CET 241: Day 2

Dr. Noori KIM

Agenda

- **Review:** Hardware overview - Generalized computer organization
- Programmer's model for ARM **emphasizing on CPU register** (i.e., Data structures, size, Core registers…):
- **Outside of CPU**: Memories and Memory map
- Software processing overview: Program execution flow - ARM ISA, assembling brief

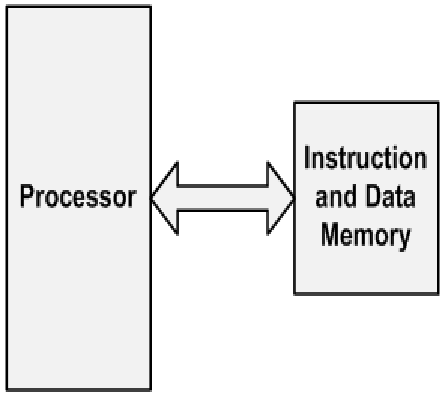
- Computer parts: C_ _, M_ _ _ _ _, I/_, B_ _
  - CPU or processor:
    - C_ _ _ _ _ _ _
    - A_ _
    - R_ _ _ _ _ _ _
  - Memory:
    - D_ _ _(i.e., RAM, volatile)
    - P_ _ _ _ _ _ _ (i.e., ROM, non-volatile)
  - Bus: collection of signals (wires)
    - D_ _ _ (Read/write data from RAM or I/O, fetch opcodes from RAM),
    - I_ _ _ _ _ _ _ _ _ _ (Fetch opcodes from ROM)

- I/O ports: physical connections between the computer and the world
  - Information enters (I ports) and exits (O ports)
  - A port: a collection of pins which can be used for either input or output
  - A pin: a place of input or output where an actual signal is connected to the microcontroller
- I/O interfaces
  - Hardware components (external to the computer, the input port) + software
  - All together perform the I/O function.

Computer Architecture

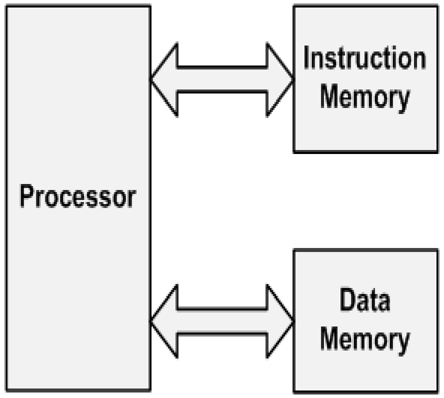
Von-Neumann

Instructions and data are stored in the same memory.



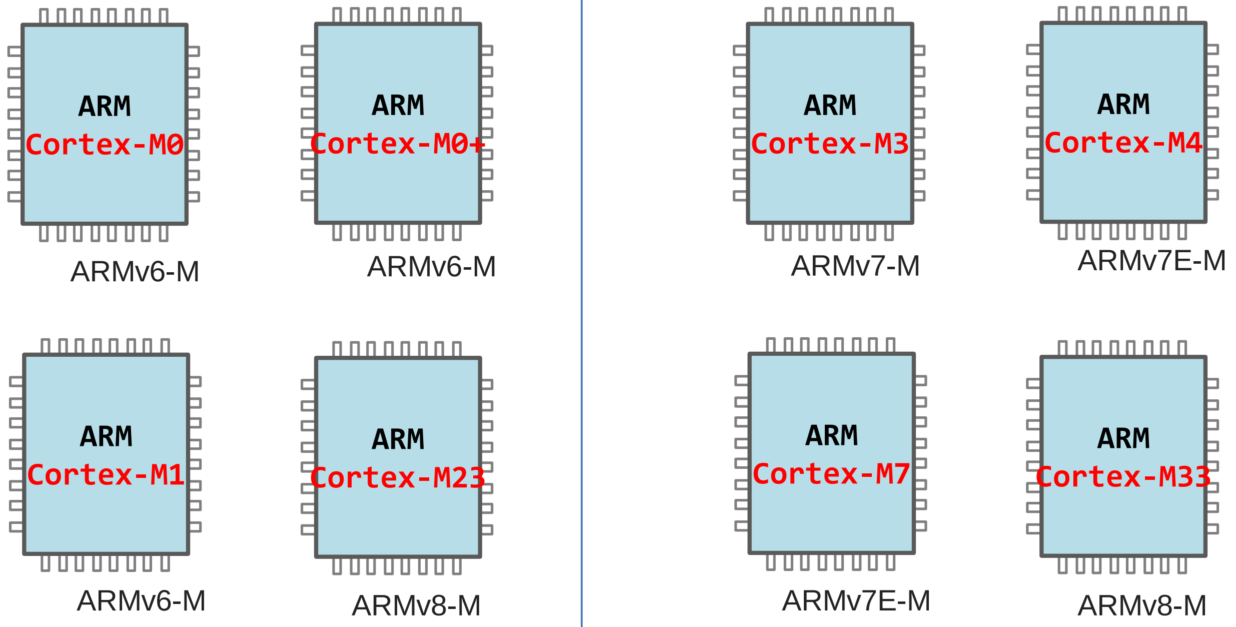
Harvard

Data and instructions are stored into separate memories.



A side bar: ARM Cortex-M Series Family

Von-Neumann **Harvard**

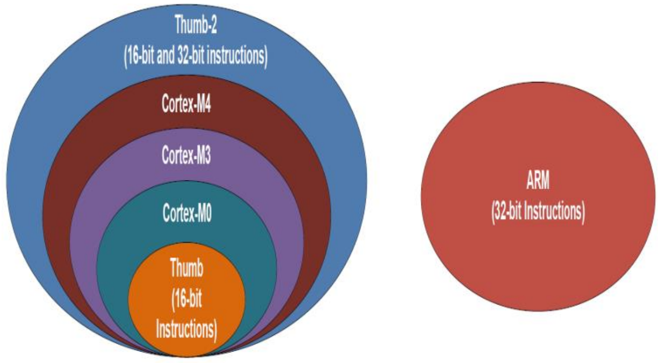


Programmer’s model for ARM **emphasizing on CPU register** (i.e., Data structures, size, Core registers…):

A programmer’s model: People program based on the given hardware/software structure and rules.

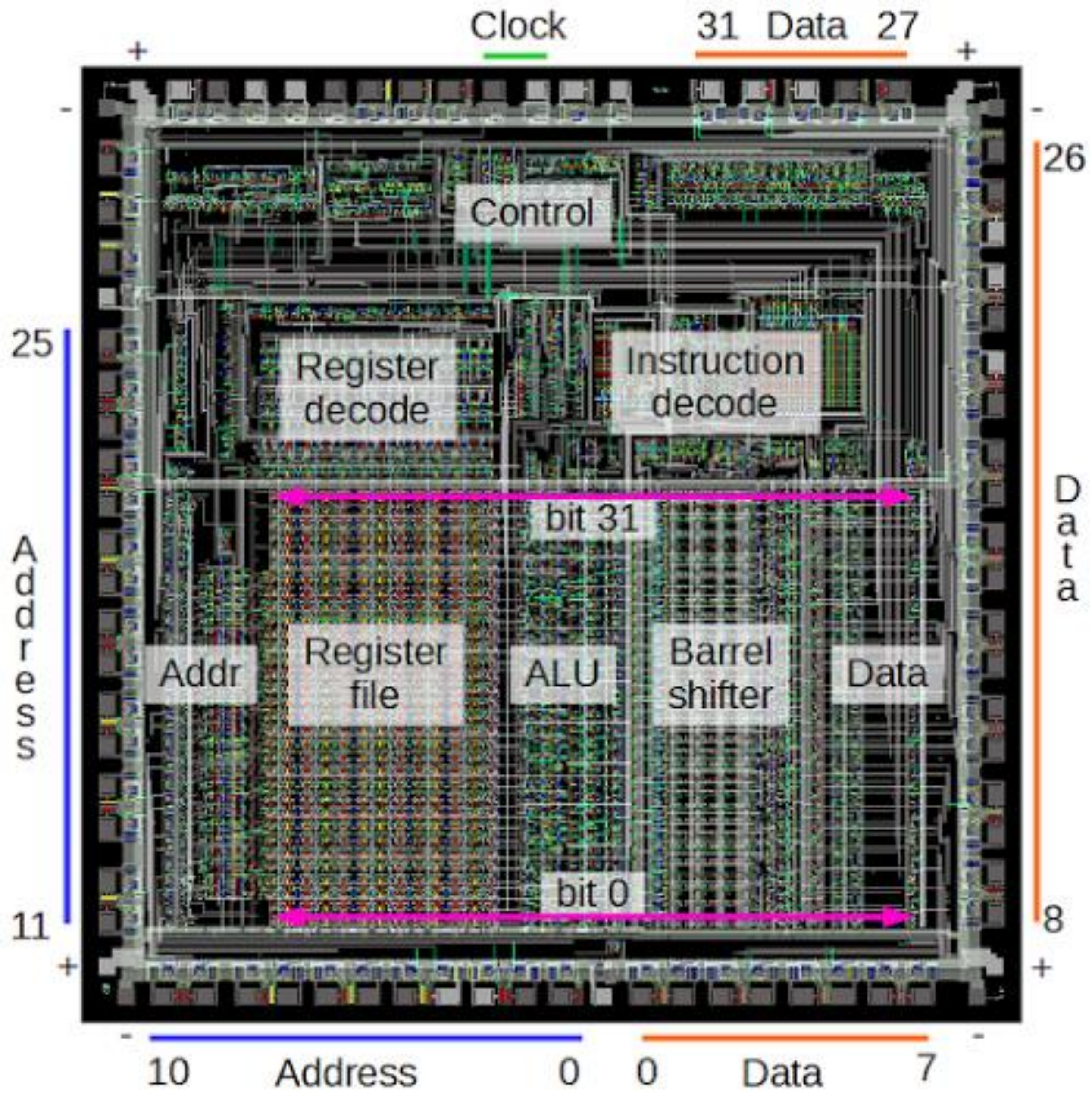
Programmer’s model (for ARM)-Brief

- Data Sizes and Instruction Sets
  - The ARM is a 32-bit architecture.
  - 1 Byte = 8 bits, 1 Word = 32 bits …
- Most ARM’s implement two instruction sets
  - 32-bit ARM Instruction Set
  - 16-bit Thumb Instruction Set
  - Or Thumb 2

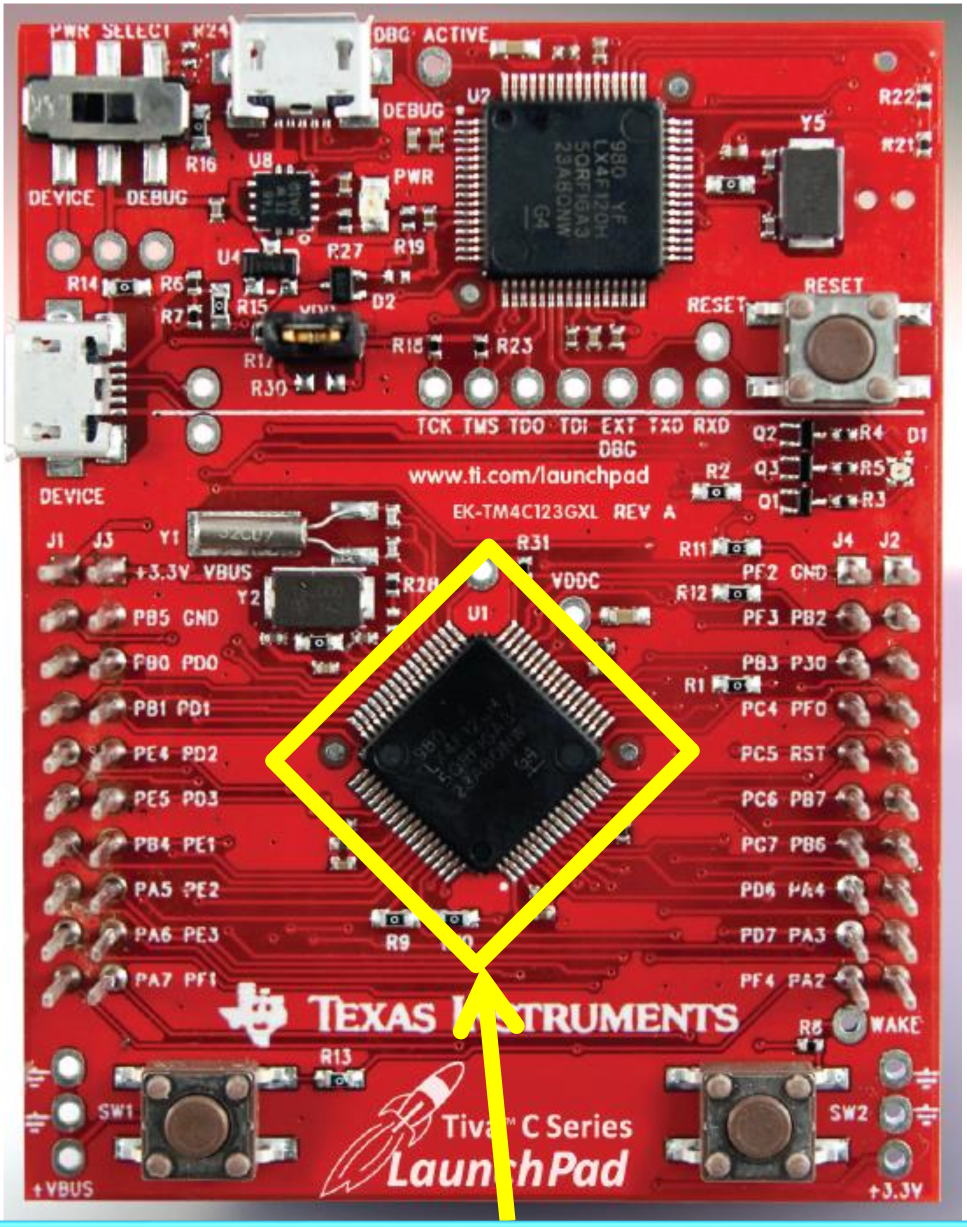


- Processor Modes
  - The ARM has seven basic operating modes, we are using User mode
- **Registers use details**

Registers?



