CSE 3430

Part A – Midterm 1

Key Points

NOTE: Answers, or partial answers in some cases, are given for some questions in parentheses.

Slide set A-1

1. What are the main components of a computer?

2. What is the smallest unit of memory in a computer system?

3. What is a bit string?

4. What kinds of data/information are stored as bit strings in a computer?

5. Are instructions stored as bit strings also?

6. How many different values can be encoded by a bit string with n bits?

7. How is memory organized in a computer?

8. What is a word of memory? How large is a word (how many bits)?

9. Are words the same size in every system (in other words, is the definition of a word universal)?

10. What is a byte?

11. Is the definition of byte universal?

12. What is the way of encoding unsigned integers which was discussed in class (B2U)? Is this method universally used for unsigned integers?

13. What is the range of values which can be encoded by a B2U number of n bits (What is the formula for the range of values, in terms of n)?

14. What are the three ways of encoding signed integers which were discussed? Answer: B2S, B2T, B2O

15. Be able to give the value of a binary encoded signed integer (of three bits) if you are given the bit string in which the value has been encoded (B2S, B2T, B2O); be sure you can do this for both negative and non-negative integer values encoded in 3 bits in B2S, B2T, and B2O.

16. What is the range of values which can be encoded by a B2T number of n bits? You should be able to give the formula in terms of n for the minimum to maximum value.

17. Be able to add two B2U encoded (unsigned) numbers of n bits (for n = 4) and determine if there is overflow. How is overflow determined by the CPU for B2U (unsigned) addition?

18. What bit value is used as the first carry to add two B2U numbers?

19. Be able to add two B2T encoded (signed) numbers of n bits (for n = 4) and determine if there is overflow. How is overflow determined by the CPU for B2T (signed) addition? Can the same method be used for B2O addition?

20. How is an integer negated in B2T (Two's complement)?

21. How is an integer negated in B2O (One's complement)?

22. When we added numbers encoded in B2O, we said the hardware has to do the addition differently if the two operands have a certain combination of signs (one negative and one non-negative). How did we say the hardware must do the addition differently in these cases? [A second addition is required, to add the last carry bit from the first addition operation back to the first sum to get a final sum.]

23. Which method of encoding signed integers is used universally in all hardware being built today? What are the reasons it is always used instead of the other two methods of encoding signed integers which we discussed?

24. How does the CPU compare the last two carries to determine if there is overflow for two's complement (B2T) operations (or one's complement, B2O)? Be sure you understand which kind of gate is used to compare the last two carries, and how it does this.

25. How can subtraction be done in two's complement by using inversion of one operand and a first carry of 1? Be sure that you can subtract one number of n bits encoded in two's complement from another number in n bits using this method.

<u>Slide set A-2</u>

26. How does IEEE 754 single-precision encoding of floating-point numbers work? How many bits, and which bit(s) is/are used for the sign? How many bits are used for the exponent? How is the exponent encoded (with a bias of 127)? How is the mantissa encoded?

27. Be sure you understand the four steps which must be done to convert an IEEE 75 single-precision encoded floating-point number to the corresponding decimal value (assume the value is a normalized number, and not a denormalized number or a special infinite/error value).

28. What kind of value is encoded by an IEEE 754 number if the true exponent is negative?

29. What is the trade-off that must be made to get the much greater range of values that can be represented by 32 bit IEEE 754 encoded numbers? [Greater range is traded for less precision.]

30. Why are many IEEE 754 encoded values only approximations to real number values? Which kinds of real values can be encoded precisely in IEEE 754 assuming we have a sufficient, but finite, number of bits?

31. What kinds of characters can be encoded in ASCII? How many bits are used to encode each character? What is the most significant bit in the ASCII code for any character?

32. Be sure you understand the fundamentals of UTF-8 character encoding. Is each character in UTF-8 encoded using the same number of bytes? How many bytes are used to encode each character in UTF-8?

33. We said that not all of the bits in a UTF-8 character code are used to identify the character (not all of the bits are significant). What are the other bits used for?

