## EENG 260 (Microcontrollers)
## Home Work 2

**Problem 1:** Logical Operations

In class we said a divide by 2 operation can be achieved by a logical shift right instruction and also said a multiply by 2 operation can be achieved by an arithmetic shift left. Without worrying about the contents of the registers write out the mathematical expressions of R0 after each of the two instructions:

a) ADD R0, R1, LSL #4

b) ADD R0, R1, R2, ASR #4

Answer:

a) R0 = **R0 + R1*16**

**- I had to check, ADD R0, R1 is equivalent to ADD R0, R0, R1**

b) R0 = **R1 + R2/16**

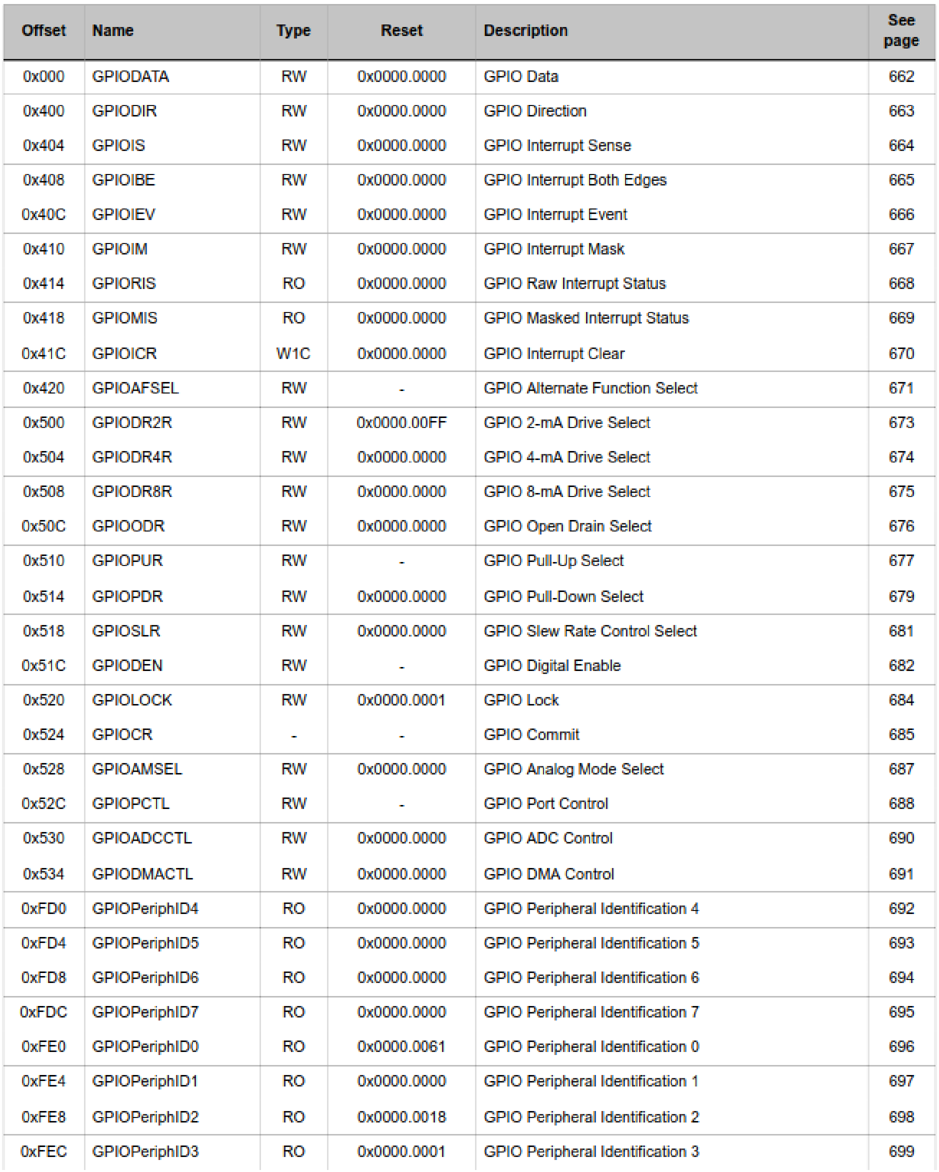
**Problem 2:** GPIO Memory Map

From the class notes titled “Lect_On_Ports_Rev1” or chapter 10 of Tiva C manual it is said that **Port A** has the address range: 0x4000.4000 – 0x4000.4FFF.

a) What components are addressable in this address range?

