Assembly VI

ECE 3710

If everything seems to be going well, you have obviously overlooked something.

- Steven Wright

Q: if datasheet says DATA for PA is GPIO 0x4000.4000

why?

Address	7	6	5	4	3	2	1	0	Name
$400F.E108	GPIOH	GPIOG	GPIOF	GPIOE	GPIOD	GPIOC	GPIOB	GPIOA	SYSCTL_RCGC2_R
$4000.43FC			DATA	DATA	DATA	DATA	DATA	DATA	GPIO_PORTA_DATA_R
$4000.4400			DIR	DIR	DIR	DIR	DIR	DIR	GPIO_PORTA_DIR_R
$4000.4420			SEL	SEL	SEL	SEL	SEL	SEL	GPIO_PORTA_AFSEL_R
$4000.451C			DEN	DEN	DEN	DEN	DEN	DEN	GPIO_PORTA_DEN_R
$4000.53FC	DATA	DATA	DATA	DATA	DATA	DATA	DATA	DATA	GPIO_PORTB_DATA_R
$4000.5400	DIR	DIR	DIR	DIR	DIR	DIR	DIR	DIR	GPIO_PORTB_DIR_R
$4000.5420	SEL	SEL	SEL	SEL	SEL	SEL	SEL	SEL	GPIO_PORTB_AFSEL_R
$4000.551C	DEN	DEN	DEN	DEN	DEN	DEN	DEN	DEN	GPIO_PORTB_DEN_R
$4000.63FC	DATA	DATA	DATA	DATA	DATA	DATA	DATA	DATA	GPIO_PORTC_DATA_R
$4000.6400	DIR	DIR	DIR	DIR	DIR	DIR	DIR	DIR	GPIO_PORTC_DIR_R
$4000.6420	SEL	SEL	SEL	SEL	SEL	SEL	SEL	SEL	GPIO_PORTC_AFSEL_R
$4000.651C	DEN	DEN	DEN	DEN	DEN	DEN	DEN	DEN	GPIO_PORTC_DEN_R
$4000.73FC	DATA	DATA	DATA	DATA	DATA	DATA	DATA	DATA	GPIO_PORTD_DATA_R
$4000.7400	DIR	DIR	DIR	DIR	DIR	DIR	DIR	DIR	GPIO_PORTD_DIR_R
$4000.7420	SEL	SEL	SEL	SEL	SEL	SEL	SEL	SEL	GPIO_PORTD_AFSEL_R
$4000.751C	DEN	DEN	DEN	DEN	DEN	DEN	DEN	DEN	GPIO_PORTD_DEN_R
$4002.43FC							DATA	DATA	GPIO_PORTE_DATA_R
$4002.4400							DIR	DIR	GPIO_PORTE_DIR_R
$4002.4420							SEL	SEL	GPIO_PORTE_AFSEL_R
$4002.451C							DEN	DEN	GPIO_PORTE_DEN_R

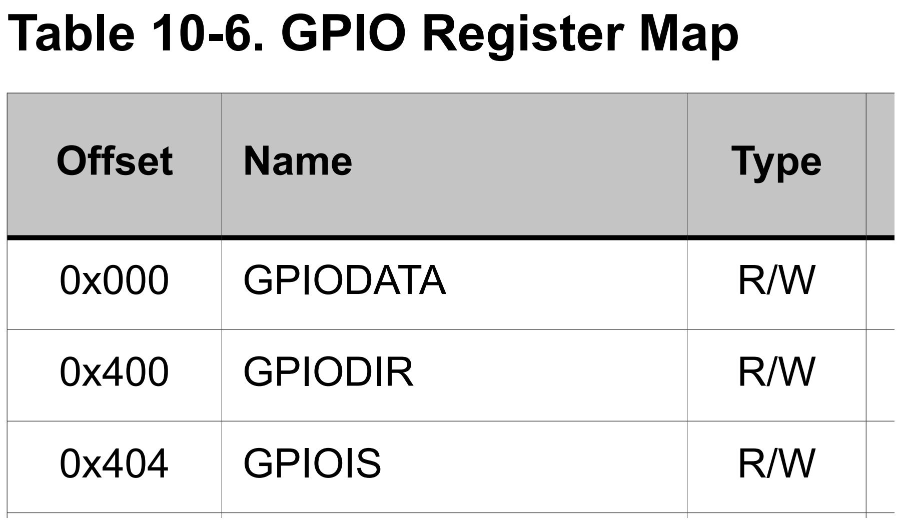


Table 10-6. GPIO Register Map

Offset	Name	Type
0x000	GPIODATA	R/W
0x400	GPIODIR	R/W
0x404	GPIOIS	R/W

notice nothing in between

all writes to 0x4000.4000--0x4000.43FF are `AND`'d with bits `[9:2]` of address before written to `DATA`

Q: if datasheet says DATA for PA is GPIO 0x4000.4000 why use 0x4000.43FC

```
                            1111111100
```

e.g.

```
LDR R1,=#0x4000.43FC
MOV R0, #0xAA
STR R0, [R1]
```

```
mem(0x4000.4000) = 0b10101010 & 0b11111111
```

all writes to 0x4000.4000--0x4000.43FF
are `AND`’d with bits `[9:2]` of address before written to `DATA`

take-away
(preferred method)

