ECE 245:
Microcontroller Applications with Internet of Things (IoT)

Lecture 1: Introduction

Instructor: Shayan (Sean) Taheri, Ph.D.

Assistant Professor

The Department of Electrical and Cyber Engineering (ECE)

The Institute for Health and Cyber Knowledge (I-HACK)

The Gannon University (GU)

Personal Informtion

- Name: Shayan (Sean) Taheri.
- Date of Birth: July/28/1991.
- Past Position: Postdoctoral Fellow at University of Florida.
- Ph.D. Degree: Electrical Engineering from the University of Central Florida.
- M.S. Degree: Computer Engineering from the Utah State University.
- University Profile: https://www.gannon.edu/FacultyProfiles.aspx?profile=taheri001

Agenda

- Course Description
  - *Book, Labs, and Equipment*
  - *Grading Criteria*
  - *Expectations/Responsibilities*
  - *Prerequisites and Reminders*
- Embedded Systems
  - *Microcontrollers*
- Product Life Cycle
  - *Analysis, Design, Implementation, Testing*
  - *Flowcharts, Data-Flow and Call Graphs*
- ARM Architecture
  - *Programming*
  - *Integrated Development Environment (IDE)*

Useful Info

- Office hours: Please refer to the Blackboard system.
- Course Components: Laboratory Assignments, Theoretical Assignments, Exams, and Projects.
- Most of the learning is in the laboratory assignments.
- Read the textbooks and the laboratory manual.
- Study the slides very well.

DOs and DON'Ts

DO

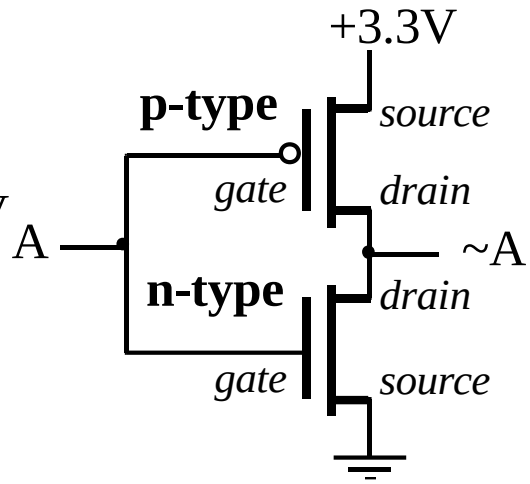
- Read
  - *Book, lab, and datasheets*
- Try before seeking help
- Follow announcements
- Discuss material with the instructor
- Do research
- Track due dates

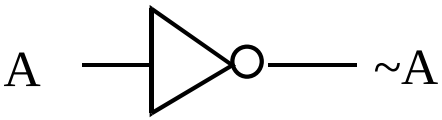
DON'T

- Don't cheat!
- **Never look at another student's code** (current or previous)
- Don't let your partner do all the work
- Don't copy software from book or web without attribution
- Don't expect handholding

Reminder: Digital Logic

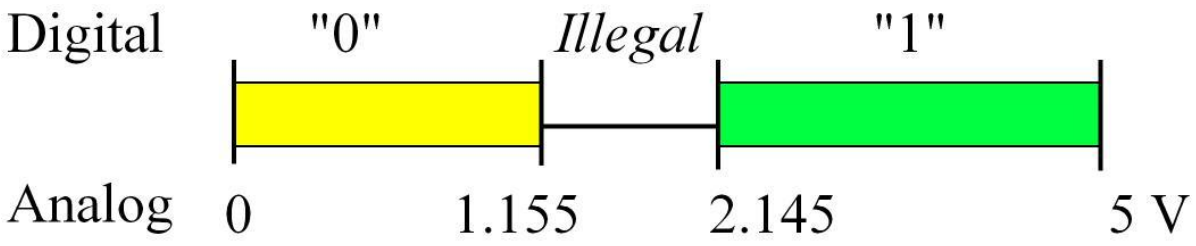
A	p-type	n-type	~A
0 V	active	off	+3.3V
+3.3V	off	active	0V





A	~A
0	1
1	0

- **'X' means '0' or '1'**
- **'Z' means neither '0' nor '1'**



- ❑ AND, OR, NOT
- ❑ Flip flops
- ❑ Registers

Positive logic:
True is higher voltage
False is lower voltage

Negative logic :
True is lower voltage
False is higher voltage

Reminder: Skills

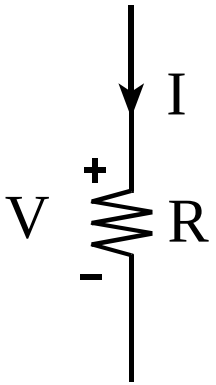
- Problem solving
- Programming
- Debugging

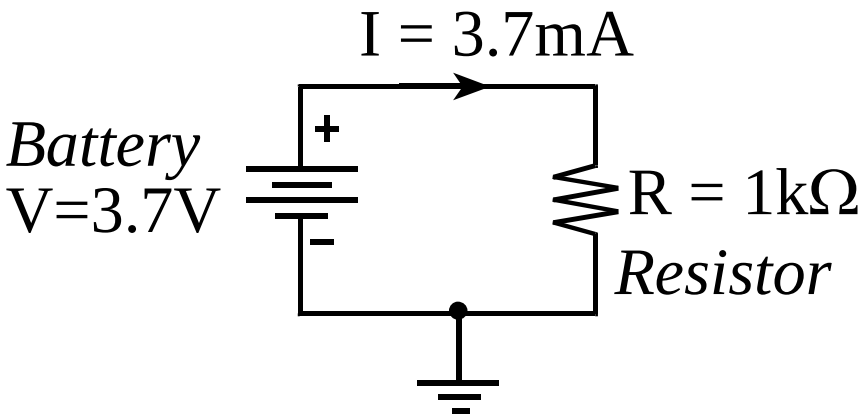
Reminder: Ohm's Law

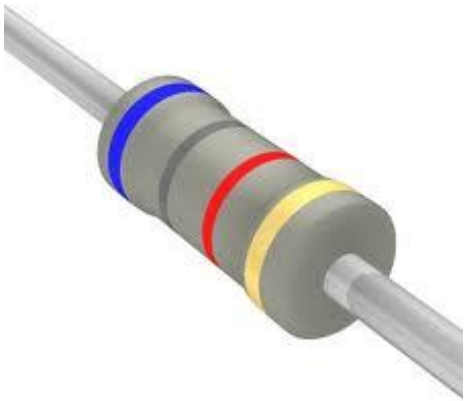
$V = I * R$ *Voltage = Current * Resistance*