**All of the device registers are addressable. We’ve worked with a subset of these to configure GPIO in labs. Some registers (notably GPIODATA) are bit-addressable(bit banded) or bit-maskable in this address space, allowing modification in a single STR instead of a read-modify-write cycle.**

**Table 10-6. GPIO Register Map**



| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| 0x000 | GPIODATA | RW | 0x0000.0000 | GPIO Data | 662 |
| 0x400 | GPIODIR | RW | 0x0000.0000 | GPIO Direction | 663 |
| 0x404 | GPIOIS | RW | 0x0000.0000 | GPIO Interrupt Sense | 664 |
| 0x408 | GPIOIBE | RW | 0x0000.0000 | GPIO Interrupt Both Edges | 665 |
| 0x40C | GPIOIEV | RW | 0x0000.0000 | GPIO Interrupt Event | 666 |
| 0x410 | GPIOIM | RW | 0x0000.0000 | GPIO Interrupt Mask | 667 |
| 0x414 | GPIORIS | RO | 0x0000.0000 | GPIO Raw Interrupt Status | 668 |
| 0x418 | GPIOMIS | RO | 0x0000.0000 | GPIO Masked Interrupt Status | 669 |
| 0x41C | GPIOICR | W1C | 0x0000.0000 | GPIO Interrupt Clear | 670 |
| 0x420 | GPIOAFSEL | RW | - | GPIO Alternate Function Select | 671 |
| 0x500 | GPIODR2R | RW | 0x0000.00FF | GPIO 2-mA Drive Select | 673 |
| 0x504 | GPIODR4R | RW | 0x0000.0000 | GPIO 4-mA Drive Select | 674 |
| 0x508 | GPIODR8R | RW | 0x0000.0000 | GPIO 8-mA Drive Select | 675 |
| 0x50C | GPIOODR | RW | 0x0000.0000 | GPIO Open Drain Select | 676 |
| 0x510 | GPIOPUR | RW | - | GPIO Pull-Up Select | 677 |
| 0x514 | GPIOPDR | RW | 0x0000.0000 | GPIO Pull-Down Select | 679 |
| 0x518 | GPIOSLR | RW | 0x0000.0000 | GPIO Slew Rate Control Select | 681 |
| 0x51C | GPIODEN | RW | - | GPIO Digital Enable | 682 |
| 0x520 | GPIOLOCK | RW | 0x0000.0001 | GPIO Lock | 684 |
| 0x524 | GPIOCR | - | - | GPIO Commit | 685 |
| 0x528 | GPIOAMSEL | RW | 0x0000.0000 | GPIO Analog Mode Select | 687 |
| 0x52C | GPIOPCTL | RW | - | GPIO Port Control | 688 |
| 0x530 | GPIOADCCTL | RW | 0x0000.0000 | GPIO ADC Control | 690 |
| 0x534 | GPIODMACTL | RW | 0x0000.0000 | GPIO DMA Control | 691 |
| 0xFD0 | GPIOPeriphID4 | RO | 0x0000.0000 | GPIO Peripheral Identification 4 | 692 |
| 0xFD4 | GPIOPeriphID5 | RO | 0x0000.0000 | GPIO Peripheral Identification 5 | 693 |
| 0xFD8 | GPIOPeriphID6 | RO | 0x0000.0000 | GPIO Peripheral Identification 6 | 694 |
| 0xFDC | GPIOPeriphID7 | RO | 0x0000.0000 | GPIO Peripheral Identification 7 | 695 |
| 0xFE0 | GPIOPeriphID0 | RO | 0x0000.0061 | GPIO Peripheral Identification 0 | 696 |
| 0xFE4 | GPIOPeriphID1 | RO | 0x0000.0000 | GPIO Peripheral Identification 1 | 697 |
| 0xFE8 | GPIOPeriphID2 | RO | 0x0000.0018 | GPIO Peripheral Identification 2 | 698 |
| 0xFEC | GPIOPeriphID3 | RO | 0x0000.0001 | GPIO Peripheral Identification 3 | 699 |

