Introduction to Microcontrollers

中山 大 学

数据科学与计算机学院

郭雪梅

Tel:39943108

Email:guoxuem@mail.sysu.edu.cn

URL1：http://human-robot.sysu.edu.cn/course

Introduction to Embedded Systems

Lecture 2: I/O, Logic/Shift Operations, Addressing modes, Memory Operations, Subroutines, Introduction to C

Read Chapters 3 & 4 of book

Agenda

- Recap
  - Embedded systems
  - Product life cycle
  - ARM programming

- Outline
  - Input/output
  - Logical/shift operations
  - Addressing modes, memory operations
  - Stack and subroutines
  - Introduction to C
    - Structure of a C program
    - Variables, expressions and assignments

Parallel I/O ports

Learning Objectives:

Know what is a parallel port

Know how a pin can be either input or output as specified by the direction register

Know the steps required to initialize a parallel port

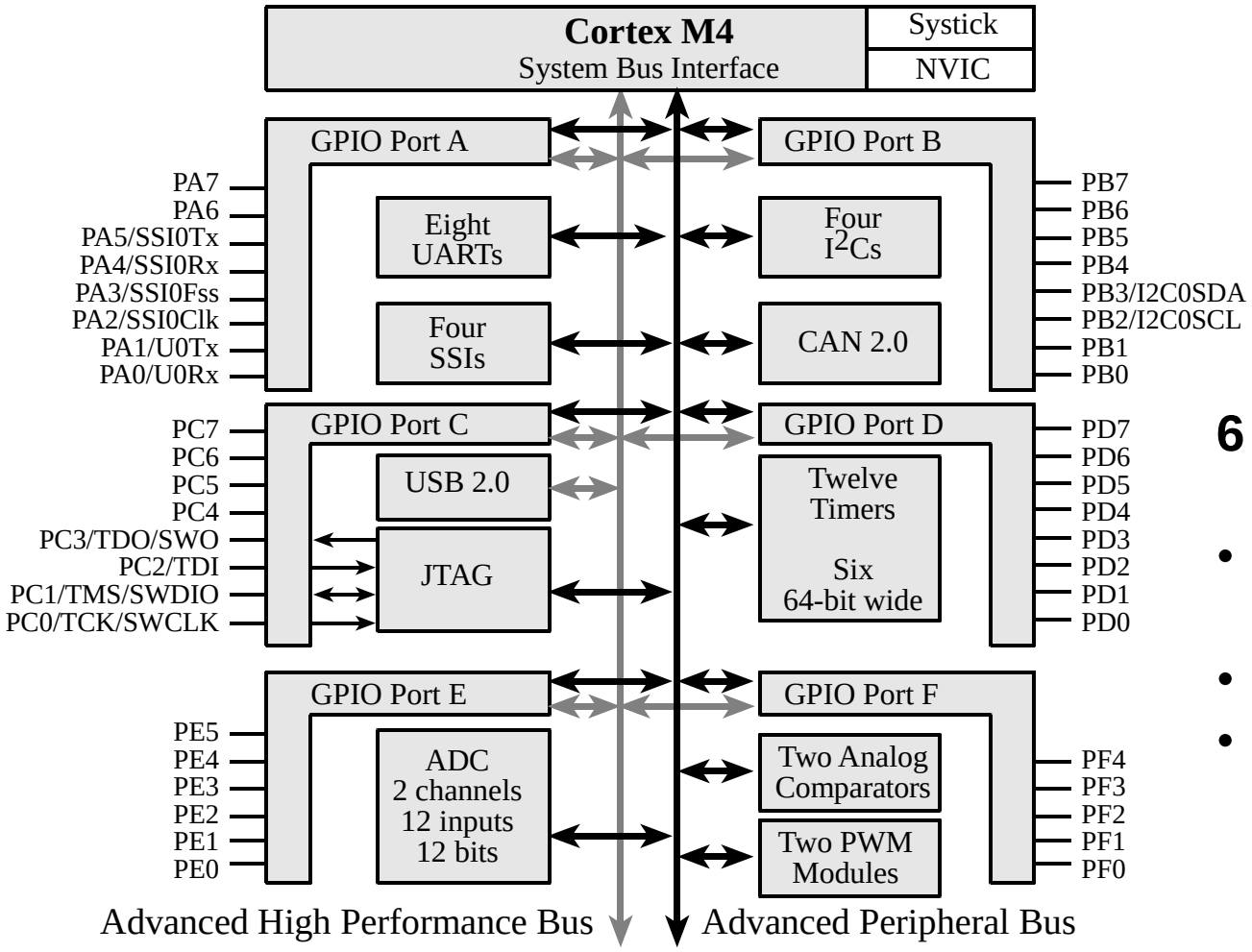
Know how to access I/O registers in a friendly manner

Know how to read data from an input port

Know how to write data to an output port

Know how to use the logic analyzer in the simulator

Input/Output: TM4C123



6 General-Purpose I/O (GPIO) ports:

- **Four 8-bit ports (A, B, C, D)**
- **One 6-bit port (E)**
- **One 5-bit port (F)**

TM4C123 I/O Pins

❿ I/O Pin Characteristics

Set AFSEL to 0

- Can be employed as an n-bit *parallel* interface
- Pins also provide *alternative functions:*

❿ UART	Universal asynchronous receiver/transmitter
❿ SSI	Synchronous serial interface
❿ I^2C	Inter-integrated circuit
❿ Timer compare	Periodic interrupts, input capture, and output
❿ PWM	Pulse width modulation
❿ ADC	Analog to digital converter, measure analog signals
❿ Analog Comparator	Compare two analog signals
❿ QEI	Quadrature encoder interface
❿ USB	Universal serial bus
❿ Ethernet	High speed network
❿ CAN	Controller area network

Set AFSEL to 1

TM4C123 LaunchPad I/O Pins

IO	Ain	0	1	2	3	4	5	6	7	8	9	14
PA2		Port		SSI0Clk								
PA3		Port		SSI0Fss								
PA4		Port		SSI0Rx								
PA5		Port		SSI0Tx								
PA6		Port			I_2C1SCL		M1PWM2					
PA7		Port			I_2C1SDA		M1PWM3					
PB0		Port	U1Rx						T2CCP0			
PB1		Port	U1Tx						T2CCP1			
PB2		Port			I_2C0SCL				T3CCP0			
PB3		Port			I_2C0SDA				T3CCP1			
PB4	Ain10	Port		SSI2Clk		M0PWM2			T1CCP0	CAN0Rx		
PB5	Ain11	Port		SSI2Fss		M0PWM3			T1CCP1	CAN0Tx		
PB6		Port		SSI2Rx		M0PWM0			T0CCP0			
PB7		Port		SSI2Tx		M0PWM1			T0CCP1			
PC4	C1-	Port	U4Rx	U1Rx		M0PWM6		IDX1	WT0CCP0	U1RTS		
PC5	C1+	Port	U4Tx	U1Tx		M0PWM7		PhA1	WT0CCP1	U1CTS		
PC6	C0+	Port	U3Rx					PhB1	WT1CCP0	USB0epen		
PC7	C0-	Port	U3Tx						WT1CCP1	USB0pflt		
PD0	Ain7	Port	SSI3Clk	SSI1Clk	I_2C3SCL	M0PWM6	M1PWM0		WT2CCP0			
PD1	Ain6	Port	SSI3Fss	SSI1Fss	I_2C3SDA	M0PWM7	M1PWM1		WT2CCP1			
PD2	Ain5	Port	SSI3Rx	SSI1Rx		M0Fault0			WT3CCP0	USB0epen		
PD3	Ain4	Port	SSI3Tx	SSI1Tx				IDX0	WT3CCP1	USB0pflt		
PD6		Port	U2Rx			M0Fault0		PhA0	WT5CCP0			
PD7		Port	U2Tx					PhB0	WT5CCP1	NMI		
PE0	Ain3	Port	U7Rx									
PE1	Ain2	Port	U7Tx									
PE2	Ain1	Port										
PE3	Ain0	Port										
PE4	Ain9	Port	U5Rx		I_2C2SCL	M0PWM4	M1PWM2			CAN0Rx		
PE5	Ain8	Port	U5Tx		I_2C2SDA	M0PWM5	M1PWM3			CAN0Tx		
PF0		Port	U1RTS	SSI1Rx	CAN0Rx		M1PWM4	PhA0	T0CCP0	NMI	C0o	
PF1		Port	U1CTS	SSI1Tx			M1PWM5	PhB0	T0CCP1		C1o	TRD1
PF2		Port		SSI1Clk		M0Fault0	M1PWM6		T1CCP0			TRD0
PF3		Port		SSI1Fss	CAN0Tx		M1PWM7		T1CCP1			TRCLK
PF4		Port					M1Fault0	IDX0	T2CCP0	USB0epen		

TM4C1294 I/O registers

Address	7	6	5	4	3	2	1	0	Name
$400F.E608			GPIOF	GPIOE	GPIOD	GPIOC	GPIOB	GPIOA	SYSCTL_RCGCGPIO_R
$400F.EA08			GPIOF	GPIOE	GPIOD	GPIOC	GPIOB	GPIOA	SYSCTL_PRGPIO_R
$4005.8000									PORTA base address
$4005.9000									PORTB base address
$4005.A000									PORTC base address
$4005.B000									PORTD base address
$4005.C000									PORTE base address
$4005.D000									PORTF base address
base+$3FC	DATA	DATA	DATA	DATA	DATA	DATA	DATA	DATA	GPIO_PORTx_DATA_R
base+$400	DIR	DIR	DIR	DIR	DIR	DIR	DIR	DIR	GPIO_PORTx_DIR_R
base+$420	AFSEL	AFSEL	AFSEL	AFSEL	AFSEL	AFSEL	AFSEL	AFSEL	GPIO_PORTx_AFSEL_R
base+$510	PUE	PUE	PUE	PUE	PUE	PUE	PUE	PUE	GPIO_PORTx_PUR_R
base+$51C	DEN	DEN	DEN	DEN	DEN	DEN	DEN	DEN	GPIO_PORTx_DEN_R
base+$524	CR	CR	CR	CR	CR	CR	CR	CR	GPIO_PORTx_CR_R
base+$528	AMSEL	AMSEL	AMSEL	AMSEL	AMSEL	AMSEL	AMSEL	AMSEL	GPIO_PORTx_AMSEL_R

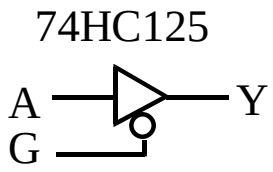
	31-28	27-24	23-20	19-16	15-12	11-8	7-4	3-0	
base+$52C	PMC7	PMC6	PMC5	PMC4	PMC3	PMC2	PMC1	PMC0	GPIO_PORTx_PCTL_R
base+$520	LOCK (32 bits)								GPIO_PORTx_LOCK_R

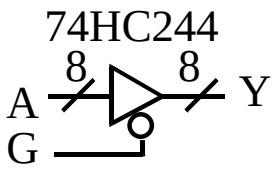
- **Four 8-bit ports (A, B, C, D)**
- **One 6-bit port (E)**
- **One 5-bit port (F)**

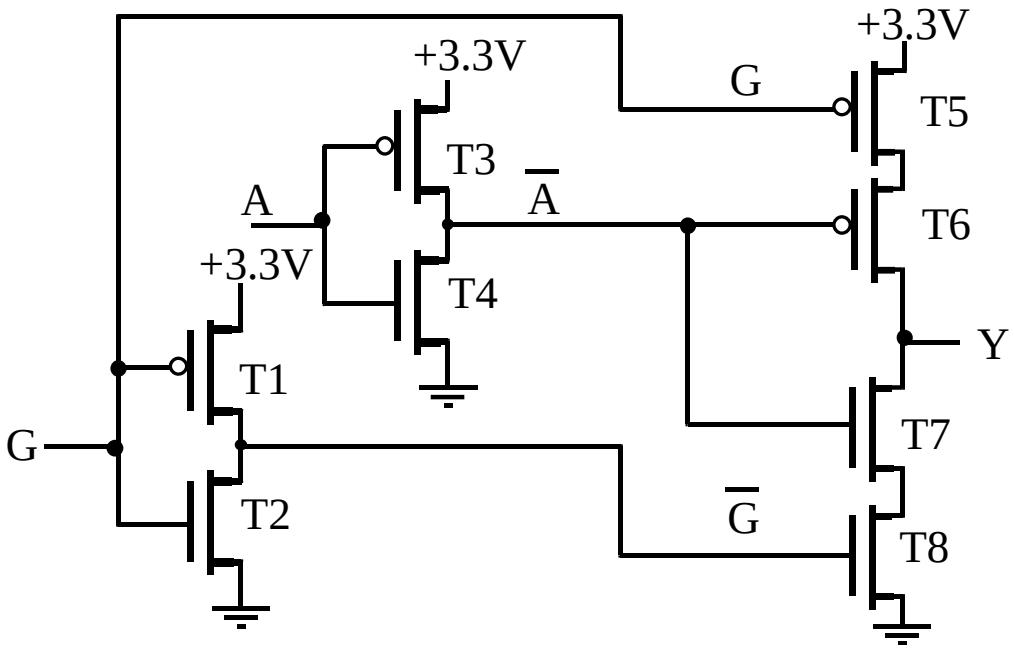
- **PA1-0 to COM port**
- **PC3-0 to debugger**
- **PD5-4 to USB device**

Bus Driver

Tristate Gate







Binary Information Implemented with MOS transistors

A binary bit can exist in one of two possible states. In positive logic, the presence of a voltage is called the ‘1’, true, or high state. The absence of a voltage is called the ‘0’, false, or low state.