$I = V / R$ *Current = Voltage / Resistance*

$R = V / I$ *Resistance = Voltage / Current*



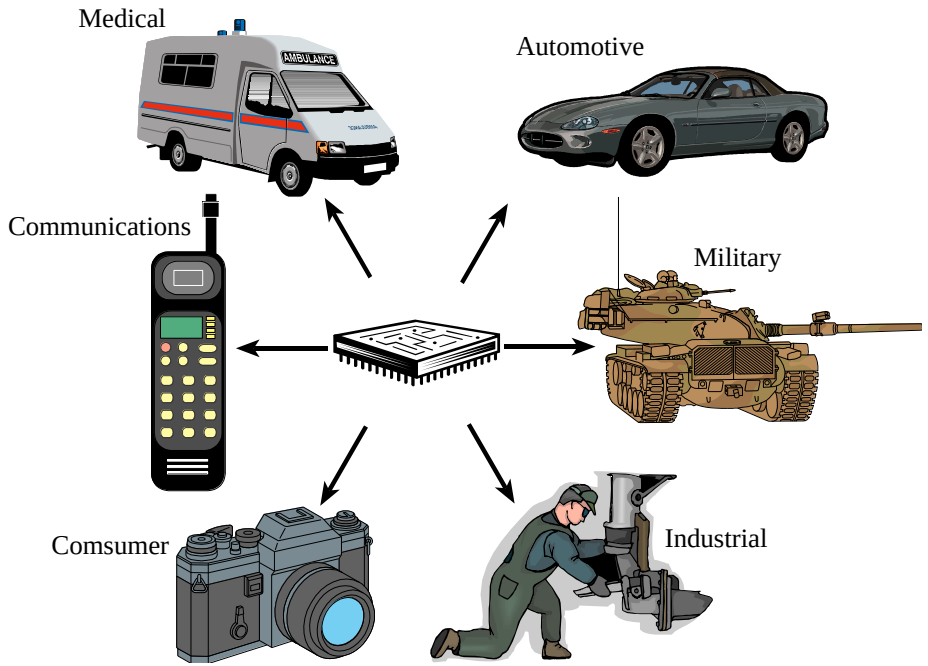




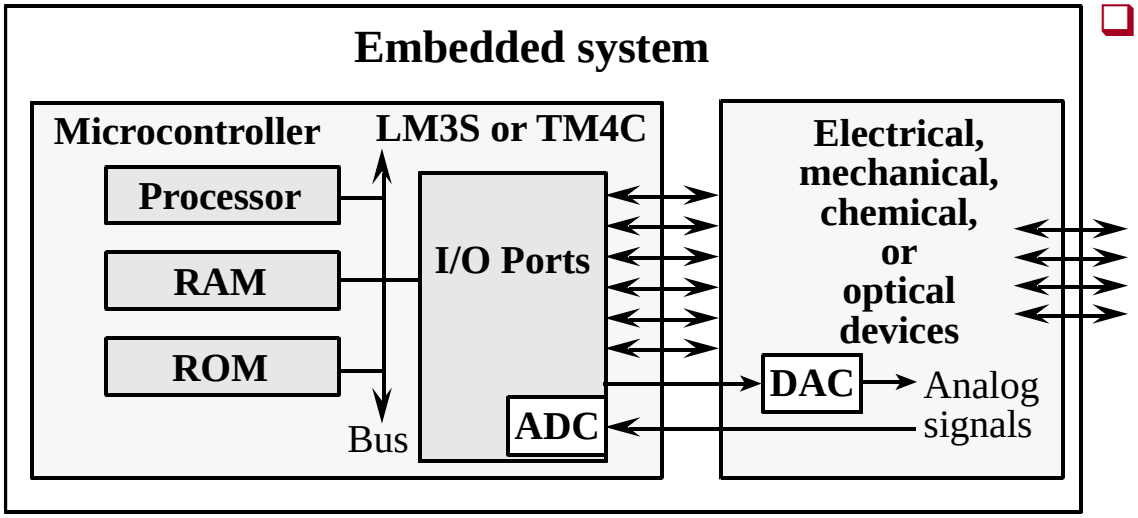
- $P = V * I$ *Power = Voltage * Current*
- $P = V^2 / R$ *Power = Voltage² / Resistance*
- $P = I^2 * R$ *Power = Current² * Resistance*

1 amp is 6.241×10^{18} electrons per second = 1 coulomb/sec

Embedded System



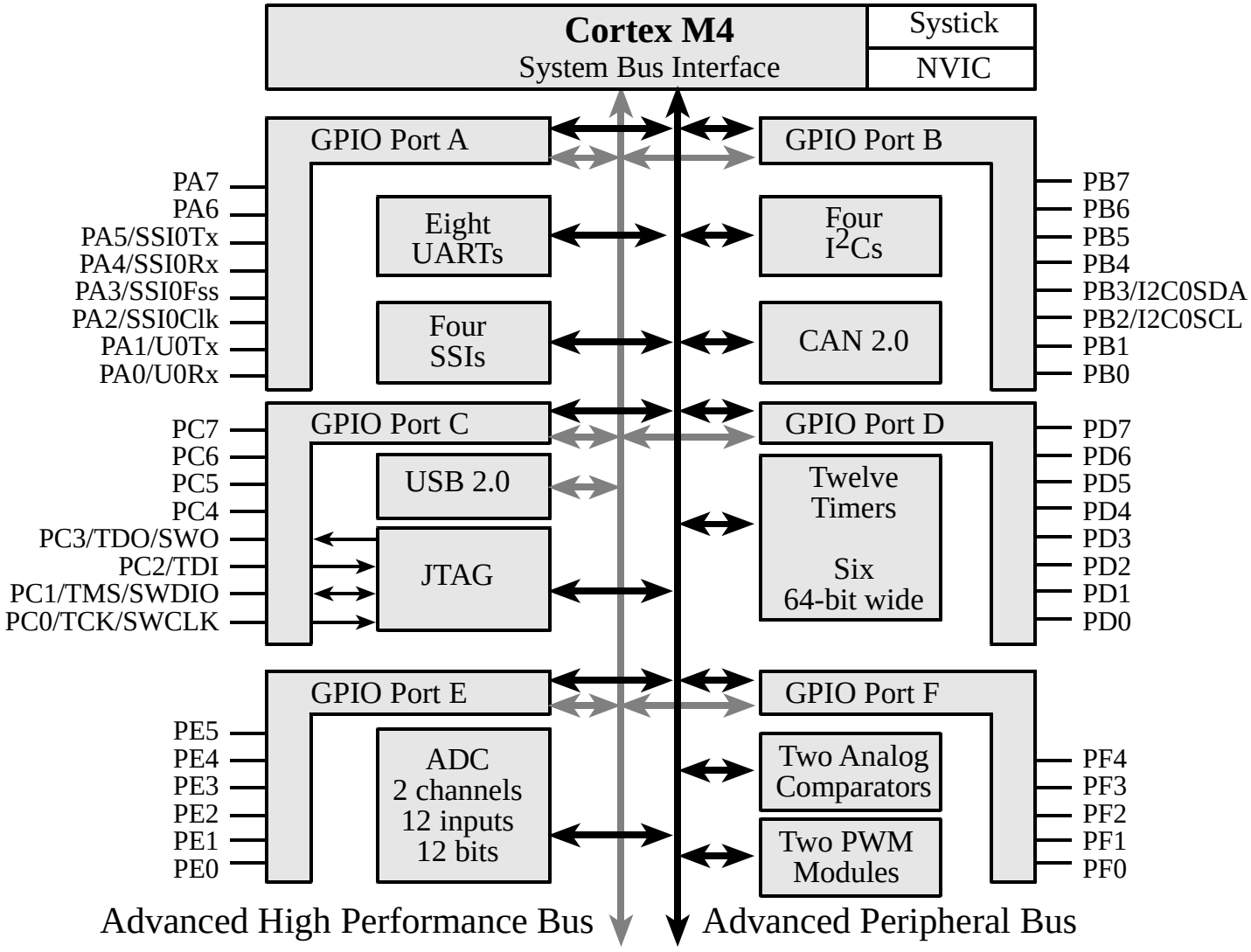
- Embedded Systems are everywhere
  - *Ubiquitous, invisible*
  - *Hidden (computer inside)*
  - *Dedicated purpose*
- Microprocessor
  - *Intel: 4004, ..8080,.. x86*
  - *Freescale: 6800, .. 9S12,.. PowerPC*
  - *ARM, DEC, SPARC, MIPS, PowerPC, Natl. Semi.,…*
- Microcontroller
  - *Processor+Memory+ I/O Ports (Interfaces)*