34. How can parity and the msb of a one-byte ASCII character code sent over a network by a sender be used by the receiver to detect possible errors in the byte received?

35. Can the hardware (the CPU) tell, by looking at a bit string, what kind of data is encoded by the bit string?

36. How is memory organized? (As an array of bytes; addresses are indexes into that array, starting at index/address 0)

37. What is the size of an address in typical systems today (how many bits)?

38. What is the difference between word-aligned access and unaligned access of data in memory?

39. Do all systems require word-aligned access?

<u>Slide set A-3</u>

40. What are the sub-parts of the CPU? [registers, ALU, control]

41. What 4 types of instructions can all CPUs perform?

42. Be able to describe how a read from memory is performed. Be able to describe how writes to memory are performed.

43. Which register in the CPU is used to hold the address of the instruction being executed?

44. Which register in the CPU holds the instruction being executed (the bit string with the encoded instruction)?

45. What kind of program converts assembly language to machine language? [Another way to ask the question: What is the program which converts assembly language to machine language called?]

46. Know the fundamental differences between RISC and CISC processors on slide 7.

47. Which type of CPU (RISC or CISC) are Intels? Which type are ARMs (or the Mac M1,M2, M3)?

48. Which of the two types of processor architectures is called load/store?

49. How many assembly language instructions typically correspond to a single statement in a high-level language (such as Java, C++, or C)? [More than one instruction.]

50. When can the address in the PC register be incremented to the address of the next instruction? [After the address has been sent to memory, so that the instruction can be read/fetched from memory.]

51. Which kinds of instructions in the CPU are used to execute if or else (or loop) statements in a high-level language? [Branch/Jump instructions.]

52. What is the Processor Status Register (or PSR)?

53. What four flags does the PSR have which were covered in class and the class slides?

54. What are labels used for in assembly language? [To mark addresses of data or instructions in a program.]

55. Do labels occupy any space in memory? Why or why not?

56. Which built-in data structures does the CPU recognize? [This is a "trick" question! The answer is ***none***! There are NO built-in data structures which the CPU recognizes!]

57. What is a subroutine (function/procedure/method)?

58. How is a subroutine called?

59. How is a subroutine call different from a branch? [A return address must be saved.]

60. In what two ways can the return address be saved?

61. Which way of saving a return address has a limitation on the number of subroutine calls which can be made before returning to the original calling subroutine? [See the HW2 key if you aren't sure.]

62. Besides being used to save a return address, what other kinds of data related to a subroutine call can a stack be used for? [Parameters and return value.]

<u>Slide set A-4</u>

<u>63</u>. How many instructions in assembly language correspond to a single machine language instruction? [One instruction always.]

64. Does every type of CPU (Intel, ARM, Mac M1, SPARC, PowerPC, MIPS etc.) use the same assembly language?

65. How many assembly language instructions does a high-level language statement (Jave, C++, C, etc.) correspond to? [Typically more than one, pehaps many more than one in some cases.]

66. Which real CPU architecture is the Y86-64 simulated CPU based on? [Intel X86-64.]

67. What is Instruction Set Architecture (ISA)? [The human programmer visible machine interface.]

68. What are the two major types of ISAs today? [RISC and CISC.]

69. How many registers does Y86-64 have? [15.]

70. Why is the number of registers in Y86 unlike a real CPU? [In real CPUs, the number of registers is always equal to a power of 2.]

71. How many condition codes (or flags) does Y86 have, and what are they? [3 flags: ZF, SF, and OF.]

72. How are the condition codes/flags in Y86 set by the processor? [See the slides for a description.]

73. What kinds of operands can the Y86 simulated CPU work on? [Signed operands.]

74. What sizes of operands can the Y86 simulated CPU work on? [Only 8 bytes (64 bits).] How is this different from real CPUs? [Real CPUs can work on data of different sizes: 1 byte, 2 bytes, 4 bytes, or 8 bytes.]