The left side shows the condition with a true bit, and the right side shows a false.

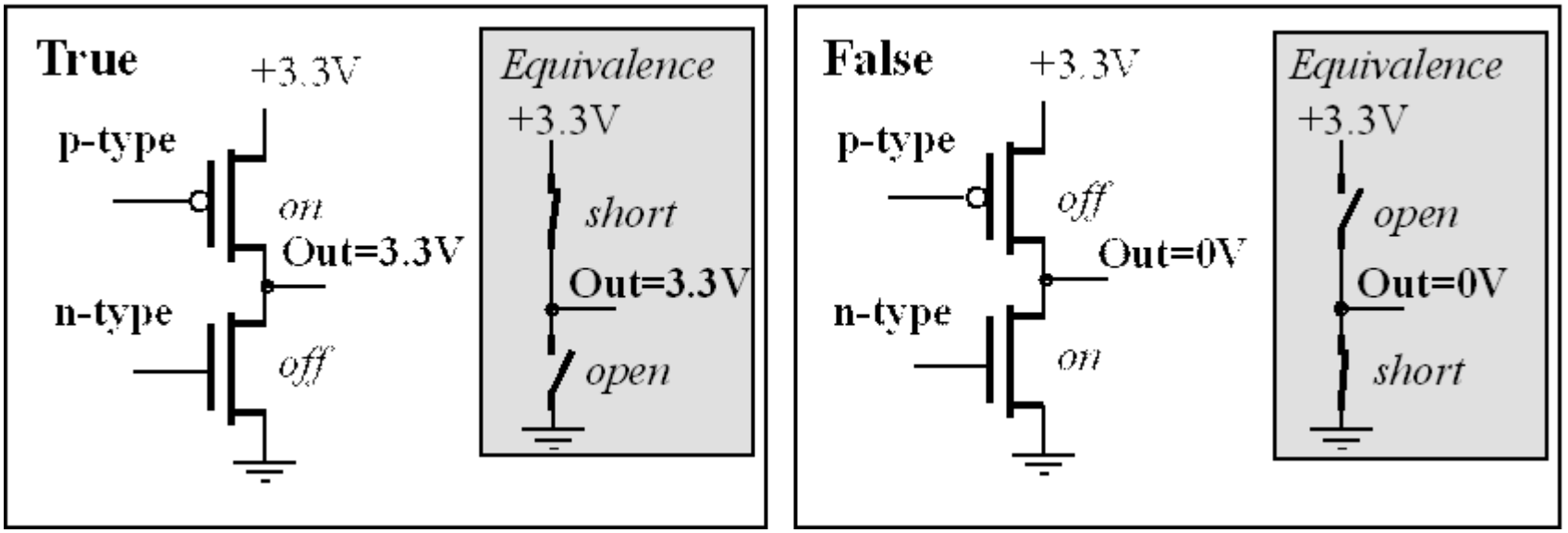
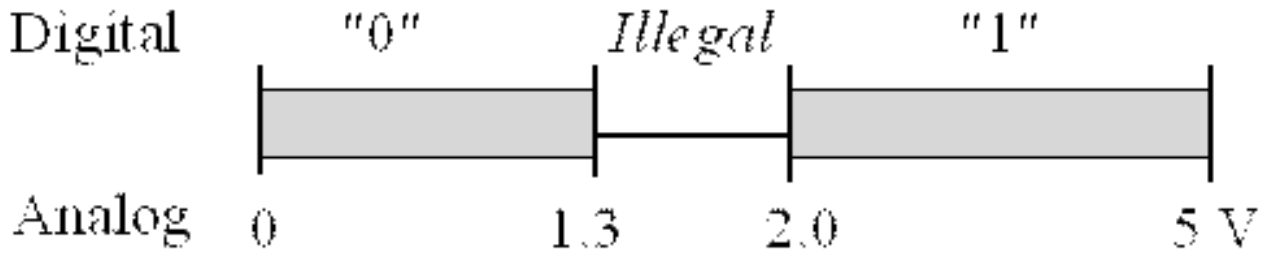


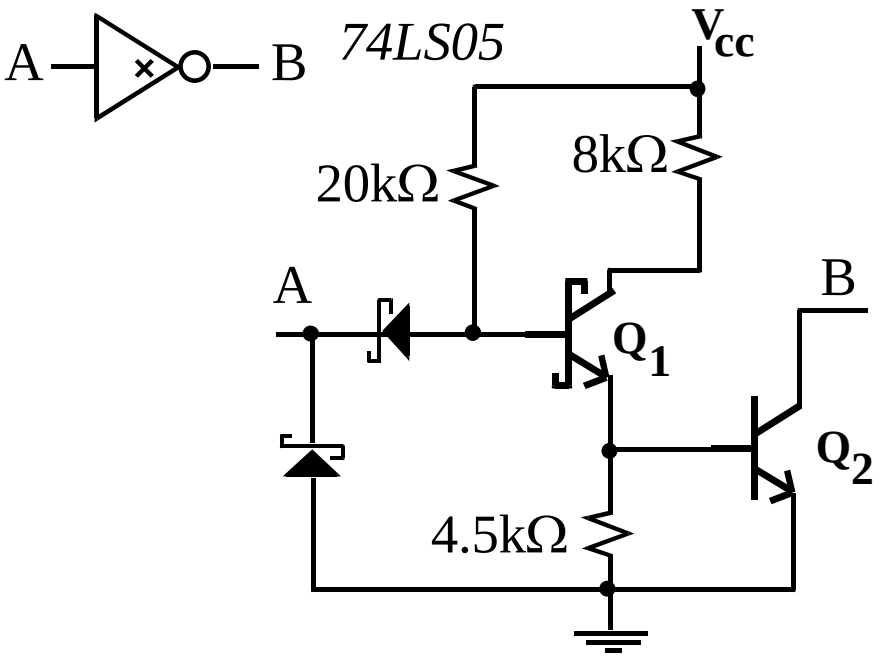
Figure A binary bit is true if a voltage is present and false if the voltage is 0.

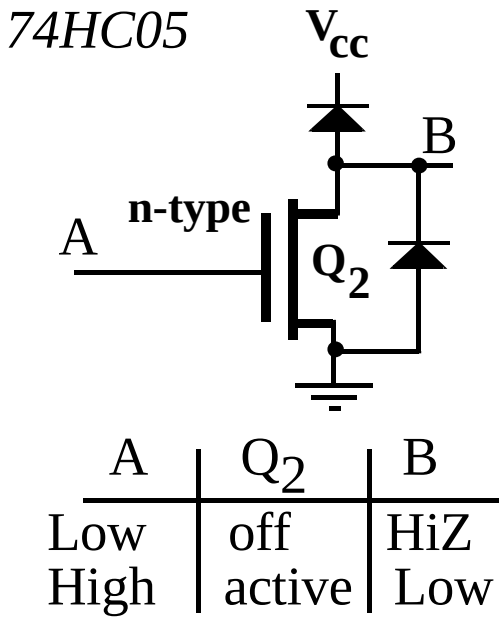
The maximum allowable voltage that the input will consider as low is called V_{IL}. For the TM4C123, V_{IL} is 1.3V. The minimum allowable voltage that the input will consider as high is called V_{IH}. For the TM4C123, V_{IH} is 2.0V.



The output pin also has a range of normal operating voltages. When the output is low, the maximum possible voltage that an output can be is called V_{OL}. For the TM4C123, V_{OL} is 0.4V. This means the output of the TM4C123 will be between 0 and 0.4V when the microcontroller is sending a low. When the output is high, the minimum possible voltage that an output can be is called V_{OH}. For the TM4C123, V_{OH} is 2.4V. This means the output of the TM4C123 will be between 2.4 and 3.3V when the microcontroller is sending a high.

Open Collector





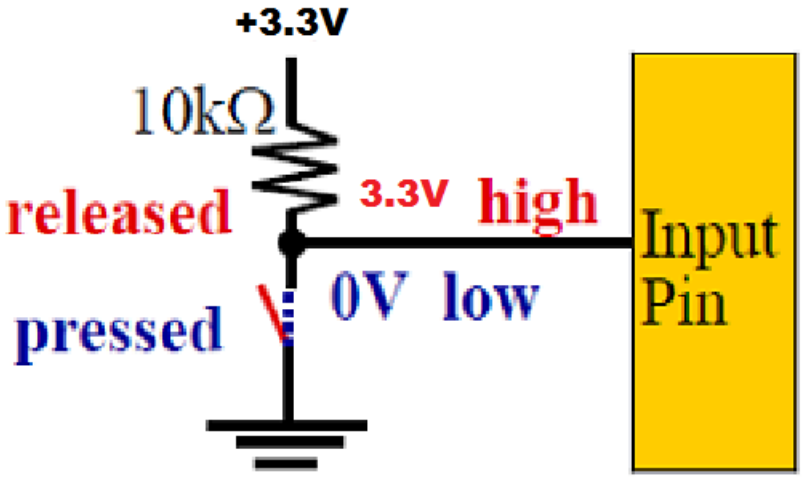
A	Q_2	B
Low	off	HiZ
High	active	Low

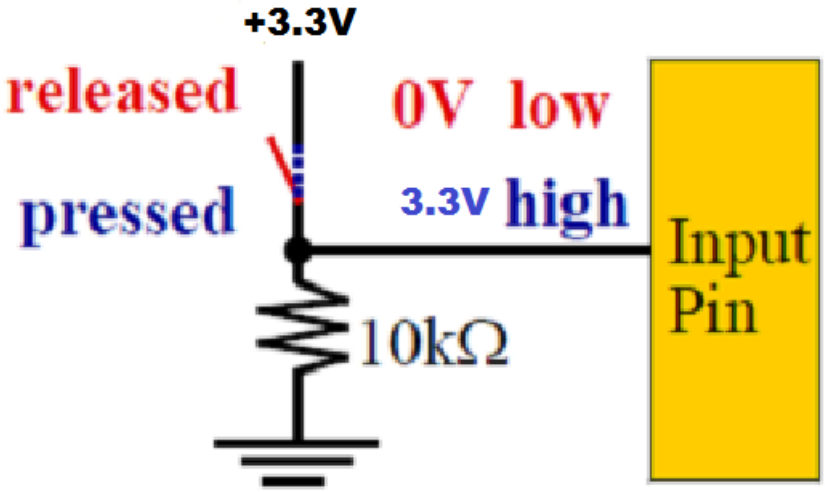
Used to control current to LED

Low allows current to flow to ground

HiZ prevents current from flowing

Switch Configuration



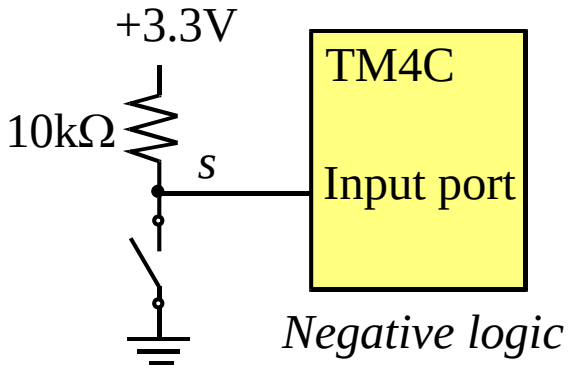


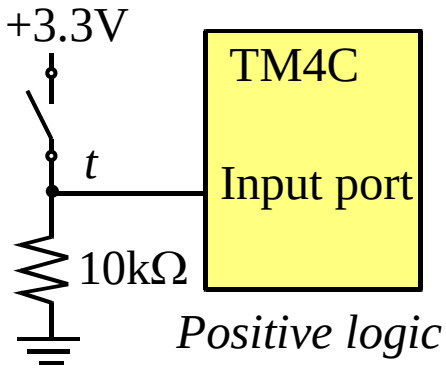
negative – pressed = '0'

positive – pressed = '1'

To convert into a digital signal, we can use a pull-down resistor to ground or a pull-up resistor to +3.3V as shown in Figure.

