NANYANG TECHNOLOGICAL UNIVERSITY

School of Mechanical & Aerospace Engineering

Design, Machine, Control and Intelligence

MA4832

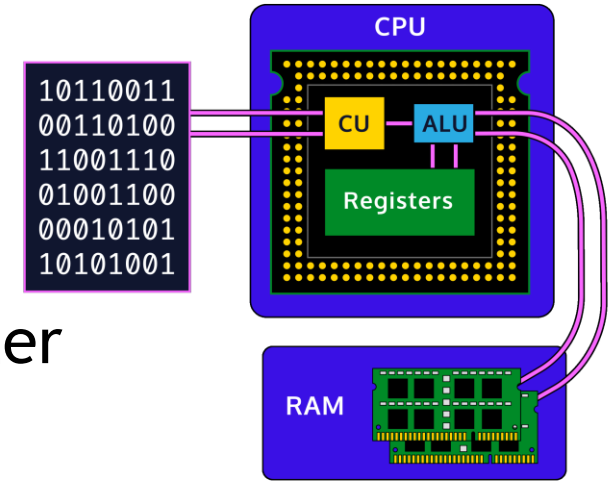
Microprocessor Systems

Xie Ming, PhD (France)

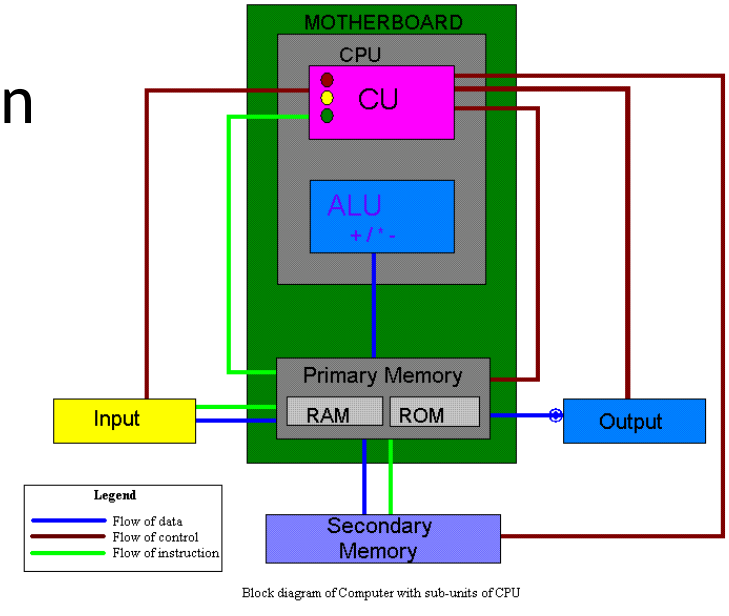
mmxie@ntu.edu.sg

http://personal.ntu.edu.sg/mmxie

Outline



- Lecture 1: Basics of ARM Microcontroller
- Lecture 2: ARM's Memories
- Lecture 3: ARM's Data Representation
- Lecture 4: ARM's Programming
- Lecture 5: ARM's Data Input/Output
- Lecture 6: ARM's Data Processing



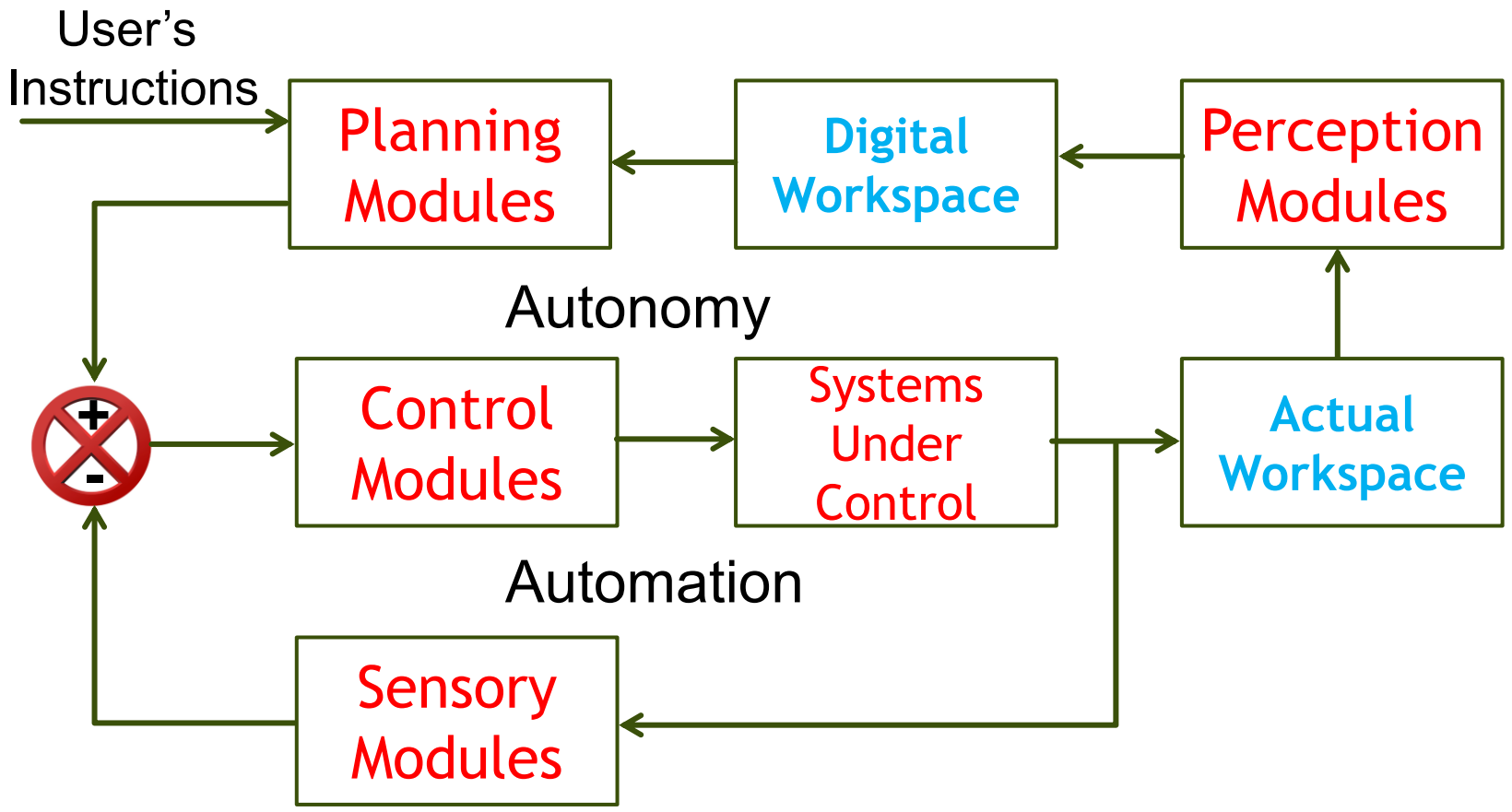
Block diagram of Computer with sub-units of CPU

Remember your mission as MAE undergraduates …

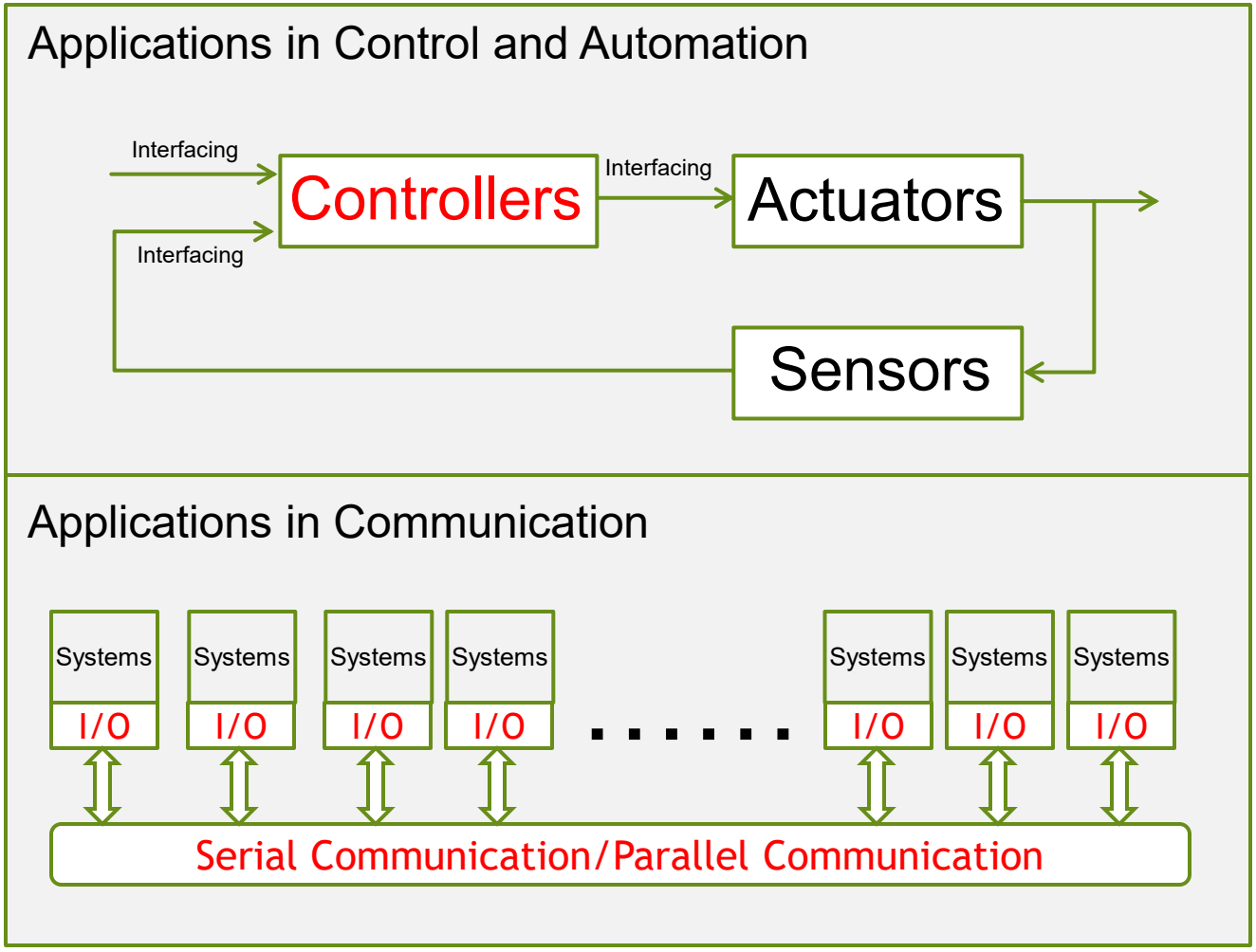
- You are here to grow your knowledge and skills so as to be able to design machines with controllable behaviors and hopefully in some intelligent ways.