↓

write/read all pins of port:
BASE + 3FC

↙

GPIO Port A (APB): 0x4000.4000
~~GPIO Port A (AHB): 0x4005.8000~~
GPIO Port B (APB): 0x4000.5000
~~GPIO Port B (AHB): 0x4005.9000~~
GPIO Port C (APB): 0x4000.6000
~~GPIO Port C (AHB): 0x4005.A000~~
GPIO Port D (APB): 0x4000.7000
~~GPIO Port D (AHB): 0x4005.B000~~
GPIO Port E (APB): 0x4002.4000
~~GPIO Port E (AHB): 0x4005.C000~~
GPIO Port F (APB): 0x4002.5000

note: use legacy registers and APB addresses
(code will work on simulator)

active low:

thing is active (on) when low (0) is given by uC

active high:

thing is active (on) when high (1) is given by uC

conditionals: if true-then statements

```
if(x == y)
   {
     x++;
   }
```

this could work but...

```
  mov r0,#0  ;x
  mov r1,#0  ;y
  cmp r0,r1  ;x ?= y
  BNE noadd
  ADD r0,#1  ;x==y
noadd        ;x!=y
  ...
```

pipeline flush is:

```
  mov r0,#0  ;x
  mov r1,#0  ;y
  cmp r0,r1  ;x ?= y
  BNE noadd
  ADD r0,#1  ;if x==y
noadd
  ...
```

conditionals: if true-then statements

```
if(x == y)
  {
    x++;
  }
```

$\longrightarrow$

```
mov r0,#0 ;x
mov r1,#0 ;y
cmp r0,r1 ;x ?= y
IT EQ
ADDEQ r0,#1 ;if x==y
```

execute
if cmp is true

syntax:

condition codes
(Table 3.2,V1.p81; e.g. NE,LT,GE)

```
   CMP ...
   IT COND
OP{COND} ...
```

conditionals: if true-then-else statements

```
if(x == y)
   {
     x++
   }
else
   {
     y++
   }
```

```
mov r0,#0 ;x
mov r1,#0 ;y
cmp r0,r1 ;x ?= y
ITE EQ
ADDEQ r0,#1 ;if x==y
ADDNE r1,#1 ;if x!=y
```

syntax:

```
    ITE COND
 OP{COND} ...
OP{~COND} ...
```

as many as you want

syntax:

`IT{x{y{z}}} COND`

`{x{y{z}}}` =
T: if cond. is true
E: else

for each T and I
must have OP

first is always:

```
        OP{COND}  ...
```

if `x` is T:

```
        OP{COND}  ...
```

if `x` is E:

```
       OP{~COND}  ...
```

conditionals: if true-then-then-else statements

```
if(x == y)
  {
    x++
    y--
  }
else
  {
    y++
  }
```

```
mov r0,#0 ;x
mov r1,#0 ;y
cmp r0,r1 ;x ?= y
ITTE EQ
ADDEQ r0,#1 ;if x==y
SUBEQ r1,#1 ;if x==y
ADDNE r1,#1 ;if x!=y
```

syntax:

```
   ITTE COND
 OP{COND} ...
 OP{COND} ...
OP{~COND} ...
```

example: PD.0 = ~PD.1

output input

```
  LDR R0,=PD0_DATA_B
  LDR R1,=PD1_DATA_B
loop
  LDR R2,[R0] ;get current output (PD0)
  LDR R3,[R1] ;get current input (PD1)
  cmp R2,R3   ;R2?=R3
  ITT EQ
  MVNEQ R2,R2   ;if R2==R3
  STREQ R2,[R0] ;if R2==R3
  b loop
```

homework one, problem five:

PC-relative addressing

```
Start
  LDR R0,=0x123123
  NOP
  NOP
  B Start
```

R0 = [PC+4]

(addr w/r/t PC)

```
0x0134 Start
0x0134 4801 LDR R0,[pc,#4]
0x0136 BF00 NOP
0x0138 BF00 NOP
0x013A E7FB B Start
0x013C 3123 DCW 0x3123
0x013E 0012 DCW 0x0012
0x0140 0000 ...
```

PC-relative addressing

R0 = [PC+4]
(addr w/r/t PC)