Switch Configuration





Negative Logic *s*

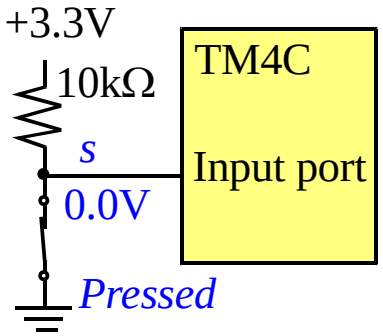
– **pressed, 0V, false**

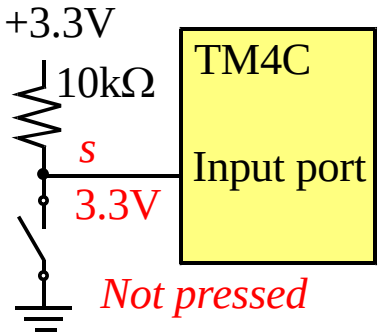
– **not pressed, 3.3V, true**

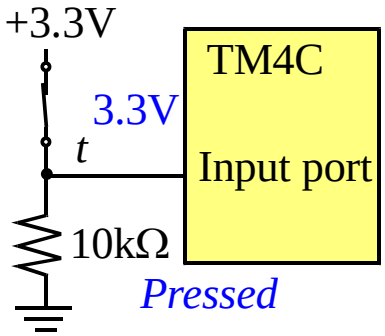
Positive Logic *t*

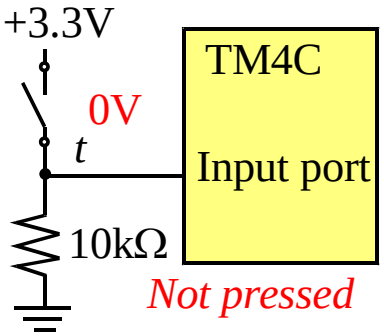
– **pressed, 3.3V, true**

– **not pressed, 0V, false**

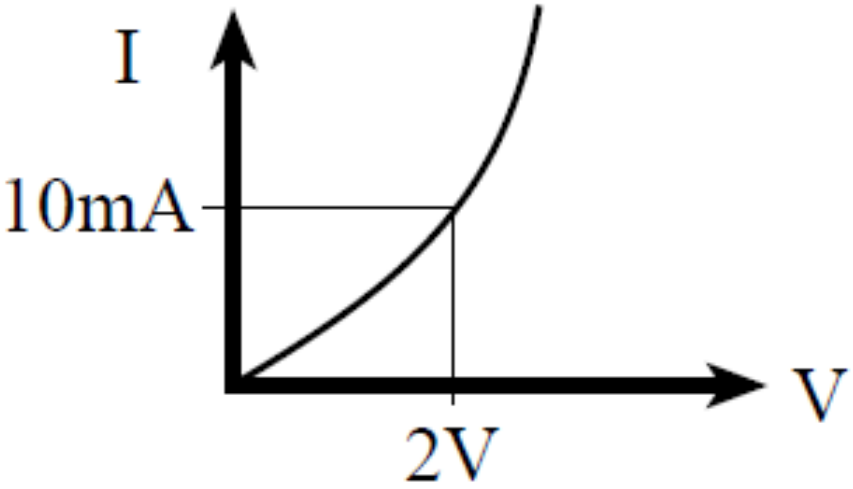








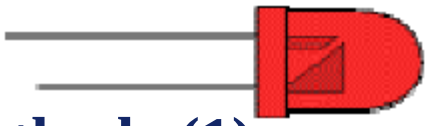
LED Interfacing



LED current v. voltage

Brightness = power = V*I

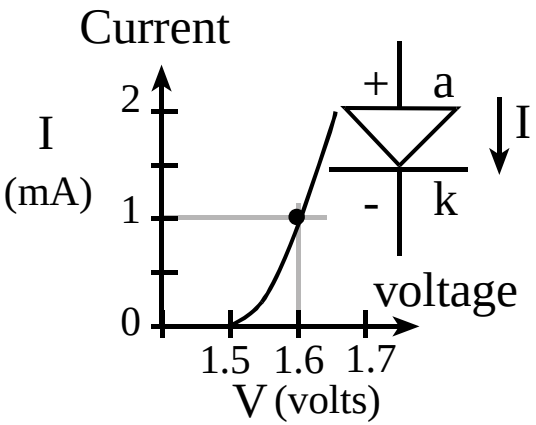
anode (+)



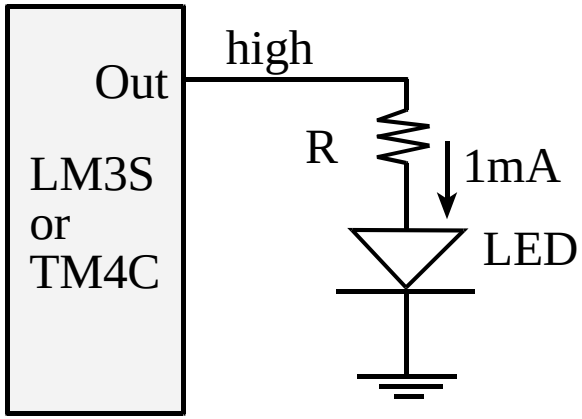
cathode (1)

"big voltage connects to big pin"

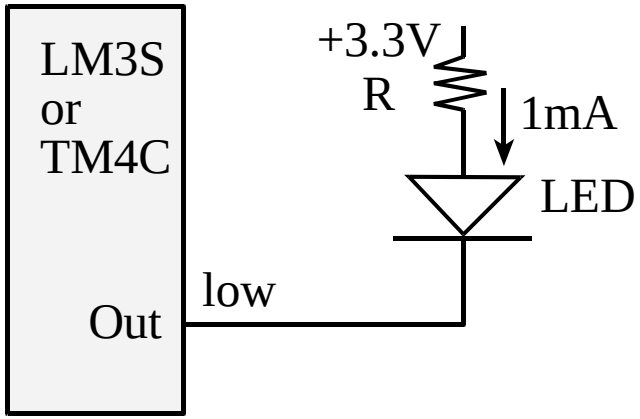
LED Configuration



(a) LED curve



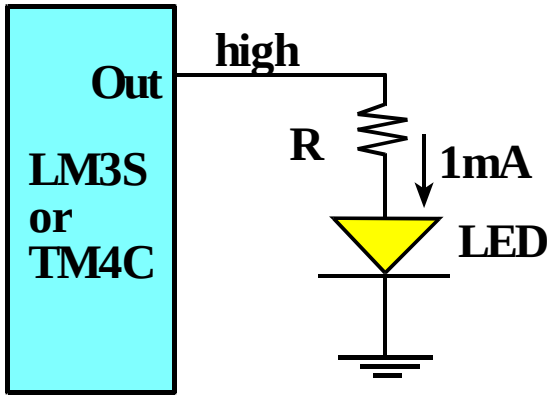
(b) Positive logic interface



(c) Negative logic interface

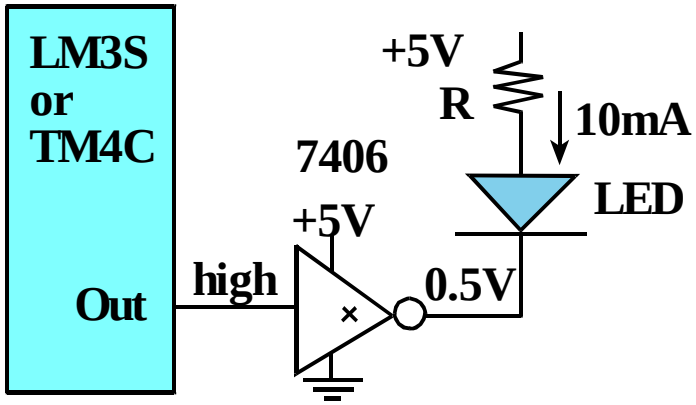
LED Interfacing

R = (3.3V – 1.6)/0.001
= 1.7 kOhm



LED current < 8 ma

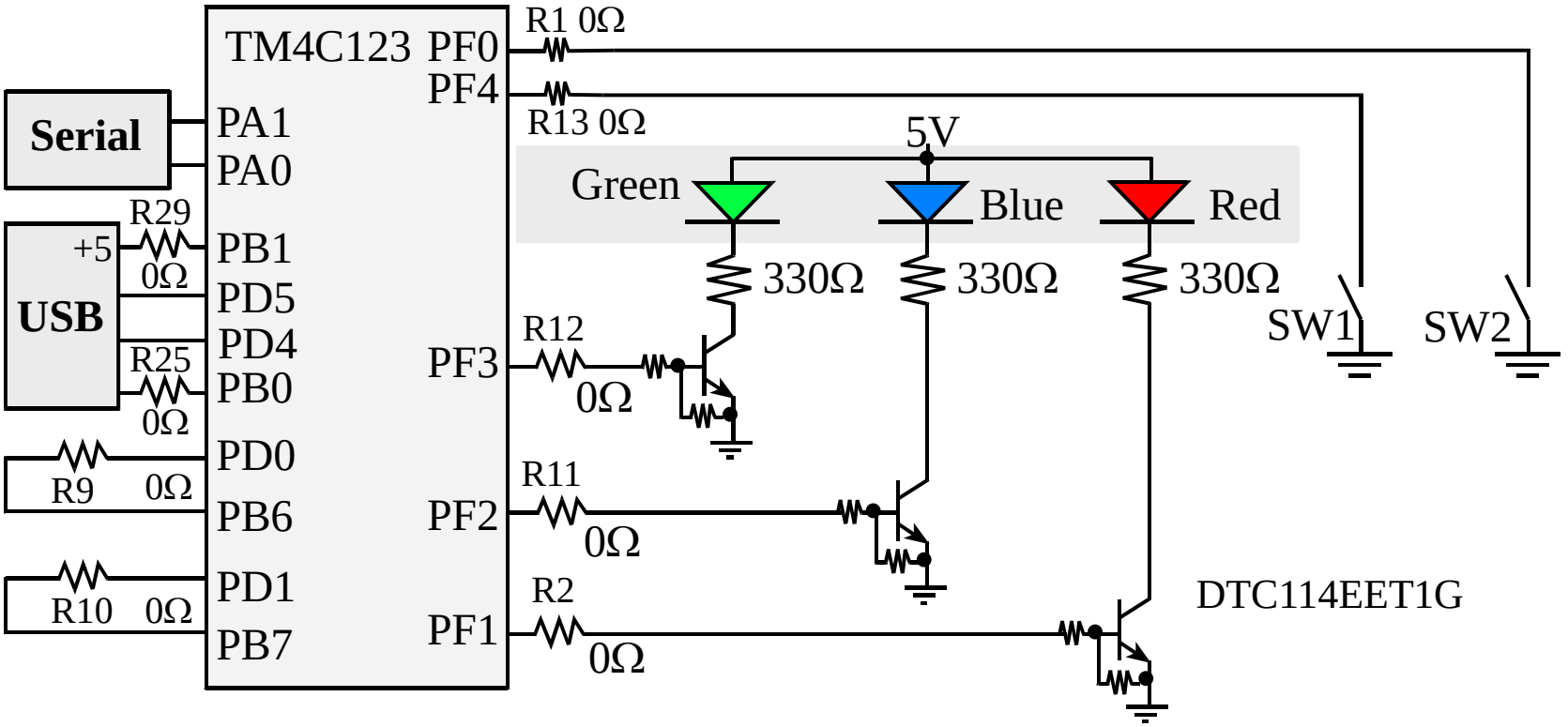
R = (5.0-2-0.5)/0.01
= 250 Ohm



LED current > 8 ma

LED may contain several diodes in series

LaunchPad Switches and LEDs



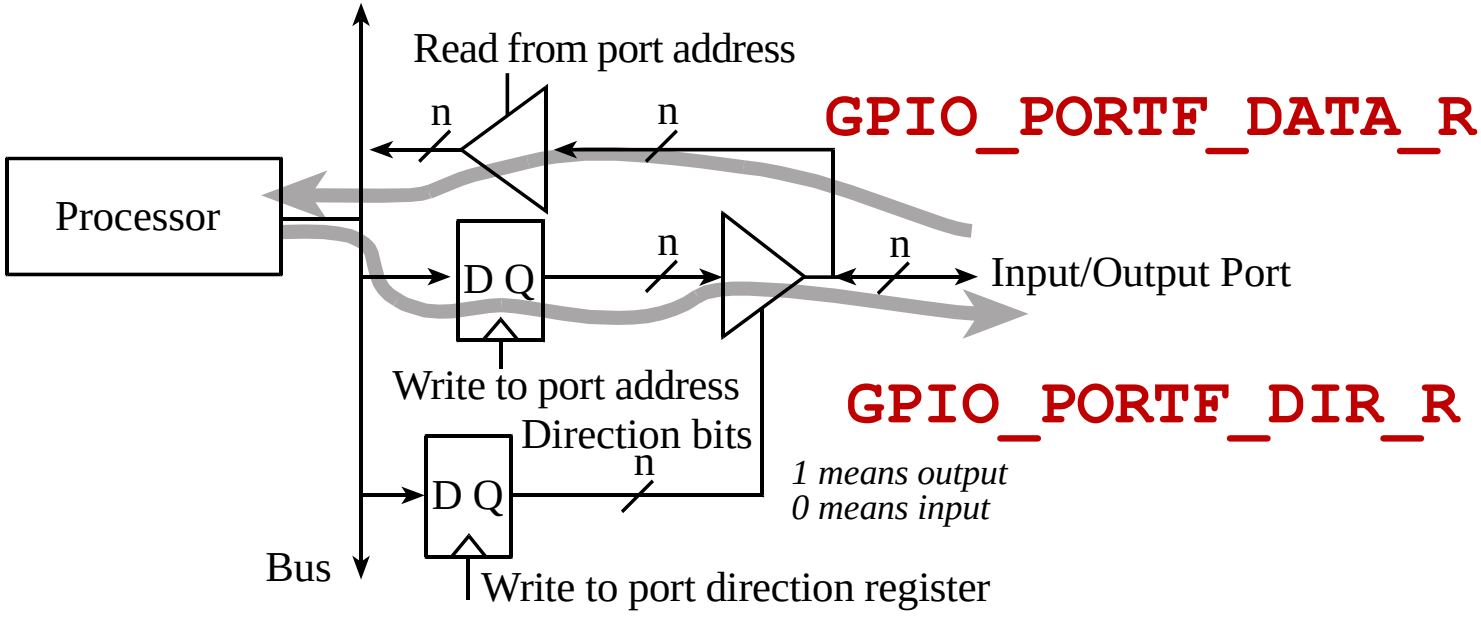
- The switches on the LaunchPad
  - Negative logic
  - Require internal pull-up (set bits in PUR)
- The PF3-1 LEDs are positive logic

Each pin has one configuration bit in the **GPIOAMSEL** register, **We set this bit to connect the port pin to the ADC or analog comparator**. we would set bits 1,0 in the **GPIO_PORTE_DEN_R** register (enable digital), For digital functions, each pin also has four bits in the **PCTL** register, which we set to specify the alternative function for that pin (0 means regular I/O port).