b) Provide addresses, names and functions for some of these components

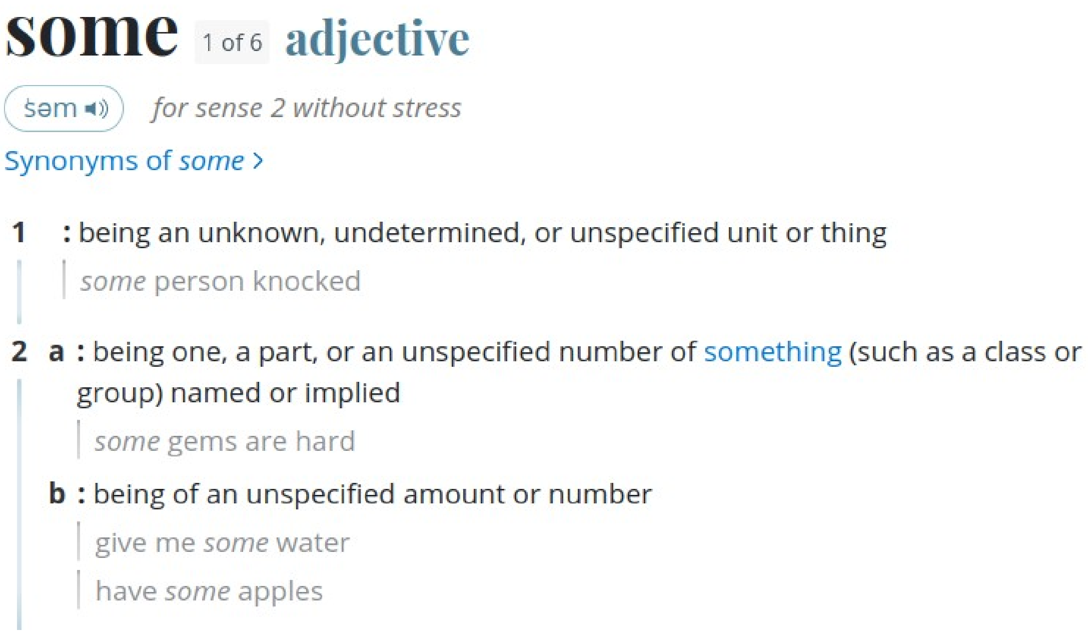
**The address of each component takes the form of (BASE_ADDRESS) + (OFFSET). The function of each register can be read in the datasheet. So, for example,**

**GPIODIR has an offset of 0x400. We add that to the PA base address (0x4000.4000) to calculate the address of 0x4000.4400 for GPIODIR. This register sets the pin to be either an input or an output.**

**GPIODR4R has an address of 0x4000.4504 and is used to set which output pins will have a 4mA output drive**

**GPIODMACTL has an address of 0x4000.4534 and is used to set a bit that will initiate DMA. This is very useful, for example, to automatically read in the values at the pins based on an external clock.**

**How many is some? I provided 3.**



**some** 1 of 6 adjective

ˈsəm *for sense 2 without stress*

Synonyms of *some* >

**1** : being an unknown, undetermined, or unspecified unit or thing

| *some* person knocked

**2 a** : being one, a part, or an unspecified number of something (such as a class or group) named or implied

| *some* gems are hard

**b** : being of an unspecified amount or number

| give me *some* water

| have *some* apples

**Problem 3:** The stack (frame and pointer)

a) Provide a stack frame for a program that saves its contents from memory location 0x2000.7FFC, assume 5 words are saved.