Microcontroller

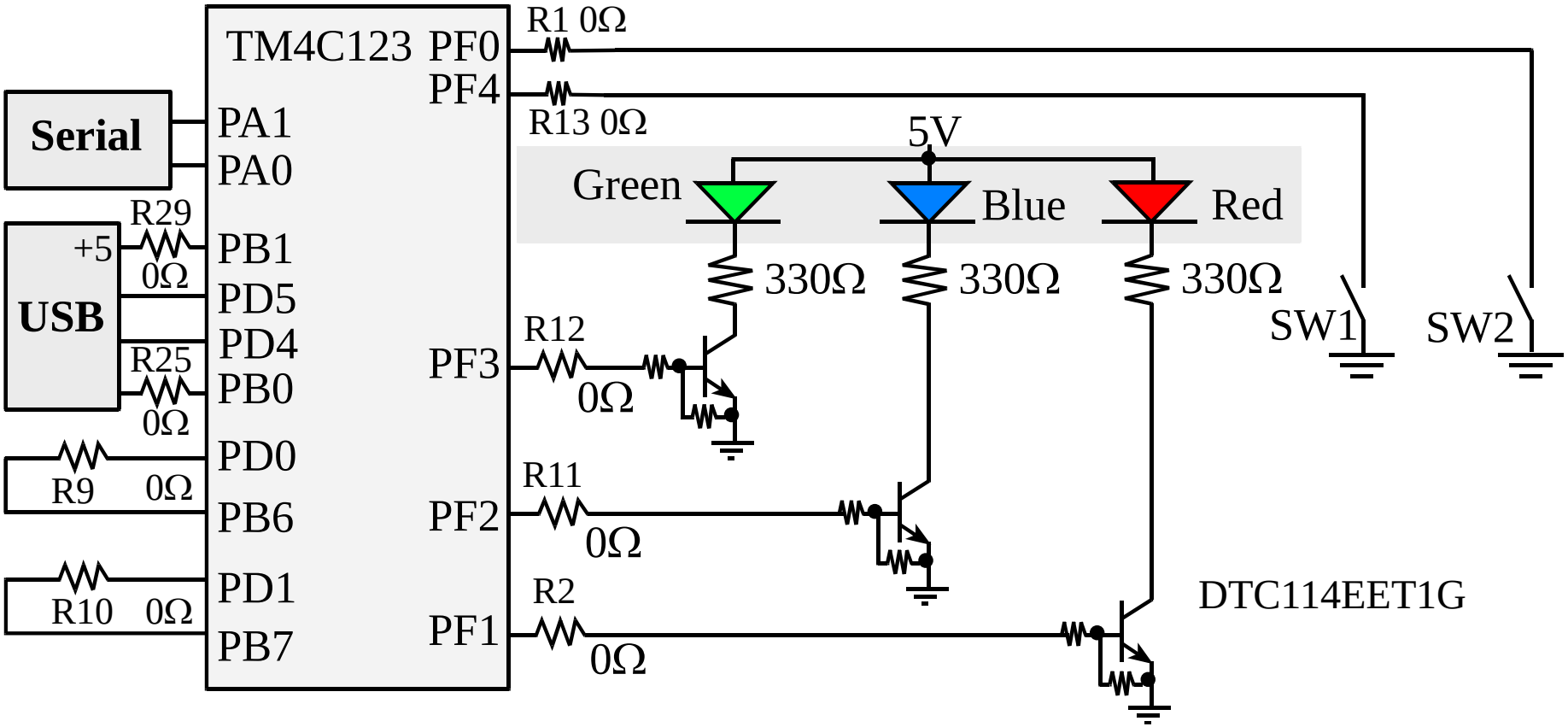
- Processor – Instruction Set + memory + accelerators
  - *Ecosystem*
- Memory
  - *Non-Volatile*
    - ROM
    - EPROM, EEPROM, Flash
  - *Volatile*
    - RAM (DRAM, SRAM)
- Interfaces
  - *H/W: Ports*
  - *S/W: Device Driver*
  - *Parallel, Serial, Analog, Time*
- I/O
  - *Memory-mapped vs. I/O-instructions (I/O-mapped)*

Texas Instruments TM4C123



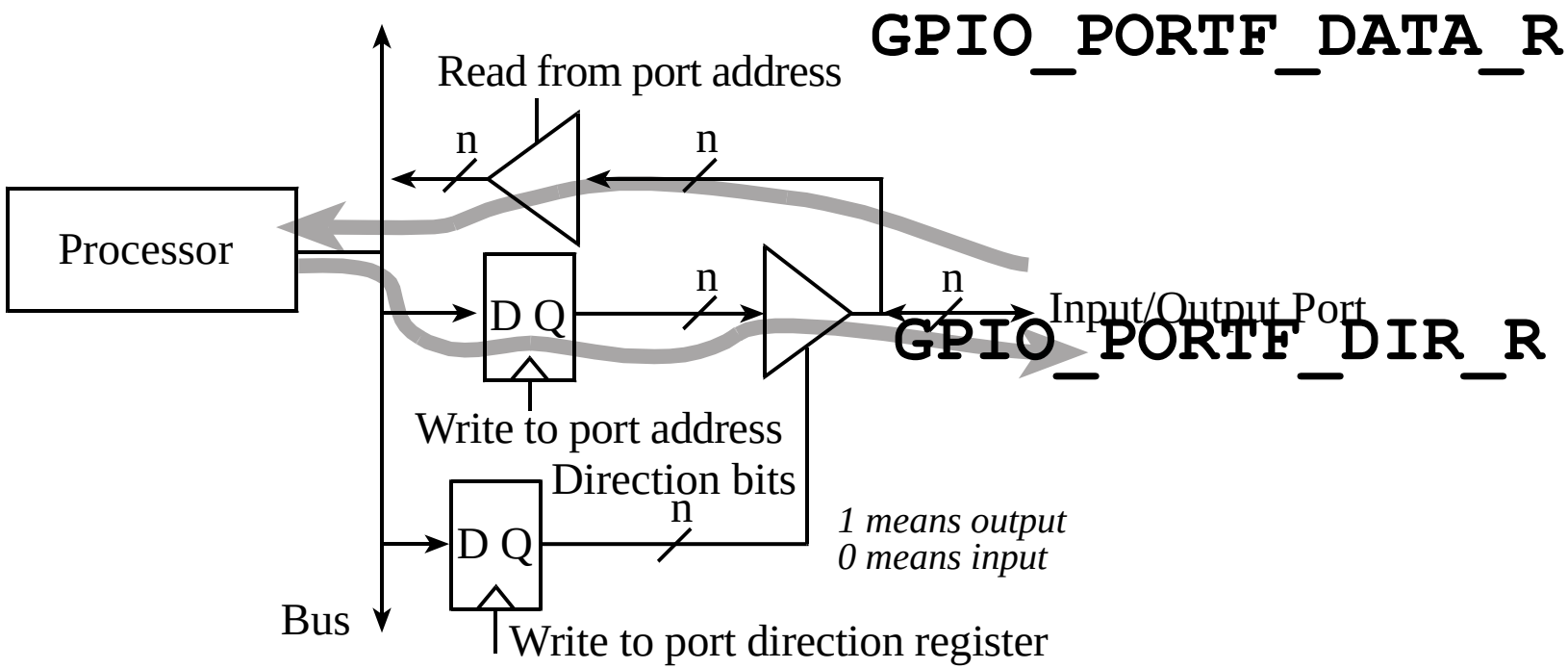
ARM Cortex-M4
+ 256K EEPROM
+ 32K RAM
+ JTAG
+ Ports
+ SysTick
+ ADC
+ UART