75. What size are addresses in Y86? [8 bytes (64 bits).]

76. What is the difference in the way multi-byte data is stored in memory in a big-endian versus a little-endian system?

77. Is Y86 (and Intel) big-endian or little-endian?

Slide set A-5

Note: In the points below, Y86-64 is referred to simply as Y86.

78. What do the 4 data movement instructions rrmovq, irmovq, rmmovq and mrmovq do in Y86? [Be sure you understand what each of the 4 types of instructions does.]

79. Is a source operand read, written, or both in Y86? [Only read.]

80. Is a destination operand read, written or both in Y86? [For ALU instructions, both read and written; for other types of instructions, only written.]

81. What is an immediate operand in Y86? [A constant value; always stored as part of the encoded bit string for the instruction.]

82. Be sure you understand address expressions for memory operands in Y86: DISP(BASE), where DISP is an optional constant displacement of the base address, and BASE is a named register (one of the 15 Y86 registers).

83. Be sure you understand how the 4 ALU instructions in Y86 work [They all use two register operands: addq, subq, andq, xorq.]

84. Be sure you understand how Y86 ALU instructions set the 3 flags (And that other types of instructions besides ALU instructions in Y86 do not affect the flags).

85. Be sure you understand the unconditional jump instruction in Y86: jmp.

86. Be sure you understand the conditional jump instructions in Y86, and how the flags must be set for the jump to be taken to the target address for each type of unconditional jump: je, jne, jl, jle, jg, jge.

87. What does it mean to say a jump (or branch) is taken in Y86? [The next instruction is the instruction at the address of the target label.] What does it mean to say that a jump (or branch) is not taken? [The next instruction executed will be the instruction immediately after the jump (or branch) instruction.]

88. What are labels in Y86? [Strings that mark addresses in memory. The assembler converts the label to the corresponding address, so the CPU does not see the label; it only sees addresses.]

89. What two things does the call instruction in Y86 do (Be sure to pay attention to the order in which these two things are done)? Be sure you can say, after a call instruction is executed, what address will be in the PC. Also be sure you can say what return address was pushed onto the stack by the call instruction.

90. What two things does the ret instruction in Y86 do (Be sure to pay attention to the order in which these two things are done)?

91. What is the portion of the stack that each subroutine (function/procedure/method) is given to use called? [A frame.]

92. Why does each subroutine (other than main) save the calling subroutine's rbp (base/frame pointer) before it sets its own base/frame pointer? [To ensure that the calling subroutine's base/frame pointer can be restored before returning to the calling subroutine.]

93. When new data is pushed onto the stack, how does the stack grow in memory? [It grows downward; that is, it grows toward lower addresses.]

94. Be sure you understand what the pushq and popq instruction do in Y86. (And be sure you understand how the address stored in register rsp changes when each of these instructions is executed.)

95. How are parameters passed on the stack in Y86 (and also in Intel)? [They are pushed onto the stack in reverse order; the last parameter is pushed first, the second last second, … and the first parameter is pushed last.]

96. What is always pushed onto the stack after all of the parameters (if any) when a subroutine is called? [The return address, when the call instruction is executed to call the suborutine.]

Slide set A-6

97. What two values are used in Boolean logic (1/0, true/false, on/off; these are all equivalent).

98. Know the truth table for Boolean AND.

99. Know the truth table for Boolean OR.

100. Know the truth table for Boolean NOT.

101. Know the precedence rule for AND/OR/NOT. (NAO rule.)

102. What is an important practical use of Boolean identities/laws? (They can be used to simplify circuits.)

103. Why is circuit simplification important? (It can reduce cost of implementing a circuit, power use, and heat.)

104. Know the symbols for AND, OR, and NOT gates.

105. Know the truth table for XOR.

106. Know the truth table for NAND

107. Know the truth table for NOR.

108. Know the symbols for XOR, NAND, and NOR gates.

109. Know why NAND and NOR are useful gates. (They are cheap to implement and are universal.)

110. Know the two inputs and two outputs of a half adder.

111. Know the three inputs and two outputs of a full adder.

112. What is a decoder used for?

113. If a decoder has n inputs, how many outputs does it have?

114. For a given input, how many of the output lines of a decoder can have the value 1?

115. What is a multiplexor used to do?

116. If a multiplexor has n inputs, how many control lines are needed to be able to select one of the n input lines to pass to the output?

