CSC 3501 Computer Organization and Design

Homework #7 Solution

Chapter 5 Review

3) Expanding Opcodes: The basic idea is that we do not have to use all the bits of an instruction word the same way for each instruction. If the instruction has more number of operands, we make the opcode short so that we have lot of bits to accommodate the operands of the instruction. If the instruction has very few operands, we can use more number of bits for the opcode, and only few bits for the operands.

4) The value 98765432 in little and big endian machines would be stored as shown below

Address→	00	01	10	11
Big Endian	98	76	54	32
Little Endian	32	54	76	98

Endian-ness represents the byte ordering used to represent the given data. It ‘matters’ because when data is moved across different machines (some of which may use little endian format and others may use big endian format to store data) each machine needs to know how the data was originally stored and then convert and store it in it’s own format. Otherwise it may end up getting the wrong value.

7) Fixed Length Instructions: Waste space but are fast and result in better performance when instruction level pipelining is used.

Variable Length Instructions: Are more complex to decode but save storage space. Can provide complex addressing modes.

16) In theory, the difference between these two modes is how they are used, but not how the operands are computed.

In **indexed addressing** mode, an index register is used to store an offset (or displacement), which is added to the operand, resulting in the effective address of the data. For example, if the operand X of the instruction LOAD X is to be addresses using indexed addressing, assuming R1 is the index register and holds the value 1, the effective address of the operand is actually X + 1.

In **Based addressing** mode a base address register is used instead of an index register. The base register holds a base address, where the address field represents a displacement from the base.

17) The various addressing modes allow us to specify a much larger range of locations than if we were limited to using one or two nodes.

19) Theoretical speedup = number of stages in a pipeline(k) = 4

In practice, Speedup $S = (n^* t_n) / ((k + n - 1) * t_p)$

Given,

No. of tasks, n = 100

No. of pipeline stages, k = 4

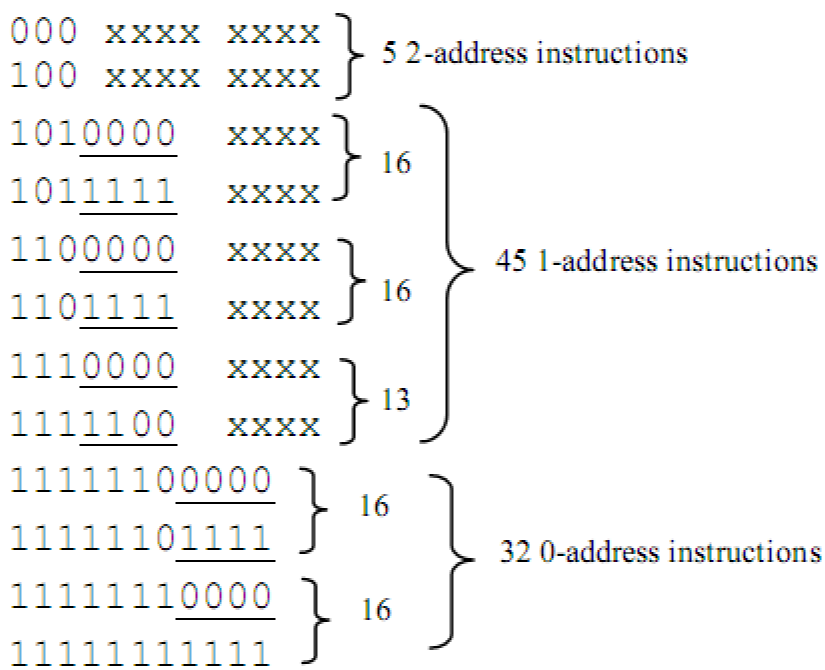
Time taken for each clock cycle, t_p = 20ns

Time for each task, $t_n = k * t_p$ = 80ns

S = (100*80) / ((4+100-1) * 20) = 3.883

Chapter 5 Exercise

11(a). Yes. The 2-address instructions could be represented 000xxxxxxxx through 100xxxxxxxx (using 000 through 100 for opcodes). The 1-address instructions could use 1010000 through 1011111 (16), 1100000 through 1101111 (16), and 1110000 through 1111100 (13 more, for a total of 45). The 0-address instructions could use 11111100000 through 11111101111 (16), and 11111110000 through 11111111111 (16). So we have:



11(b). Assume the two-address instructions use bit patterns 000 xxxx xxxx through 101 xxxx xxxx. Assume also that the zero-address instructions are of the format 11111101000 through 11111101111 (8), 11111110000 through 11111110111 (8), and 11111111000 through 11111111111 (8) (These constitute the last 16 binary numbers possible with 11 bits). Then all instructions beginning with 110 (1100000 xxxx through 1101111 xxxx) could be one address instructions (16). In addition, 1110000 xxxx through 1111101 xxxx could be one address instructions, giving us 14 more, for a total of 30 1-address instructions.

14)

Mode	Value
Immediate	500
Direct	100
Indirect	600
Indexed	800

20) For one instruction, there is no speedup. The speedup comes with the parallel execution of multiple instructions. While the first instruction is decoding, the second can be fetched; while the first instruction is performing the ALU instruction, the second can be decoding, and the third can be fetched, etc.