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

I/O Ports and Control Registers



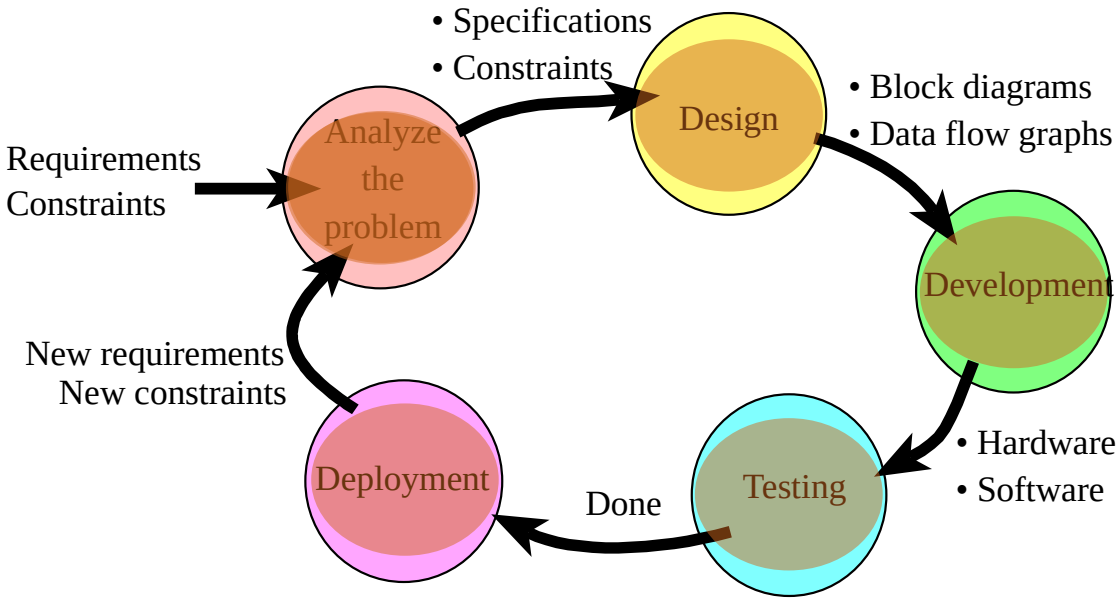
- *The input/output direction of a bidirectional port is specified by its direction register.*
- **GPIO_PORTF_DIR_R** , specify if corresponding pin is input or output:
  - *0 means input*
  - *1 means output*

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`**
  2. Wait two bus cycles (two NOP instructions)
  3. *Unlock PF0 (PD7 also needs unlocking)*
  4. Set *DIR* to 1 for output or 0 for input
  5. Clear *AFSEL* bits to 0 to select regular I/O
  6. *Set PUE bits to 1 to enable internal pull-up*
  7. Set *DEN* bits to 1 to enable data pins
- **Input/output from pin**
  6. Read/write **`GPIO_PORTF_DATA_R`**

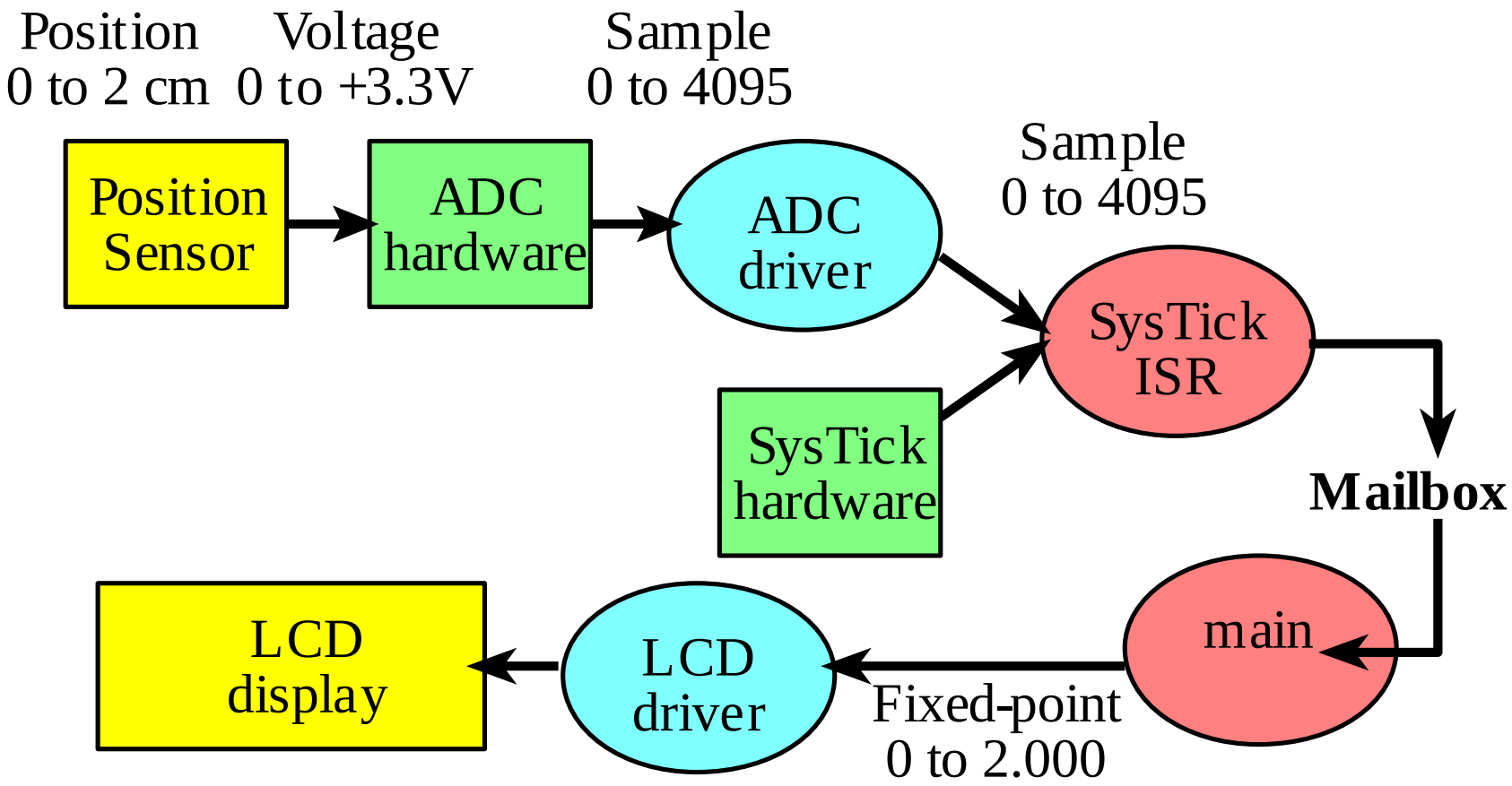
Product Life Cycle



- Analysis (What?)
  - *Requirements -> Specifications*
- Design (How?)
  - *High-Level: Block Diagrams*
  - *Engineering: Algorithms, Data Structures, Interfacing*
- Implementation(Real)
  - *Hardware, Software*
- Testing (Works?)
  - *Validation:Correctness*
  - *Performance: Efficiency*
- Maintenance (Improve)