The most important functional blocks of the ARM1 chip

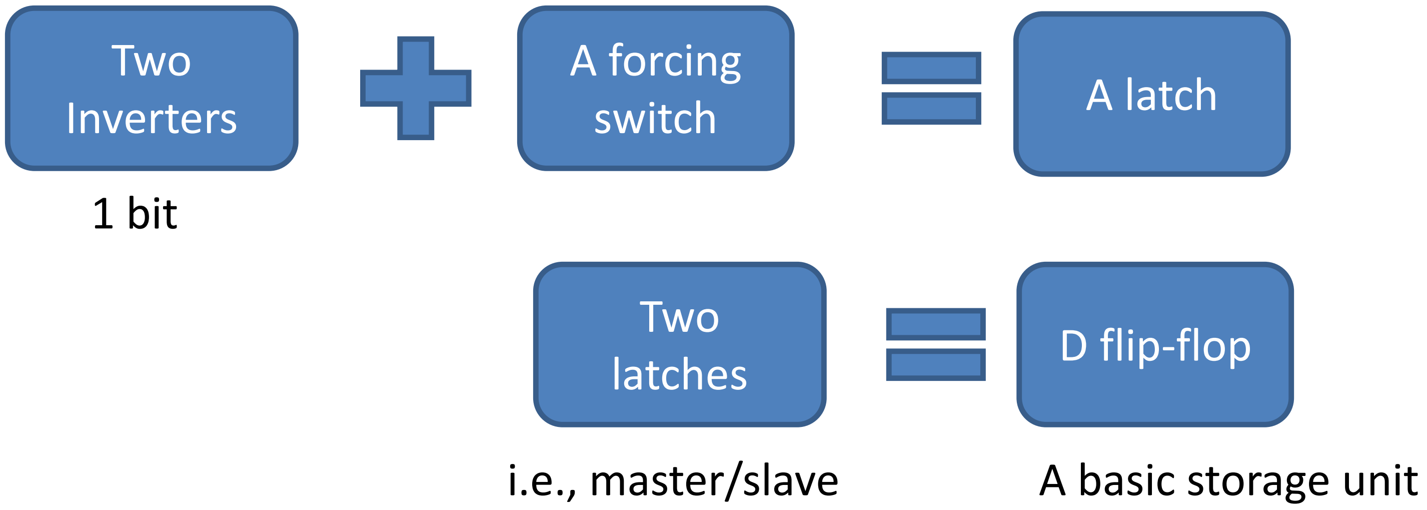


We have talked about this MCU ARM chip.
Registers are located inside of this chip

Registers?

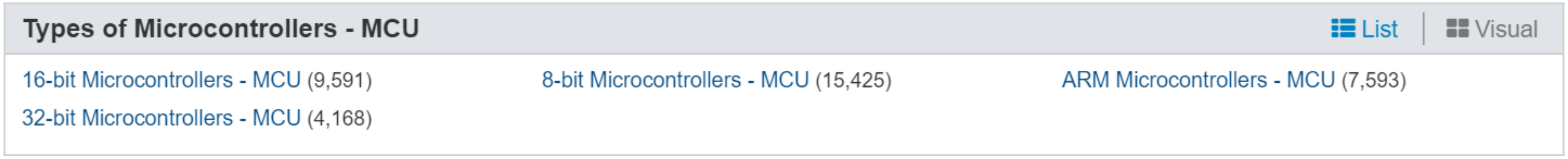
- What are they?
  - Small but the fastest, a kind of “memory” in the CPU chip
  - Temporal storage spaces for computing
- How fast (in general)?
  - More than 100 times fasters than RAM
  - 10 times than cache
  - (ROM? Not worthy to be mentioned i.t.o. speed)
- Base components for manufacturing?
  - Mostly, (D) flip-flop ← A very important component

An oversimplified description of a D flip-flop



- How many core registers?
  - Depending on a chip, but not many (countable)
- When we categorize 32-bits MCU or 16-bits MCU ....

http://www.mouser.sg/Semiconductors/Embedded-Processors-Controllers/Microcontrollers-MCU/_/N-a85i8

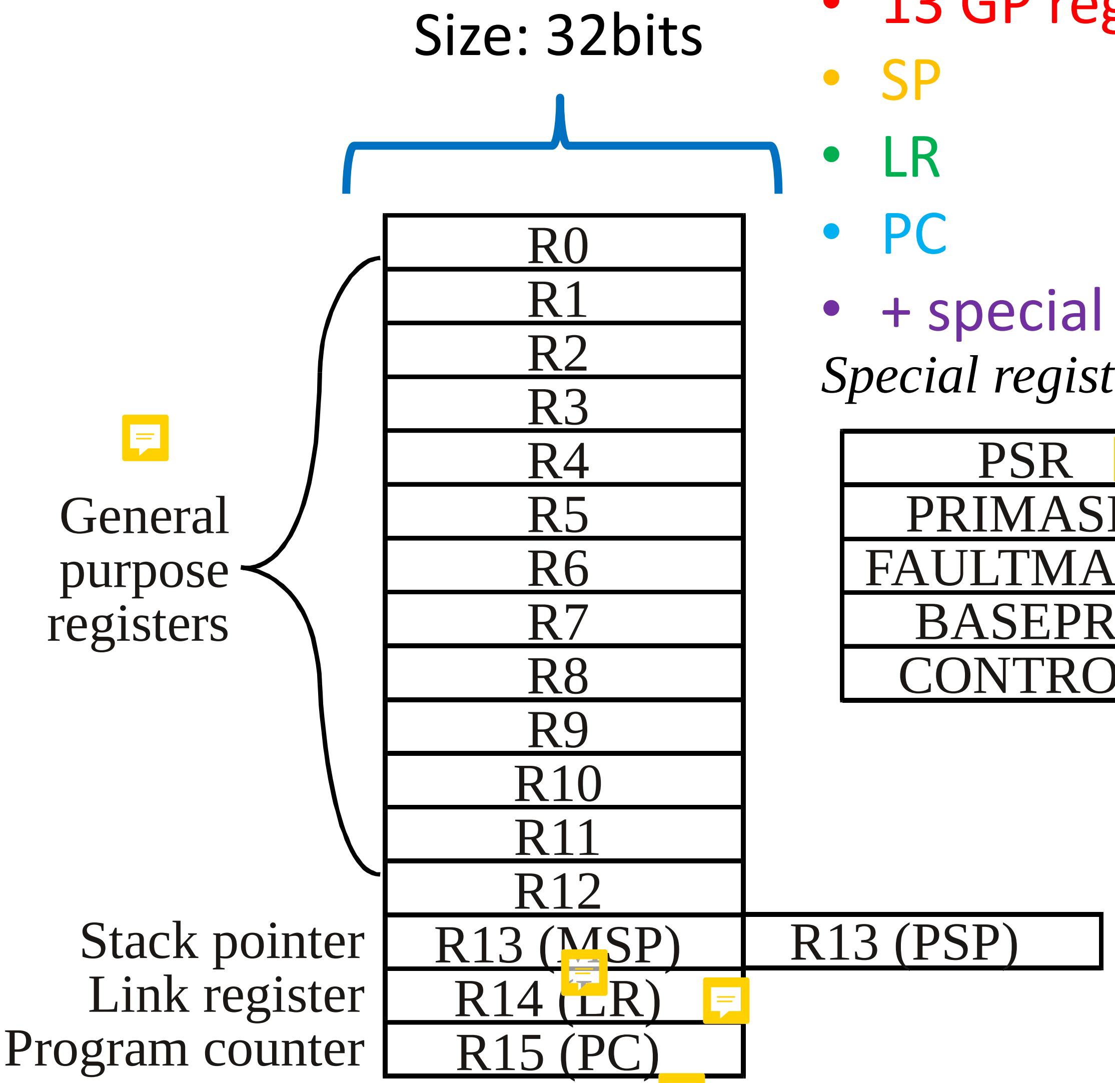

Types of Microcontrollers - MCU

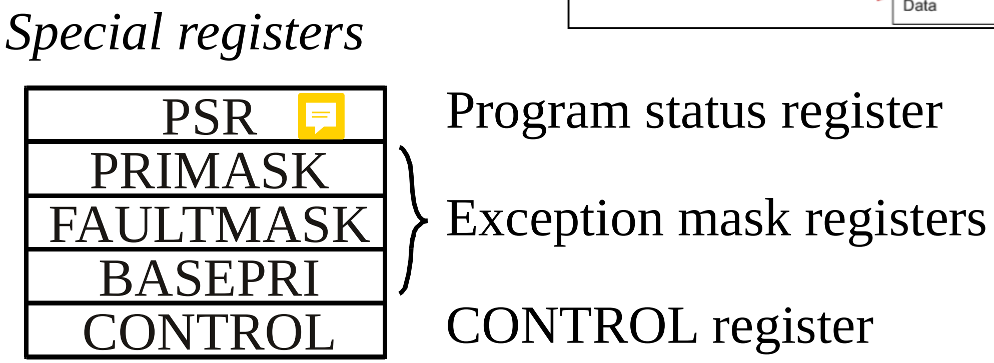
16-bit Microcontrollers - MCU (9,591)	8-bit Microcontrollers - MCU (15,425)	ARM Microcontrollers - MCU (7,593)
32-bit Microcontrollers - MCU (4,168)		

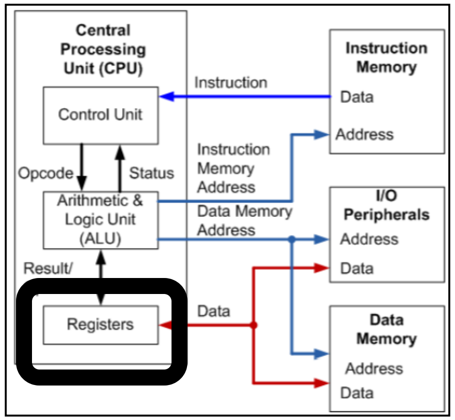
  - It refers to the size of registers and buses, or word size