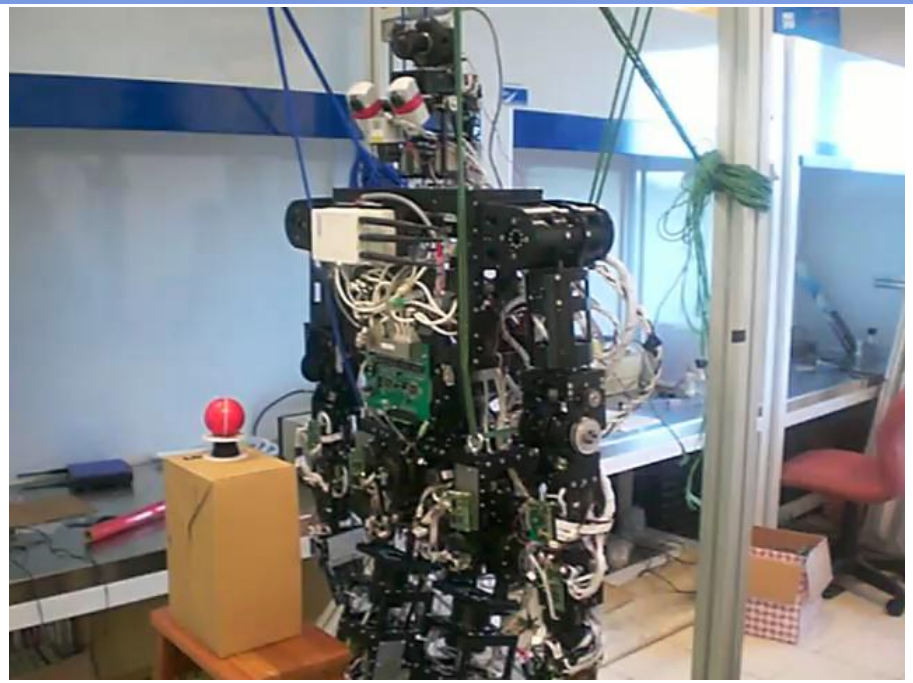
How to fulfill your mission?

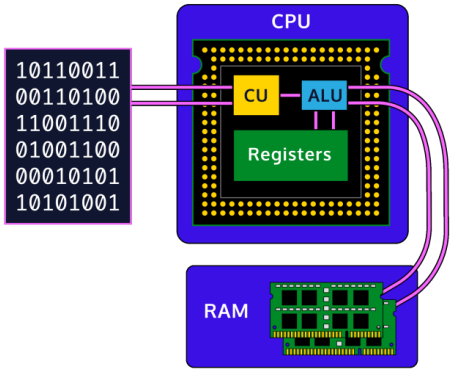
- To apply learnt knowledge and skills into the implementation of the following universal blueprint underlying all the intelligent machines or systems.

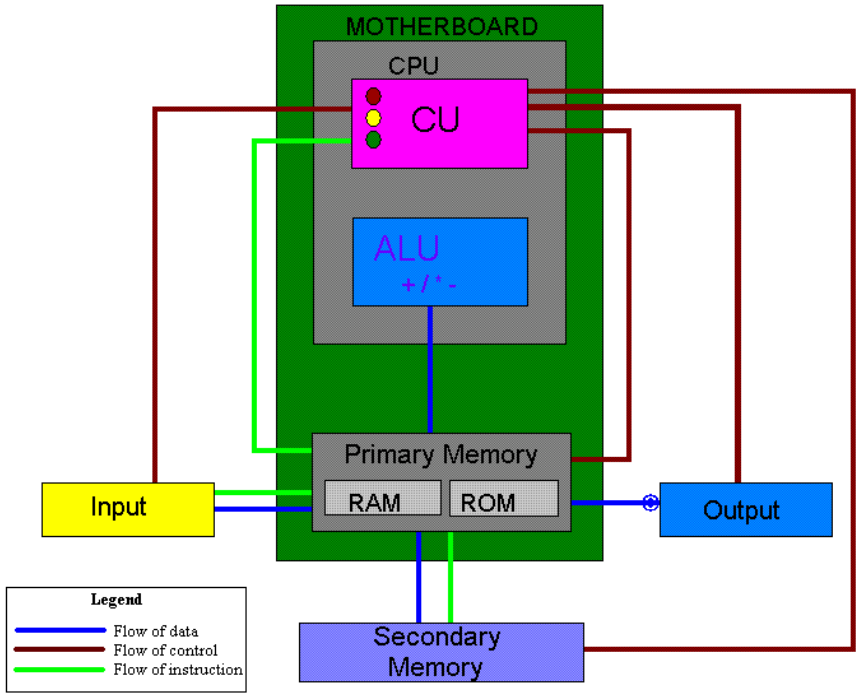


Why to study?


Applications in Control and Automation
Interfacing
Interfacing
Controllers
Interfacing
Actuators
Sensors
Applications in Communication
Systems
I/O
Systems
I/O
Systems
I/O
Systems
I/O
Systems
I/O
Systems
I/O
Systems
I/O
Serial Communication/Parallel Communication



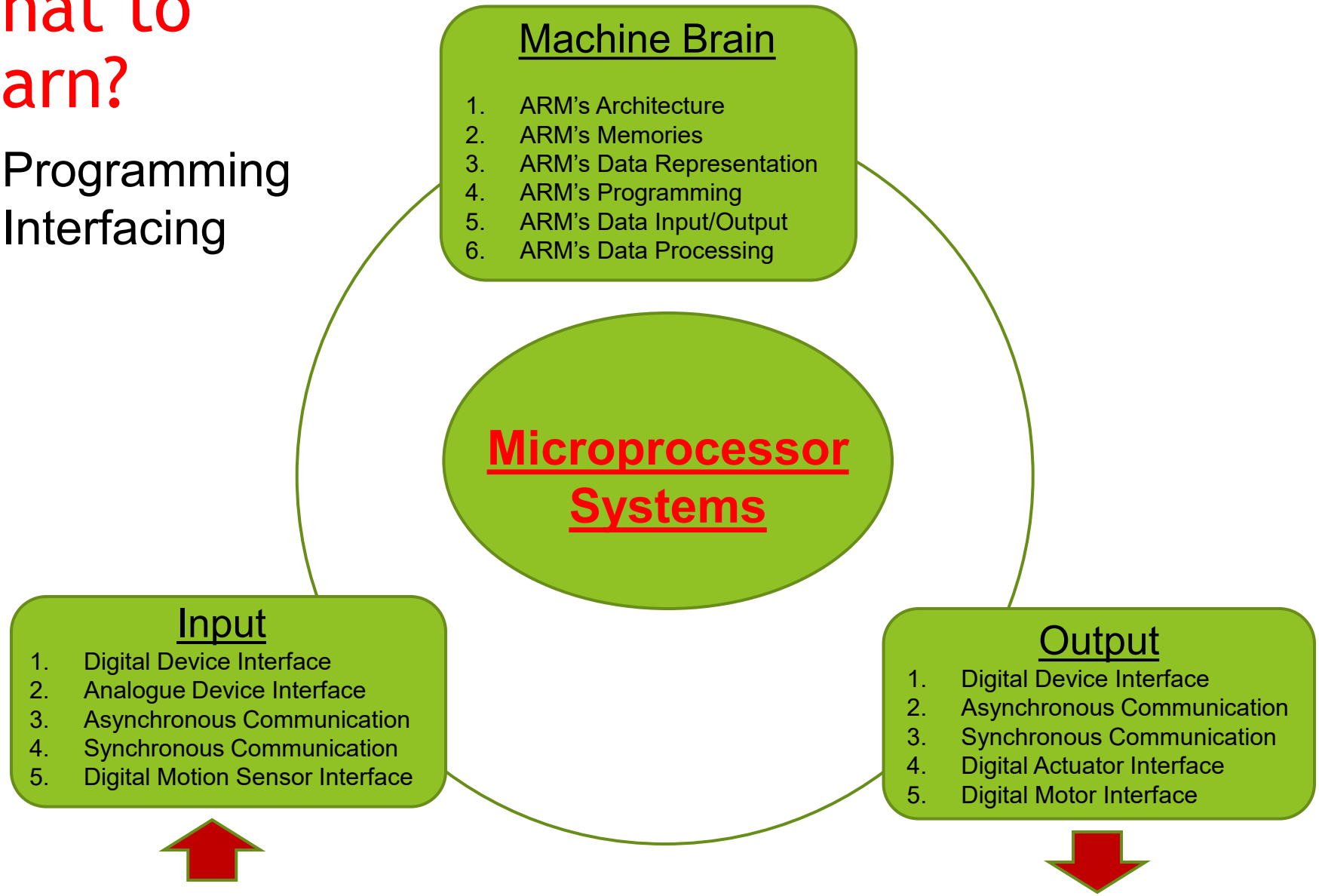




Block diagram of Computer with sub-units of CPU

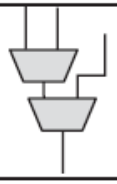
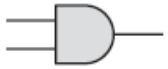
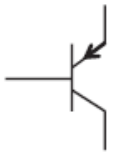
What to learn?

- Programming
- Interfacing



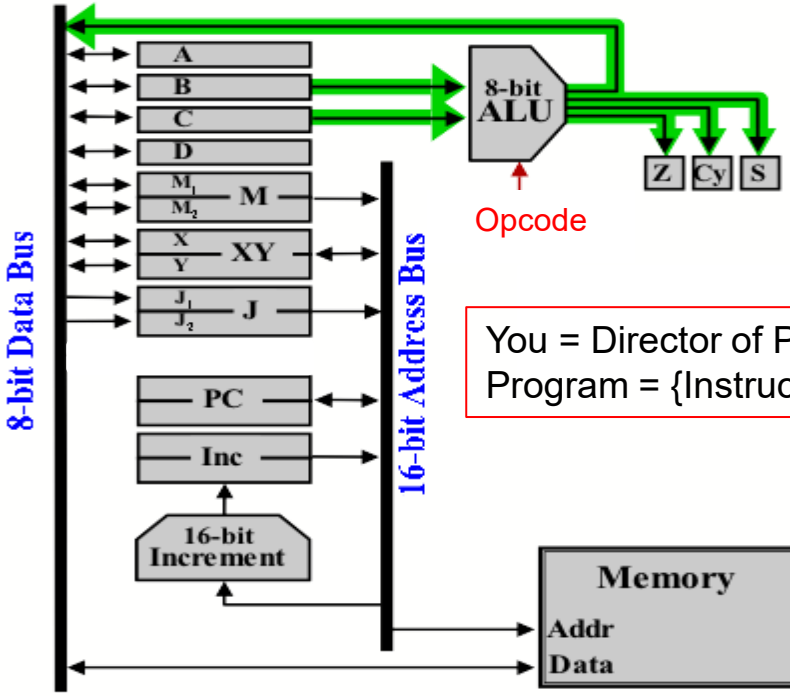
What is your role?

- Data = {Values, Symbols, Addresses, Instructions}
- Instructions = {Op Code + Addresses + Value/Symbol}

Applications/OS	
Languages	C++, Java
ISA	EOR r3,r2,r1 BEQ Table
Microarchitecture	
Gates	
Transistors	

Algorithms or Solutions

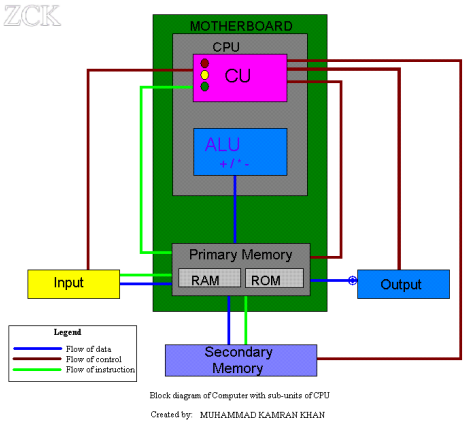
Problems to be solved



You = Director of Program
Program = {Instructions}

Memory = {Address + Data/Instruction}

How to learn?