For example, if we wished to use UART7 on pins PE0 and PE1, we would set bits 1,0 in the **DEN** register (enable digital), clear bits 1,0 in the **AMSEL** register (disable analog), write a 0001,0001 to bits 7–0 in the **GPIO_PORTE_PCTL_R** register (enable UART7 functionality), and set bits 1,0 in the **GPIO_PORTE_AFSEL_R** register (enable alternate function).

Pin	Ain	0	1	2	3	4	5	6	7	8	9	14
PA0		*Port*	*U0Rx*							CAN1Rx		
PA1		*Port*	*U0Tx*							CAN1Tx		
PA2		Port		SSI0Clk								

I/O Ports and Control Registers



- The input/output direction of a bidirectional port is specified by its direction register.
- GPIO_PORTF_DIR_R , specify if corresponding pin is input or output:
  - 0 means input
  - 1 means output

I/O Ports and Control Registers

Address	7	6	5	4	3	2	1	0	Name
400F.E608	-	-	**GPIOF**	GPIOE	GPIOD	GPIOC	GPIOB	GPIOA	SYSCTL_RCGCGPIO_R
4002.53FC	-	-	-	DATA	DATA	DATA	DATA	DATA	GPIO_PORTF_DATA_R
4002.5400	-	-	-	DIR	DIR	DIR	DIR	DIR	GPIO_PORTF_DIR_R
4002.5420	-	-	-	SEL	SEL	SEL	SEL	SEL	GPIO_PORTF_AFSEL_R
4002.551C	-	-	-	DEN	DEN	DEN	DEN	DEN	GPIO_PORTF_DEN_R

- **Initialization (executed once at beginning)**
  1. **Turn on clock in `SYSCTL_RCGCGPIO_R(1 Enable)`**
  2. **Wait two bus cycles (two NOP instructions)**
  3. **Set *DIR* to 1 for output or 0 for input**
  4. **Clear *AFSEL* bits to 0 to select regular I/O**
  5. **Set *DEN* bits to 1 to enable data pins**
     Input/output from pin
  6. **Read/write `GPIO_PORTF_DATA_R`**

返回

I/O Programming and the Direction Register

On most embedded microcontrollers, the I/O ports are memory mapped. The software can access an input/output port simply by reading from or writing to the appropriate address. we make symbolic definitions for the I/O ports, for example we set direction register GPIO_PORTF_DIR_R, the alternate function register GPIO_PORTF_AFSEL_R, the bit in the enable register GPIO_PORTF_DEN_R.

To initialize an I/O port for general use we perform seven steps. First, we activate the clock for the port. Second, we unlock the port; unlocking is needed only for pins PC3-0, PD7, PF0 on the LM4F and TM4C. Third, we disable the analog function of the pin, because we will be using the pin for digital I/O. Fourth, we clear bits in the PCTL (Table 4.1) to select regular digital function. Fifth, we set its direction register. Sixth, we clear bits in the alternate function register, and lastly, we enable the digital port. We need to add a short delay between activating the clock and accessing the port registers. A DIR bit of 0 means input and 1 means output.

Initialization Ritual

- **Initialization (executed once at beginning)**
  1. **Turn on Port F clock in `SYSCTL_RCGCGPIO_R` Wait two bus cycles (two `NOP`)**
  2. **Unlock PF0 (PD7 also needs unlocking)**
  3. **Clear *AMSEL* to disable analog**
  4. **Clear *PCTL* to select GPIO**
  5. **Set *DIR* to 0 for input, 1 for output**
  6. **Clear *AFSEL* bits to 0 to select regular I/O**
  7. **Set *PUE* bits to 1 to enable internal pull-up**
  8. **Set *DEN* bits to 1 to enable data pins**
- **Input from switches, output to LED**
  10. **Read/write `GPIO_PORTF_DATA_R`**

Prog 4.1, show `PortF_Init` function in InputOutput_xxxasm

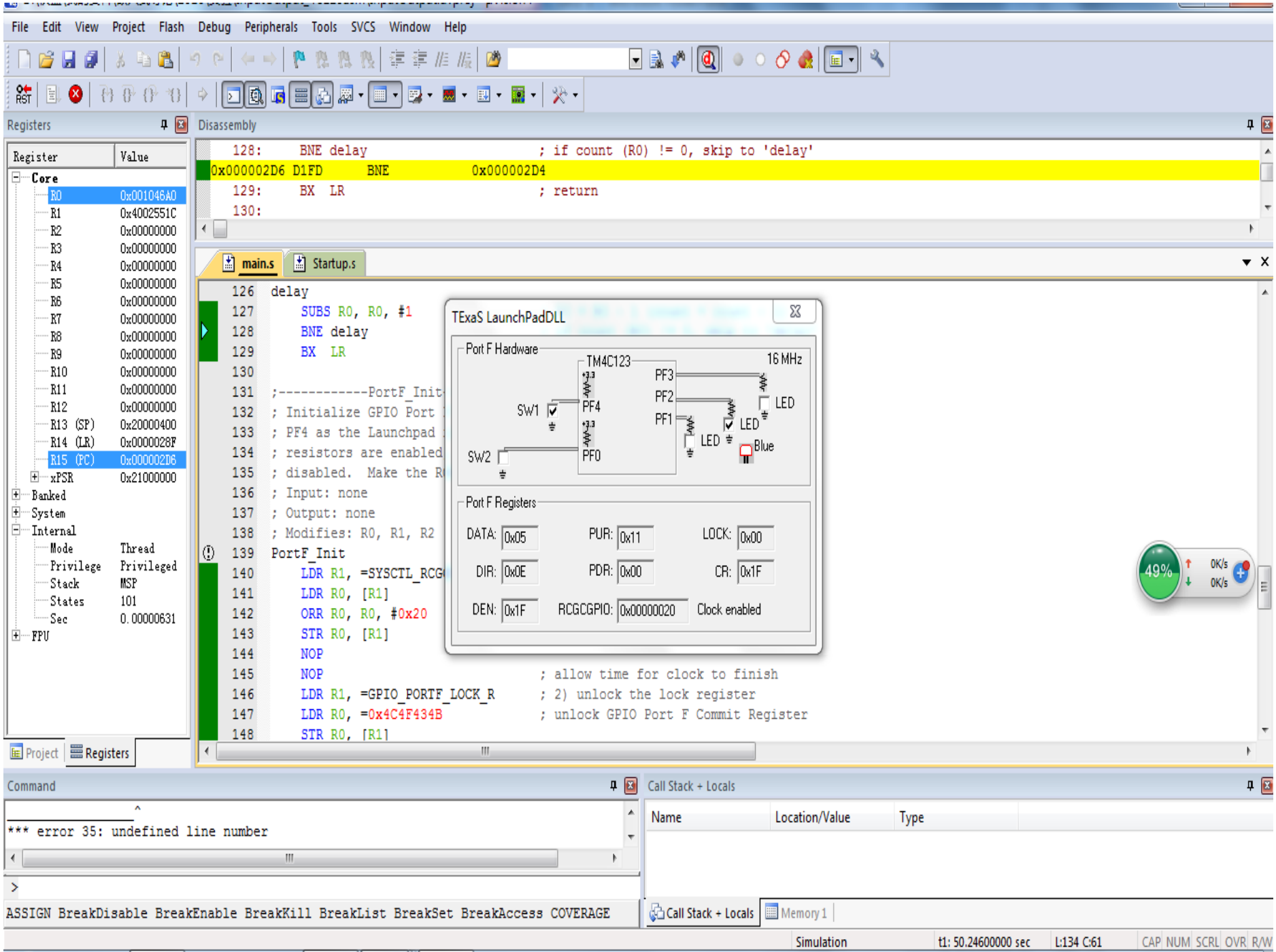
Initialization Port F

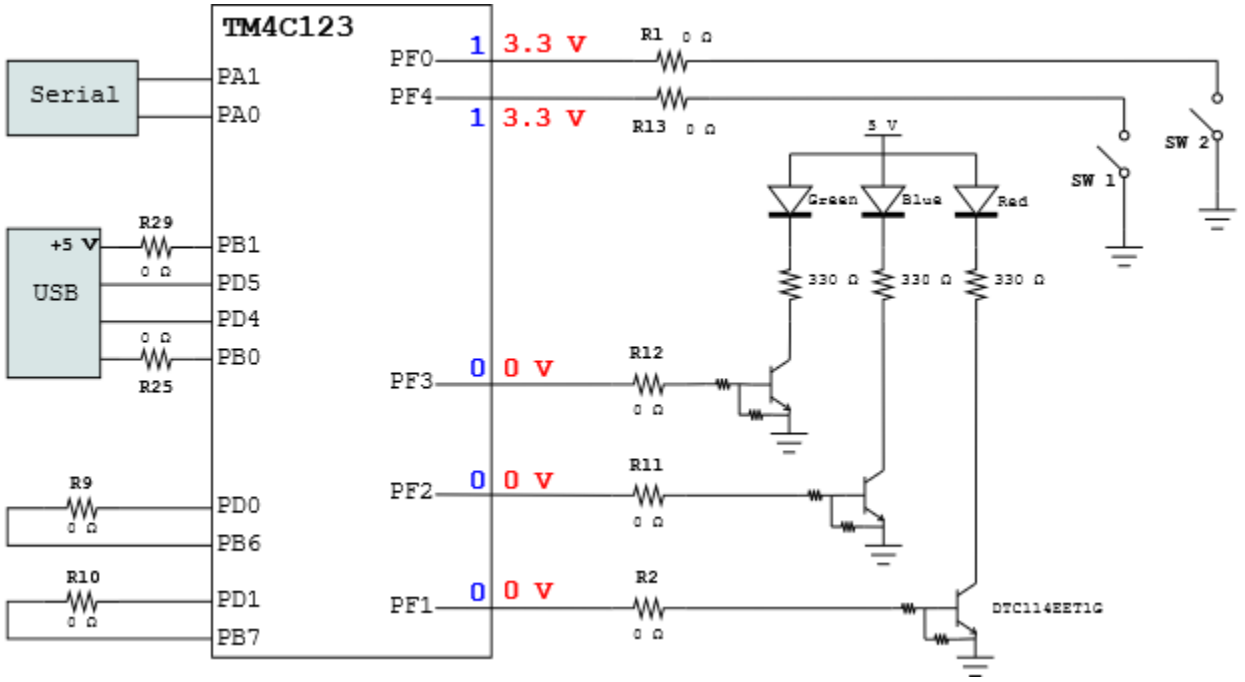
```
PortF_Init
   LDR R1, =SYSCTL_RCGCGPIO_R     ; 1) activate clock for Port F
                                  ;SYSCTL_RCGCGPIO_R  EQU   0x400FE608
   LDR R0, [R1]
   ORR R0, R0, #0x20              ; set bit 5 to turn on Port F clock
   STR R0, [R1]
   NOP
   NOP                            ; allow time for clock to finish
   LDR R1, =GPIO_PORTF_LOCK_R     ; 2) unlock the lock register
   LDR R0, =0x4C4F434B            ; unlock GPIO Port F Commit Register
   STR R0, [R1]
   LDR R1, =GPIO_PORTF_CR_R       ; allow changes to PF4-0
   MOV R0, #0x1F                  ; 1 means allow access
   STR R0, [R1]
   LDR R1, =GPIO_PORTF_AMSEL_R    ; 3) disable analog functionality
   MOV R0, #0                      ; 0 means analog is off
   STR R0, [R1]
```

