►Solution◄

Question 1: (0 points)
Bank of Questions

Assembly Code for Pointers (V16, V17

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                 };
SB-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 dinamicaly allocated array of `page_lists` (the actual allocation call is omitted 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 RISC-V assembly code for each one of the following statements in the program above:

Question 2: (10 points)
`valid[i-1] = valid[i];`

Solution: The question states that "the global array `valid` starts at the address gp+4", which means that the first character of the array valid is at the address given by gp+4.

```
# 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]
```

Question 3: (10 points)

`*page_pointers++;`

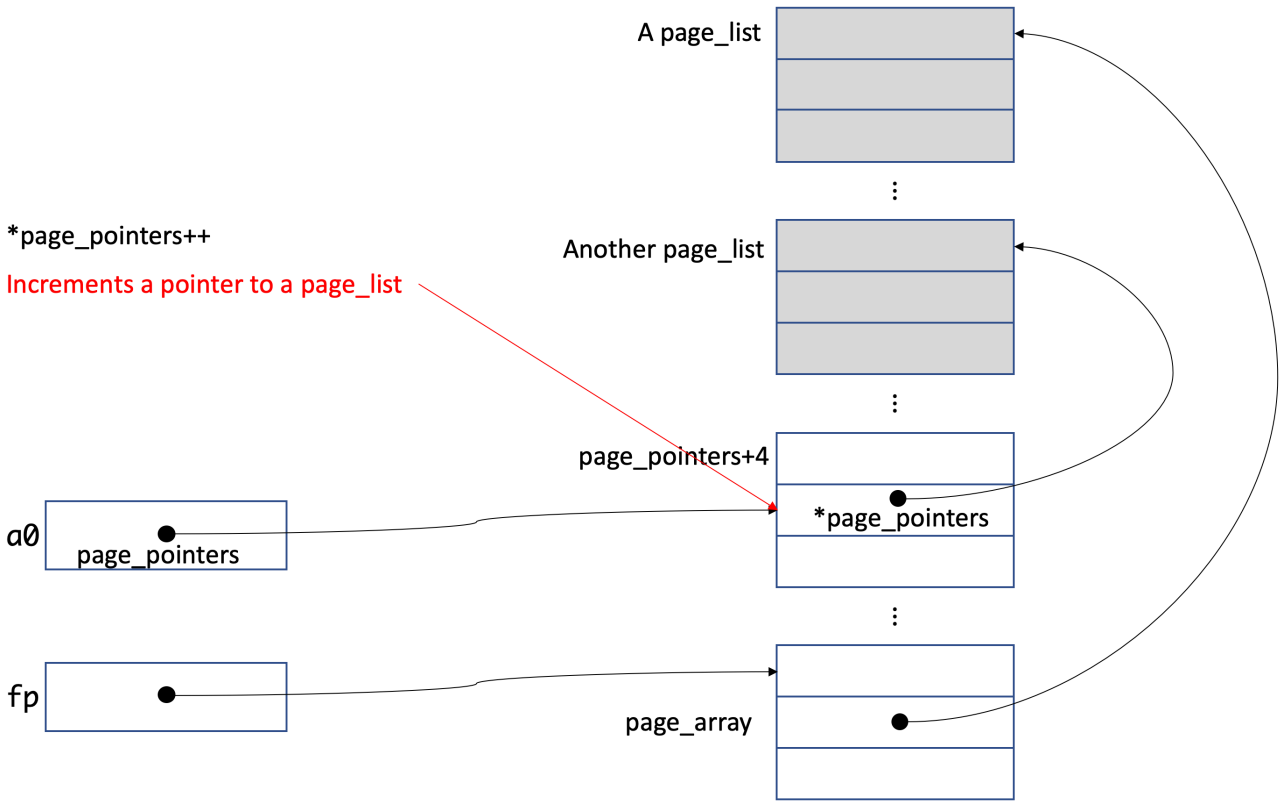

Figure 1: Illustrates `(*page_pointers)++;`

Solution:

It has long been said that the powerful C programming language gives you as much rope as you need to hang yourself. My mistake in setting up this question illustrates the issue. I was careless with the operator precedence in C. My intention was to ask

for the code for the statement `(*page_pointers)++` to create the situation illustrated in Figure **??**. In that case, one would have to load the value from the memory location specified by `a0`, increment it by twelve because `*page_pointers` is a pointer to a `page_list` and a `page_list` has twelve bytes, and store the new value in the location specified by `a0`. I realized my mistake when someone asked a question in Slack, but it was too late to change it. I did explain this situation at the start of the morning and of the afternoon classes on Friday to reward students that decided to attend these classes - a bit of participation marks.