when executed

b/c pipeline

PC is here

+4

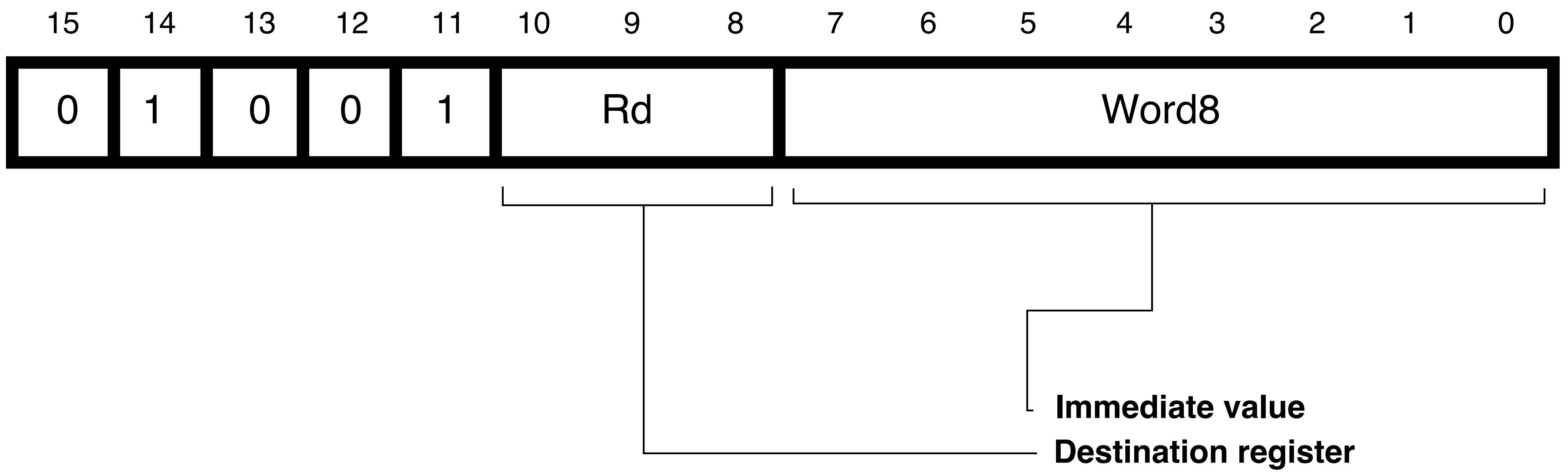
0x138+0x4
=0x13C $\longrightarrow$ gets loaded

```
0x0134 Start
0x0134 4801 LDR R0,[pc,#4]
0x0136 BF00 NOP
0x0138 BF00 NOP
0x013A E7FB B Start
0x013C 3123 DCW 0x3123
0x013E 0012 DCW 0x0012
0x0140 0000 ...
```

ooo...look what assembler has done for us
(word at 0x13C = 0x123123)

PC-relative LDR

5-16, Thumb Instruction Manual:



load addr. PC+Word << 2 ⟵ left shift two

↑

will be +4 of instruction addr

(zero-th bit forced to zero for word alignment) verify:

```
0x0134 4801 LDR R0,[pc,#4]
0x013C 3123 DCW 0x3123
0x013E 0012 DCW 0x0012
```

0b00000001<<2

= 0b0000000100

=0x4

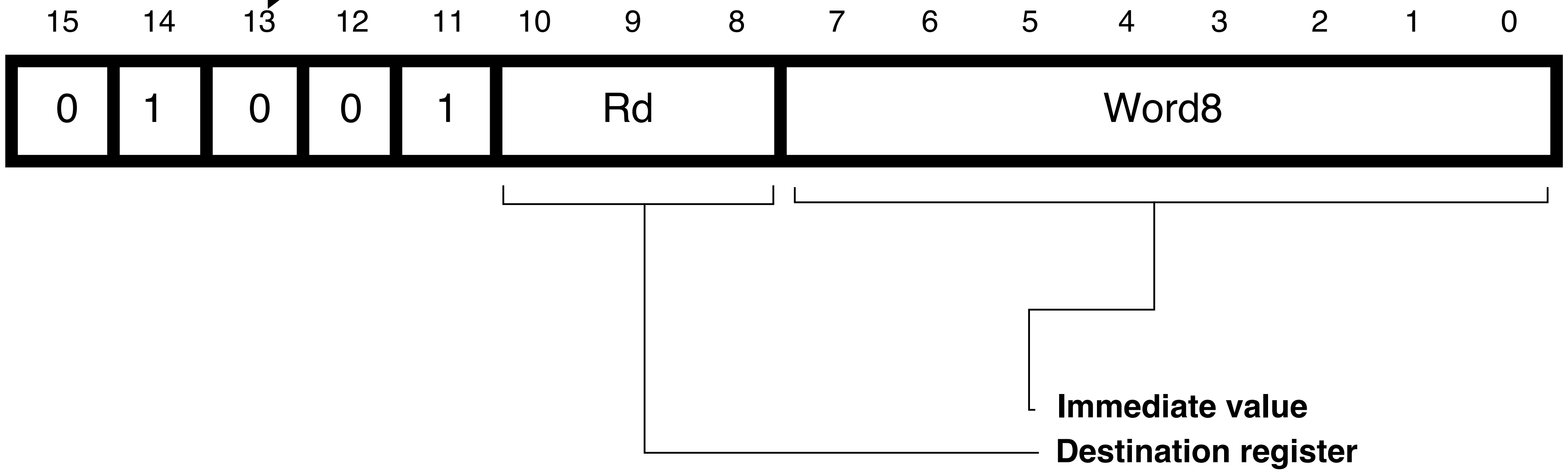
0x4801 = 0b0$\underline{100}$1$\underline{000}$$\overline{00000001}$

PC+4+4

=0x134+4+4

=0x13C

these check out:


15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
0
1
0
0
1
Rd
Word8
Immediate value
Destination register