- To understand data flows inside a microcontroller.
- To translate your solutions into data flows.
- To pay attention to <memory address> and <memory data>.
  - Memory Address: Address Label/Name and Address Value.
  - Memory Data: Data Label/Name and Data Value.

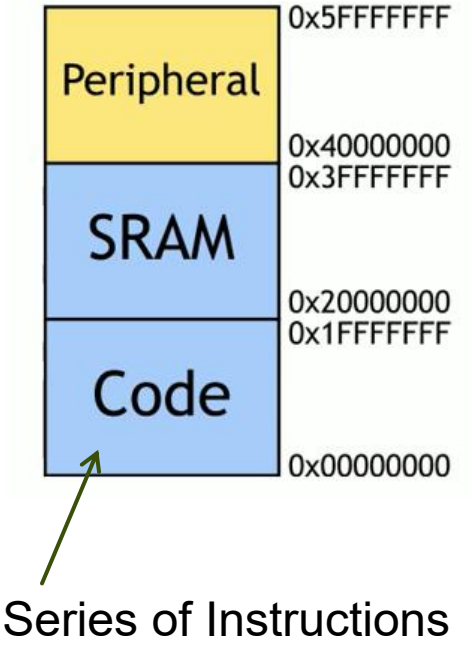
Memory Content

Data	Memory Address

- To pay attention to <memory address> and <memory code>.
  - Memory Address: Address Label/Name and Address Value.
  - Memory Code: Code Label/Name and Code Value.

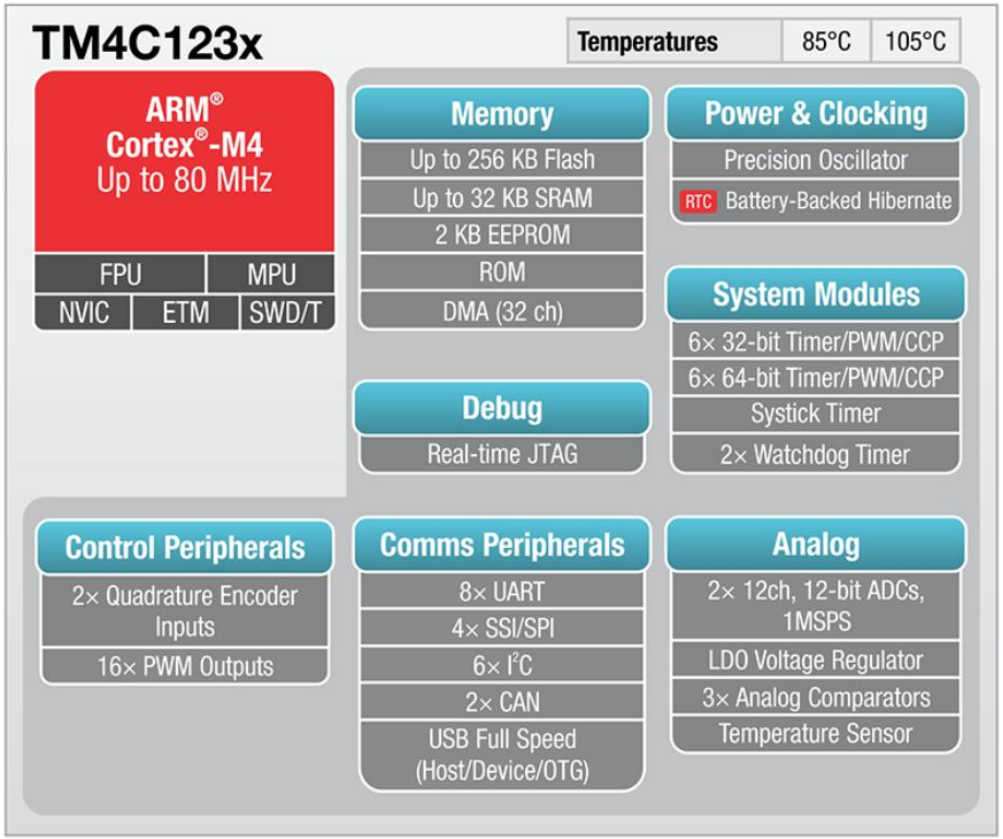
Memory Content

Instruction	Memory Address



Example of Using I/O Modules

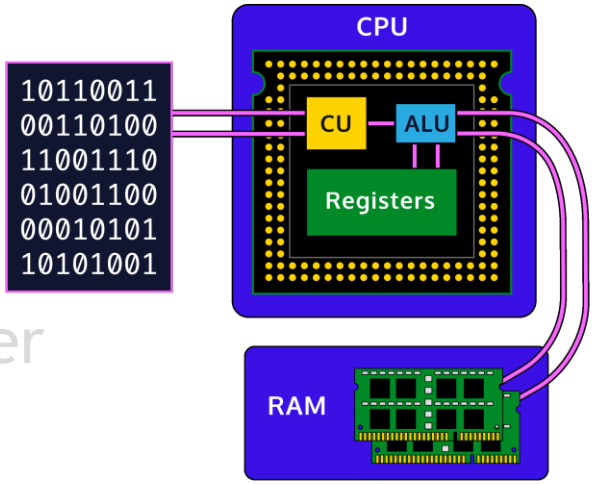
- Configure Control Registers
- Clear/Monitor Status Registers
- Read/Write Data Registers
- Instructions:
  - MOV <address of destination>, <source of value>
  - LDR <address of destination>, <source of value>
  - STR <source of value>, <address of destination>



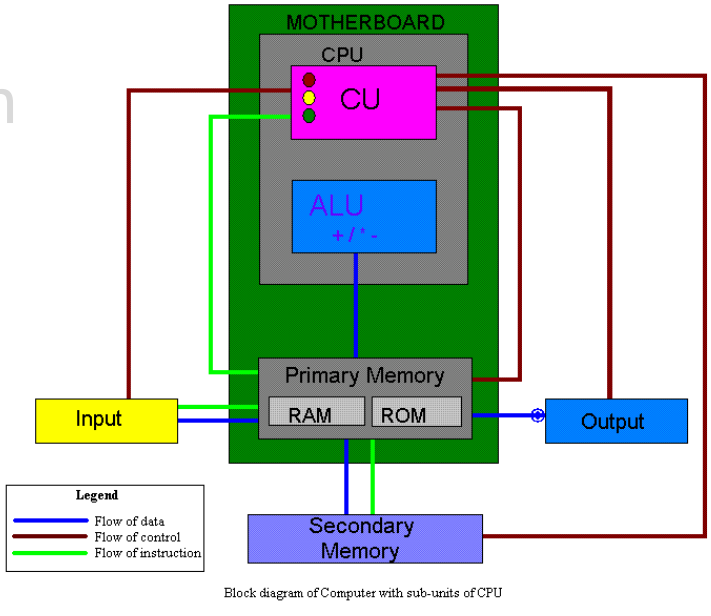

```
MOV   R0, #0x11
MOV   R1, #2560
MVN   R2, #4
MOVW  R3, #0xC0DE
MOVT  R3, #0xFEED
MOV   R4, R1
```

Word = 2 Bytes

Today's Lecture …



- Lecture 1: Basics of ARM Microcontroller
- Lecture 2: ARM's Memories
- Lecture 3: ARM's Data Representation
- Lecture 4: ARM's Programming
- Lecture 5: ARM's Data Input/Output
- Lecture 6: ARM's Data Processing



Block diagram of Computer with sub-units of CPU

NANYANG TECHNOLOGICAL UNIVERSITY

School of Mechanical & Aerospace Engineering

Design, Machine, Control and Intelligence

MA4832

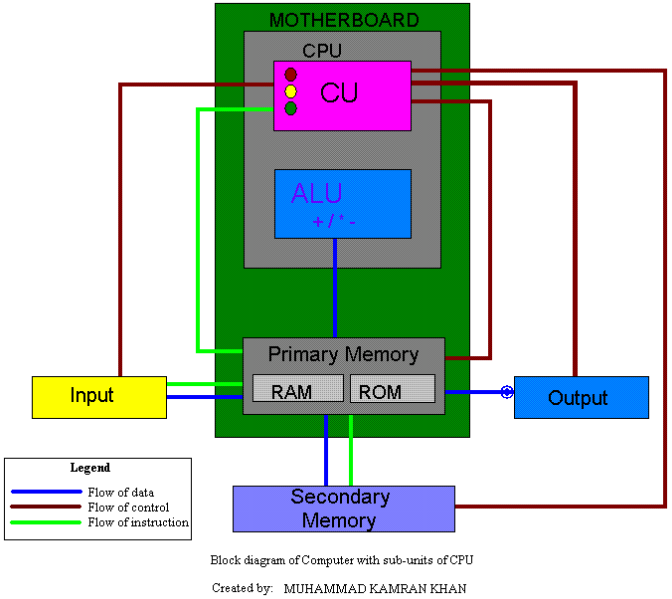
ARM's Data Input and Output

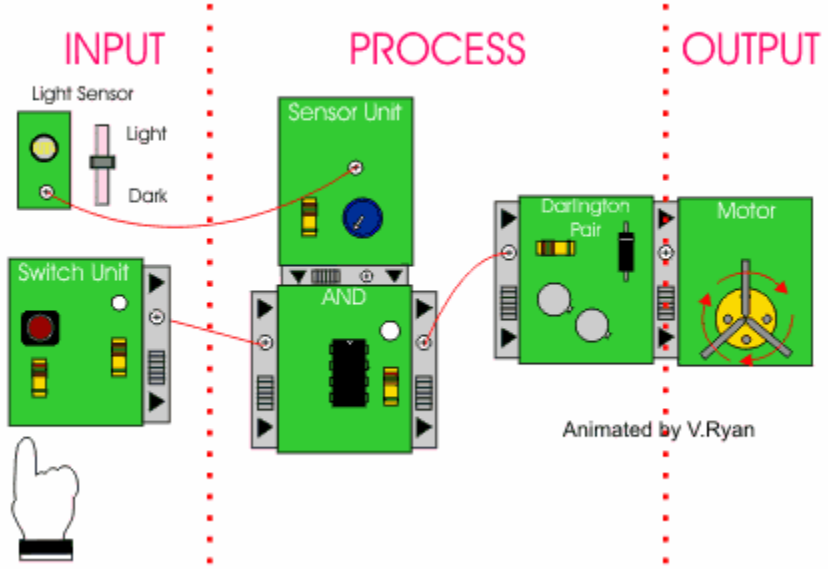
Xie Ming, PhD (France)

http://personal.ntu.edu.sg/mmxie

Outline

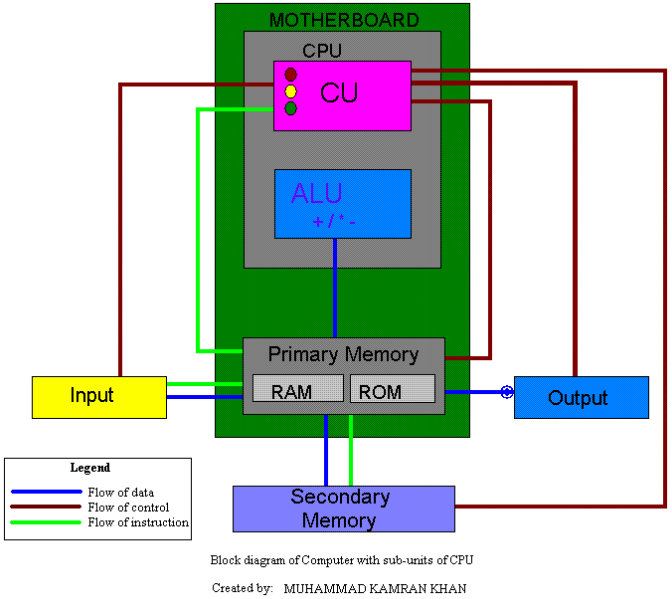
- Nature of Data Input and Output
- Memory-Mapped I/O Devices
- Basics of Address Indexing
- ARM Instructions for Data I/O
- Maskable Data Registers in ARM

