Matt Kennedy CIS 351
Parker Skarzynski Lab 08

1. The address of the first instruction run in the program is 0x00400000.
2. In MIPS, after a program is assembled, the Text Segment section has an Address column which displays this information.
3. Using the la $t0, val command, the address returned is 0x10010000
4. I think this is what you're referring to?

0x10000000 .extern
0x10001000 .data
0x10040000 heap
current $gp
current $sp
0x00400000 .text
0x90000000 .kdata
0xffff0000 MMIO

5. 2^{32}
6. The first line is lui $1, 0x00000001. Next is ori $1, $1, 0x00000002. Lastly, the program runs add $2, $2, $1
7. Lui stands for "load upper immediate"
8. 32767 in hex is 0x7FFF, 65538 is 0x10002. The max limit for 32 bit constants is less than 0x10002, so the assembler needs to break down the instruction to handle this into several pieces.
9. From http://people.cs.pitt.edu/~xujie/cs447/MIPS_Instruction.htm, "The MIPS instruction set addresses this principal by making constants part of arithmetic instructions. Furthermore, by loading small constants into the upper 16-bits of a register.
10. Hex representation is 1001.
11. This is the beginning of the Data Segment section. All declared data segments are stored here.
12. The second command is lw $8, 0x00000000 ($1). The first part, $8, refers to register $t0, which was provided. The second part is the offset of the address stored in register 1 to load. The third part, $1, is the register that holds the address of the value we want to load.
13. The offset parameter could be avoided. We used this on previous labs. For example: In the example, assume $a0 holds the data and $t3 is a word counter that is incremented by 4 when we want to access some index.

```
la $t0, ($a0)
add $t1, $t0, $t3
lw $t2, 0($t1)
```

14. Instead of having to load an offset or use an lui command every time we want to access something other than the base address, the 3 parameter lw is useful in reducing the redundancy and making the common case fast by allowing us to specify the word offset.
15. If there are only 2 parameters, the computer will always check the base address. In the 3 parameter example, however, the computer has to find the base address, then calculate the offset, then pull that value, which adds steps to what the computer has to do, which slows it down.
16. For the li command, the real instruction used is addiu.
17. For the move command, the real instructions is addu.
18. Overall, I don’t believe so. It follows the principle of Make the Common Case Fast, from question 9. If we have more base instructions, then the processor takes longer to find and perform each. If we keep it simple, then pseudo commands may take slightly longer, but it would be overall less than having to look for each command.
19. See attached file
20. This is due to the size of the attached address. For la $a0 string1, the i-type instruction only has 16 bits to work with for an address, where addresses are 32 bits. Therefore, it breaks it up into 2 parts to make the loaded address 32 bits. For li $v0 4, the pseudo command addiu can handle the number 4 being added so no need to break it into 2 separate commands.
21. 0x00000018
22. This is the number of instructions away the exit block is, roughly. The assembler will calculate based on the word offset (add 1 and multiply by 4) to get to the block.
23. The instruction in hex is 0x08100004. This is 000010 00000100000000000000000100 in binary. The 26 bit address translates to 00100004 in hex.
24. This is the word offset for the label that the assembler is looking for. It calculates this based on where the word is in context to the program.