Daniel Martinez

CIS 11- 34595

Assignment 1

Section 1.1.3

1. An assembler takes assembly language and makes it into a language the computer can understand and execute. A linker takes the multiple files created by the assembler and *links* them together to make a functioning program.

3. High level languages have a one-to-many relationship meaning they can have one line of code carrying out many functions like x = (y \* 8) + 4. Machine language, on the other hand, can only take on one-to-one relationship code so a mathematical equation like the one above would need 3 lines of code just for the right side of the equation.

8. I suppose that type checking, using pointers, is probably stricter in assembly language as far as how you request it in the code. Since in assembly language you have more freedom it means you have to be very specific in what it is you are requesting. However, if looked at from the point of view of available objects that pointers can reference then C and C++ are stricter because you can’t easily access any storage information as freely as in assembly language.

12. Assembly language for x = (y \* 4) + 3 mov eax, y mov ebx, 4 imul ebx add eax, 3 mov x, eax

Section 1.2.1

1. I suppose the translation method executes faster because it converts the entire program from L1 to L0 all at once rather than just a little bit at a time while it is also trying to run the program as it decodes it. Basically, it’s just easier for the computer to use translation because it’s one thing at a time rather than switching back and forth.
2. True, except I would include that the L1 program is running as it is being decoded.

10. They are translated to Instruction set architecture (ISA)

Section 1.3.9

2. a.) 11111000 🡪 00000111 🡪 00001000 🡪 8

b.) 11001010 🡪 00110101 🡪 00110110 🡪 44

c.) 11110000 🡪 00001111 🡪 00010000 🡪 16

3. a.) 11110000 + 00000010 🡪 11110010

b.) 11010101 + 01101011 🡪 (1)01000000 (1) only displayed if there is room

c.) 00001111 + 00001111 🡪 00011110

5. a.) 7 b.) 9 c.) 15

6. a.) 0011 0101 1101 1010 🡪 35DA

b.) 1100 1110 1010 0011 🡪 CEA3

c.) 1111 1110 1101 1011 🡪 FEDB

7. a.) A4693FBC 🡪 1010 0100 0110 1001 0011 1111 1011 1100

b.) B697C7A1 🡪 1011 0110 1001 0111 1100 0111 1010 0001

c.) 2B3D9461 🡪 0010 1011 0011 1101 1001 0100 0110 0001

Section 1.4.2

|  |  |  |
| --- | --- | --- |
| X | Y | X /\ Y |
| T | T | T |
| T | F | F |
| F | T | F |
| F | F | F |

1. 2.

|  |  |  |  |
| --- | --- | --- | --- |
| X | ~X | Y | ~X \/ Y |
| T | F | T | T |
| T | F | F | F |
| F | T | T | T |
| F | T | F | T |

3. ( T /\ F ) \/ T 🡪 F \/ T 🡪 T

4. ~( F \/ T ) 🡪 ~T 🡪 F

5. ~F \/ ~T 🡪 T \/ F 🡪 T

Section 1.7.1

1. The one furthest to the left

2. a.) 00110101 🡪 53

b.) 10010110 🡪 128 + 16 + 4 + 2 🡪 150

c.) 11001100 🡪 128 + 64 + 8 + 4 🡪 204

3. a.) 10101111 + 11011011 🡪 (1)10001010

b.) 10010111 + 11111111 🡪 (1)10010110

c.) 01110101 + 10101100 🡪 (1)00100001

4. 00001101 + 11111001 = (1)00000110

5. a.) word – 16

b.) doubleword – 32

c.) quadword – 64

d.) double quadword- 128

6. a.) 12 b.) 16 c.) 16

7. a.) 0011 0101 1101 1010 🡪 35DA

b.) 1100 1110 1010 0011 🡪 CEA3

c.) 1111 1110 1101 1011 🡪 FEDB

8. a.) 0126F9D4 🡪 0000 0001 0010 0110 1111 1001 1101 0100

b.) 6ACDFA95 🡪 0110 1010 1100 1101 1111 1010 1001 0101

c.) F69BDC2A 🡪 1111 0110 1001 1011 1101 1100 0010 1010

9. a.) 3A 🡪 48 + 10 🡪 58

b.) 1BF 🡪 256 + 176 + 15 🡪 447

c.) 1001 🡪 4096 + 1 🡪 4097

10. a.) 62 🡪 96 + 2 🡪 98

b.) 4B3 🡪 1024 + 176 + 3 🡪 1203

c.) 29F 🡪 512 + 144 + 15 🡪 671

11. a.) -24 🡪 18 🡪 E7

b.) -331 🡪 14B 🡪 EB4

12. a.) -21 🡪 15 🡪 EA

b.) -45 🡪 2D 🡪 D2

13. a.) 6BF9 🡪 24576 + 2816 + 240 + 9 🡪 27641

b.) C123 🡪 3EDC 🡪 12288 + 3584 + 208 + 12 🡪 -16092

14. a.) 4CD2 🡪 16384 + 3072 + 208 + 2 🡪 19666

b.) 8230 🡪 7DBF 🡪 28672 + 3328 + 176 + 15 🡪 -32191

15. a.) 10110101 🡪 01001010 🡪 01001011 🡪 64 + 8 + 2 + 1 🡪 -75

b.) 00101010 🡪 32 + 8 + 2 🡪 42

c.) 11110000 🡪 00001111 🡪 00010000 🡪 -16

16. a.) 10000000 🡪 -128

b.) 11001100 🡪 00110011 🡪 00110100 🡪 32 + 16 + 4 🡪 -52

c.) 10110111🡪 01001000 🡪 01001001 🡪 64 + 8 + 1 🡪 -73

17. a.) -5 🡪 00000101 🡪 11111010 🡪 11111011

b.) -42 🡪 00101010 🡪 11010101 🡪 11010110

c.) -16 🡪 00010000 🡪 11101111 🡪 11110000

18. a.) -72 🡪 01001000 🡪 10110111 🡪 10111000

b.) -98 🡪 01100010 🡪 10011101 🡪 10011110

c.) -26 🡪 00011010 🡪 11100101 🡪 11100110

19. a.) 6B4 + 3FE 🡪 AC3

b.) A49 + 6BD 🡪 1217

20. a.) 7C4 + 3BE 🡪 B83

b.) B69 + 7AD 🡪 1427

21. hexadecimal = 42 ; decimal = 642

22. hexadecimal = 47; decimal = 647

25.

|  |  |  |  |
| --- | --- | --- | --- |
| A | B | A \/ B | ~( A \/ B ) |
| T | T | T | F |
| T | F | T | F |
| F | T | T | F |
| F | F | F | T |

Section 1.7.2

9. To subtract binary numbers work from right to left subtracting the bottom from the top like usual. 1 subtracted from 1 and 0 subtracted from 0 should come out to 0. When there is a 0 subtracted from 1 it should come out to 1 in that ‘slot’ of your answer. Now, when there is a 1 being subtracted from a 0 you must go to the left until you find a 1. When you do you will carry the 1 to the next spot now counting it as an 11. Whereas if you need to keep carrying the 1 to get to your 0 minus 1 ‘slot’ then the 11 will become a 10. When subtracting the 1 from the 11 once it has been moved over enough it will be counted as a 10 and all the 10’s will be represented by 1’s in your answer. Anytime you end up with a 10 minus 1 subtraction it will come out to a 0 in your answer. This number system goes into further detail in the book as to how these numbers are used.