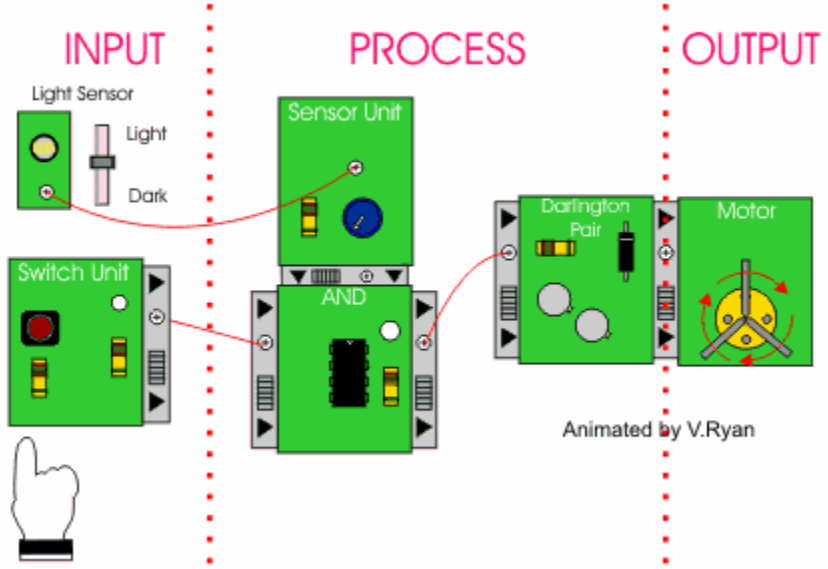




Outline

- Nature of Data Input and Output
- Memory-Mapped I/O Devices
- Basics of Address Indexing
- ARM Instructions for Data I/O
- Maskable Data Registers in ARM



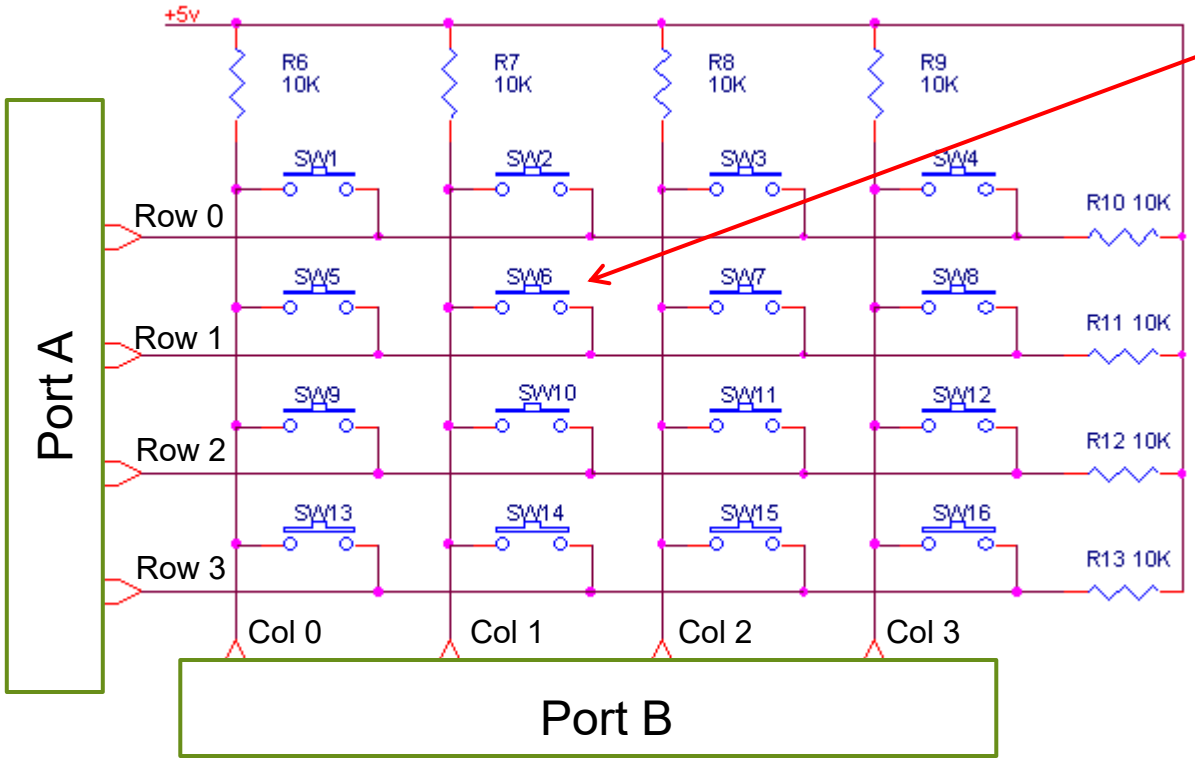


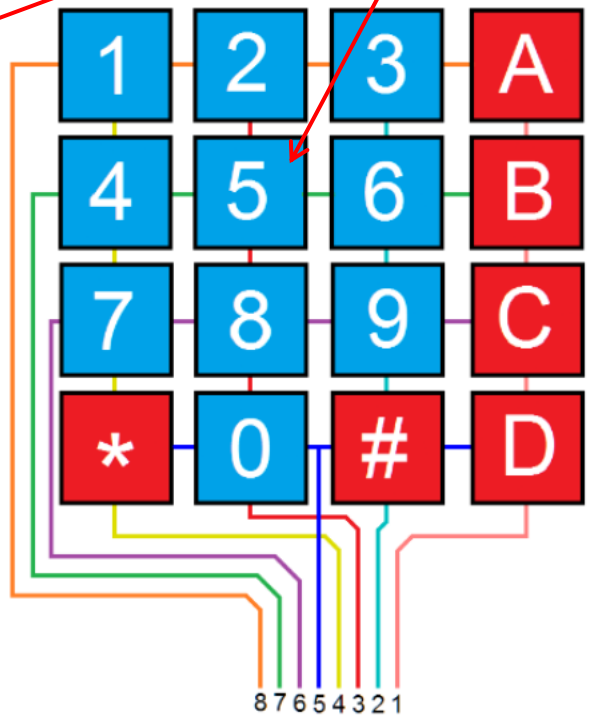
Example of Machines with Controllable Activities

Example of Data Input

- CPU sequentially **outputs** 1110, 1101, 1011, 0111 to port **A**
- CPU **inputs** data from port **B**
- By default, all the keys are not pressed.
- When a key is pressed, the corresponding column line will be in logic low (i.e. "0").

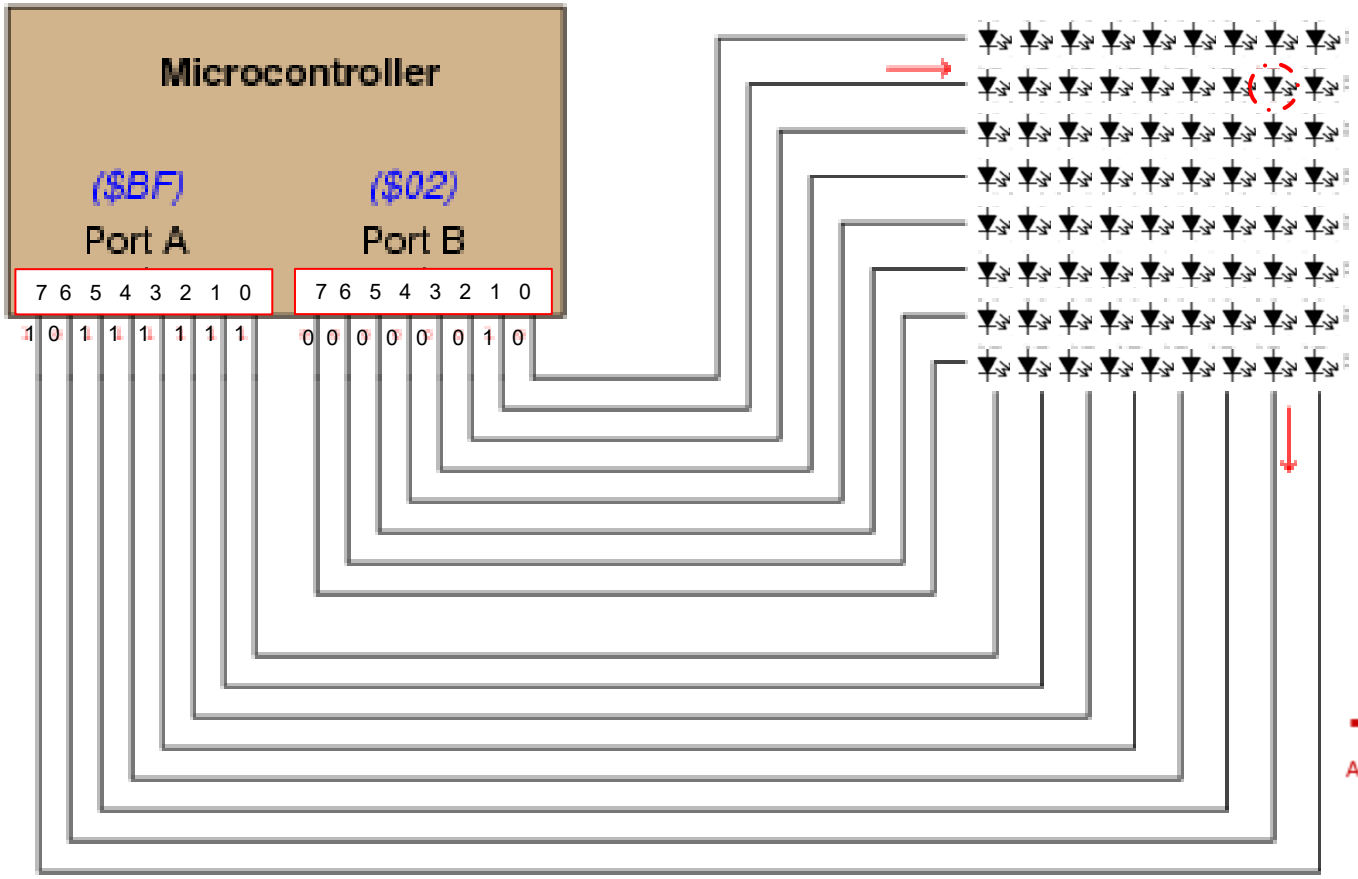
Port A = 1101
Port B = 1101
Which key?
Answer: 5

