The second method uses the **bit-specific addressing**. The LM4F/TM4C family implements a flexible way to access port pins. This bit-specific addressing doesn't work for all the I/O registers, just the parallel port data registers. The mechanism allows collective access to 1 to 8 bits in a data port. We define eight address offset constants in Table 6.3. Basically, if we are interested in bit *b*, the constant is $4*2^b$. There 256 possible bit combinations we might be interested in accessing, from all of them to none of them. Each possible bit combination has a separate address for accessing that combination. For each bit we are interested in, we add up the corresponding constants from Table 6.3 and then add that sum to the base address for the port. The base addresses for the data ports can be found Table 6.2. For example, assume we are interested in Port A bits 1, 2, and 3. The base address for Port A is 0x4000.4000, and the constants are 0x0008, 0x0010, and 0x0020. The sum of 0x4000.4000+0x0008+0x0010 +0x0020 is the address 0x4000.4038. If we read from 0x4000.4038 only bits 1, 2, and 3 will be returned. If we write to this address only bits 1, 2, and 3 will be modified.

If we wish to access bit	*Constant*
7	0x0200
6	0x0100
5	0x0080
4	0x0040
3	0x0020
2	0x0010
1	0x0008
0	0x0004

Table 6.3. Address offsets used to specify individual data port bits.

The base address for Port A is 0x4000.4000. If we want to read and write all 8 bits of this port, the constants will add up to 0x03FC. Notice that the sum of the base address and the constants yields the 0x4000.43FC address used in Table 6.2 and Program 6.1. In other words, read and write operations to **GPIO_PORTA_DATA_R** will access all 8 bits of Port A. If we are interested in just bit 5 of Port A, we add 0x0080 to 0x4000.4000, and we can define this in C and in assembly as

```
#define PA5   (*((volatile unsigned long *)0x40004080))
```

```
PA5 EQU 0x40004080
```

Now, a simple write operation can be used to set **PA5**. The following code is friendly because it does not modify the other 7 bits of Port A.

```
PA5 = 0x20;       // make PA5 high
```

A simple write sequence will clear **PA5**. The following code is also friendly.

```
PA5 = 0x00;       // make PA5 low
```

A read from **PA5** will return 0x20 or 0x00 depending on whether the pin is high or low, respectively. If **PA5** is an output, the following code is also friendly.

```
PA5 = PA5^0x20;   // toggle PA5
```

Note that the base address when computing the bit-specific address for PortA is 0x40004000, the following table lists the base addresses for the other ports.

Port	Base address
PortA	0x40004000
PortB	0x40005000
PortC	0x40006000
PortD	0x40007000
PortE	0x400240000
PortF	0x40025000

Table 6.4. Base Addresses for bit-specific addressing of ports A-F.

Help

CHECKPOINT 6.5

What happens if we write to location 0x4000.4000?

Hide Answer

Nothing happens. Since none of the address bits are selected, none of the port bits are affected.

CHECKPOINT 6.6

Specify a #define that allows us to access bits 7 and 1 of Port A. Use this #define to make both bits 7 and 1 of Port A high.

Hide Answer

The base address for Port A is 0x4000.4000.
#define PA71 (*((volatile unsigned long *)0x40004208))
PA71 = 0x82; // sets PA7 PA1, other 6 bits are not affected

CHECKPOINT 6.7

Specify a #define that allows us to access bits 6, 1, 0 of Port B. Use this #define to make bits 6, 1 and 0 of Port B high.

Hide Answer

The base address for Port B is 0x4000.5000.
#define PB610 (*((volatile unsigned long *)0x4000510C))
PB610 = 0x43; // sets PB6 PB1 PB0, other 5 bits are not affected

To understand the port definitions in C, we remember **#define** is simply a copy paste. E.g.,

```
data = PA5;
```

becomes

```
data = (*((volatile unsigned long *)0x40004080));
```

To understand why we define ports this way, let's break this port definition into pieces. First, 0x40004080 is the address of Port A bit 5. If we write just **#define PA5 0x40004080** it will create

```
data = 0x40004080;
```

which does not read the contents of PA5 as desired. This means we need to dereference the address. If we write **#define PA5 (*0x40004080)** it will create

```
data = (*0x40004080);
```

This will attempt to read the contents at 0x40004080, but doesn't know whether to read 8 16 or 32 bits. So the compiler gives a syntax error because the type of data does not match the type of (*0x40004080). To solve a type mismatch in C we **typecast**, placing a (new type) in front of the object we wish to convert. We wish force the type conversion to unsigned 32 bits, so we modify the definition to include the typecast,

```
#define PA5   (*((volatile unsigned long *)0x40004080))
```

The **volatile** is added because the value of a port can change beyond the direct action of the software. It forces the C compiler to read a new value each time through a loop and not rely on the previous value.

Help

Help