS code for statement $i = A[2]$;))

Answer:

*Third element of integer array has offset = 8 bytes (4 * Array Index).*

```
lw $t0, 8($s1) # Load t0, address of A in $s1, offset is 8
sw $t0, ($s2) # $s2 has address of i. Store $t0 to i.
```

If you used [sw \$t0, i] as the second instruction, then 1/2 point would be deducted.

Since address of i is known, why use sw \$t0, i. MIPS assembler allows such instructions, but this would be slow. Also, typically i is a local variable on the stack. Its address is always known. So use the address directly. That is what a compiler would do.