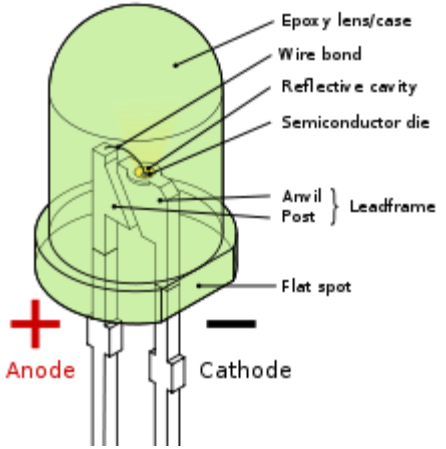




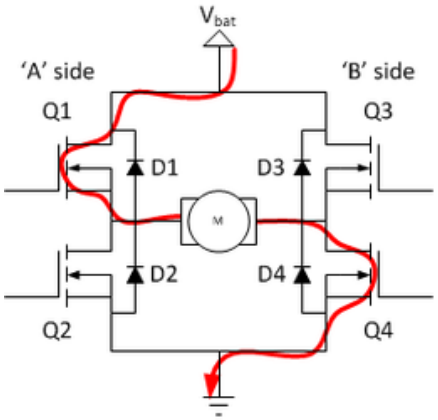
Example of Data Output

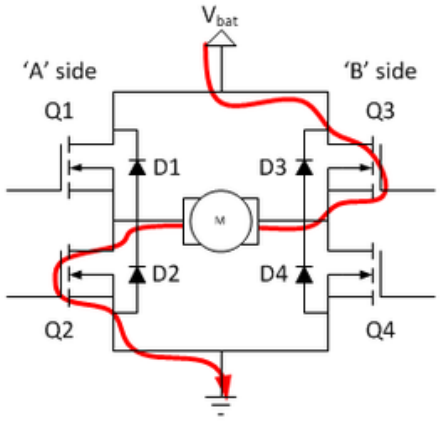
- CPU outputs $BF to port A and $02 to port B.
- Which LED will light up?
- Answer: the one at row 1 and column 6



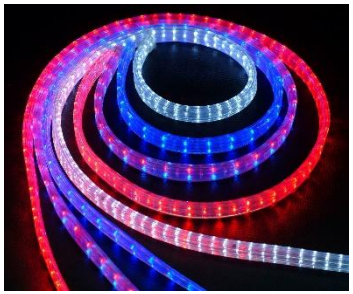


Examples of Digital Devices for Output



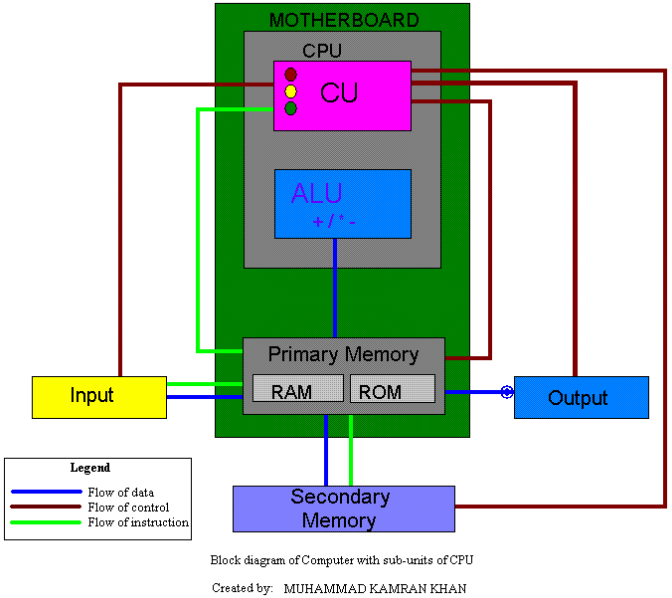


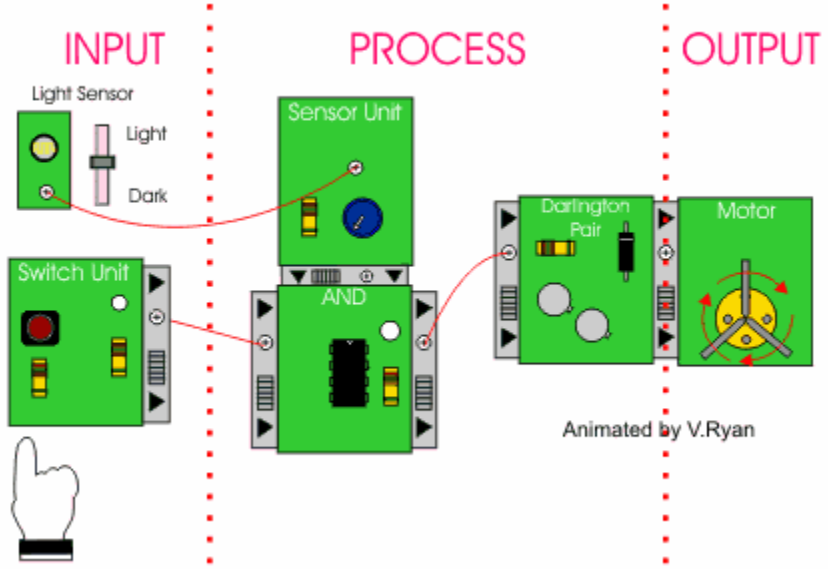
Examples of Digital Devices of Input/Output



Outline

- Nature of Data Input and Output
- Memory-Mapped I/O Devices
- Basics of Address Indexing
- ARM Instructions for Data I/O
- Maskable Data Registers in ARM





A microcontroller animates data flow …

Data = {Values, Symbols, Addresses, Instructions}

Data flows …

From Source Memory

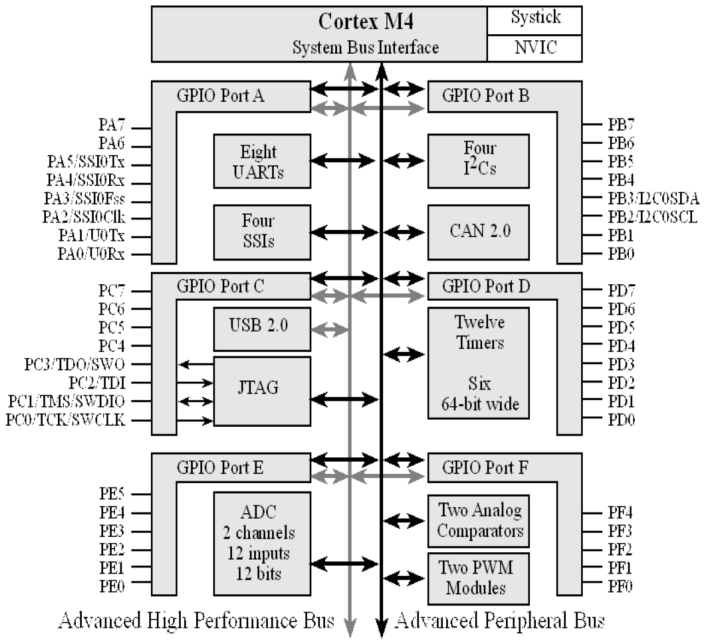
- A memory is a space which has:
  - Memory Location
  - Memory Address*
  - Memory Label

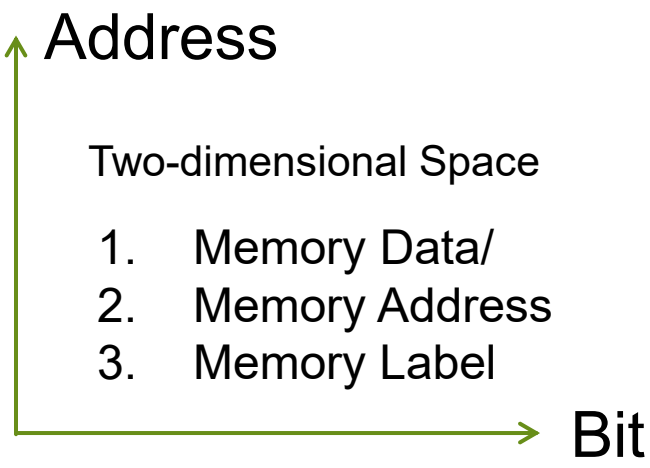
To Destination Memory

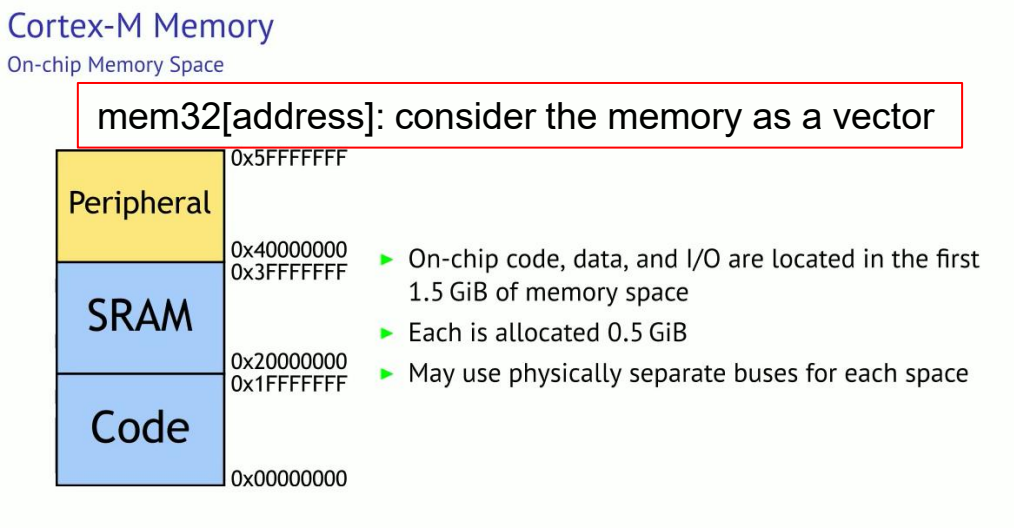
- A memory is a space which has:
  - Memory Location
  - Memory Address*
  - Memory Label

Basic Unit of Input/Output Data

- A port has eight pins.
- Hence, the basic unit of input/output data is one-Byte.
- However, inside ARM Cortex, the data bus transfer 4-Bytes per cycle.







