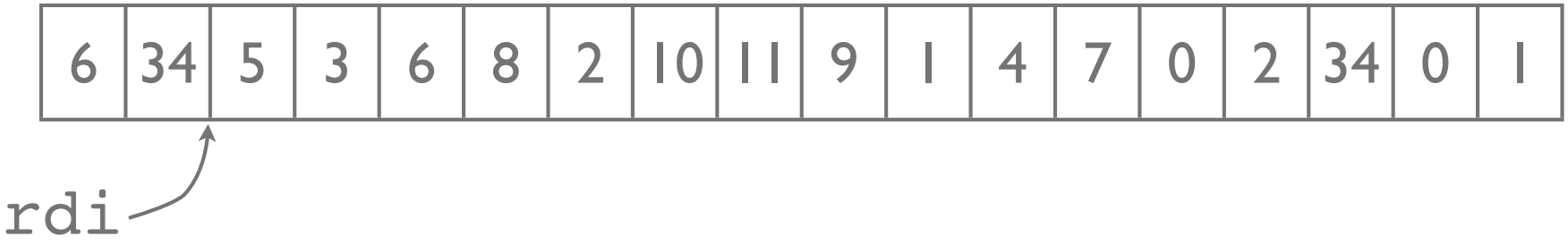
Setup for examples/lab session

Assembly code examples

- **General setup**: start with `rdi` pointing to a zero-terminated array of 32 bit words with at least one element:



- **Goal**: Perform some computation and return a result (if appropriate) in `eax` (possibly also modifying the array)
- **Example 1**: Return the length of the input array.
- **Example 2**: Return the largest number from the array.

... continued:

- **Example 3**: Return the position of the largest number in the array.
- **Example 4**: Return the average value of the numbers in the array using integer division and ignoring any remainder.
- **Example 5**: Reverse the order of the elements in the array, without using any additional storage.
- **Example 6**: Sort the elements in the array into increasing numerical order, without using any additional storage. (No algorithmic sophistication should be expected here!)

Test program (main.c)

```c
#include <stdio.h>

extern int f(int*);

void printArray(char* msg, int* a) {
  for (; *a; ++a) {
    printf("%s%d", msg, *a);
    msg = ", ";
  }
  printf("\n");
}

int main() {
  int a[] = { 5, 3, 6, 8, 2, 10, 11, 9, 1, 4, 7, 0 };
  printArray("Before: ", a);
  printf("first result is %d\n", f(a));
  printArray("After: ", a);
  ...
  printf("\n");
  return 0;
}
```

Assembly code skeleton

```
        .file   "linux.s"
        .text
        .globl  f
f:

### This is where your code begins ...

        movl    $42, %eax       # replace this with your code!

### This is where your code ends ...

        ret
```

Register use conventions

- The input parameter is in `rdi`.
- `rsp` is the stack pointer and should not be used for other purposes.
- The code can freely use the following registers:

`rax`	(return result)
`rsi`, `rdx`, `rcx`, `r8`, `r9`	(arguments 2-6)
`r10`, `r11`	(caller saved)

- If the code uses `rbx`, `rbp`, `r12`, `r13`, `r14`, or `r15`, then it MUST restore them to their original values before it ends. (e.g., `pushq` the register value on to the stack at the beginning of the code and then `popq` it off at the end.)