Initialization Port F

```
LDR R1, =GPIO_PORTF_PCTL_R      ; 4) configure as GPIO
  MOV R0, #0x00000000            ; 0 means configure Port F as GPIO
  STR R0, [R1]
  LDR R1, =GPIO_PORTF_DIR_R      ; 5) set direction register
  MOV R0,#0x0E                   ; PF0 and PF7-4 input, PF3-1 output
  STR R0, [R1]
  LDR R1, =GPIO_PORTF_AFSEL_R    ; 6) regular port function
  MOV R0, #0                     ; 0 means disable alternate function
  STR R0, [R1]
  LDR R1, =GPIO_PORTF_PUR_R      ; pull-up resistors for PF4,PF0
  MOV R0, #0x11                  ; enable weak pull-up on PF0 and PF4
  STR R0, [R1]
  LDR R1, =GPIO_PORTF_DEN_R      ; 7) enable Port F digital port
  MOV R0, #0xFF                  ; 1 means enable digital I/O
  STR R0, [R1]
  BX  LR
```

Show program InputOutput_4C123asm


File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Registers
Register
Value
Core
R0 0x001046A0
R1 0x4002551C
R2 0x00000000
R3 0x00000000
R4 0x00000000
R5 0x00000000
R6 0x00000000
R7 0x00000000
R8 0x00000000
R9 0x00000000
R10 0x00000000
R11 0x00000000
R12 0x00000000
R13 (SP) 0x20000400
R14 (LR) 0x0000028F
R15 (PC) 0x000002D6
xPSR 0x21000000
Banked
System
Internal
Mode Thread
Privilege Privileged
Stack MSP
States 101
Sec 0.00000631
FPU
Project
Registers
Disassembly
128: BNE delay ; if count (R0) != 0, skip to 'delay'
0x000002D6 D1FD BNE 0x000002D4
129: BX LR ; return
130:
main.s
Startup.s
126 delay
127 SUBS R0, R0, #1
128 BNE delay
129 BX LR
130
131 ;------------PortF_Init
132 ; Initialize GPIO Port
133 ; PF4 as the Launchpad
134 ; resistors are enabled
135 ; disabled. Make the R
136 ; Input: none
137 ; Output: none
138 ; Modifies: R0, R1, R2
139 PortF_Init
140 LDR R1, =SYSCTL_RCG
141 LDR R0, [R1]
142 ORR R0, R0, #0x20
143 STR R0, [R1]
144 NOP
145 NOP ; allow time for clock to finish
146 LDR R1, =GPIO_PORTF_LOCK_R ; 2) unlock the lock register
147 LDR R0, =0x4C4F434B ; unlock GPIO Port F Commit Register
148 STR R0, [R1]
TExaS LaunchPadDLL
Port F Hardware
TM4C123
16 MHz
PF3
PF2
PF1
PF4
PF0
SW1
SW2
LED
LED
LED
Blue
Port F Registers
DATA: 0x05
PUR: 0x11
LOCK: 0x00
DIR: 0x0E
PDR: 0x00
CR: 0x1F
DEN: 0x1F
RCGCGPIO: 0x00000020
Clock enabled
49%
0K/s
0K/s
Command
*** error 35: undefined line number
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE
Call Stack + Locals
Name
Location/Value
Type
Call Stack + Locals
Memory 1
Simulation
t1: 50.24600000 sec
L:134 C:61
CAP NUM SCRL OVR R/W



	bitfield:	31	30	29	28	27 - 8	7	6	5	4	3	2	1	0	
0x400253FC										1	0	0	0	1	GPIO_PORTF_DATA_R
0x40025400										0	1	1	1	0	GPIO_PORTF_DIR_R
0x4002551C										1	1	1	1	1	GPIO_PORTF_DEN_R
0x40025510										1	0	0	0	1	GPIO_PORTF_PUR_R
0x40025420										0	0	0	0	0	GPIO_PORTF_AFSEL_R

Port Address (Table 4.3,4.4)

Port	Base address
PortA	0x40004000
PortB	0x40005000
PortC	0x40006000
PortD	0x40007000
PortE	0x40024000
PortF	0x40025000

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

Base Addresses for bit-specific addressing of ports A-F

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.

I/O Port Bit-Specific

- I/O Port bit-specific addressing is used to access port data register
  - Define address offset as $4*2^b$, where b is the selected bit position
  - 256 possible bit combinations (0-8)
  - Add offsets for each bit selected to base address for the port
  - Example: PF4 and PF0

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

Port F = 0x4005.D000

0x4005.D000+0x0004+0x0040

= 0x4005.D044

Provides friendly and atomic access to port pins

Logic Operations

A **Rn**	B **Operand2**	A&B **AND**	A|B **ORR**	A^B **EOR**	A&(~B) **BIC**	A|(~B) **ORN**
0	0	0	0	0	0	1
0	1	0	1	1	0	0
1	0	0	1	1	1	1
1	1	1	1	0	0	1

❑ Logic Instructions

```
AND{S} {Rd,} Rn, <op2>   ; Rd=Rn&op2
ORR{S} {Rd,} Rn, <op2>   ; Rd=Rn|op2
EOR{S} {Rd,} Rn, <op2>   ; Rd=Rn^op2
BIC{S} {Rd,} Rn, <op2>   ; Rd=Rn&(~op2)
ORN{S} {Rd,} Rn, <op2>   ; Rd=Rn|(~op2)
```

To set

The or operation to set bits 1 and 0 of a register.

The other six bits remain constant.

Friendly **software modifies just the bits that need to be.**

```
GPIO_PORTD_DIR_R |= 0x03; // PD1,PD0 outputs
```

Assembly:

```
LDR  R0,=GPIO_PORTD_DIR_R
LDR  R1,[R0]       ; read previous value
ORR  R1,R1,#0x03   ; set bits 0 and 1
STR  R1,[R0]       ; update
```

c_7	c_6	c_5	c_4	c_3	c_2	c_1	c_0	
0	0	0	0	0	0	1	1	0x03 constant
c_7	c_6	c_5	c_4	c_3	c_2	1	1	result of the ORR

value of R1

To toggle

The exclusive or operation can also be used to toggle bits.

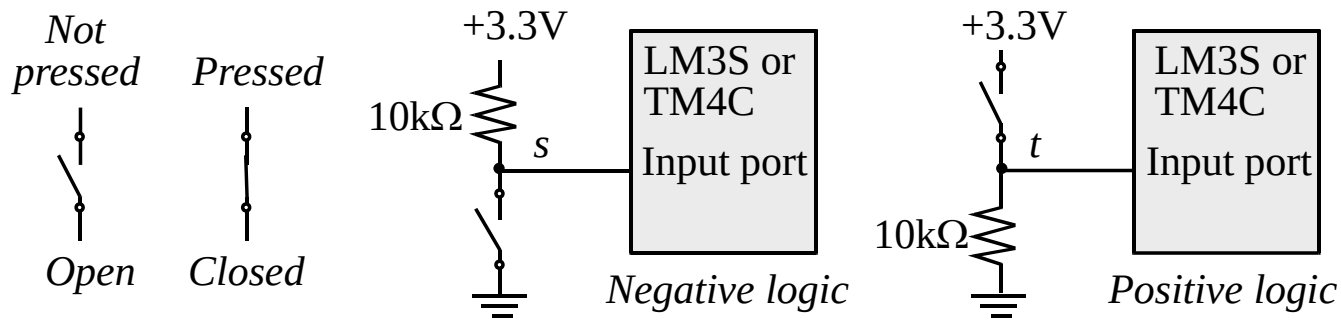
```
GPIO_PORTD_DATA_R ^= 0x80; /* toggle PD7 */
```

Assembly:

```
LDR  R0,=GPIO_PORTD_DATA_R
LDR  R1,[R0]       ; read port D
EOR  R1,R1,#0x80   ; toggle bit 7
STR  R1,[R0]       ; update
```

$b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0$	value of R1
$1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0$	0x80 constant
$\sim b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0$	result of the EOR

Switch Interfacing



The and operation to extract, or *mask*, individual bits:

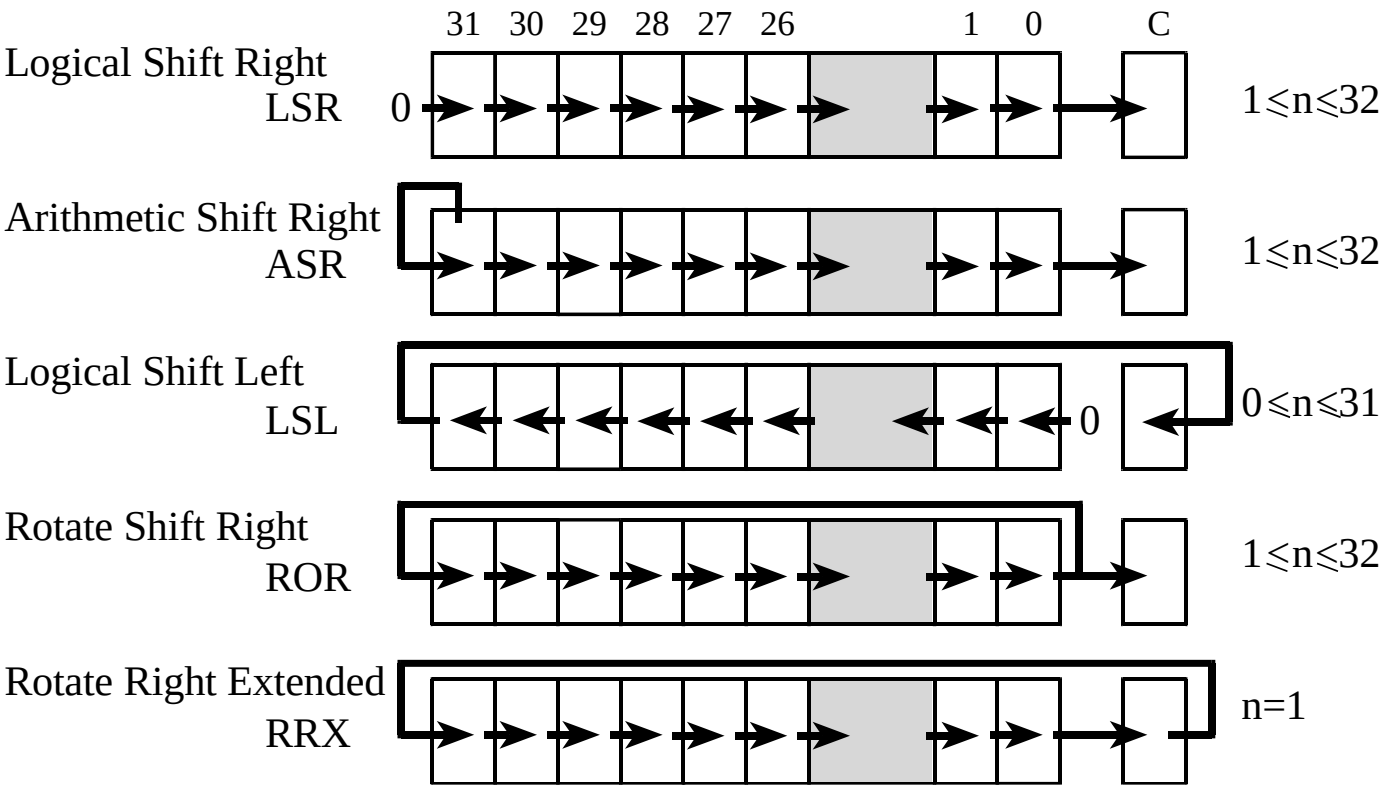
```
Pressed = GPIO_PORTA_DATA_R & 0x40;
                        //true if the PA6 switch pressed
```

Assembly:

```
LDR  R0,=GPIO_PORTA_DATA_R
LDR  R1,[R0]        ; read port A
AND  R1,R1,#0x40    ; clear all bits except bit 6
LDR  R0,=Pressed    ; update variable
STR  R1,[R0]        ; true if switch pressed
```

a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	
0	1	0	0	0	0	0	0	value of R1
0	a_6	0	0	0	0	0	0	0x40 constant
								result of the AND

Shift Operations



Use the `ASR` instruction when manipulating signed numbers,

and use the `LSR` instruction when shifting unsigned numbers

Shift Example

High and Low are unsigned 4-bit components, which will be combined into a single unsigned 8-bit Result.

```
Result = (High<<4)|Low;
```

Assembly:

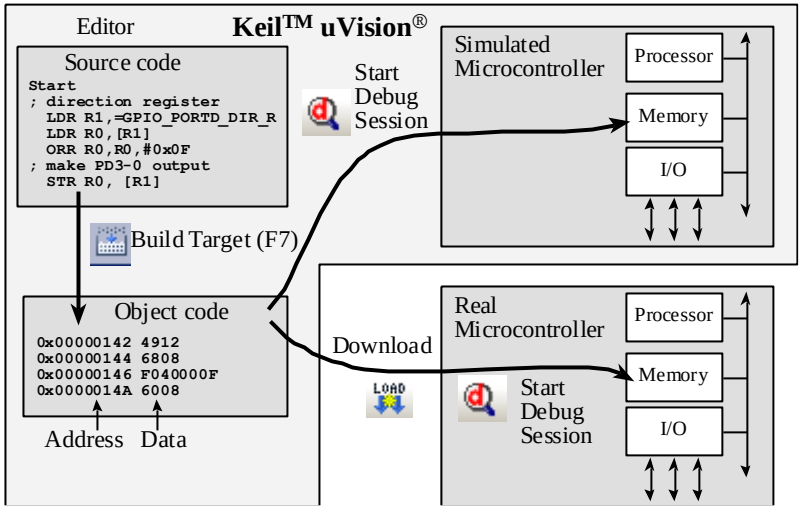
```
LDR   R0,=High
LDR   R1,[R0]          ; read value of High
LSL   R1,R1,#4         ; shift into position
LDR   R0,=Low
LDR   R2,[R0]          ; read value of Low
AND   R2,R2,#0x0F      ; clear all bits except low 4 bit
ORR   R1,R1,R2         ; combine the two parts
LDR   R0,=Result
STR   R1,[R0]          ; save the answer
```

$0\ \ 0\ \ 0\ \ 0\ \ h_3\ h_2\ h_1\ h_0$	value of High in R1
$h_3\ h_2\ h_1\ h_0\ 0\ \ 0\ \ 0\ \ 0$	after last LSL
$0\ \ 0\ \ 0\ \ 0\ \ l_3\ l_2\ l_1\ l_0$	value of Low in R2
$h_3\ h_2\ h_1\ h_0\ l_3\ l_2\ l_1\ l_0$	result of the ORR instruction

Process

⑩ Solution in NOTGate-asm.zip

Start	**Activate clock for port**
	Enable PD3, PD0 pins
	Make PD0 (switch) pin an input
	Make PD3 (LED) pin an output
Loop	**Read input from Switch (0 or 1)**
	not function LED = 8*(not Switch)
	Write output (8 or 0)
	branch Loop



❑ Build it, run it, test it with logic analyzer

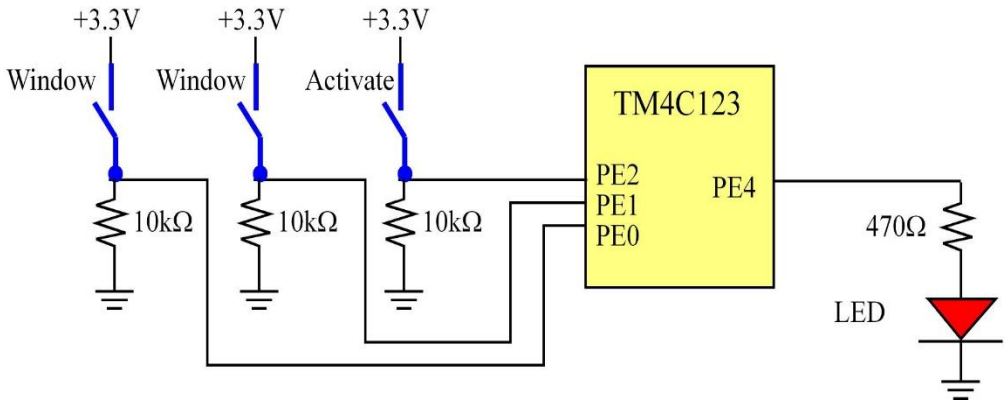
```
GPIO_PORTD_DATA_R  EQU 0x400073FC
GPIO_PORTD_DIR_R   EQU 0x40007400
GPIO_PORTD_LOCK_R  EQU 0x4C4F434B
GPIO_PORTD_AFSEL_R EQU 0x40007420
GPIO_PORTD_DEN_R   EQU 0x4000751C
SYSCTL_RCGCGPIO_R  EQU 0x400FE108
      AREA    |.text|, CODE, READONLY,
ALIGN=2
      THUMB
      EXPORT  Start
GPIO_Init
   ; 1) activate clock for Port D
   LDR R1, =SYSCTL_RCGCGPIO_R
   LDR R0, [R1]                ; R0 = [R1]
   ORR R0, R0, #0x08 ; R0 = R0|0x08
   STR R0, [R1]                ; [R1] = R0
   NOP
   NOP
   NOP
   NOP          ; allow time to finish activating
   ; 3) set direction register
   LDR R1, =GPIO_PORTD_DIR_R      ; R1 =
&GPIO_PORTD_DIR_R
   LDR R0, [R1]                ; R0 = [R1]
   ORR R0, R0, #0x08     ; R0 = R0|0x08 (make
                                PD3 output)
   BIC R0, R0, #0x01    ; R0 = R0 & NOT(0x01)
                               (make PD0 input)
   STR R0, [R1]                ; [R1] = R0
   ; 4) regular port function
   LDR R1, =GPIO_PORTD_AFSEL_R     ; R1 =
              &GPIO_PORTD_AFSEL_R
   LDR R0, [R1]                ; R0 = [R1]
   BIC R0, R0, #0x09           ; R0 = R0&~0x09
                  (disable alt funct on PD3,PD0)
   STR R0, [R1]              ; [R1] = R0
   ; 5) enable digital port
   LDR R1, =GPIO_PORTD_DEN_R      ; R1 =
                        &GPIO_PORTD_DEN_R
   LDR R0, [R1]                ; R0 = [R1]
   ORR R0, R0, #0x09      ; R0 = R0|0x09
               (enable digital I/O on PD3,PD0)
   STR R0, [R1]                ; [R1] = R0
   BX  LR
Start
   BL  GPIO_Init
   LDR R0, =GPIO_PORTD_DATA_R
loop
    LDR R1,[R0]
    AND          R1,#0x01        ; Isolate PD0
    EOR          R1,#0x01 ; NOT state of PD0
                              read into R1
    LSL R1,#3   ;SHIFT left negated state of
                 PD0 read into R1
    STR R1,[R0] ; Write to PortD DATA
              register to update LED on PD3
    B loop      ; unconditional branch to 'loop'

   ALIGN        ; make sure the end of this
                section is aligned
   END                      ; end of file
```

Process

⑩ **Solution in Lab1_EE319K_C.zip**

Start	**Activate clock for port**
	Enable PE0- PE4 pins
	Make PE0-PE2 (switch) pin an input
	Make PE4 (LED) pin an output
Loop	**Read input from Switch (0 or 1、3)**
	toggle output for alarm
	branch Loop

❑ **Build it, run it, test it with logic analyzer**




```
#include <stdint.h>
#include "inc/tm4c123gh6pm.h"
unsigned long arm,sensor;
void EnableInterrupts(void);
int main(void){ unsigned long volatile delay;
//TExaS_Init();// activate multimeter, 80 MHz
SYSCTL_RCGC2_R |= 0x10; // Port E clock
delay = SYSCTL_RCGC2_R;// wait 3-5 bus cycles
GPIO_PORTE_DIR_R |= 0x10;// PE4 output
GPIO_PORTE_DIR_R &= ~0x07;// PE2,1,0 input
GPIO_PORTE_AFSEL_R &= ~0x17; // not alternative
GPIO_PORTE_AMSEL_R &= ~0x17;// no analog
GPIO_PORTE_PCTL_R &= ~0x000F0FFF; // bits for PE4,PE2,PE1,PE0
GPIO_PORTE_DEN_R |= 0x17;// enable PE4,PE2,PE1,PE0
```

```
while(1){
      arm = GPIO_PORTE_DATA_R&0x04; // arm 0 if deactivated, 1 if
activated(PE2)
      sensor = GPIO_PORTE_DATA_R&0x03; // 1 means ok, 0 means
                                                  break in
if((arm==0x04)&&(sensor != 0x03)){
 GPIO_PORTE_DATA_R ^= 0x10; // toggle output for alarm
 delay=100;  // 100ms delay makes a 5Hz period
   }else{
GPIO_PORTE_DATA_R &= ~0x10; // LED off if deactivated
       }
  }
}
```

ARM Assembly Language

Assembly format

```
Label   Opcode   Operands        Comment
init    MOV      R0, #100     ; set table size
        BX       LR
```

Comments

- Comments should explain *why* or *how*
- Comments should *not* explain the opcode and its operands
- Comments are a major component of self-documenting code

Simple Addressing Modes

- Second operand - *<op2>*

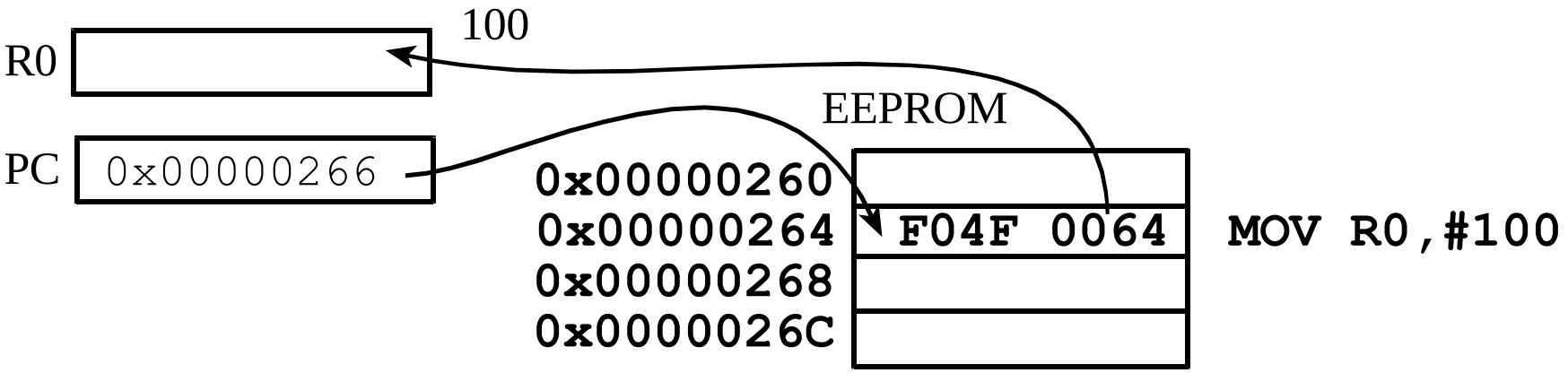
```
ADD  Rd, Rn, <op2>
```

  - Constant
    - `ADD Rd, Rn, #constant   ; Rd = Rn+constant`
  - Shift
    - `ADD R0, R1, LSL #4      ; R0 = R0+(R1*16)`
    - `ADD R0, R1, R2, ASR #4  ; R0 = R1+(R2/16)`
- Memory accessed only with LDR STR
  - Constant in ROM: `=Constant / [PC,#offs]`
  - Variable on the stack: `[SP,#offs]`
  - Global variable in RAM: `[Rx]`
  - I/O port: `[Rx]`

Addressing Modes

- **Immediate addressing**
  - **Data is contained in the instruction**

```
MOV   R0,#100        ; R0=100, immediate addressing
```