Allocation of Memory is One Major Task of Programmers …

Data = {Values, Symbols, Addresses, Instructions}

To Store Variable Values

- Assign a memory label to a variable value, which indicates:
  - A fixed memory address
    - Useful to users
  - Multiple instances of value
    - Needed by users
  - Example:

```
TravelledDistance   DCD   0
```

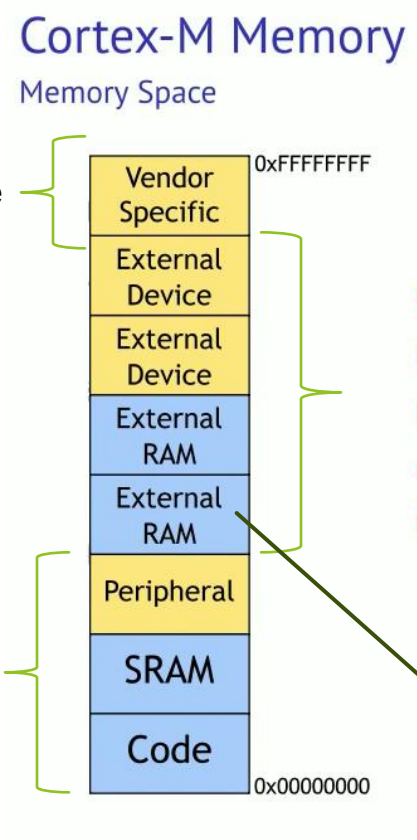
To Store Fixed Values

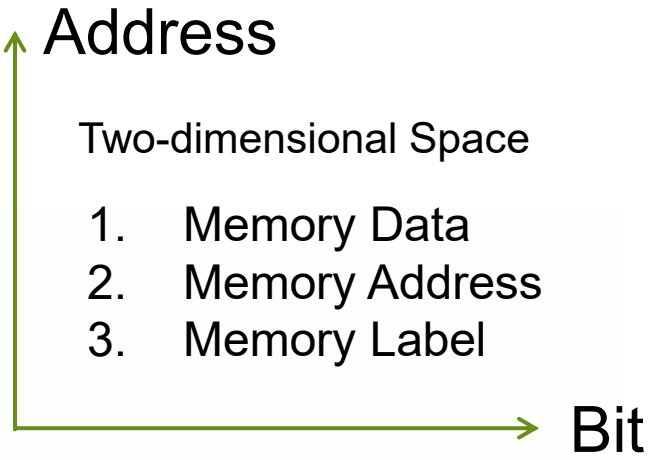
- Assign a memory label to a fixed value, which indicates:
  - Multiple memory addresses
    - Not useful to users
  - Single instance of value
    - Needed by users
  - Example:

```
GravityAcceleration   EQU   9.8
```

ARM Cortex-M's Memory Space

- On-chip Memory Sub-space
- Off-chip Memory Sub-space
- Private Memory Sub-space



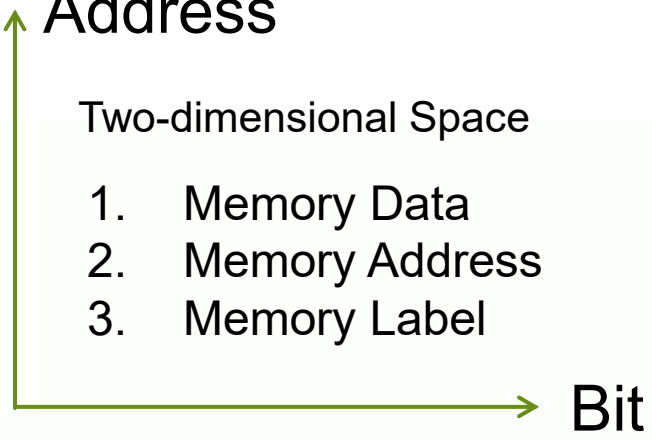


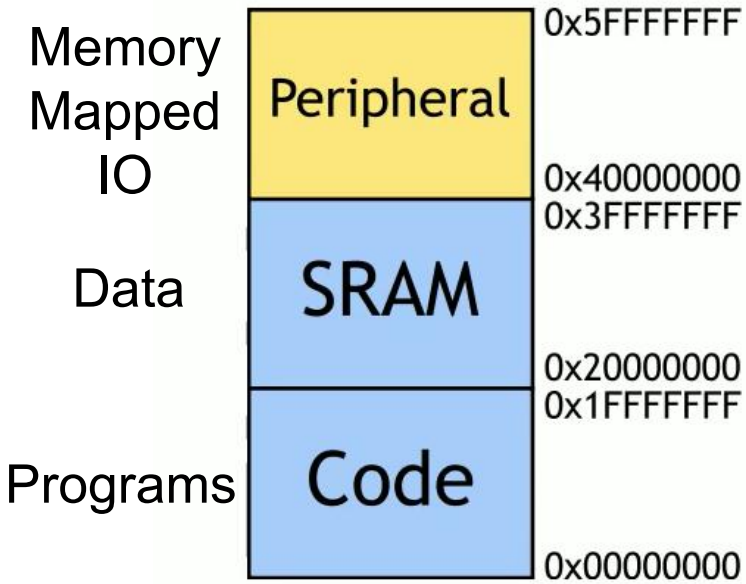
- 32-bit addresses support 4 GiB memory space
- Code, data, and I/O share same memory space
- Data types are bytes, halfwords, and words
- Memory addresses are byte addresses
- Predefined regions have distinct characteristics
  - Executable
  - Device or Strongly-ordered
  - Shareable

Analogy:
1. Main Building with Guest Rooms.
2. Annex Buildings with Functional Rooms.

On-chip Memory Sub-space in ARM

Data = {Values, Symbols, Addresses, Instructions}

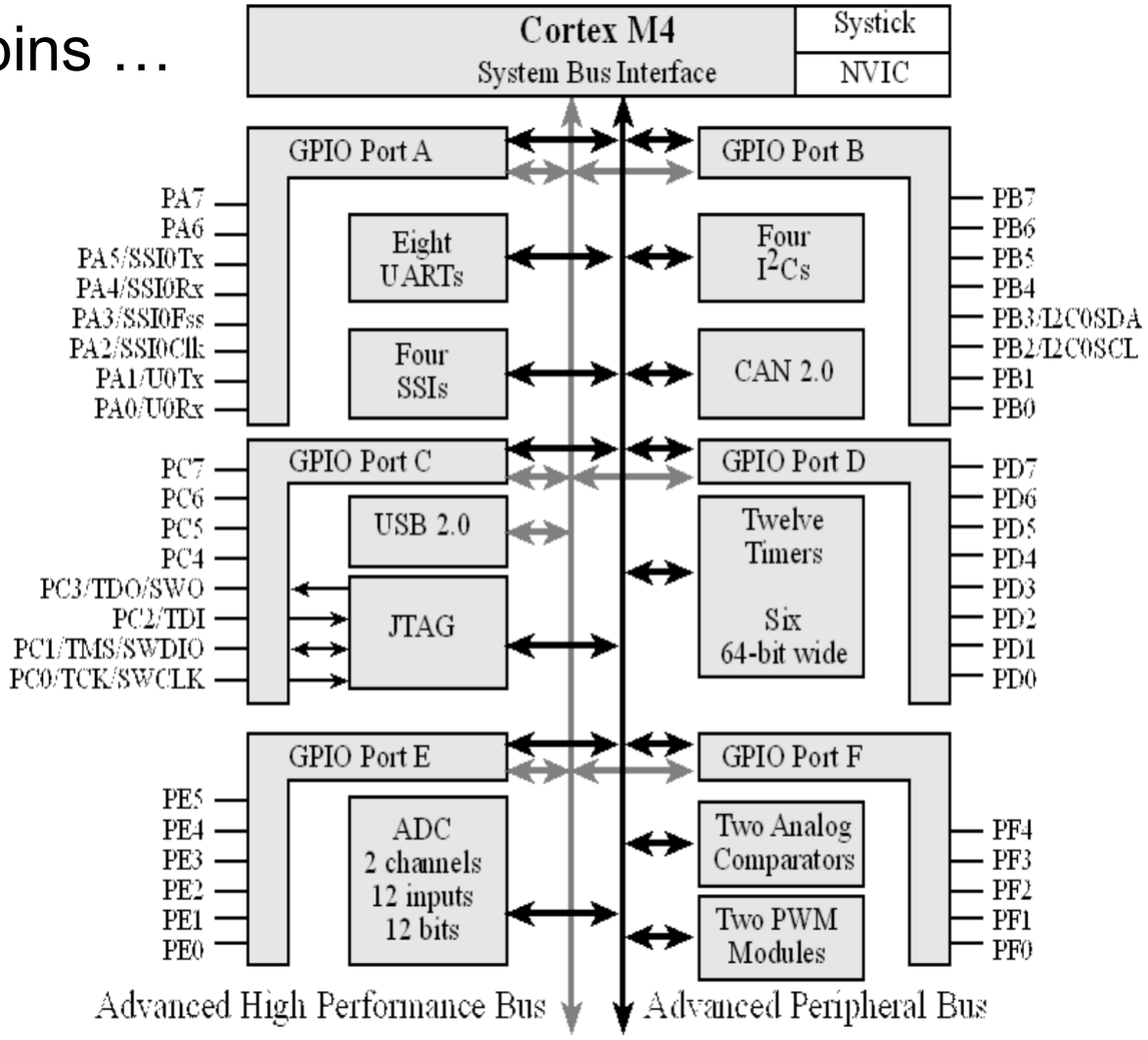




- On-chip code, data, and I/O are located in the first 1.5 GiB of memory space
- Each is allocated 0.5 GiB
- May use physically separate buses for each space

ARM's Interfaces for Input/Output

Each port has eight pins …



Address

Two-dimensional Space

1. Memory Data
2. Memory Address
3. Memory Label

Bit

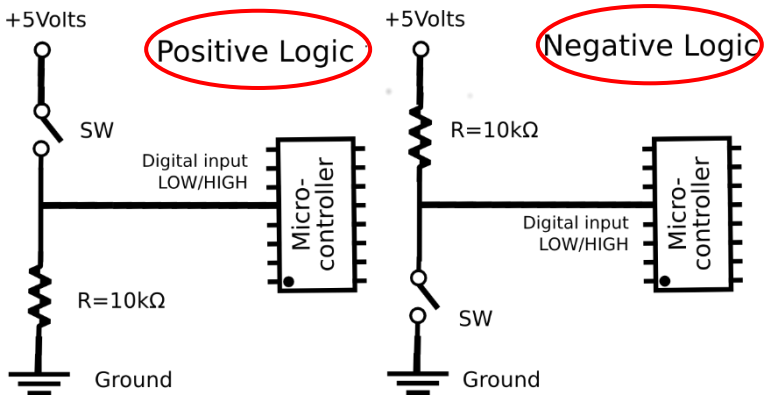
Example of Internal Data Input from Port F

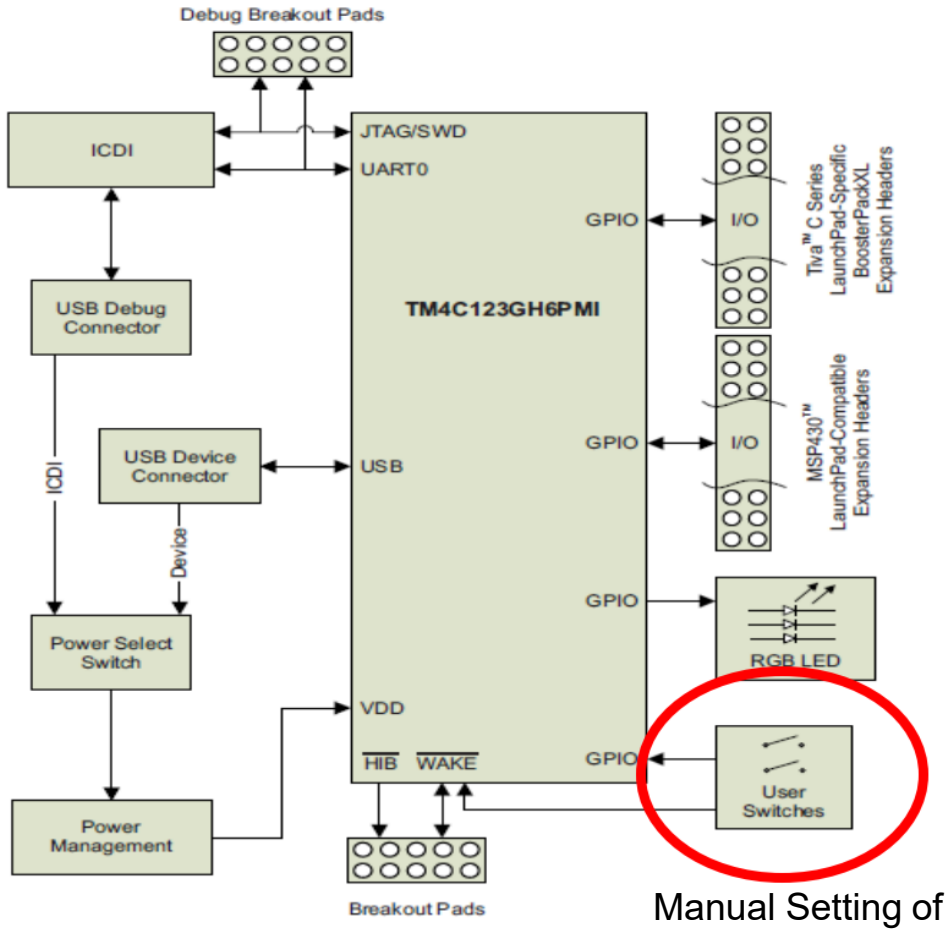
- Hardware Address of Port F's Data: **R4 = 0x4002.5FFF**