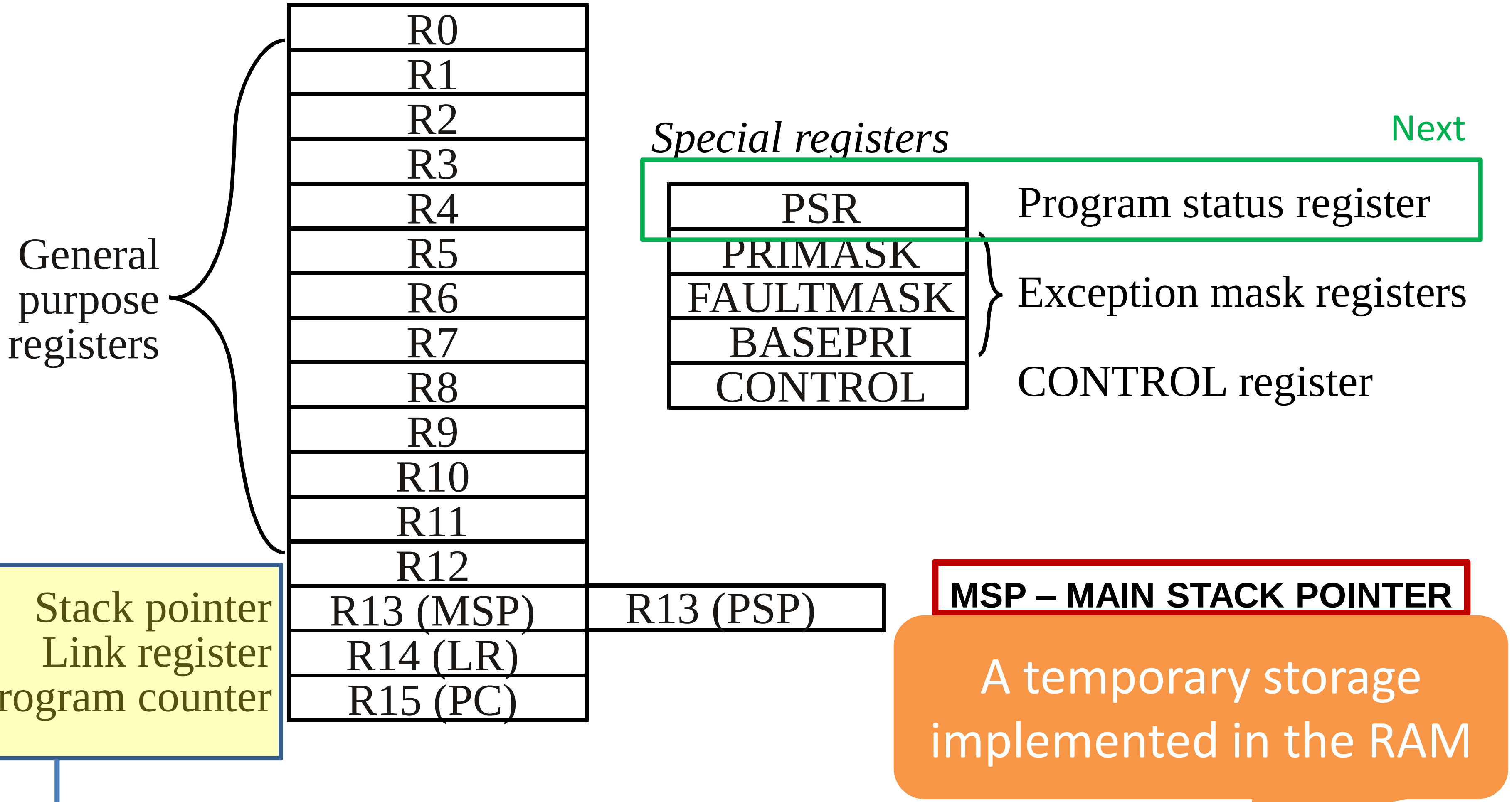
• Ex) Cortex-M4’s core register structure

- 13 GP registers
- SP
- LR
- PC
- + special registers





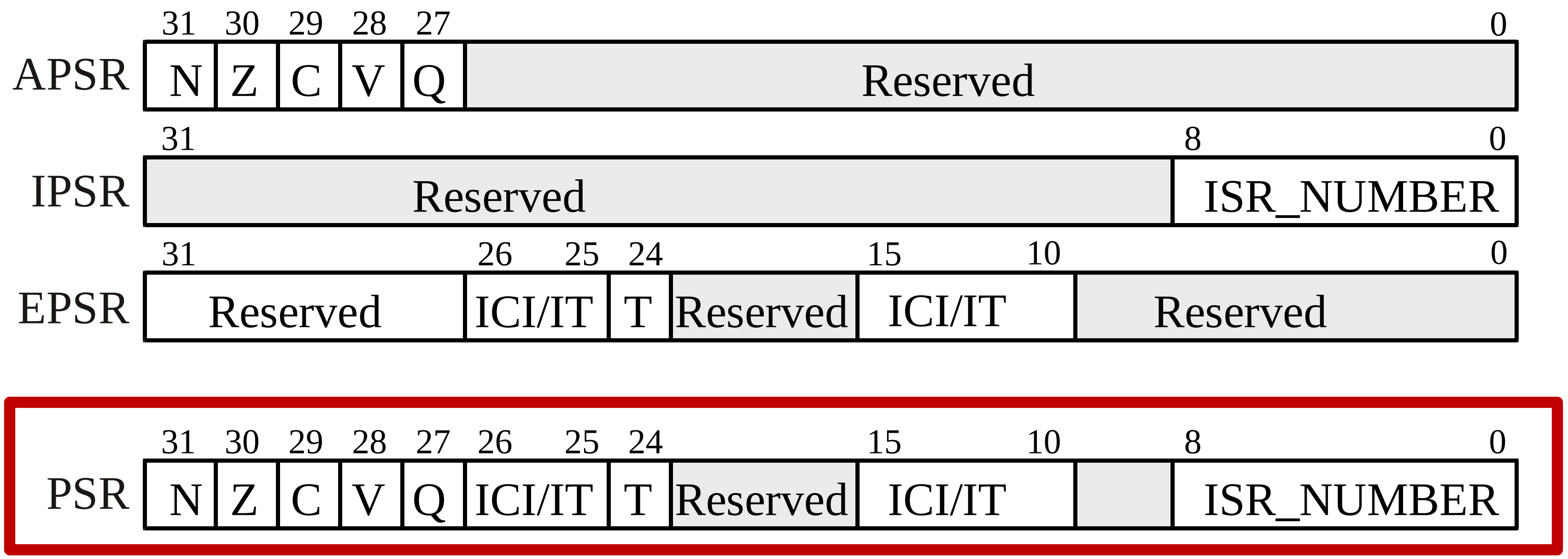




Hold

- SP: address of the top of the stack
- LR: return address of the subroutines
- PC: address of the **next** instruction to be fetched from the memory

Cortex-M4: State Register (SR)



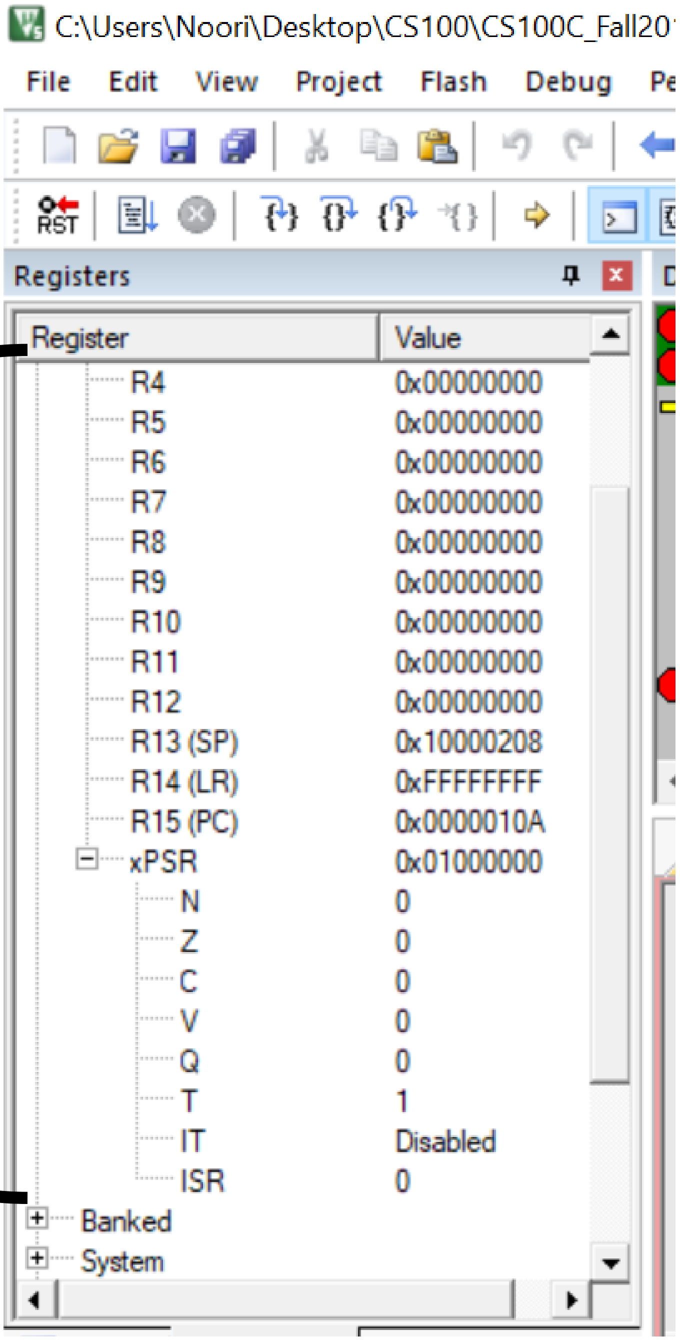
<u>PSR – PROGRAM STATUS REGISTER, COMBINATION OF</u>

- APPLICATION PSR
- INTERRUPT PSR, AND
- EXECUTION PSR

Q – SATURATION BIT
ICI – INTERRUPT CONTINUABLE INSTRUCTION
IT – IF/THEN INSTRUCTION BLOCK

Condition bits are directly tested in Assembly language (flags)

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


C:\Users\Noori\Desktop\CS100\CS100C_Fall20
File Edit View Project Flash Debug Pe
Registers

Register	Value
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x10000208
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x0000010A
xPSR	0x01000000
N	0
Z	0
C	0
V	0
Q	0
T	1
IT	Disabled
ISR	0
Banked	
System	

No addresses, placed in CPU.

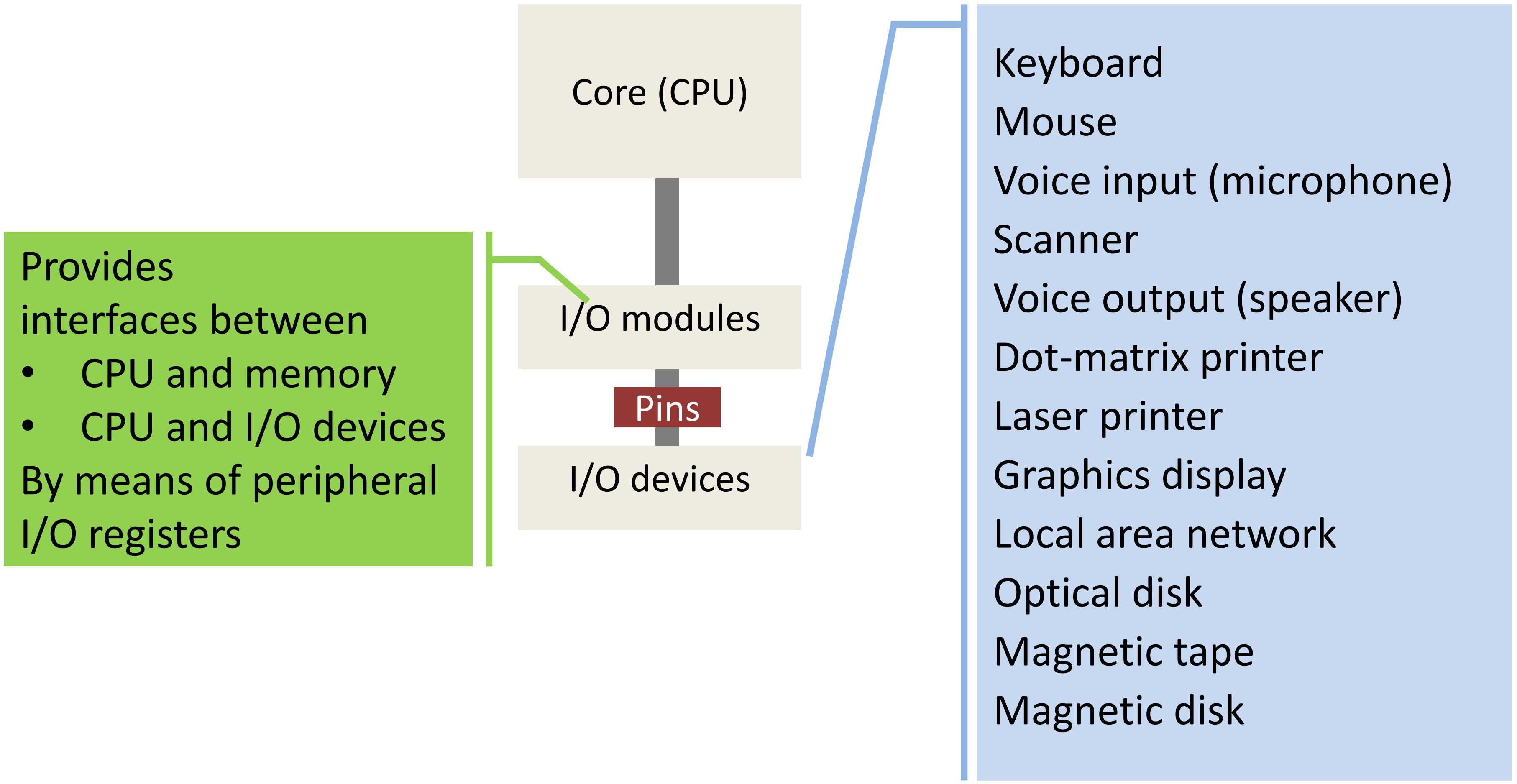
Here they are

Outside of CPU:
How does CPU access peripherals?
➔Memories and Memory map structure