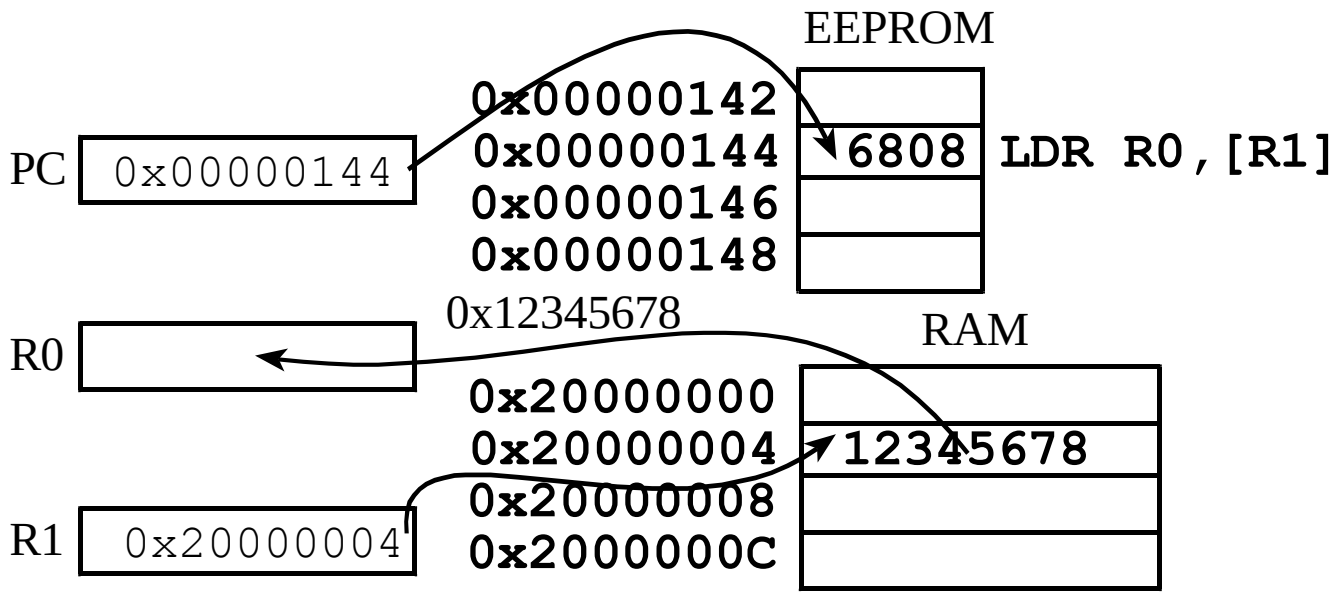


Addressing Modes

⑩ Indexed Addressing

- Address of the data in memory is in a register

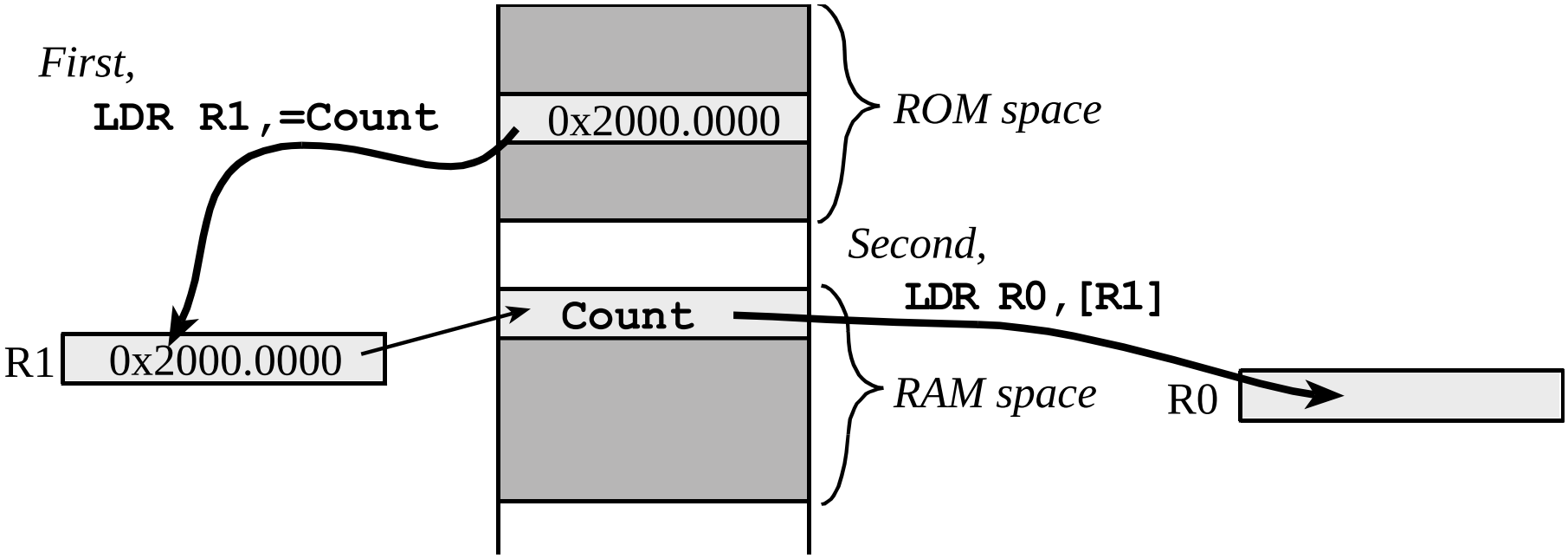
```
LDR   R0,[R1]        ; R0= value pointed to by R1
```



Addressing Modes

⑩ PC Relative Addressing

- Address of data in EEPROM is indexed based upon the Program Counter



Memory Access Instructions

- Loading a register with a constant, address, or data
  - `LDR  Rd, =number`
  - `LDR  Rd, =label`
- `LDR` and `STR` used to load/store RAM using register-indexed addressing
  - `Register [R0]`
  - `Base address plus offset [R0,#16]`

Load/Store Instructions

⑩ General load/store instruction format

```
LDR{type} Rd,[Rn]       ;load memory at [Rn] to Rd
STR{type} Rt,[Rn]       ;store Rt to memory at [Rn]
LDR{type} Rd,[Rn, #n]   ;load memory at [Rn+n] to Rd
STR{type} Rt,[Rn, #n]   ;store Rt to memory [Rn+n]
LDR{type} Rd,[Rn,Rm,LSL #n] ;load [Rn+Rm<<n] to Rd
STR{type} Rt,[Rn,Rm,LSL #n] ;store Rt to [Rn+Rm<<n]
```

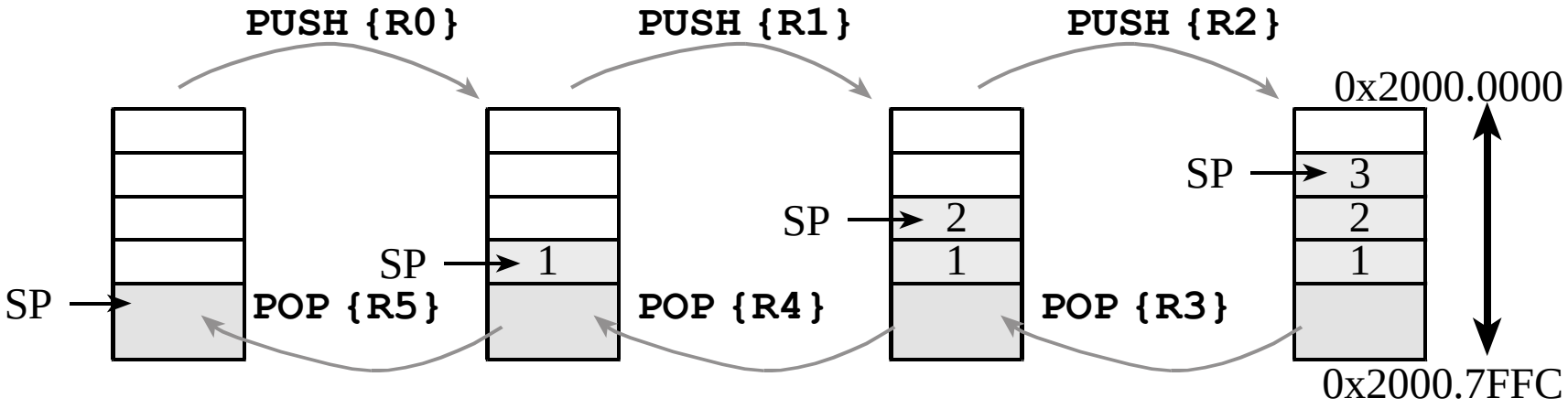
{type}	*Data type*	*Meaning*
	32-bit word	0 to 4,294,967,295 or -2,147,483,648 to +2,147,483,647
B	Unsigned 8-bit byte	0 to 255, Zero pad to 32 bits on load
SB	Signed 8-bit byte	-128 to +127, Sign extend to 32 bits on load
H	Unsigned 16-bit halfword	0 to 65535, Zero pad to 32 bits on load
SH	Signed 16-bit halfword	-32768 to +32767, Sign extend to 32 bits on load
D	64-bit data	Uses two registers

The Stack

- Stack is last-in-first-out (LIFO) storage
  - 32-bit data
- Stack pointer, SP or R13, points to top element of stack
- Stack pointer *decremented* as data placed on stack
- `PUSH` and `POP` instructions used to load and retrieve data

The Stack

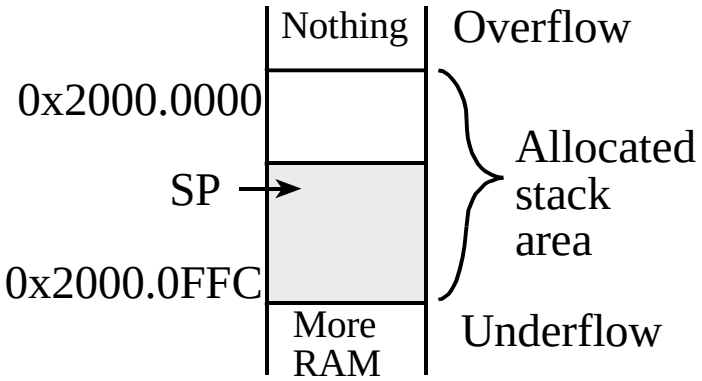
- Stack is last-in-first-out (LIFO) storage
  - 32-bit data
- Stack pointer, SP or R13, points to top element of stack
- Stack pointer *decremented* as data placed on stack (*incremented* when data is removed)
- `PUSH` and `POP` instructions used to load and retrieve data



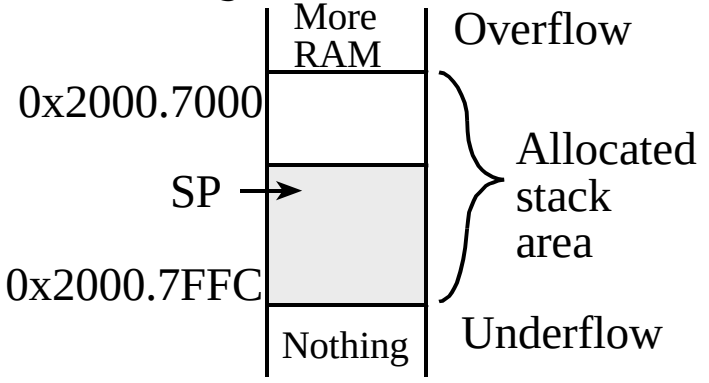
Stack Usage

❑ Stack memory allocation

Stack starting at the first RAM location



Stack ending at the last RAM location



❑ Rules for stack use

- ❖ **Stack should always be balanced, i.e. functions should have an equal number of pushes and pops**
- ❖ **Stack accesses (push or pop) should not be performed outside the allocated area**
- ❖ **Stack reads and writes should not be performed within the free area**

The assembly language syntax is as follows:

PUSH {R0} ; R13=R13-4, then Memory[R13] = R0

POP {R0} ; R0 = Memory[R13], then R13 = R13 + 4

You can PUSH or POP multiple registers in one instruction:

```
subroutine_1
PUSH {R0-R7, R12, R14}      ; Save registers
…                           ; Do your processing
POP {R0-R7, R12, R14}       ; Restore registers
BX R14                      ;  Return to calling function
```

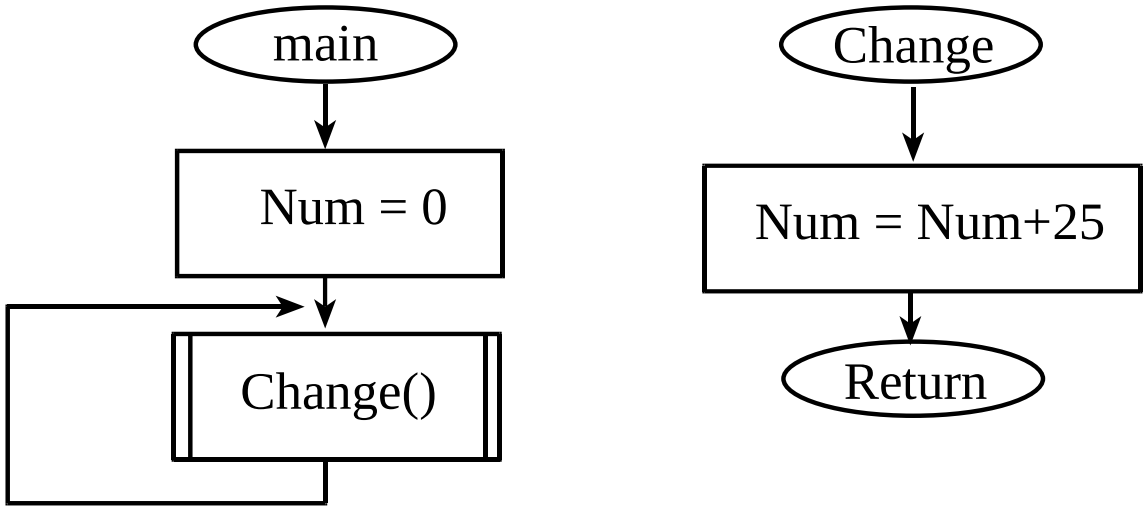
Link Register R14

R14 是链接寄存器(LR). 当一个子程序或函数被调用时，LR用来存储返回的程序计数器。

当你使用BL (branch and link) 指令时:

```
main ; Main program
…
BL function1 ; Call function1 using Branch with Link
             ; instruction.
             ; PC = function1 and
             ; LR = the next instruction in main
…
function1
…            ; Program code for function 1
BX LR        ; Return
```