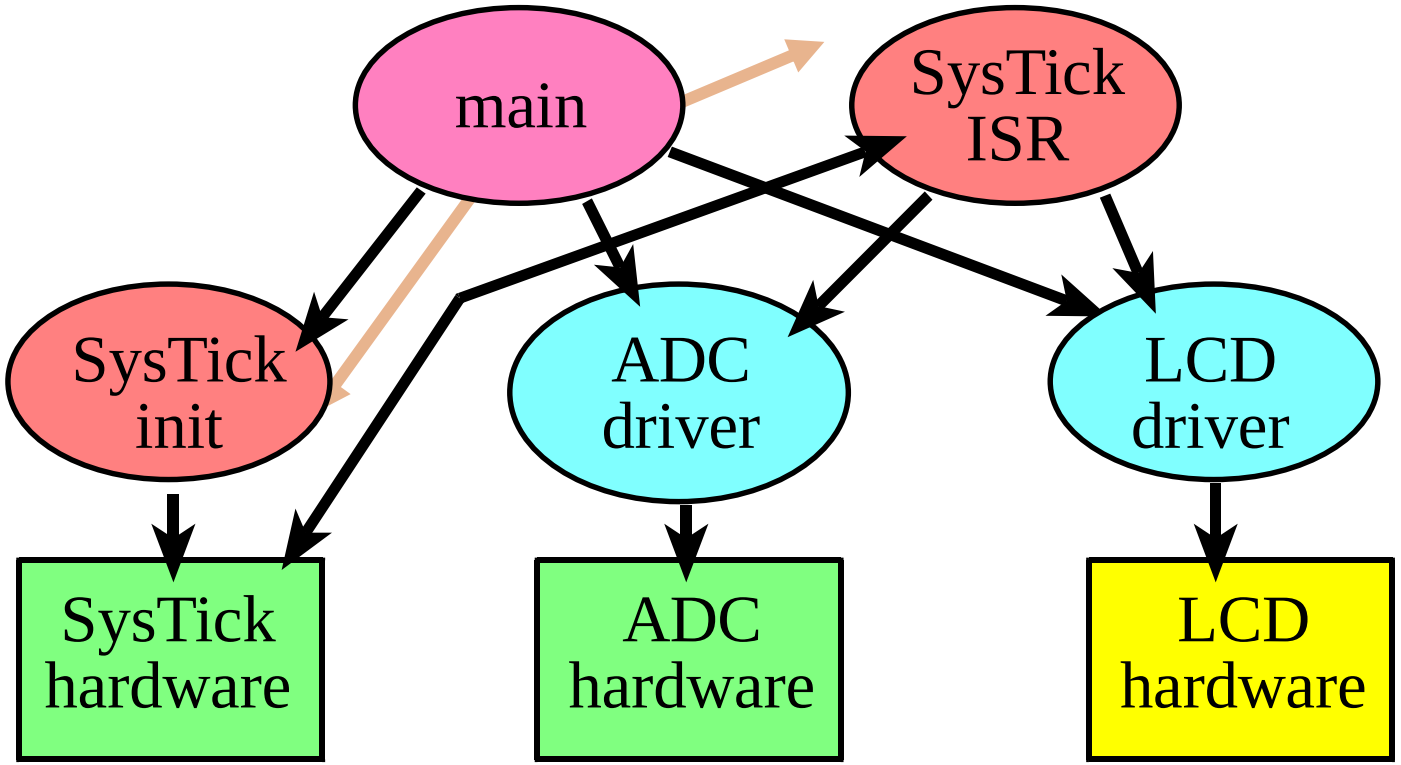
Data Flow Graph

Position Measurement System



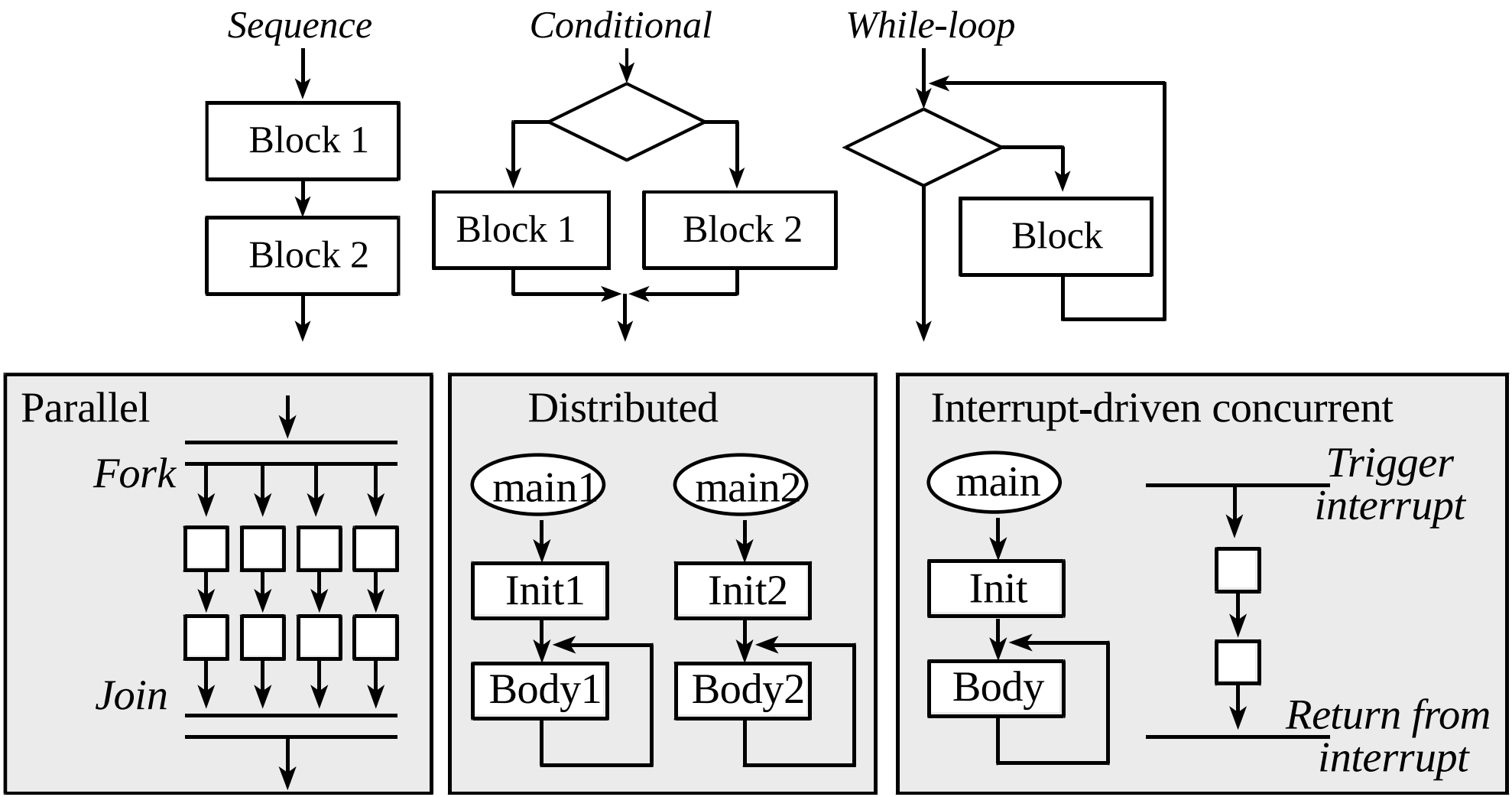
Call Flow Graph

Position Measurement System



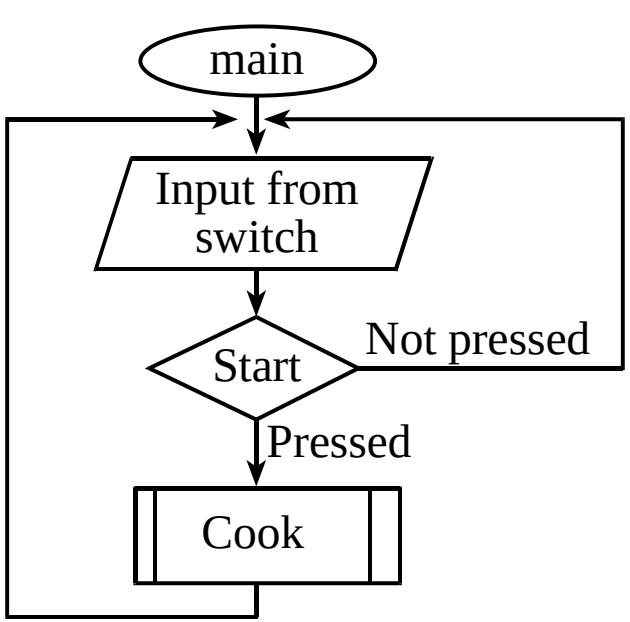
Structured Programming

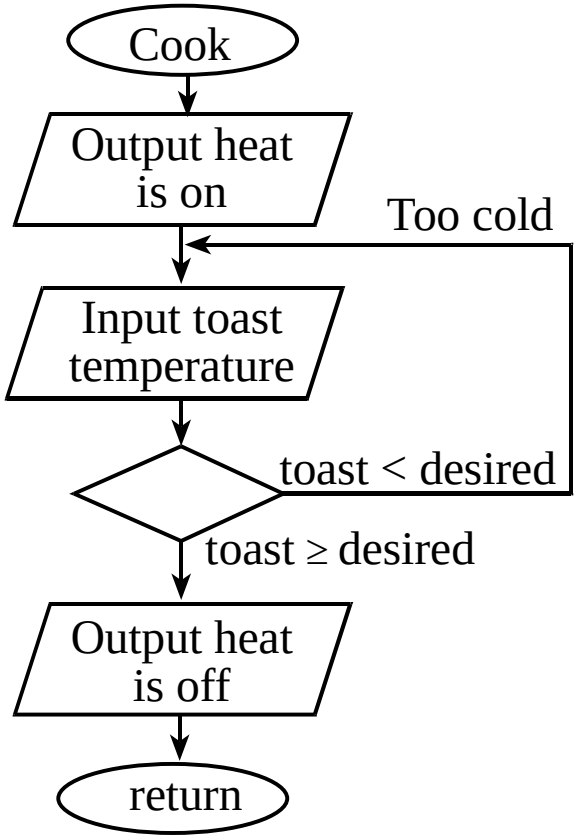
- Common Constructs (as Flowcharts)