Examples of I/O Devices

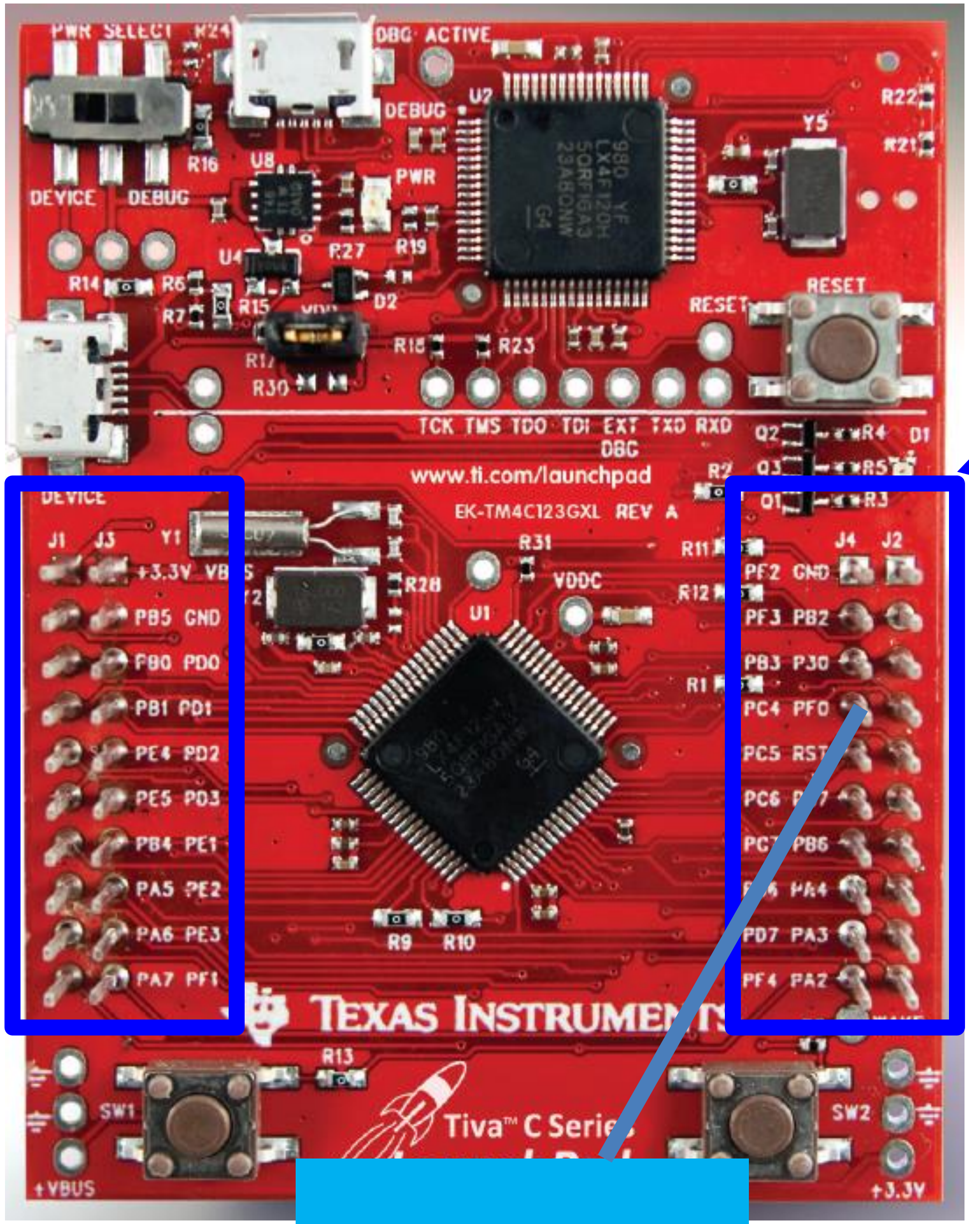
Device	Input/Output	Date Rate (Kbytes/s)
Keyboard	[redacted]	0.01
Mouse	[redacted]	0.02
Microphone	[redacted]	0.02
Scanner	[redacted]	200
Speaker	[redacted]	0.5
Laser printer	[redacted]	100
Graphics display	[redacted]	30,000
Local area network	[redacted]	200 – 20,000
Optical disk	[redacted]	500
Magnetic tape	[redacted]	2,000
Magnetic disk	[redacted]	2,000

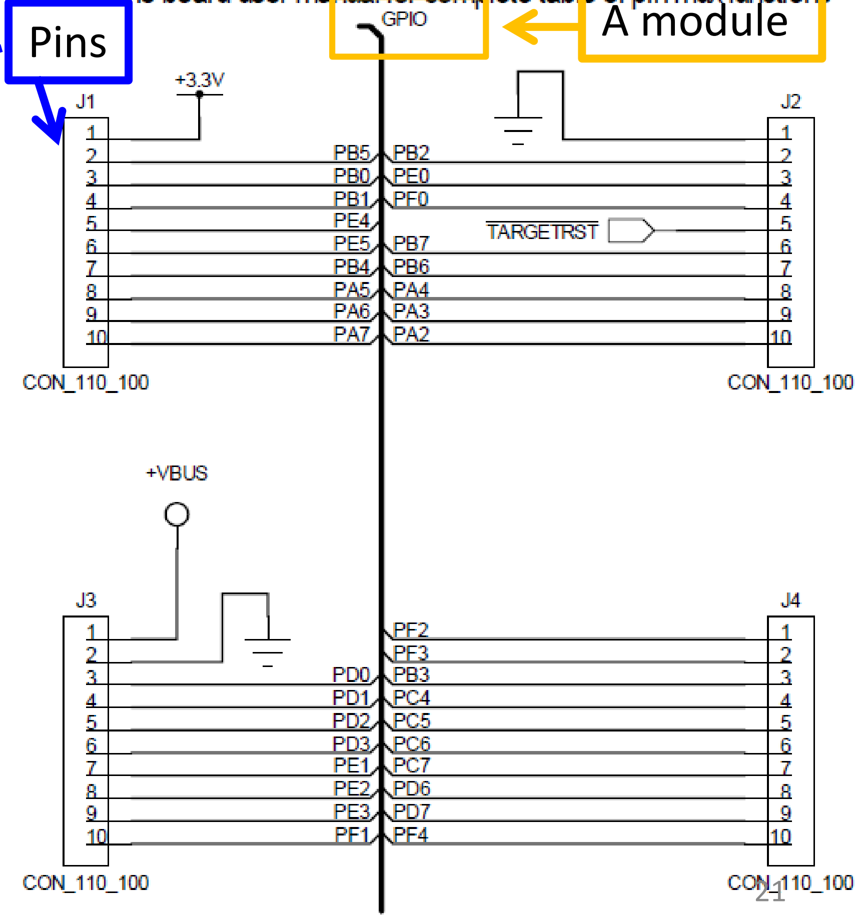
I/O Configurations (1 of 2)



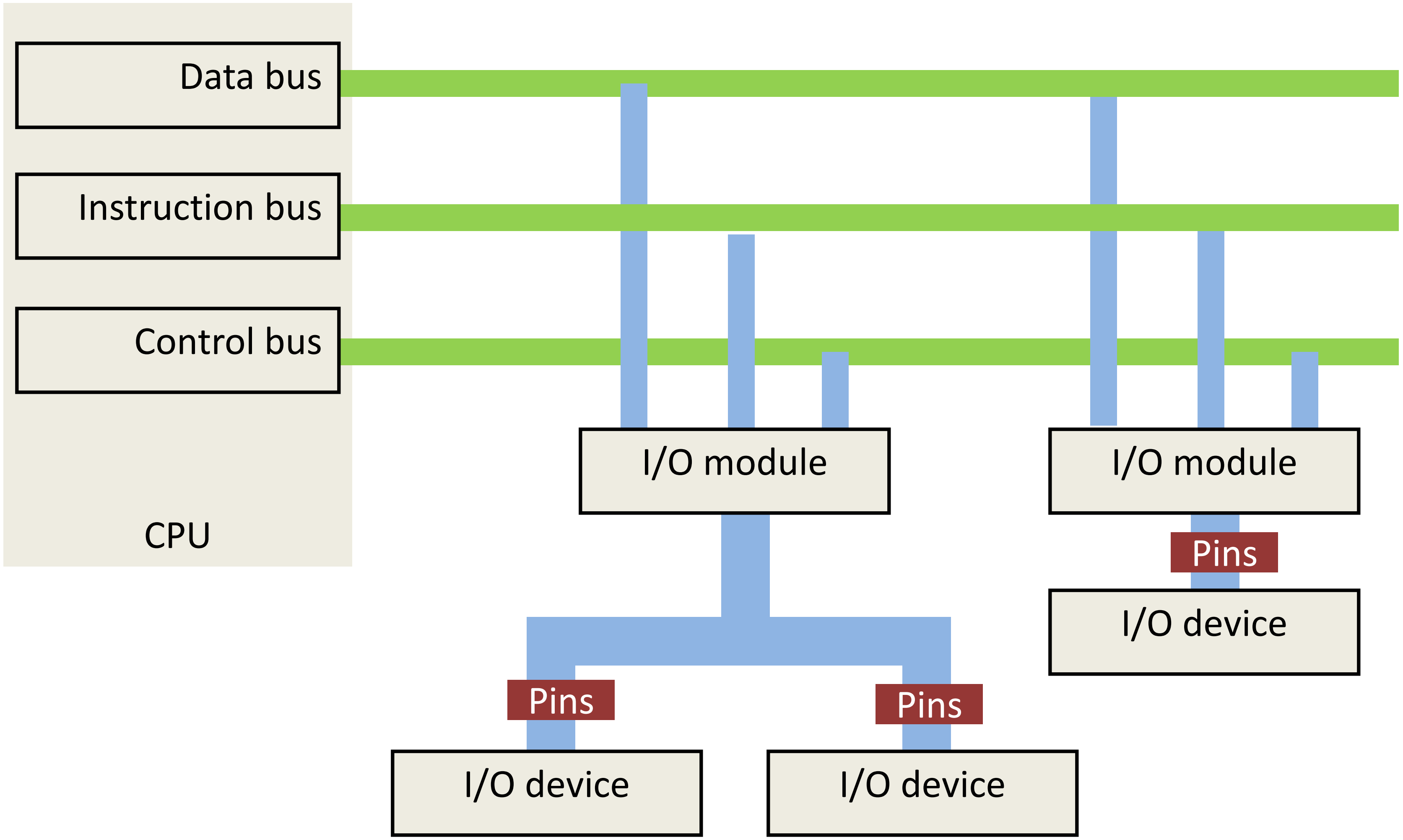
- Vol2, page 76
- Vol1, page 145
- LaunchPadUsersManual.pdf page 20

J1 and J2 provide compatability with
Booster Packs designed for MSP430 Launchpad
J3 and J4 sit 100 mils inside J1 and J2 to provide
extended functions specific to this board.
See the board user manual for complete table of pin mux functions





