|  |  |  |
| --- | --- | --- |
| BLG351E  Experiment 3 “Advanced Assembly Coding”  REPORT | CRN | 12633 |
| Group | G8 |
| Name #1 | Eyüpcan Çakır |
| Name #2 | Furkan Salık |
| Name #3 | Ahmet Fatih Akcan |
| Name #4 | Muhammed Yavuz Köseoğlu |
| Q1) (30 pts.) How did you reach the value held inside the variables or the values held inside the hash table? Briefly explain. | | |
| We defined the variables in the data section and then in the text section, we assigned addresses of the variables to general purpose registers using immediate addresses of labels with mov instruction.  For example “mov #hash,R12” for hash map, “mov #aa,R9” for variable aa. After that we wrote the general purpose registers containing the variable addresses to the memory browser to observe the variables and values. And in order to use these variables and hash table in the rest of the program,we used indirect autoincrement, indexed and absolute addressing modes in instructions. For example “mov @R14+,&aa” for num array, “mov &aa,0(R12)” for hash map and “rla &cc” for variable cc. | | |
| Q2) (30 pts.) During the update of the hash table, why did you increase the address held in the register by 2? | | |
| We split the id number into 3 parts and each of them includes 3 digits. This means they can take value between 0-999 and the maximum decimal number can be represent with 1 byte is 255, thus we should use 2 bytes for every splitted parts. So we allocated 58 bytes memory for the hash map. Actually the size of the hash map was already given in the experiment file. In addition to these, the size of the registers and words are 2 bytes. And insctructions with “.w” extension or without any extension use 2 byte words as operands. When these instructions read or write data from memory, since each memory location formed by 1 byte, they use 2 contiguous memory blocks, one for the least significant byte at the even memory address and the other for the most significant byte at the odd address. So we increased addresses by 2 due to the size of our data and to avoid from overwritten contiguous memory blocks. | | |
| Q3) (40 pts.) If the 29th location in the hash table is full, we need to go back to first location. How did you handle this situation? | | |
| We were already checking indexes for linear probing. So we were checking whether the current index was empty and if not, we were directly incrementing the current address by 2. To handle the circularity situation, we added an extra checkpoint before double increment instruciton. We checked for whether current index is the last index of the hash table and if it is, we returned the current index to the base index which is the address of the first element of the hash table. We realized this check operation with “mov #hash,R10//add #038h,R10//cmp R10,R12” instructions. The reason we used #038h was to get the address of the last 2 bytes of the hash map and we used R12 general purpose register for it contains the current index. And lastly we realized return the first location operation with “mov #hash,R12” instruction. | | |
|  | | |