Flowchart

Toaster oven:

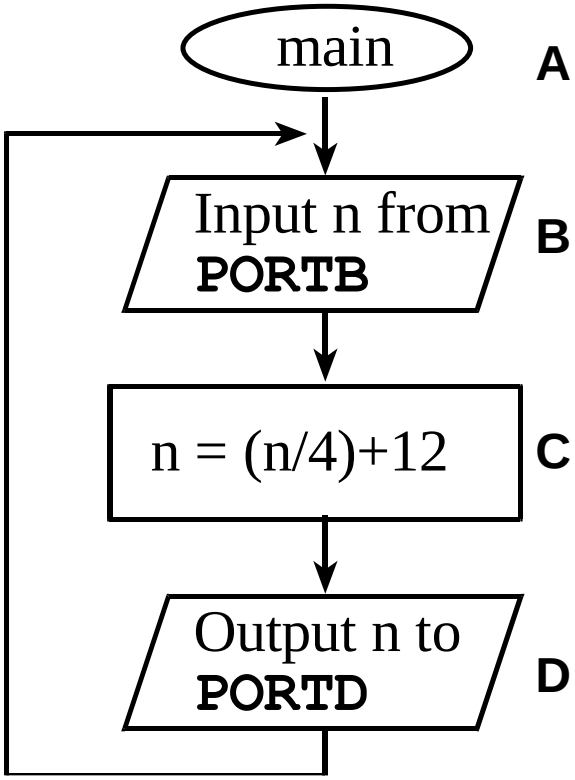


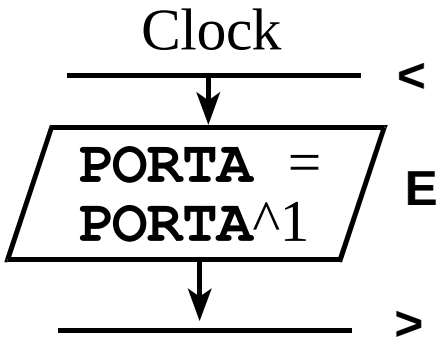


Coding in assembly and/or high-level language (C)

Flowchart

- Design a flowchart for a system that performs two independent tasks. The first task is to output a 20 kHz square wave on **PORTA** in real time (period is 50 ms). The second task is to read a value from **PORTB**, divide the value by 4, add 12, and output the result on **PORTD**. This second task is repeated over and over.

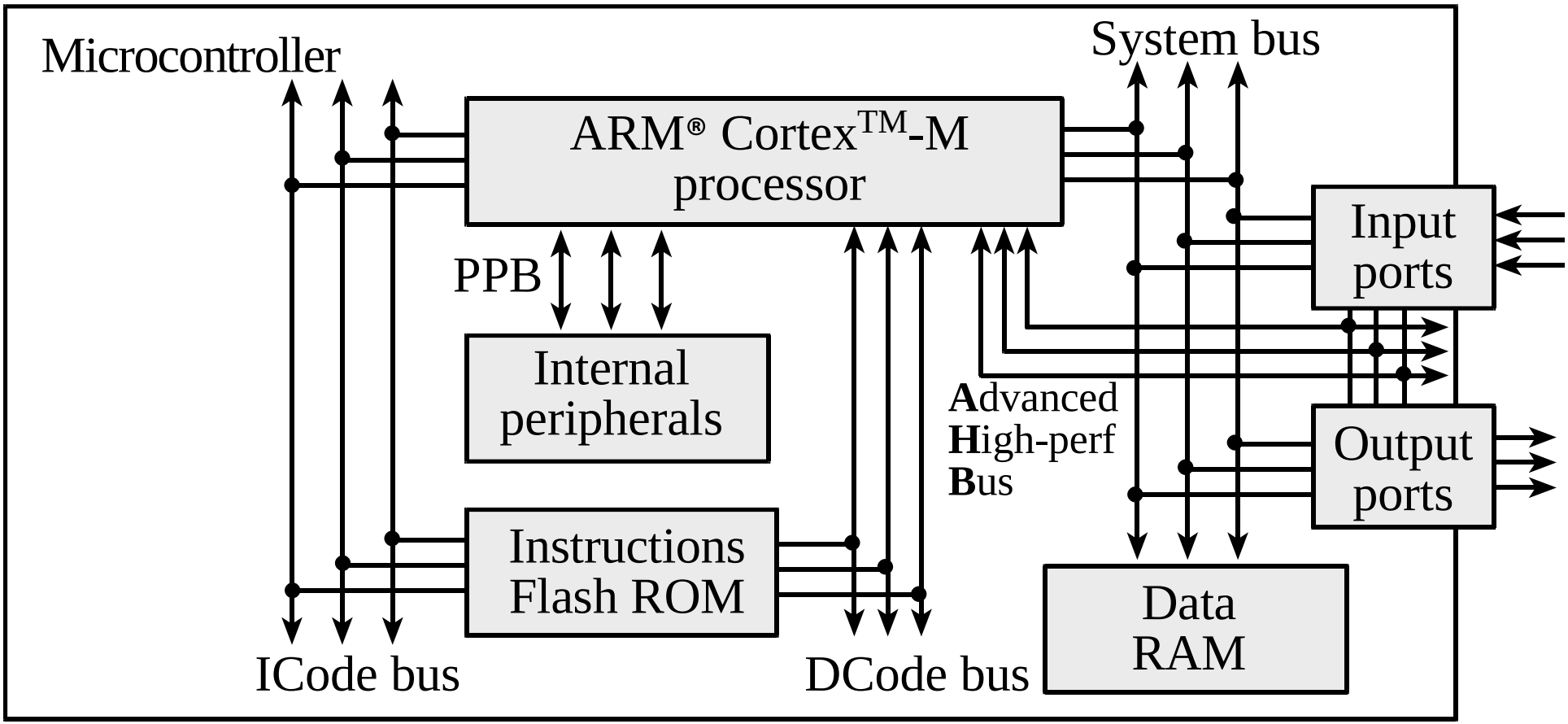





```
void  SysTick_Handler(void){
  PORTA = PORTA^0x01;  <-- E
}                      >

void main(void){       <-- A
unsigned long n;
  while(1){
    n = PORTB;         <-- B
    n = (n/4)+12;      <-- C
    PORTD = n;         <-- D
  }
}
```

ARM Cortex M4-based System



- ARM Cortex-M4 processor
- *Harvard* architecture
  - Different busses for instructions and data