At Pin/Bit 4

GPIO Pin	Pin Function	USB Device
PF4	GPIO	SW1
PF0	GPIO	SW2
PF1	GPIO	RGB LED (Red)
PF2	GPIO	RGB LED (Blue)
PF3	GPIO	RGD LED (Green)

Two Types of Switches





Manual Setting of Pin Values

Continued …

- Sample Code: Hexadecimal(0x10) = Binary(0001 0000)

```
loop            BL  SSR_On          ; function switch on LED
waitforpress1   LDR R0, [R4]        ; R0 = [R4] (read status of PF4)
                CMP R0, #0x10       ; R0 == 0x 10
                BEQ waitforpress1   ; Negative Logic Switch
waitforrelease1 LDR R0, [R4]        ; R0 = [R4]
                CMP R0, #0x10       ; R0 != 0x10?
                BNE waitforrelease1 ; Negative Logic Switch
                BL  SSR_Off         ; function switch off LED
waitforpress2   LDR R0, [R4]        ; R0 = [R4]
                CMP R0, #0x 01      ; R0 == 0x 01
                BEQ waitforpress2   ; Negative Logic Switch
waitforrelease2 LDR R0, [R4]        ; R0 = [R4]
                CMP R0, #0x 01      ; R0 != 0x 01
                BNE waitforrelease2 ; Negative Logic Switch
                B   loop
```

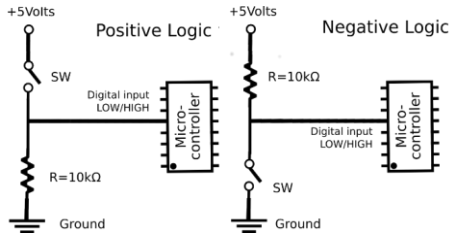
If not pressed, Continue the loop.

If not released, Continue the loop.

If not pressed, Continue the loop.

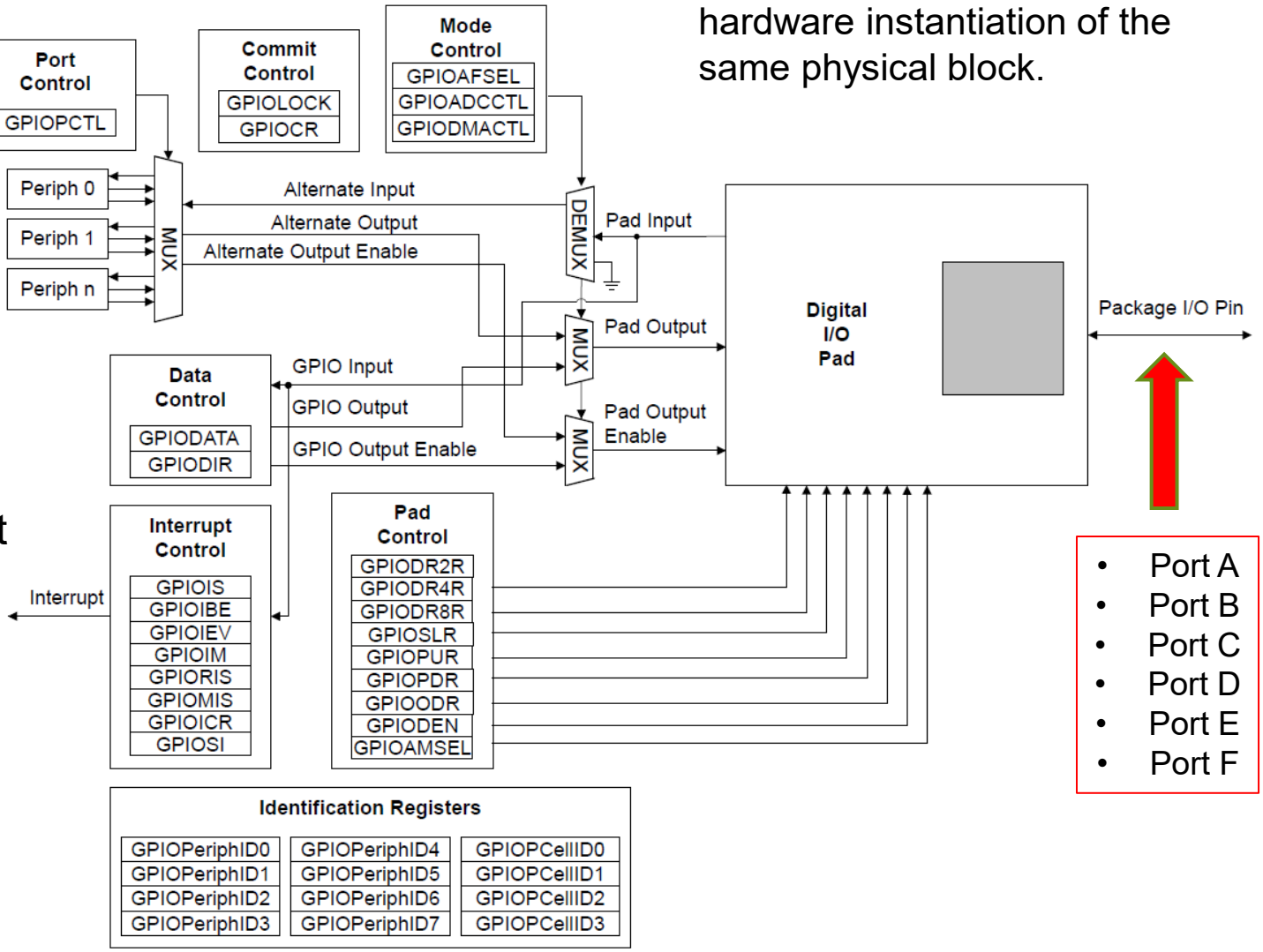
If not released, Continue the loop.

Address of Port F's Data: R4 = 0x4002.5FFF



ARM's GPIO Module

Each GPIO port is a separate hardware instantiation of the same physical block.

