Question 1 (25 points): The code for function `make_big_endian` in the C programming language is as follows:

```
00  struct page_list {
01                  page_list*       prev;
02                  page_list*       next;
03                  unsigned int     page;
04                 };
05  int  *page_count;
06  char valid[1000];
07
08  void  make_big_endian(page_list **page_pointers)
09  {
10    page_list *page_array;
11    unsigned int      i;
12    ...
13    valid[i-1] = valid[i];
14    *page_pointers++;
15    page_array++;
16    *page_pointers = page_array;
17    *page_count = i;
18     ....
19  }
```

In this code `page_array` is a dinamically allocated array of `page_lists` (the actual allocation call is ommitted above) and `page_pointers` is an array of pointers to `page_list`. Assume that the variable `page_array` is stored in the stack at the address given by `$fp+4`; the global variable `page_count` is at the address given by `$gp` and the global array `valid` starts at the address `$gp+4`. The parameter passed to `make_big_endian` is the address of the first position of the array of pointers to `page_list` called `page_pointers`. Assume that in this architecture an integer is stored in 32 bits and a memory address also occupies 32 bits. Assume that `i` is in `$t0`. Write MIPS assembly code for each one of the following statements in the program above:

1. (**5 points**) `valid[i-1] = valid[i];`

```
# valid is an array of chars, thus no need to multiply index
addi    $t2, $gp, $t0  #  $t2 <-- $gp+i = Address(valid[i-4])
lb      $t3, 4($t2)    #  $t3 <-- MEM[$gp+i+4] = valid[i]
sb      $t3, 3($t2)    #  MEM[$gp+i+3] = valid[i-1] <-- valid[i]
```

2. (**5 points**) `*page_pointers++;`

```
# *page_pointers is a pointer to a page_list
lw      $t2, 0($a0)     #  $t2 <-- *page_pointers
addi    $t2, $t2, 4     #  $t2 <-- $t2+4
sw      $t2, 0($a0)     #  *page_pointers <-- *page_pointers + 4
```

3. (**5 points**) `page_array++;`

```
# page_array points to a page_list which is formed by 12 bytes
lw      $t1, 4($fp)    #  $t1 <-- page_array
addi    $t1, $t1, 12   #  $t1 <-- page_array+1
sw      $t1, 4($fp)    #  page_array <- page_array+1
```

4. (**5 points**) `*page_pointers = page_array;`

```
lw    $t1, 0($a0)     #  $t1 <-- *(page_pointers)
lw    $t2, 4($fp)     #  $t2 <-- page_array
sw    $t2, 0($t1)     #  *page_pointers <-- page_array
```

5. (**5 points**) `*page_count = i;`

```
lw     $t2, 0($gp)     #  $t2 <-- Address(page_count)
sw     $t0, 0($t2)     #  *page_count <-- i
```