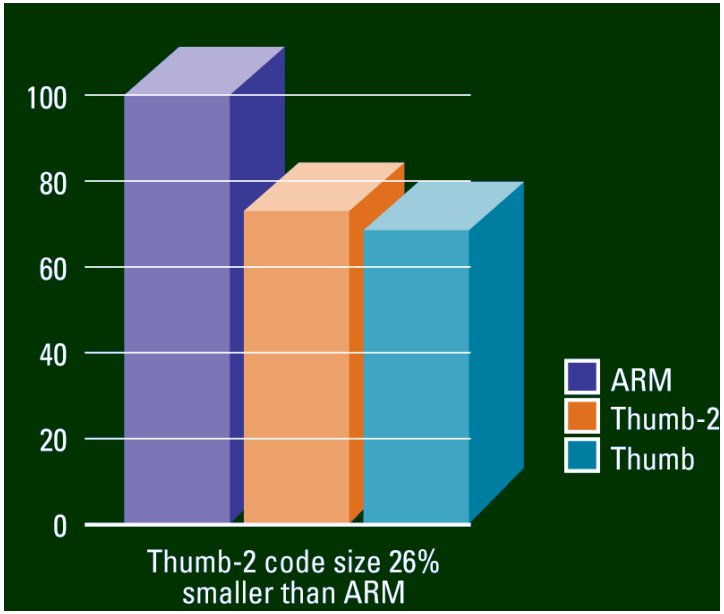
ARM Cortex M4-based System

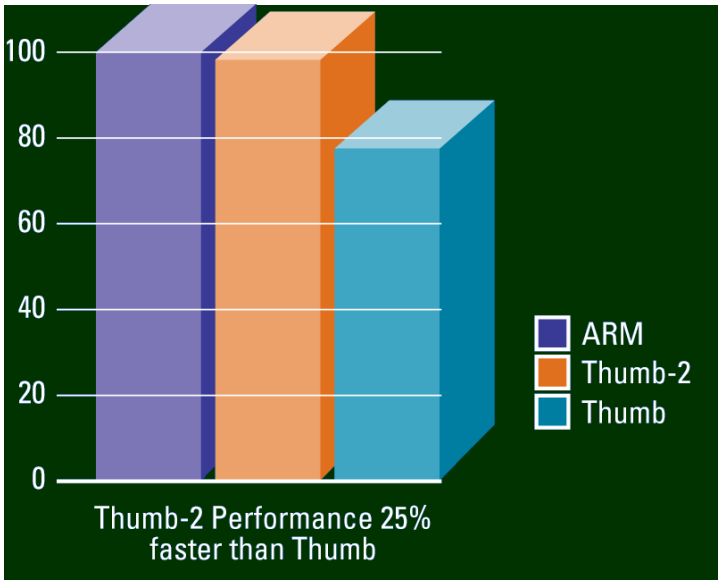
- RISC machine
  - *Pipelining* effectively provides single cycle operation for many instructions
  - Thumb-2 configuration employs both 16 and 32 bit instructions

CISC	*RISC*
Many instructions	Few instructions
Instructions have varying lengths	Instructions have fixed lengths
Instructions execute in varying times	Instructions execute in 1 or 2 bus cycles
Many instructions can access memory	Few instructions can access memory • Load from memory to a register • Store from register to memory
In one instruction, the processor can both • read memory and • write memory	No one instruction can both read and write memory in the same instruction
Fewer and more specialized registers. • some registers contain data, • others contain addresses	Many identical general purpose registers
Many different types of addressing modes	Limited number of addressing modes • register, • immediate, and • indexed.

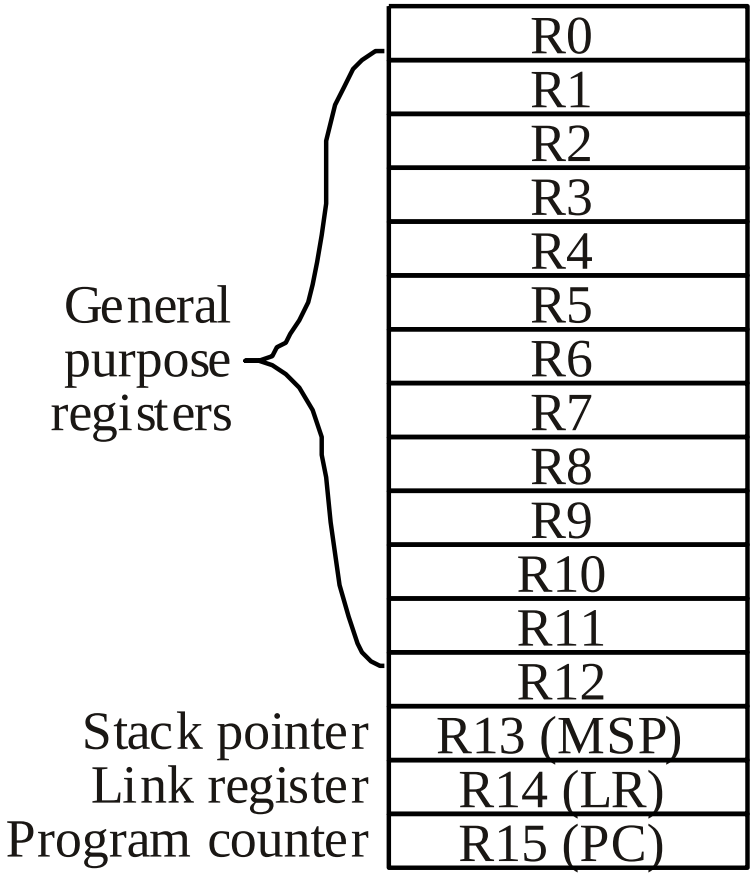
ARM ISA: Thumb2 Instruction Set

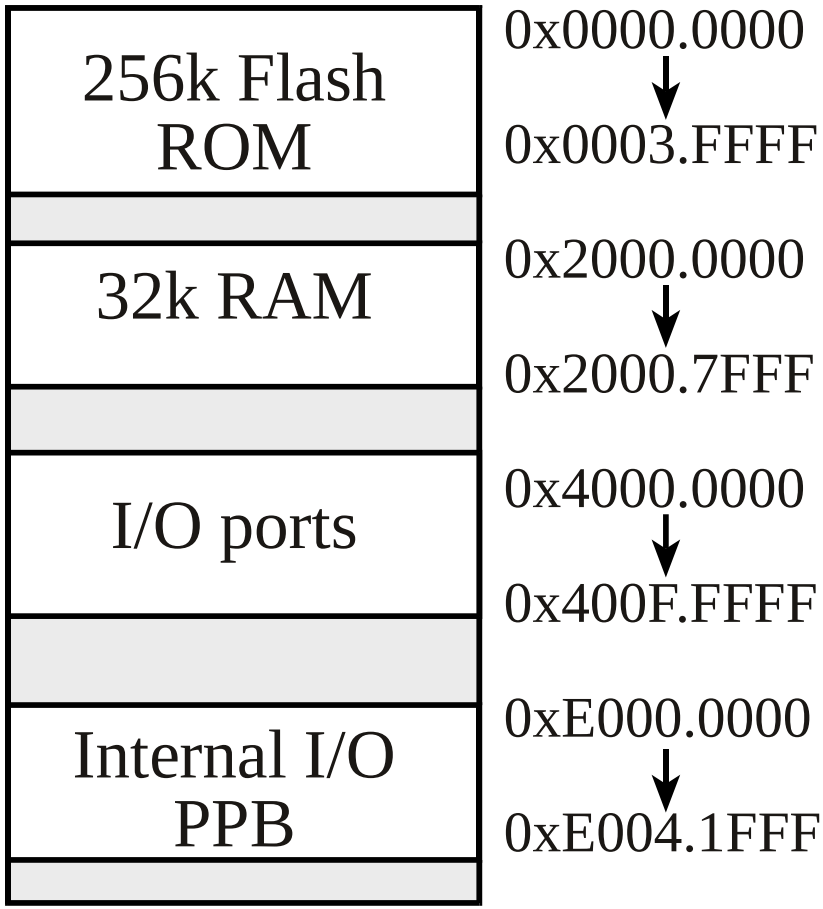
- Variable-length instructions
  - ARM instructions are a fixed length of 32 bits
  - Thumb instructions are a fixed length of 16 bits
  - Thumb-2 instructions can be either 16-bit or 32-bit
- Thumb-2 gives approximately 26% improvement in code density over ARM
- Thumb-2 gives approximately 25% improvement in performance over Thumb





ARM ISA: Registers, Memory-map





Condition Code Bit	***s***	***Indicates***
N	negative	Result is negative
Z	zero	Result is zero
V	overflow	Signed overflow
C	carry	Unsigned overflow

TI TM4C123 Microcontroller

LC3 (RISC) to ARM (RISC) - Data Movement

- LEA R0, Label ;R0 <- PC + Offset to Label
- ADR R0,Label or LDR R0,=Label
- LD R1,Label ; R1 <- M[PC + Offset]
- LDR R0,=Label ; Two steps: (i) Get address into R0
  LDRH R1,[R0] ; (ii) Get content of address [R0] into R1
- LDR R1,R0,n ; R1 <- M[R0+n]
- LDRH R1,[R0,#n]
- LDI R1,Label ; R1 <- M[M[PC + Offset]]
- ; Three steps!!
- ST R1,Label ; R1 -> M[PC + Offset]
- LDR R0,=Label ; Two steps: (i)Get address into R0
  STRH R1,[R0] ; (ii) Put R1 contents into address in R0
- STR R1,R0,n ; R1 -> M[R0+n]
- STRH R1,[R0,#n]
- STI R1,Label ; R1 -> M[M[PC + Offset]]
- ; Three steps!!