**Ordinarily in ARM the stack grows DOWN, so the initial stack pointer is the "top" value of the stack. Given this, our five words will be stored at:**
**Word 1: 0x2000.7FF8**
**Word 2: 0x2000.7FF4**
**Word 3: 0x2000.7FF0**
**Word 4: 0x2000.7FEC**
**Word 5: 0x2000.7FE8 <--- This is the new value of SP**

b) Where does the return address of a subroutine get stored i.e., when a subroutine completes how does the program know where to start?

**The link register. When branching back from a subroutine, the link-register is loaded into the program counter**

c) What is a stack overflow?

**When PUSH is used without POP a number of times greater than the available words on the stack, the stack pointer will go below the lower-bound of the stack space. This could result in the stack clobbering the heap, or simply a bus error if there isn't something to write at that address**

d) What is a stack underflow?

**When POP is used more times than PUSH has been used, the stack pointer will be above the assigned space for the stack. This could silently write to more ram in that location, potentially overwriting important data, or cause a bus error if there isn't anything to write in that location.**

**Problem 4:** Exceptions and Interrupts

a) Please state the difference between an exception and an interrupt

**An exception is, broadly, software based, while an interrupt is usually external to the cpu. Section 2.5 in the TM4C datasheet indicated that exceptions are all handled by the NVIC, and that IRQ (interrupts) are exceptions also.**

b) What is the purpose of a vector table?

**The vector table stores addresses (function pointers, to use C terminology) to the subroutines/functions that should be called when a particular exception occurs. At the top of the NVIC is the initial stack pointer, followed by the address of the reset handler – when the CPU turns on, it loads the stack pointer into SP from the top of the NVIC, then loads the reset handler address into the PC, starting program execution.**

c) What is the purpose of an interrupt service routine?

**An ISR is the function that gets called when an interrupt occurs. This could be for any number of purposes – a pin just changed state, an input buffer is full, a timer finished… The ISR is what allows asynchronous events to be handled outside of the normal program flow.**

d) Interrupts $IRQ_0$ and $IRQ_1$ have the same priority and both happen to occur at the same time. Which of the two interrupts will be serviced first?

**The interrupt with the lowest IRQ number is processes first: IRQ_0**

e) Provide the steps taken by a processor to enter into an exception and how it returns from exception

**The processor pushes the PC, LR and all registers that are Caller-saved, according to the ABI, to the stack. Then the ISR is entered. Once the ISR completes the LR and other registers are popped back off, then the PC is popped off the stack, returning to the prior execution state.**

**Problem 5:** GPIODATA Register

You have the following assembly instructions in your assembly code:

```
MOVW        R1, #0x4000         ;load lower half of reg R1 with 0x4000
MOVT        R1, #0x4000         ;load upper half of reg R1 with 0x4000
LDRB        R0, #0x05           ;load value 5 into R0
STRB        R0, [R1]            ;store content of R) at memory location
                                pointed to by R1
```

After these four instructions execute:

a) What port is this program manipulating?

**Port A**

b) What register of this specific port is being manipulated?

**GPIODATA, incorrectly, since the address needs a mask for the write to take effect! The STRB instruction tries to write to the base address, with no offset, which is GPIODATA. GPIODATA can't be written without an offset, however.**

c) How are this port's pins configured?

**They are not, in this code. Are you asking how to configure them? What are you trying to configure them to? You need to enable the port, then set GPIODIR according to the direction, GPIOPUR or GPIOR4R or others depending on the specific use.**

d) Now that the port's pins are set up what register would be used to Read date from or write data to the port?

**GPIODATA, using an appropriate mask left shifted by 2 bits and used as an offset for the base address of the port – GPIODATA has no offset.**

e) Which pins of the port will be written to and which pins will be read from?

**NONE. See b)**