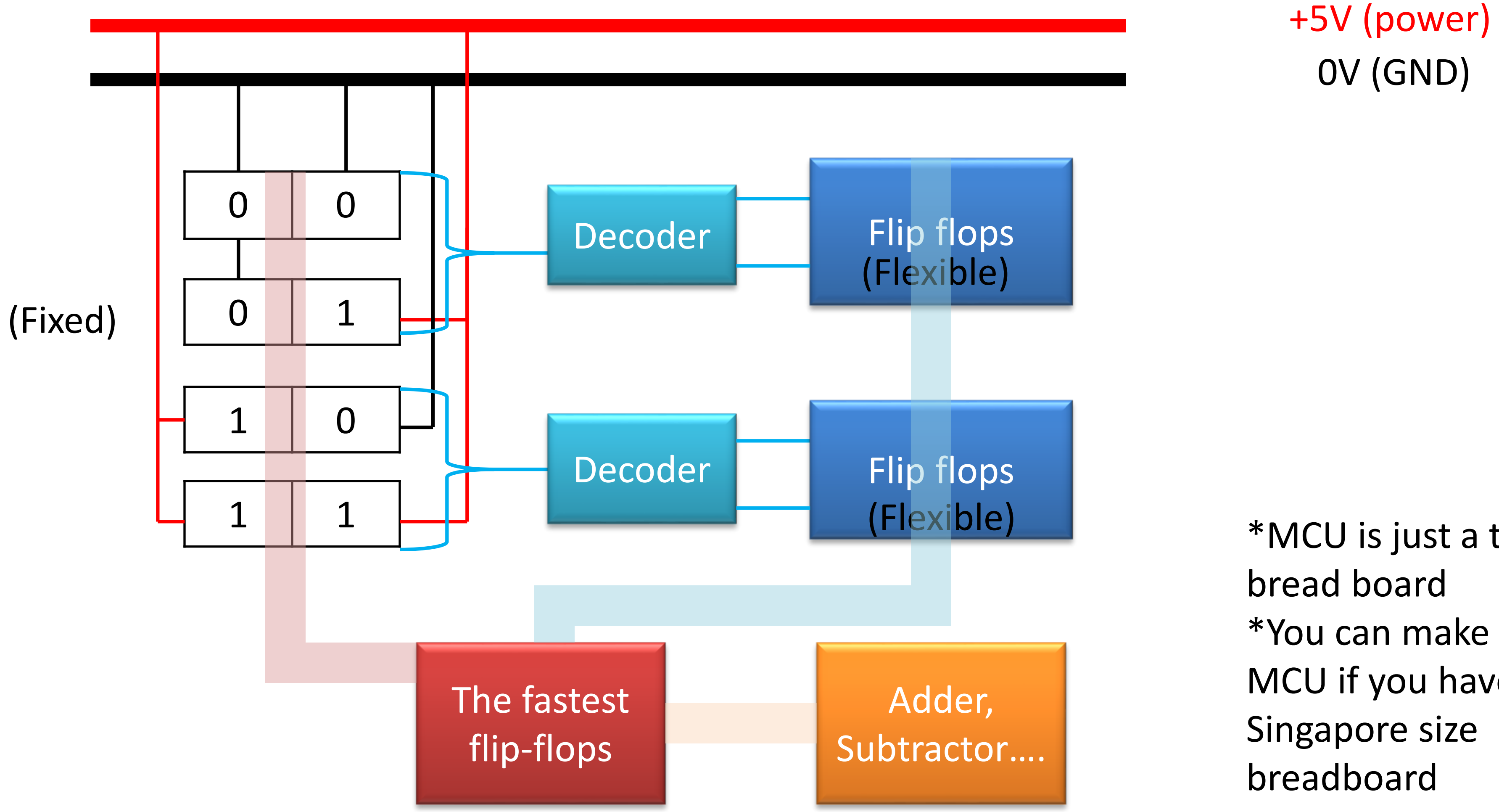
I/O Configurations (2 of 2)



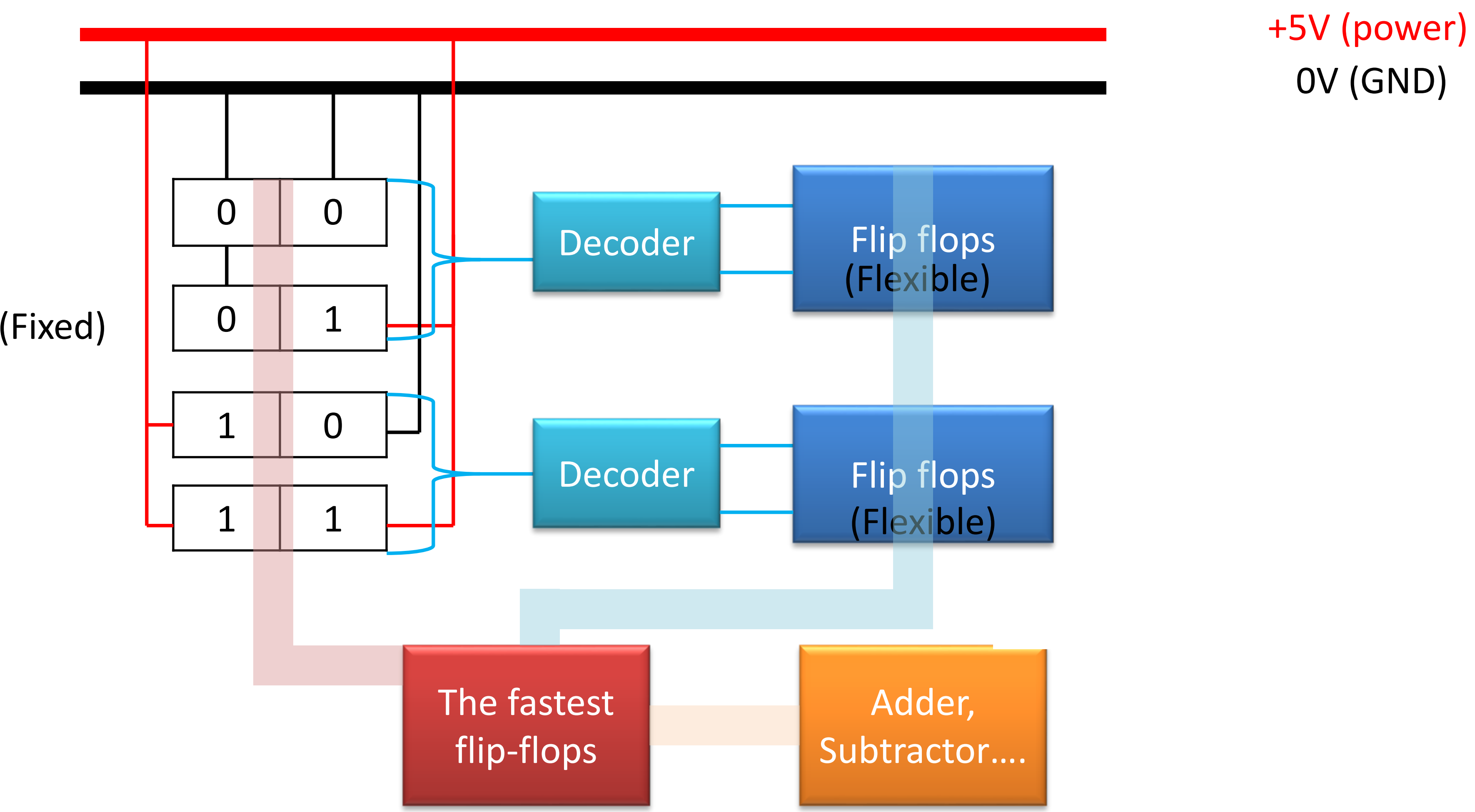
Basic concepts of I/O

- Input signal mapping
  - Zero volt to logic level false (0)
  - 3.3 volt to logic level true (1)
- Output signal mapping: same
- Four types of I/O interfaces
  - Parallel: a bunch of bits at the same time
  - Serial: one bit at a time on a single line
  - Time: data are encoded as a period, freq, pulse width, or phase shift
  - Analog: data are decoded as an electrical voltage, current, or power
- We focus on **the Parallel interface**: <u>GPIO ports</u>

Recap: Decoder, Flip-flops, Adder, subtractor…. (logic gate implementations)
You have a breadboard and many IC chips



*MCU is just a tiny bread board
*You can make a MCU if you have a Singapore size breadboard


+5V (power)
0V (GND)
(Fixed)
0
0
0
1
1
0
1
1
Decoder
Decoder
Flip flops (Flexible)
Flip flops (Flexible)
The fastest flip-flops
Adder, Subtractor….

Types of I/O modules

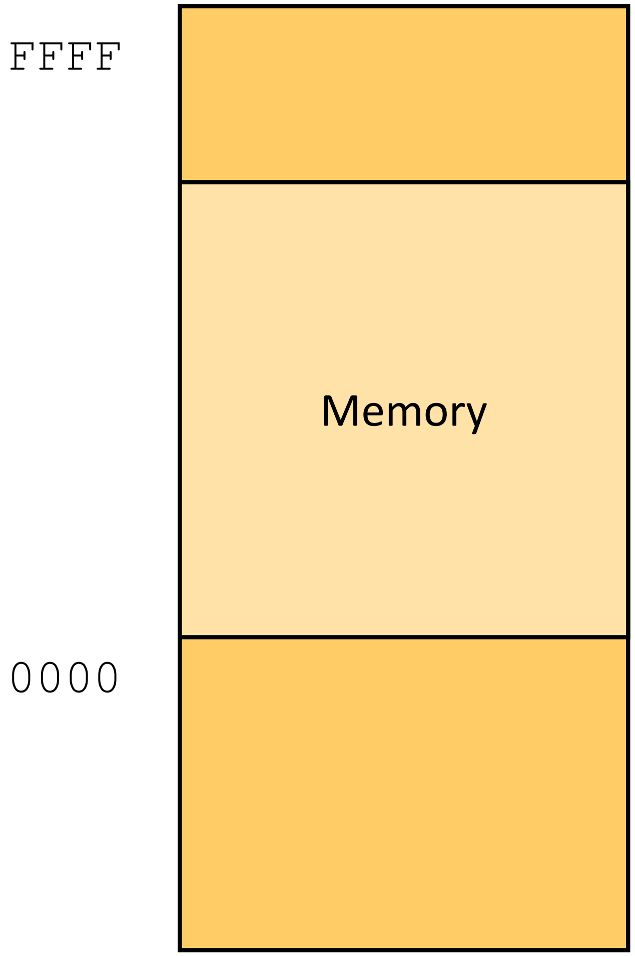
- Two possibilities
  - Memory-mapped I/O
  - I/O-mapped I/O