117. Be able to explain the difference between a combinational circuit and a sequential circuit.

118. Which type of circuit, combinational or sequential, is used to implement memory?

119. Which type of circuit, combinational or sequential, are full-adders, multiplexors, and decoders?

120. Which type of circuit, combinational or sequential, uses a clock input?

121. Which combination of input bit values did we say must be prevented from occurring for an SR-gate (flip-flop, latch), because it makes the circuit unstable (the output is unknown)?

122. When a clocked SR-gate/latch/flip-flop is used to implement a D gate/latch/flip-flop, which input (S or R) is used for D, and which (S or R) is used for ~D (inverted D)?

Slide set A-7

123. When the address in the PC is sent to the memory to "fetch" the instruction to be executed, what does fetching mean? (Reading the instruction by the memory, and sending it back to the CPU.)

124. When can the address in the PC be incremented by the CPU? (After it is sent to the memory.)

125. In the simple accumulator machine, be sure you understand the format of each one byte instruction (three msbs are 3 bit opcode for instruction, and 5 lsbs are address of operand in memory.)

126. Be sure you understand what each of the following does in the simple accumulator machine:

PC

INC

ACC

ALU

IR

DECODE

MAR

MDR

Bus (Remember that this is the internal bus in the CPU, separate from the memory bus.)

127. Be sure you understand how the MAR, MDR, and read enable line are used to read data (or an instruction) from memory.

128. Be sure you understand how the MAR, MDR, and write enable line are used to write data to memory.

129. How many clock cycles does each instruction require in the simple accumulator machine?

130. Why does the simple accumulator machine only allow sequential execution of instructions? (There are no branch/jump instructions, and no call or return instructions.)

131. How does use of a cache improve performance? (The CPU will not have to wait for data/instructions to come from memory if they are in the cache, and very often, they are.)

132. About how large are CPU caches? (Typically less than 0.1% of the size of main memory/RAM.)

133. When data is moved to/from the cache, are single words moved? (No, whole blocks are moved; a block is usually 16 words or more.)

134. In the direct-mapped cache we looked at, be sure you understand how the 16 bit address is divided into bits: 5 tags bits – 7 block bits – 4 word bits; also be sure you understand what each part of the address is used for (tag bits must be stored when a memory block is stored; block bits and word bits are not stored; block bits are used by the cache to determine which cache block the block from memory should be placed/stored in).

135. What is cache hit rate? What is cache miss rate? What are typical cache hit rates we mentioned in class for many types of software?

136. What are typical cache hit rates in many types of programs? (Hit rates around 90% are common.)

137. What is the difference between write-through and write-back protocol for updating data in the cache in memory? Which protocol uses a dirty bit, and how is the dirty bit used?

138. What is the fundamental idea behind pipelining a CPU? (Overlap the stages of execution of various instructions, so that more than one instruction can be executed at any given time.)

146. With our 4-stage execution model, how many clock cycles are required to execute each instruction if a pipeline is not used? How many instructions complete per clock cycle (once the pipeline is fully loaded) if a pipeline is used? How much of an improvement in performance is there theoretically?

147. What can we say about the general improvement in performance when a pipeline is used? Will a pipeline with more stages offer more improvement in performance, or less?

148. What did we say is a typical number of stages in modern pipelined CPUs? [More than 20]

149. What are the potential problems in a pipeline? [Instruction hazards and data hazards.]

150. How much do pipeline hazards typically reduce the theoretical maximum performance improvement? [Less than 10%]

151. How do multi-core processors differ from single-core processors? [Each of the multiple cores has its own registers, including PC, IR, PSR, and data registers, and also has all of the ALU hardware to read/fetch, and execute instructions. Generally, each CPU has a cache also, though there may also be a higher-level cache that is shared by all cores.]