As the question stood it was much less interesting. The statement is equivalent to `*(page_pointers++` because the postfix increment `++` in C has a higher precedence than the dereference operator `*`: the `*` has no function and the statement is equivalent to `page_pointers++`. Because `page_pointers` is a pointer to a pointer, its increment must be by 4.

```
# ++ has priority over *  thus, this statement is
# like page_pointers = page_pointers+4  because
# page_pointers is a pointer to a pointer.
addi    a0, a0, 4     #  page_pointers = page_pointers+4
```

Question 4: (10 points)

```
page_array++;
```

Solution:

```
# 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
```

Question 5: (10 points)

```
*page_pointers = page_array;
```

Solution:

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

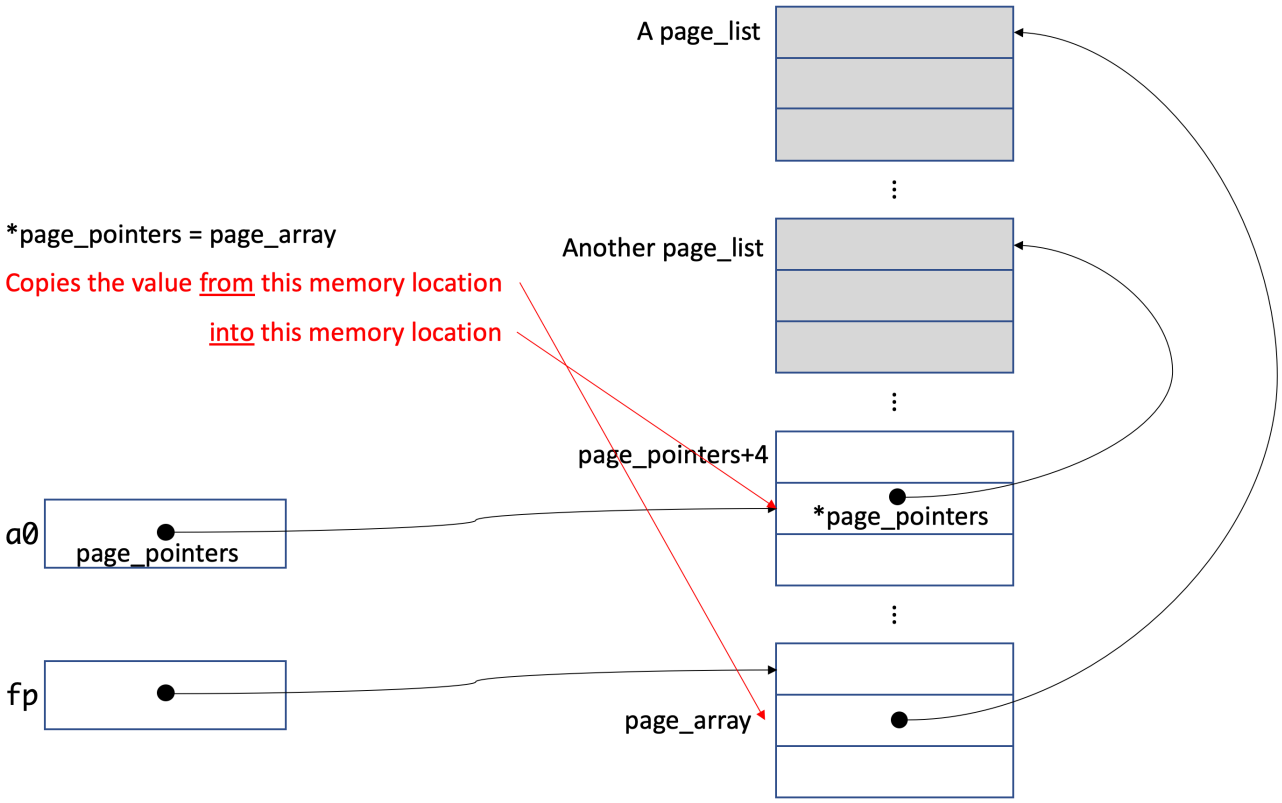

Figure 2: Illustrates `*page_pointers = page_array;`

Question 6: (10 points)

```
*page_count = i;
```

Solution:

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