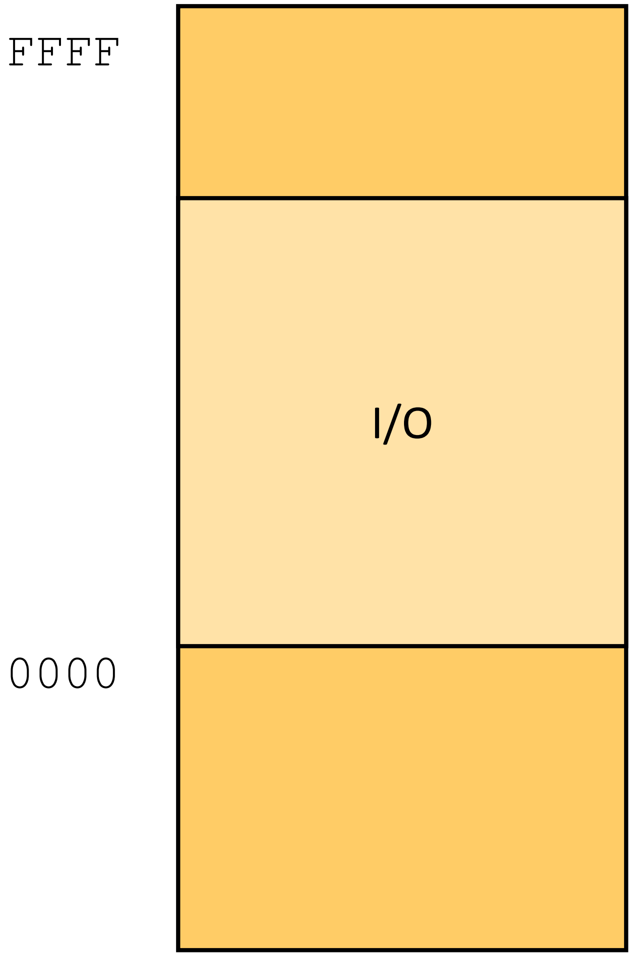
I/O-Mapped I/O

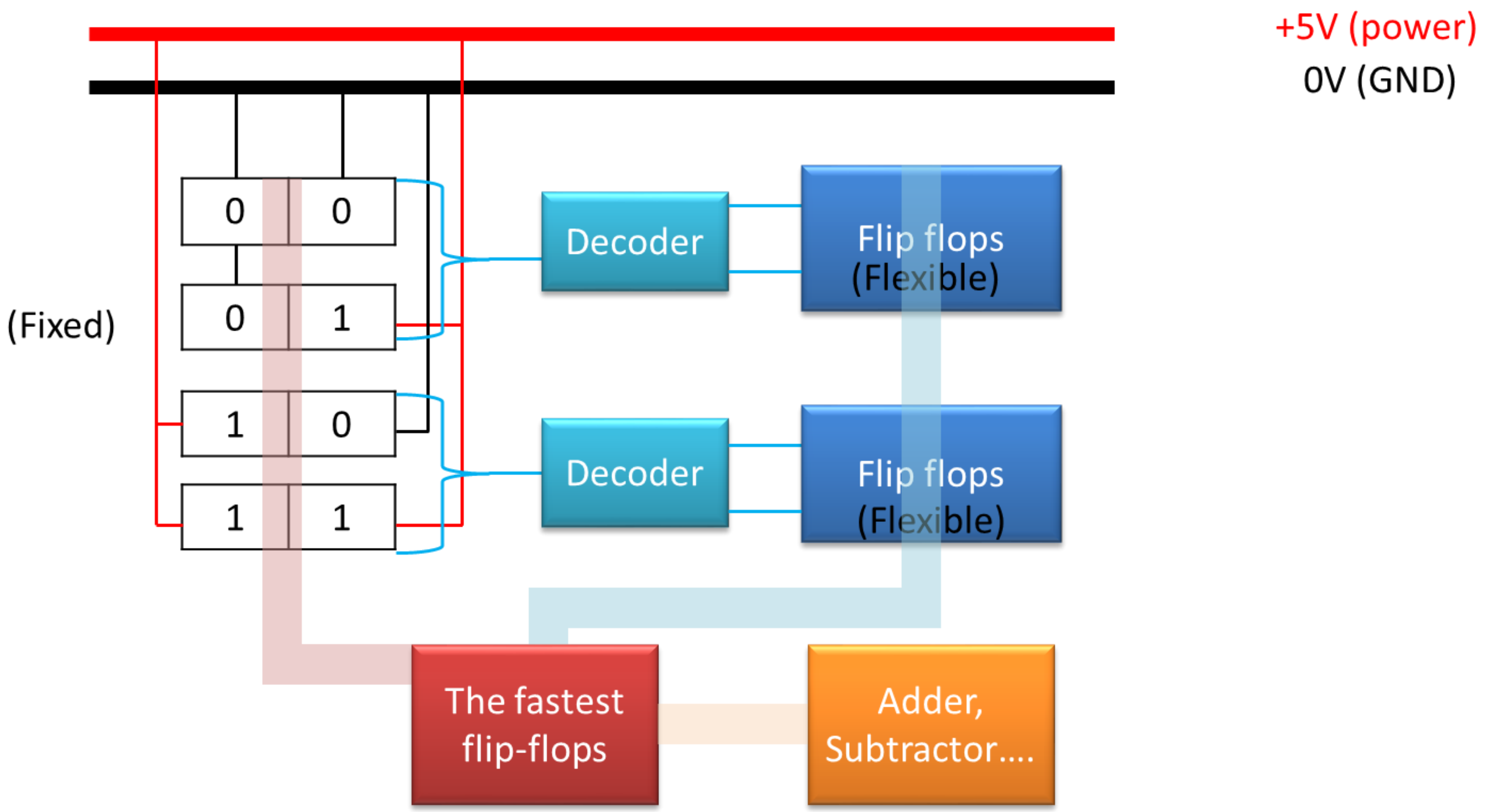
- Memory and I/O…
  - Occupy different “spaces”
  - Are accessed by unique instructions
- Differentiated by instructions
  - i.e.,8086

```
IN  AL, 19H     ;8-bits are saved to AL from I/O port 19H.
IN  EAX, DX     ;32-bits are saved to EAX.
OUT DX, EAX     ;32-bits are written to port DX from EAX.
OUT 19H, AX     ;16-bits are written to I/O port 0019H.
```

Memory Maps (I/O mapped I/O)



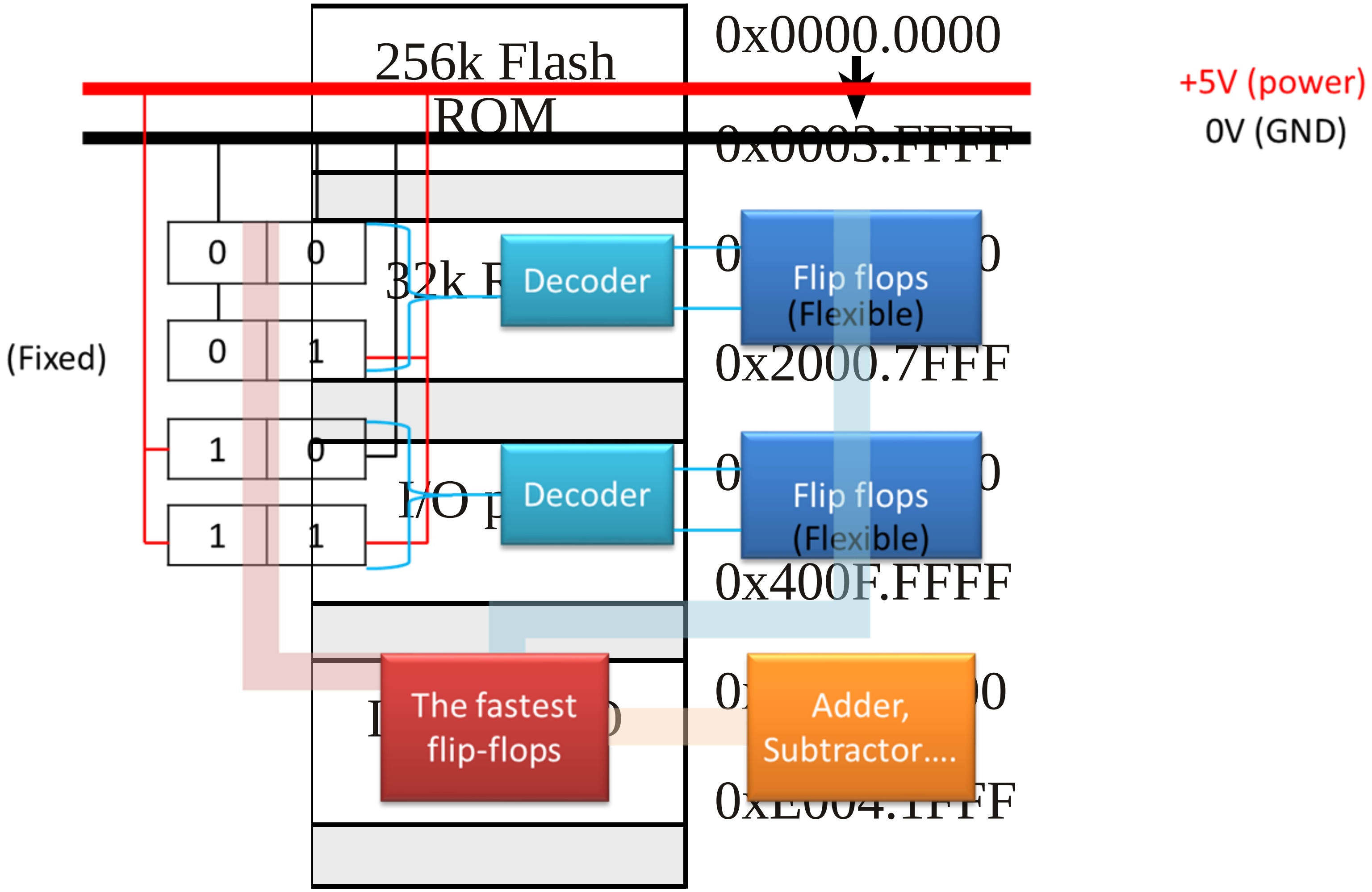



+5V (power)
0V (GND)
(Fixed)
0
0
0
1
1
0
1
1
Decoder
Decoder
Flip flops (Flexible)
Flip flops (Flexible)
The fastest flip-flops
Adder, Subtractor….

Memory-Mapped I/O

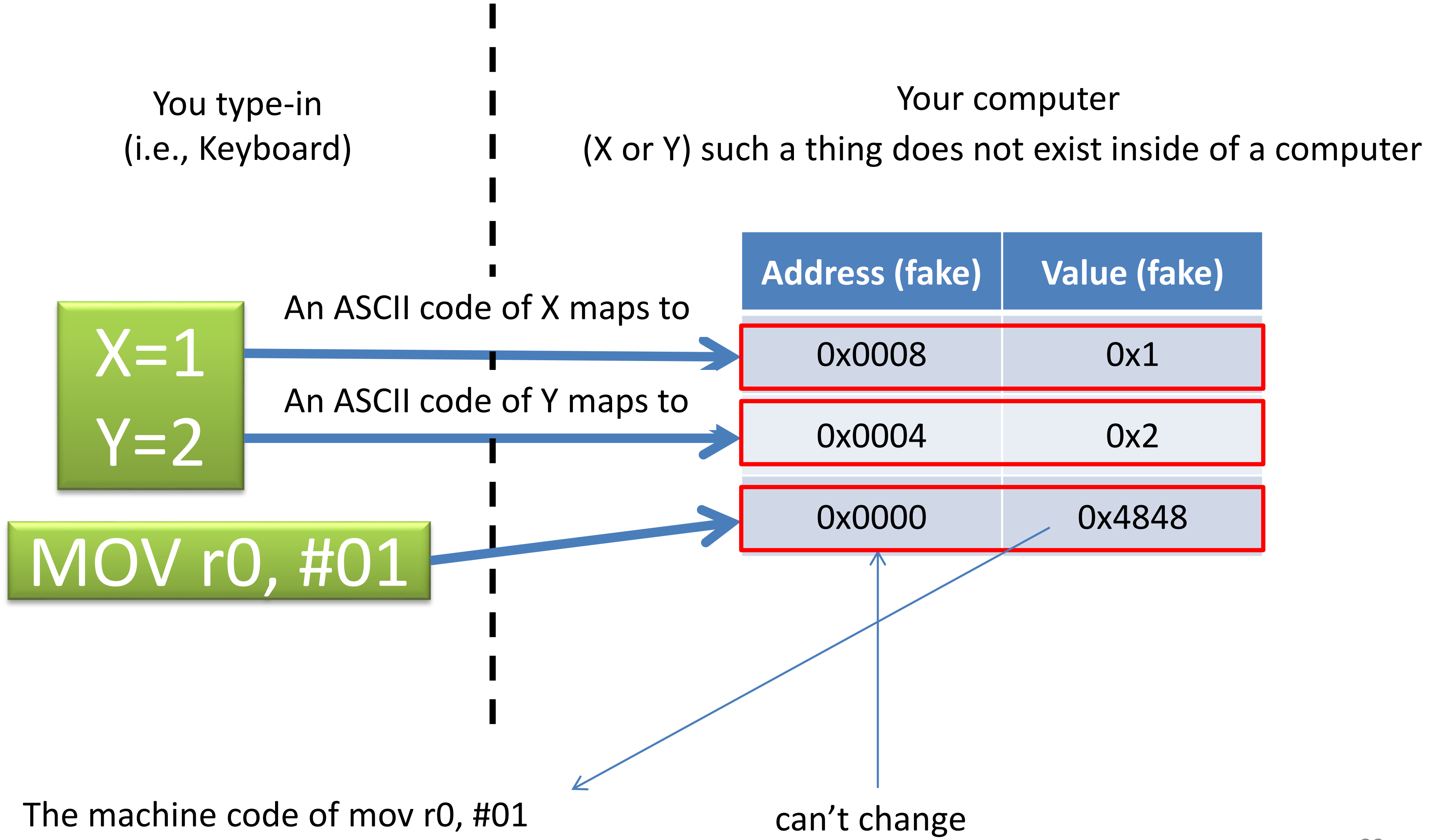
- Memory and I/O…
  - reside in the same “space”
  - are accessed in the same manner
- Differentiated only by their addresses

Memory Map (Memory-Mapped I/O)



Remember? Our MCU is Memory-Mapped I/O

A simplest view of memory

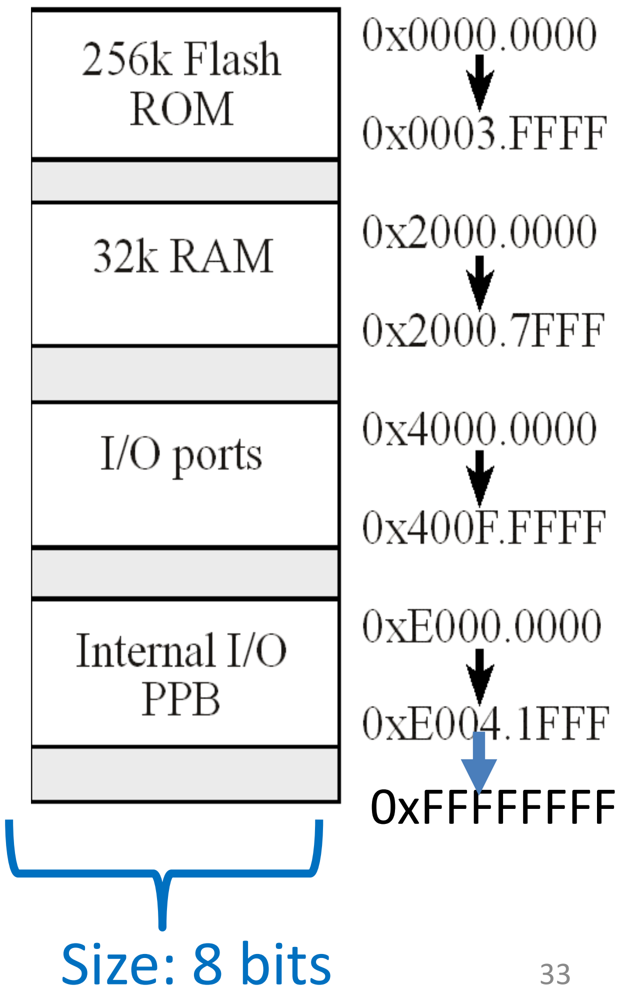


You type-in
(i.e., Keyboard)
Your computer
(X or Y) such a thing does not exist inside of a computer
X=1
Y=2
MOV r0, #01
An ASCII code of X maps to
An ASCII code of Y maps to
The machine code of mov r0, #01
can’t change