Functions




```
Change  LDR   R1,=Num    ; 5) R1 = &Num
        LDR   R0,[R1]    ; 6) R0 = Num
        ADD   R0,R0,#25  ; 7) R0 = Num+25
        STR   R0,[R1]    ; 8) Num = Num+25
        BX    LR         ; 9) return
main    LDR   R1,=Num    ; 1) R1 = &Num
        MOV   R0,#0      ; 2) R0 = 0
        STR   R0,[R1]    ; 3) Num = 0
loop    BL    Change     ; 4) function call
        B     loop       ; 10) repeat
```

```
unsigned long Num;
void Change(void){
  Num = Num+25;
}
void main(void){
  Num = 0;
  while(1){
    Change();
  }
}
```

Subroutines

```
;------------Rand100------------
; Return R0=a random number between
; 1 and 100. Call Random and then
   divide
; the generated number by 100
; return the remainder+1
Rand100
       PUSH {LR}   ; SAVE Link
       BL           Random
   ;R0 is a 32-bit random number
     LDR  R1,=100
     BL   Divide
     ADD          R0,R3,#1
     POP  {LR}   ;Restore Link back
     BX   LR

                  POP {PC}
```

```
;------------Divide------------
; find the unsigned quotient and
   remainder
; Inputs:  dividend in R0
;          divisor in R1
; Outputs: quotient in R2
;          remainder in R3
;dividend = divisor*quotient +
   remainder
Divide
  UDIV R2,R0,R1    ;R2=R0/R1,R2 is
   quotient
  MUL  R3,R2,R1    ;R3=(R0/R1)*R1
  SUB  R3,R0,R3    ;R3=R0%R1,
                   ;R3 is remainder of
   R0/R1
  BX   LR          ;return

       ALIGN
       END
```

One function calls another, so LR must be saved

Reset, Subroutines and Stack

- A *Reset* occurs immediately after power is applied and when the reset signal is asserted (Reset button pressed)
- The Stack Pointer, SP (R13) is initialized at *Reset* to the 32-bit value at location 0 (Default: 0x20000408)
- The Program Counter, PC (R15) is initialized at *Reset* to the 32-bit value at location 4 (Reset Vector)
- The Link Register (R14) is initialized at Reset to 0xFFFFFFFF
- Thumb bit is set at *Reset*
- Processor automatically saves return address in LR when a subroutine call is invoked.
- User can push and pull multiple registers on or from the *Stack* at subroutine entry and before subroutine return.

Debugging

- Aka: Testing, Diagnostics, Verification
- Debugging Actions
  - Functional debugging, input/output values
  - Performance debugging, input/output values with time
  - Tracing, measure sequence of operations
  - Profiling,
    - measure percentage for tasks,
    - time relationship between tasks
  - Performance measurement, how fast it executes
  - Optimization, make tradeoffs for overall good
    - improve speed,
    - improve accuracy,
    - reduce memory,
    - reduce power,
    - reduce size,
    - reduce cost

Debugging Intrusiveness

- Intrusive Debugging
  - degree of perturbation caused by the debugging itself
  - how much the debugging slows down execution
- Non-intrusive Debugging
  - characteristic or quality of a debugger
  - allows system to operate as if debugger did not exist
  - e.g., logic analyzer, ICE, BDM

- Minimally intrusive
  - negligible effect on the system being debugged
  - e.g., dumps(ScanPoint) and monitors
- Highly intrusive
  - print statements, breakpoints and single-stepping

Debugging Aids in Keil

Interface

- Breakpoints
- Registers including xPSR
- Memory and Watch Windows
- Logic Analyzer, GPIO Panel
- Single Step, StepOver, StepOut, Run, Run to Cursor
- Watching Variables in Assembly

```
EXPORT  VarName[DATA,SIZE=4]
```

- Command Interface (Advanced but useful)

```
WS 1, `VarName,0x10
LA (PORTD & 0x02)>>1
```

… Debugging

- Instrumentation: Code we add to the system that aids in debugging
  - E.g., print statements
  - Good practice: Define instruments with specific pattern in their names
  - Use instruments that test a run time global flag
    - leaves a permanent copy of the debugging code
    - causing it to suffer a runtime overhead
    - simplifies “on-site” customer support.
  - Use conditional compilation (or conditional assembly)
    - Keil supports conditional assembly
    - Easy to remove all instruments
- Visualization: How the debugging information is displayed

ARM ISA : ADD, SUB and CMP

ARITHMETIC INSTRUCTIONS

```
ADD{S} {Rd,} Rn, <op2>      ;Rd = Rn + op2
ADD{S} {Rd,} Rn, #im12      ;Rd = Rn + im12
SUB{S} {Rd,} Rn, <op2>      ;Rd = Rn - op2
SUB{S} {Rd,} Rn, #im12      ;Rd = Rn - im12
RSB{S} {Rd,} Rn, <op2>      ;Rd = op2 - Rn
RSB{S} {Rd,} Rn, #im12      ;Rd = im12 - Rn
CMP Rn, <op2>               ;Rn - op2
CMN Rn, <op2>               ;Rn - (-op2)
```

Addition
C bit set if unsigned overflow
V bit set if signed overflow

Subtraction
C bit *clear* if unsigned overflow
V bit set if signed overflow

ARM ISA : Multiply and Divide

32-BIT MULTIPLY/DIVIDE INSTRUCTIONS

```
MUL{S} {Rd,} Rn, Rm          ;Rd = Rn * Rm
MLA     Rd, Rn, Rm, Ra ;Rd = Ra + Rn*Rm
MLS     Rd, Rn, Rm, Ra ;Rd = Ra - Rn*Rm
UDIV   {Rd,} Rn, Rm          ;Rd = Rn/Rm
unsigned
SDIV   {Rd,} Rn, Rm          ;Rd = Rn/Rm
signed
```

Multiplication does not set C,V bits

Random Number Generator

```
;Program demonstrates Arithmetic operations
; ADD, SUB, MUL and IDIV
; LR loss when sub calls sub - Solution
Seed    EQU          123456789
        THUMB
        AREA    DATA, ALIGN=2
        EXPORT  M [DATA,SIZE=4]
M       SPACE   4
        ALIGN
        AREA |.text|, CODE, READONLY, ALIGN=2
        EXPORT  Start
        EXPORT  M [DATA,SIZE=4]
       ; Need this to be able to watch RAM
       ;  Variable M (in Assembly only)
Start   LDR R2,=M       ; R2 points to M
        LDR R0,=Seed    ; Initial seed
        STR R0,[R2]     ; M=Seed
loop    BL  Random      ; R0 has rand number
        B   loop
```

```
;------------Random------------
; Return R0= 32-bit random number
; Linear congruential generator
; from Numerical Recipes by Press et
    al.
Random LDR R2,=M    ; R2 points to M
       LDR R0,[R2]  ; R0=M
       LDR R1,=1664525
       MUL R0,R0,R1 ; R0 = 1664525*M
       LDR R1,=1013904223
       ADD R0,R1  ;
   1664525*M+1013904223
       STR R0,[R2]  ; store M
       BX  LR
```

Function *Random*
- How it is called
- How it returns

Global variable *M*
- How it is defined
- How it is written
- How it is read

Introduction to C

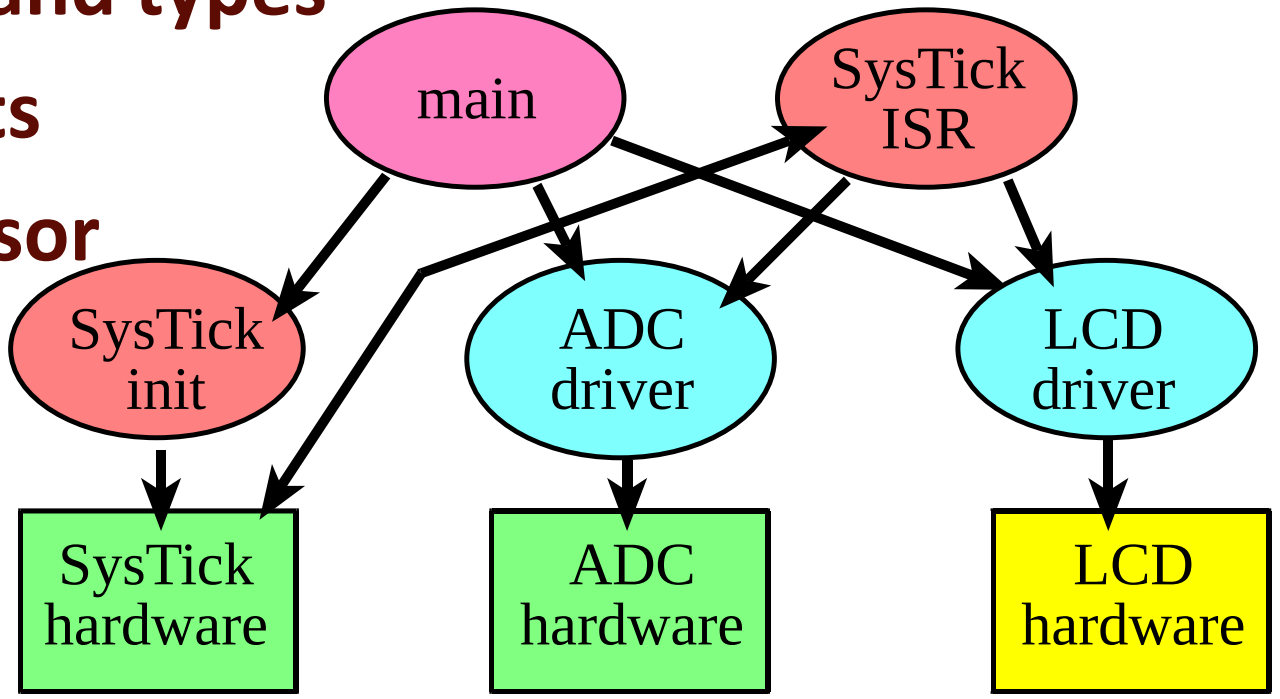
- C is a high-level language
  - Abstracts hardware
  - Expressive
  - Readable
  - Analyzable
- C is a *procedural language*
  - The programmer explicitly specifies steps
  - Program composed of procedures
    - `Functions/subroutines`
- C is compiled (not interpreted)
  - Code is analyzed as a whole (not line by line)

Why C?

- C is popular
- C influenced many languages
- C is considered close-to-machine
  - Language of choice when careful coordination and control is required
  - Straightforward behavior (typically)
- Typically used to program low-level software (with some assembly)
  - Drivers, runtime systems, operating systems, schedulers, …

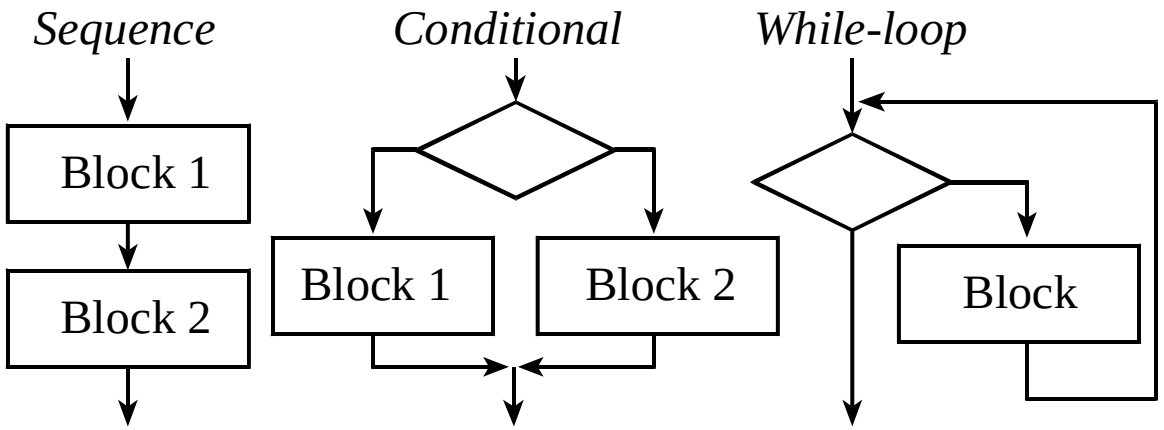
Introduction to C

- Program structure
  - Subroutines and functions
- Variables and types
- Statements
- Preprocessor
- DEMO



C Program (demo)

- Preprocessor directives
- Variables
- Functions
- Statements
- Expressions
- Names
- Operators
- Comments
- Syntax



Important Notes

- C comes with a lot of “built-in” functions
  - printf() is one good example
  - Definition included in *header files*
  - #include<header_file.h>

- C has one special function called *main*()
  - This is where execution starts (reset vector)

- C development process
  - Compiler translates C code into assembly code
  - Assembler (e.g. built into uVision4) translates assembly code into object code
  - Object code runs on machine