LC3 to ARM – Arithmetic/Logic

- ADD R1, R2, R3 ; R1 <- R2 + R3
- ADD R1,R2,R3 ; 32-bit only
- ADD R1,R2,#5 ; R1 <- R2 + 5
- ADD R1,R2,#5 ; 32-bit only, Immediate is 12-bit
- AND R1,R2,R3 ; R1 <- R2 & R3
- AND R1, R2, R3 ; 32-bit only
- AND R1,R2,#1 ; R1 <- Bit 0 of R2

  AND R1, R2, #1 ; 32-bit only
- NOT R1,R2 ; R1 -> ~(R2)
- EOR R1,R2,#-1 ; -1 is 0xFFFFFFFF,
- ; so bit

  XOR with 1 gives complement

LC3 to ARM – Control

- BR Target ; PC <- Address of Target
- B Target
- BRnzp Target ; PC <- Address of Target
- B Target
- BRn Target ; PC <- Address of Target if N=1
- BMI Target ; Branch on Minus
- BRz Target ; PC <- Address of Target if Z=1
- BEQ Target
- BRp Target ; PC <- Address of Target if P=1
- No Equivalent
- BRnp Target ; PC <- Address of Target if Z=0
- BNE Target
- BRzp Target ; PC <- Address of Target if N=0
- BPL Target ; Branch on positive or zero (Plus)
- BRnz Target ; PC <- Address of Target if P=0
- No Equivalent

LC3 to ARM – Subs,TRAP, Interrupt

```
JSR  Sub        ; PC <- Address of Sub, Return address in R7
        BL Sub      ; PC<-Address of Sub, Ret. Addr in R14 (Link Reg)
JSRR  R4        ; PC <- R4, Return address in R7
        BLX R4      ; PC <-R4, Return address in R14 (Link Reg)
RET             ; PC <- R7  (Implicit JMP to address in R7)
        BX LR       ; PC <- R14 (Link Reg)
JMP R2          ; PC <- R2
        BX R2       ; PC <- R14 (Link Reg)
TRAP x25        ; PC <- M[x0025], Return address in R7
        SVC #0x25   ; Similar in concept but not implementation
RTI             ;  Pop PC and PSR from Supervisor Stack…
        BX LR       ; PC <- R14 (Link Reg)  [same as RET]
```

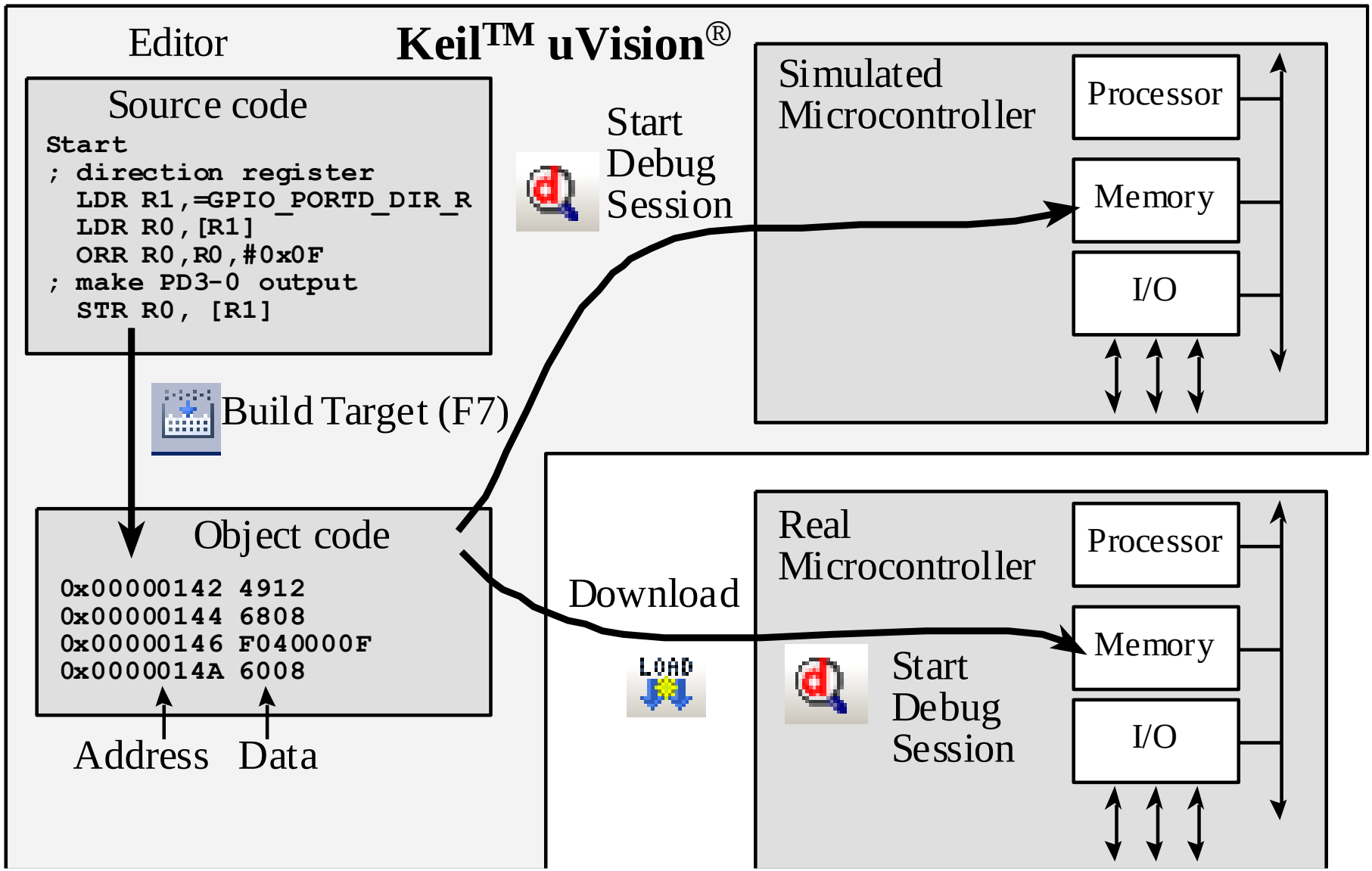
ARM is a Load-Store machine

- **Code to set (to 1) bit 5 of memory address x400FE608**

- SYSCTL_RCGCGPIO_R EQU 0x400FE608
- *; EQU psedo-op allows use of*
- *; symbolic name to represent a constant*

- LDR R1, =SYSCTL_RCGCGPIO_R *; R1 holds x400FE608*
- LDR R0, [R1] *; R0 holds contents of*
- *; location x400FE608*
- ORR R0, R0, #0x20 *; bit5 of R0 is set to 1*
- STR R0, [R1] *; write R0 contents back to*
- *; location x400FE608*

SW Development Environment



Questions?