Address (fake)	Value (fake)
0x0008	0x1
0x0004	0x2
0x0000	0x4848

Memory map: TM4C123

- Memory map is arranged as a series of "locations"
  - e.g. the memory location at address `0x080001B0` contains the byte value `0x70`, i.e., 112
- The number of locations in memory is limited
  - e.g. 4 GB of RAM, 1 Gigabyte (GB) = 2^{30} bytes
  - 2^{32} locations ➔ 4,294,967,296 locations!
- Address is represented with 32-bit size
  - 0x00000000 (the lowest memory location.)
  - 0xFFFFFFFF (the highest memory location)
- Byte addressable memory:
  - A unique address for each byte
  - Each location holds 8 bits of information



Little/Big Endian: ways to store data

STORING 0X03E8

2 bytes info:
03 and E8

Address	Data
0x2000.0850	0x03
0x2000.0851	0xE8

Big Endian

Address	Data
0x2000.0850	0xE8
0x2000.0851	0x03

Little Endian

STORING 0X12345678

Address	Data
0x2000.0850	0x12
0x2000.0851	0x34
0x2000.0852	0x56
0x2000.0853	0x78

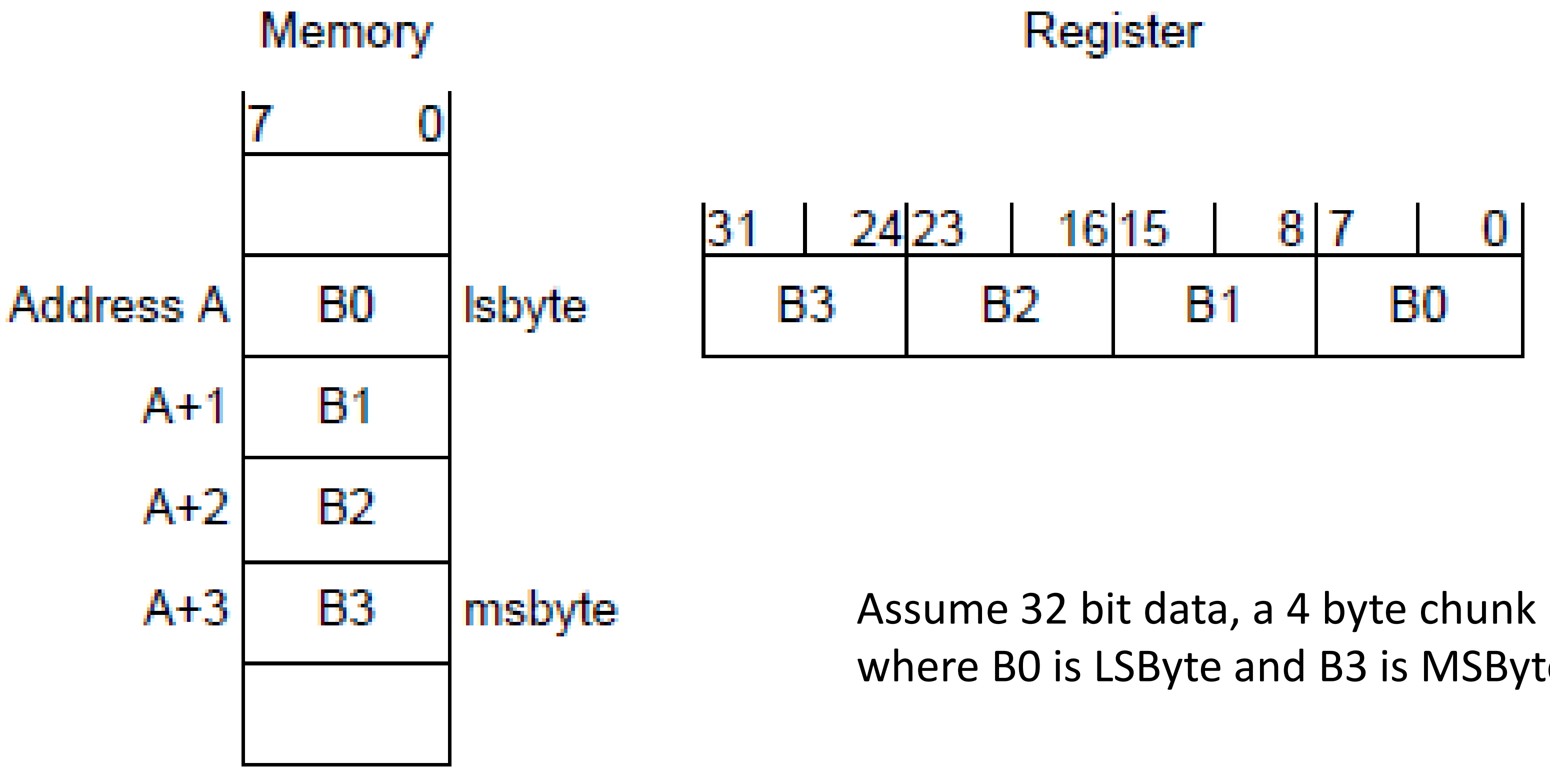
Big Endian

Address	Data
0x2000.0850	0x78
0x2000.0851	0x56
0x2000.0852	0x34
0x2000.0853	0x12

Little Endian

- **ENDIANESS:**
  - BIG ENDIAN – MSB STORED IN SMALLEST ADDRESS
  - **LITTLE ENDIAN – LSB STORED IN SMALLEST ADDRESS**

Little endian memory access



Assume 32 bit data, a 4 byte chunk where B0 is LSByte and B3 is MSByte

LE/BE examples

Figure out followings:

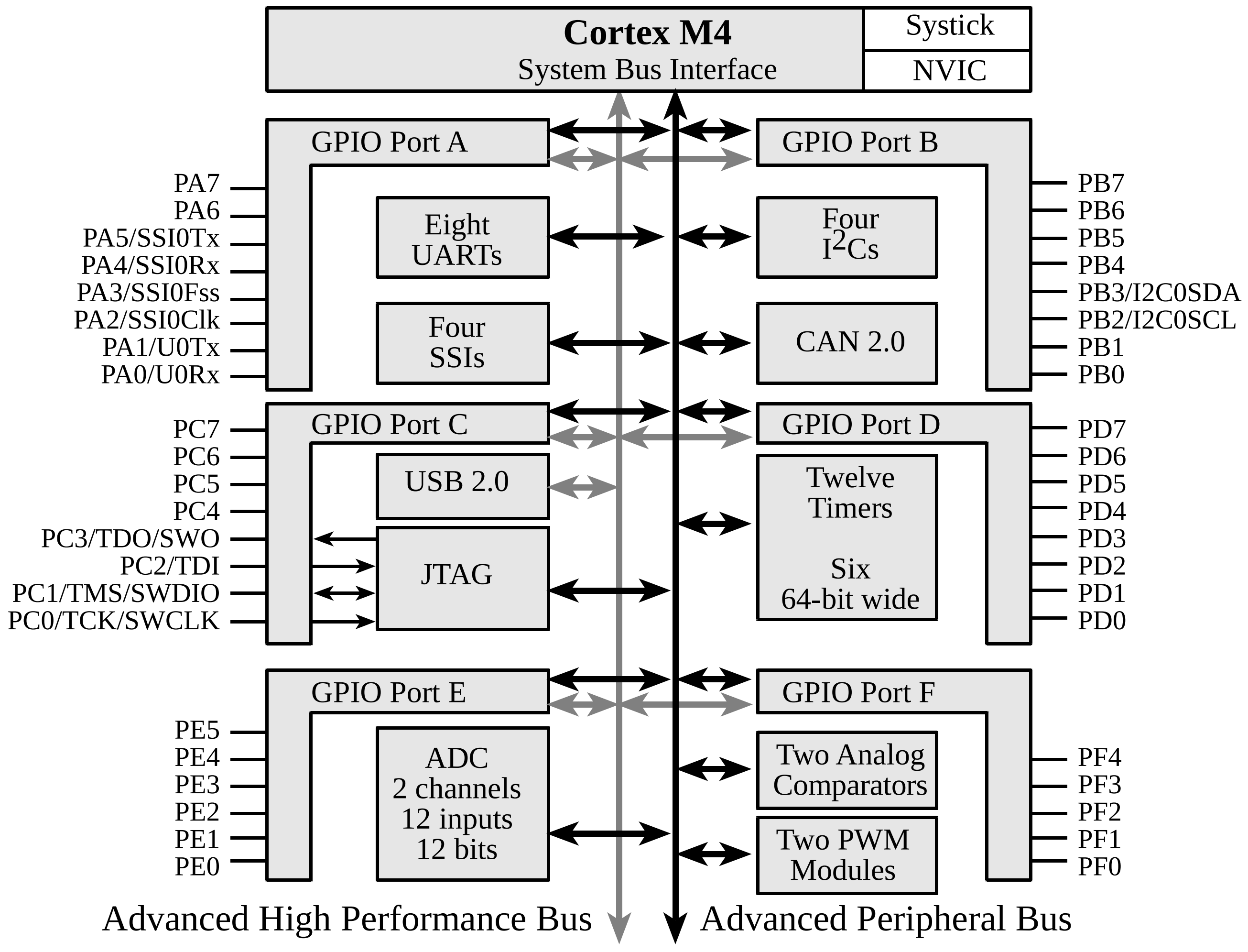
1. The number of bytes (or bits) to store each of the following variables;
2. How the variables are stored for both big and little Endian;

- char x = -120;
- short x = 0x82BA;
- int x[2];

  x[0] = 0x9381DC7E;

  x[1] = 0x274B1A9D;

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 Characteristics

- Can be employed as an n-bit *parallel* interface: Set AFSEL to 0
  - Pins also provide *alternative functions* (Set AFSEL to 1):

• UART	Universal Asynchronous Receiver/Transmitter
• SSI	Synchronous Serial Interface
• I^2C	Inter-Integrated Circuit
• Timer	Periodic interrupts, input capture, and output compare
• PWM	Pulse Width Modulation
• ADC	Analog to Digital Converter, measure analog signals
• Analog	Compare two analog signals comparator
• QEI	Quadrature Encoder Interface
• USB	Universal Serial Bus
• Ethernet	High speed network
• CAN	Controller Area Network

- UART: used for serial, asynchronous, and simultaneous communication
- SSI or SPI (Serial peripheral interface): used for medium-speed I/O devices such as graphic display
- I^2C: a simple I/O bus to interface low speed peripheral devices
- PWM: used to apply variable power to a motor interfaces or to create a DAC

- ADC: important in data acquisition systems and used to measure the amplitude of analog signals.
- USB: a high speed serial communication channel
- Ethernet: used to bridge the MCU to the Internet or a local area network.
- CAN: creates a high-speed communication channel between microcontrollers

Software processing overview (brief)

- What is a C program?
  - Simply just a text file ≠ understandable
    - Contains only human readable characters
    - Can be opened with notepad
  - Computer stores characters via ASCII code mapping
    - Every character is represented by bits

- A heart of computers (the generalization of computing process)

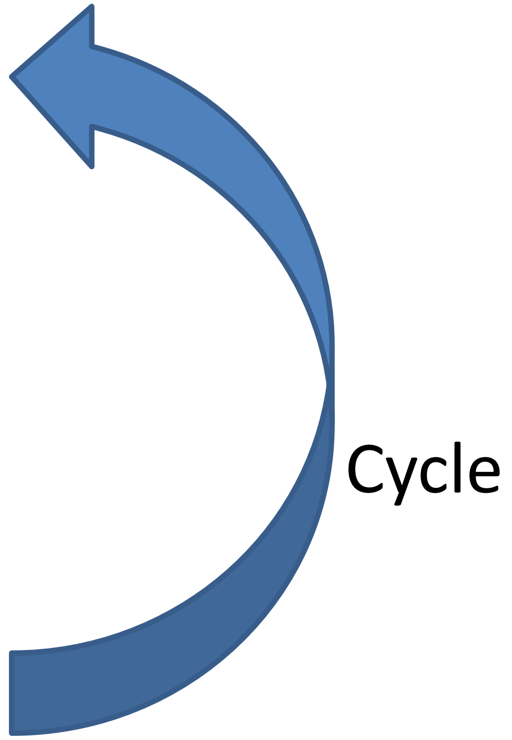
CU

1. Fetch: instructions
   - From memory into CU
2. Decode: split instructions
   - Opcode, Operand..
   - Pass values to ALU

ALU

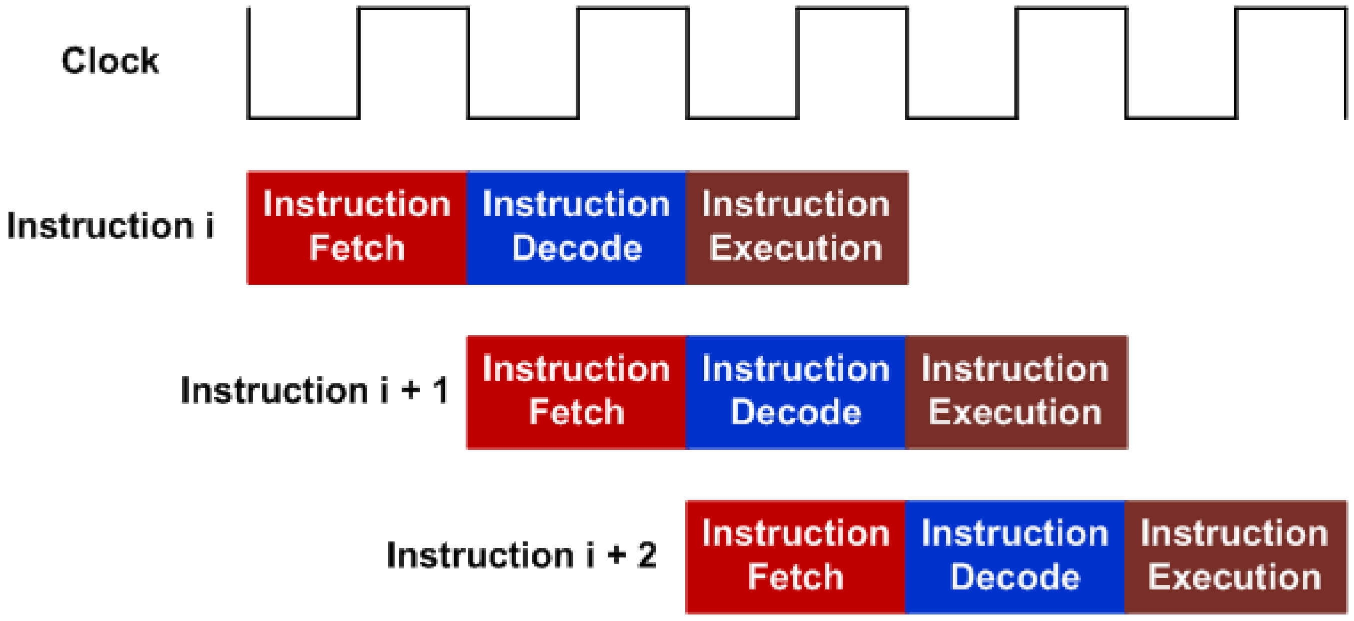
3. Execute: perform the operation

(4. Write back)



Three-state pipeline: Fetch, Decode, Execution (Cortex-M3)

- **Pipelining** allows hardware resources to be fully utilized
- One 32-bit instruction or two 16-bit instructions can be fetched.



Pipeline of 32-bit instructions

Levels of Program Code

C Program

```
int main(void){
  int i;
  int total = 0;
  for (i = 0; i < 10; i++) {
    total += i;
  }
  while(1); // Dead loop
}
```

Compile →

Assembly Program

```
        MOVS r1, #0
        MOVS r0, #0
        B    check
loop    ADD  r1, r1, r0
        ADDS r0, r0, #1
check   CMP  r0, #10
        BLT  loop
self    B    self
```

Assemble →

Machine Program

```
0010000100000000
0010000000000000
1110000000000001
0100010000000001
0001110001000000
0010100000001010
1101110011111011
1011111100000000
1110011111111110
```

- **High-level language**
  - Level of abstraction closer to problem domain
  - Provides for productivity and portability
- **Assembly language**
  - Textual representation of instructions
- **Hardware representation**
  - Binary digits (bits)
  - Encoded instructions and data

C Code

```
int main(void){
    int a = 0;
    int b = 1;
    int c;
    c = a + b;
    return 0;
}
```

compiler

Assembly Code

```
MOVS r1, #0x00      ; int a = 0
MOVS r2, #0x01      ; int b = 1
ADDS r3, r1, r2     ; c = a + b
MOVS r0, 0x00       ; set return value
BX lr               ; return
```

assembler

Machine Code

```
0010000100000000
0010001000000001
0001100010001011
0010000000000000
0100011101110000
```

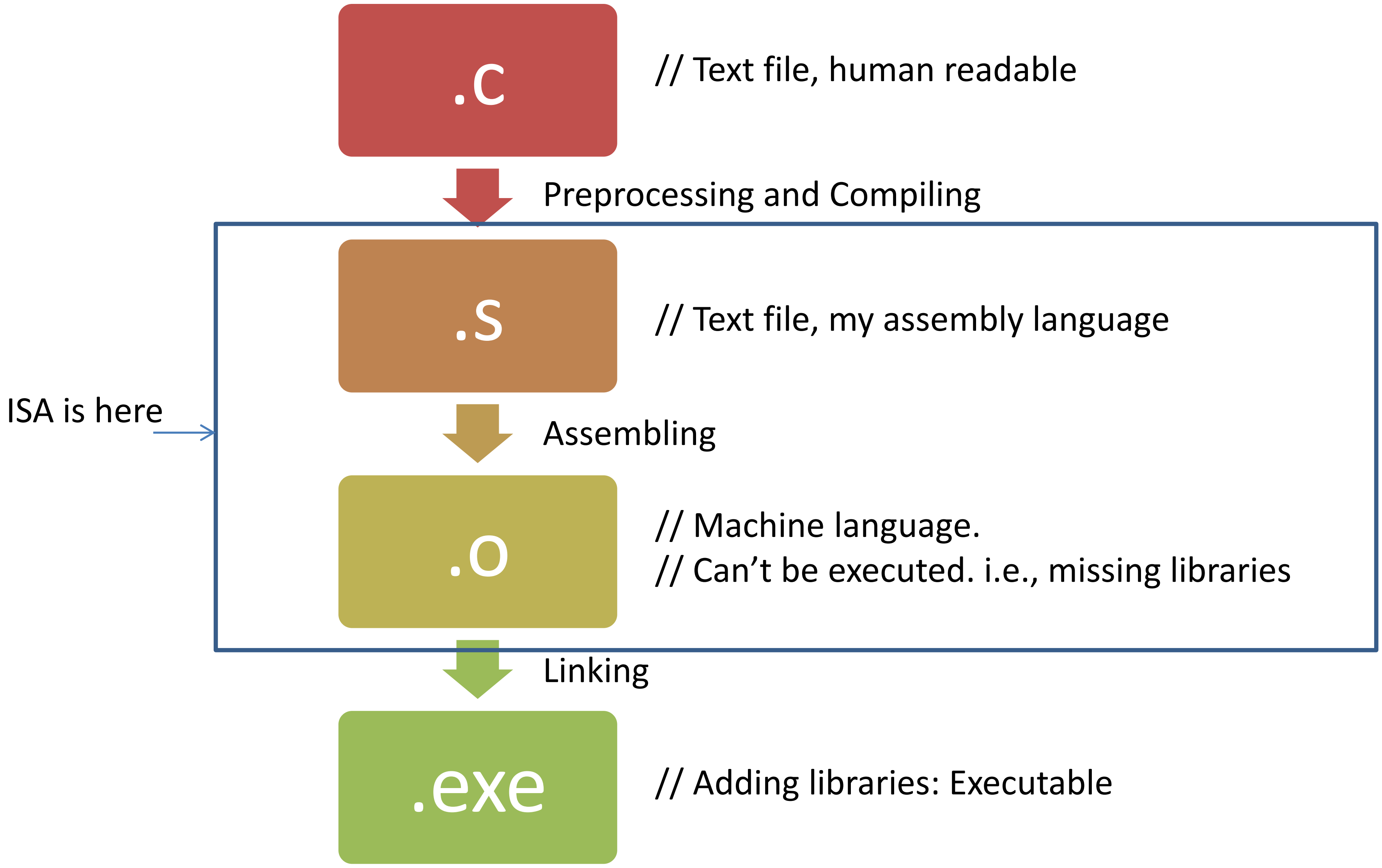
In Binary

```
2100
2201
188B
2000
4770
```

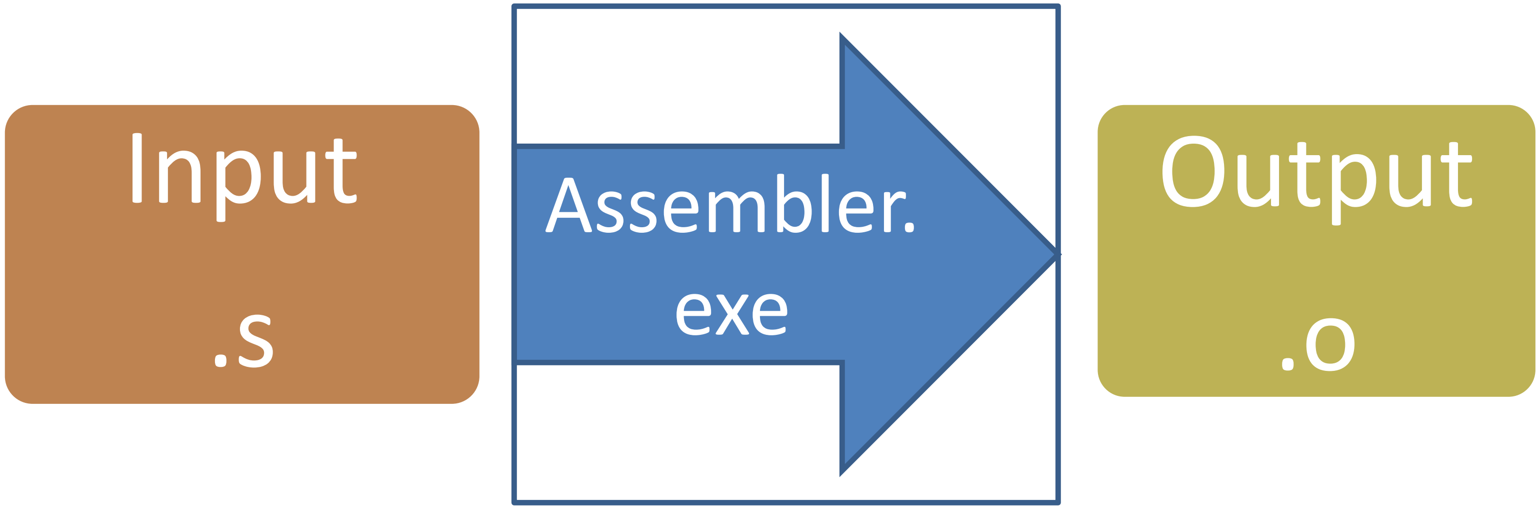
In Hex

```
; MOVS    r1, #0x00
; MOVS    r2, #0x01
; ADDS    r3, r1, r2
; MOVS    r0, #0x00
; BX      lr
```

- In short, the C program execution flow


.c
// Text file, human readable
Preprocessing and Compiling
.s
// Text file, my assembly language
ISA is here
Assembling
.o
// Machine language.
// Can't be executed. i.e., missing libraries
Linking
.exe
// Adding libraries: Executable

- Each process requires F-D-E process



1. Fetch instructions
2. Decode via **ISA**
3. Execute

(4. Write back)

Instruction **S**et **A**rchitecture: **ISA**

Machine language	Assembly language
Encoding 0’s and 1’s: binary	Writing in textual form
Copy the value from “Register 9” into “Register 3.”	
1110 0001 1010 0000 0011 0000 0000 1001	MOV R3, R9
Assembler	

- ISA: The design of the machine language encoding