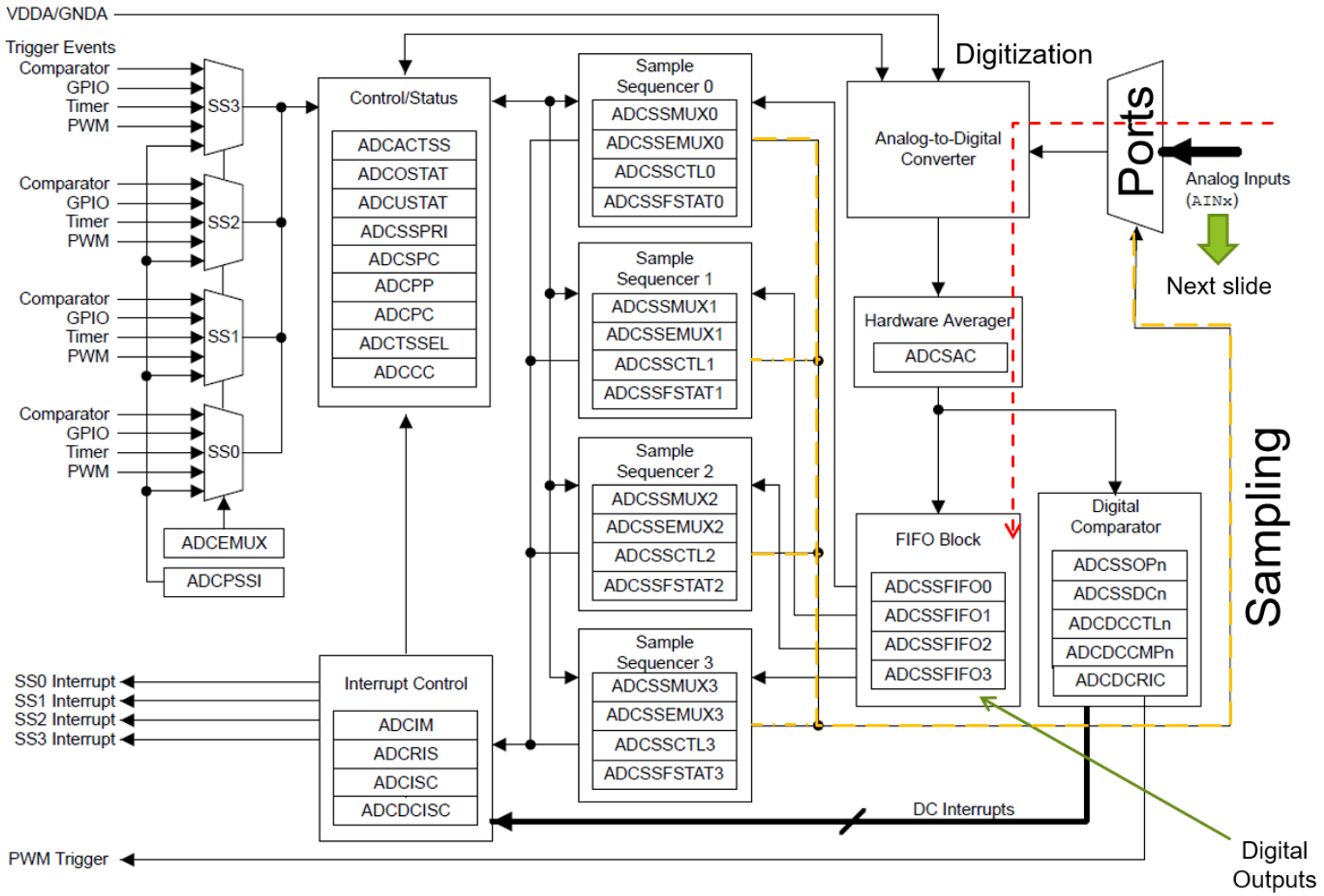


One interrupt per port

- Port A
- Port B
- Port C
- Port D
- Port E
- Port F

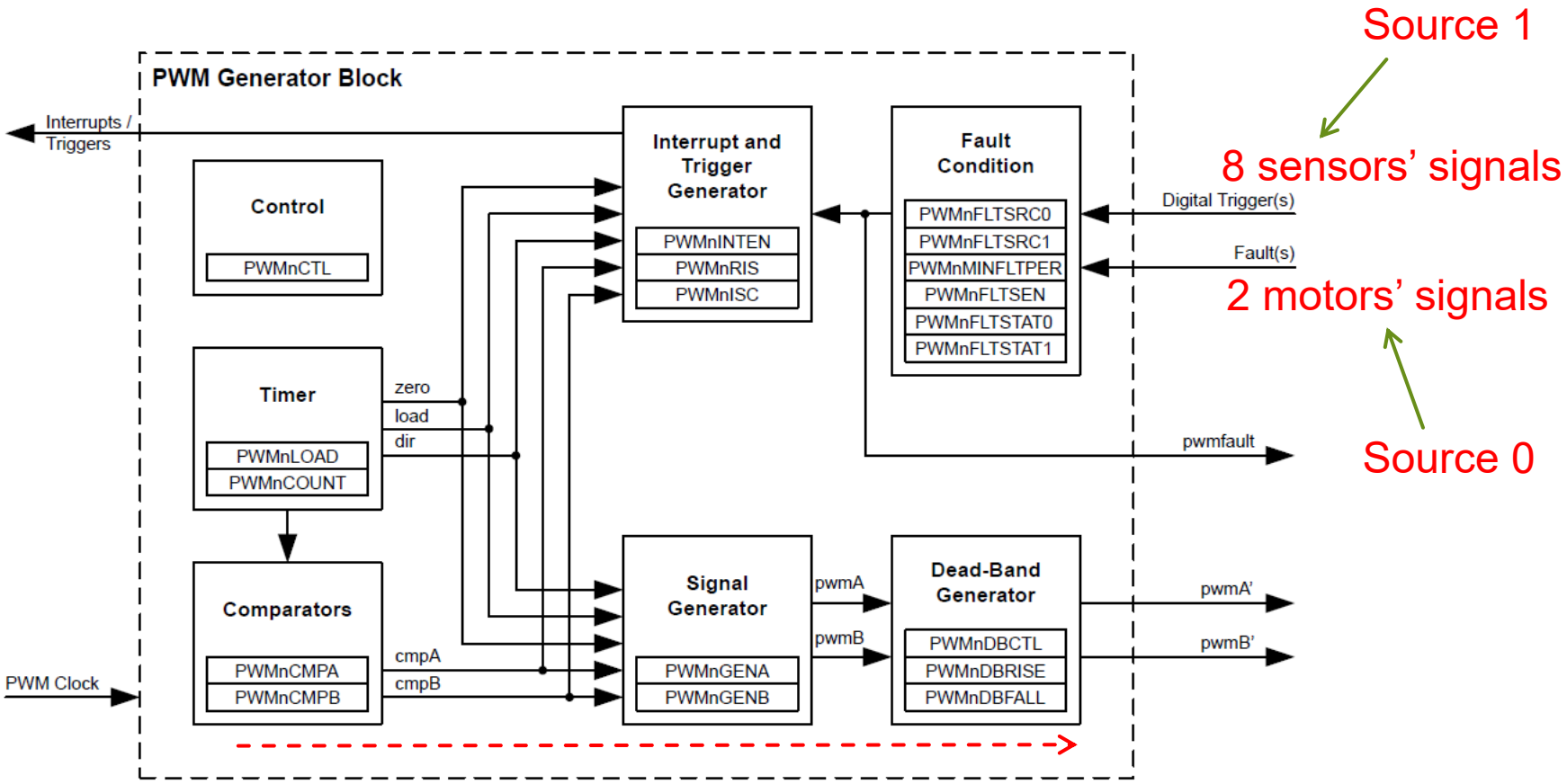
ARM's ADC Module

(Some pins could receive analogue values)



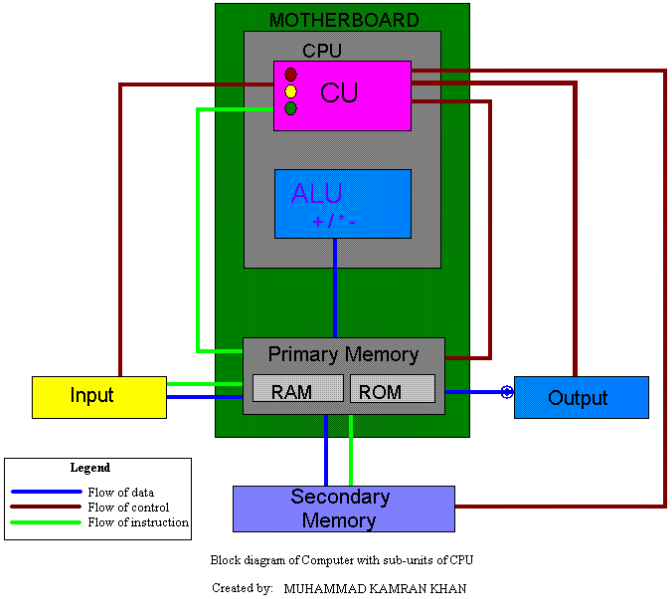
ARM's Module for PWM

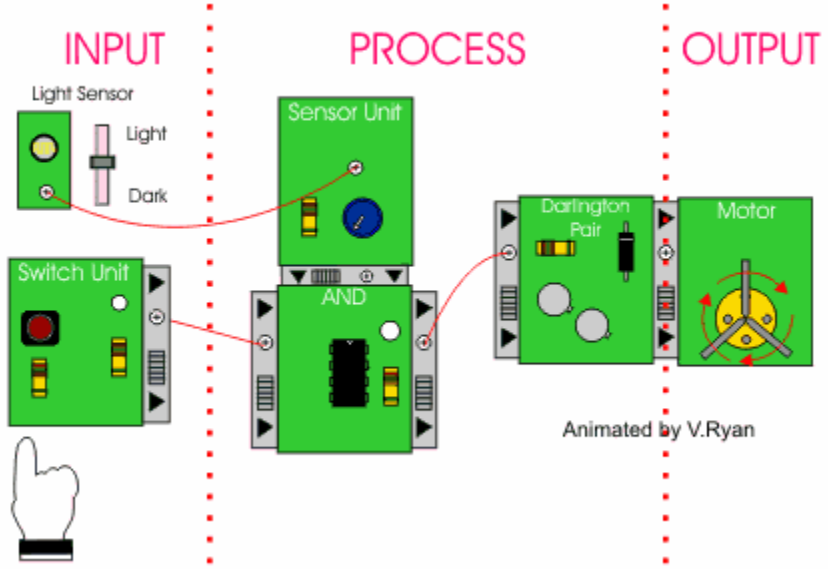
(Two PWM Modules, Four PWM Generators per Module)



Outline

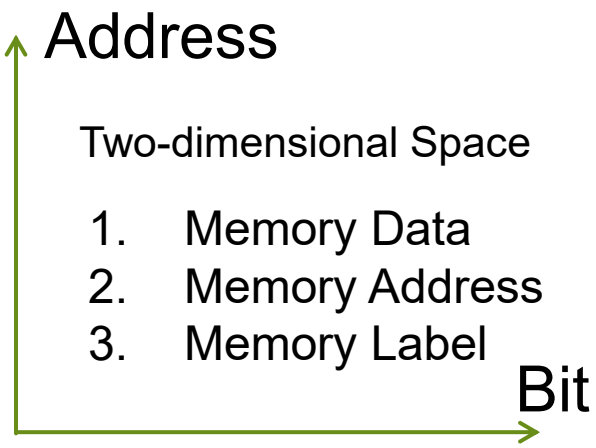
- Nature of Data Input and Output
- Memory-Mapped I/O Devices
- Basics of Address Indexing
- ARM Instructions for Data I/O
- Maskable Data Registers in ARM

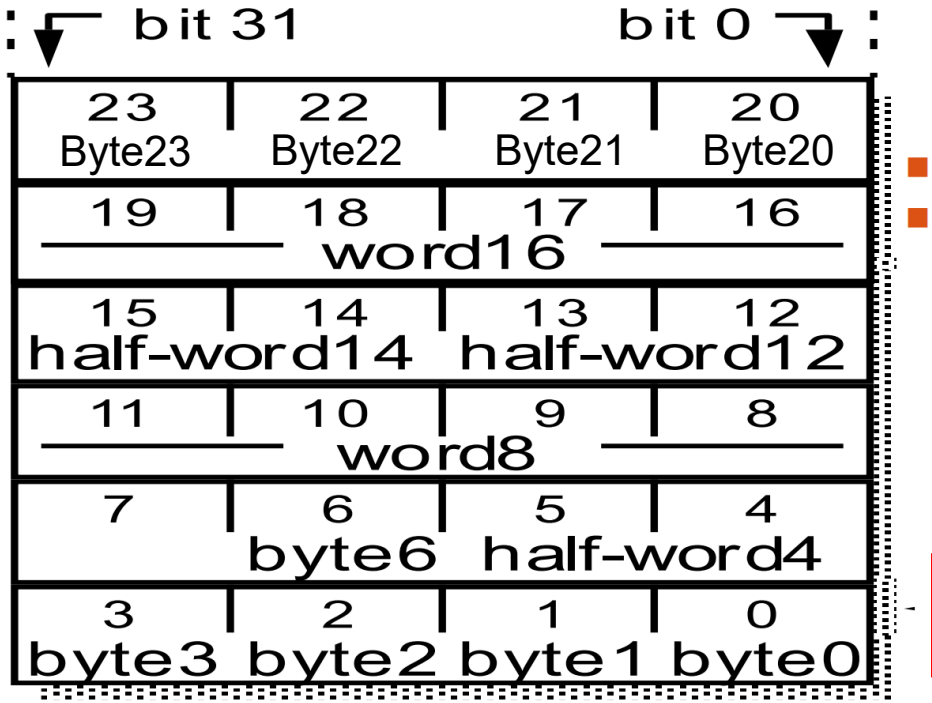




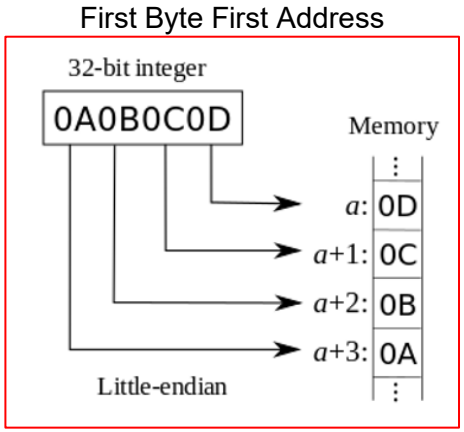
Notations of Data's Organizations

- Data is organized into 8 bits: one byte (basic unit of data)
- Data is organized into 16 bits: half-word
- Data is organized into 32 bits: one word
- etc …





- Address bus: 32 bits
- 1 word = 32 bits



ARM Cortex is 32-bit microcontroller

- Memory Unit in Hardware: 1-byte
- Memory Unit in Software: 4-byte by default

Notation of Memory's Address and Value

- Format:
  - mem{data size} [data address] : get or set the value of "data size" from "data address".

- Examples:

mem32[address] = data

Source

```
LDR r0, [r1]        ; equiv: LDR r0, [r1, #0]
                    ; r0 = mem32[r1]
STR r0, [r1]        ; equiv: STR r0, [r1, #0]
                    ; mem32[r1]= r0
```

Get or return a value

Set a value

Source

- **Register *r1 (in example)* is called the *base address register***

(In terms of technical language, a memory is a vector. Hence, an address is an index to a vector)

Definition of Address Indexing

- Base Address: It is a common reference for a series of related addresses.
- Address Offset: It is an individual offset from a common reference address.
- Address Indexing: It refers to the operation of adding offset to base address.

Example: A configuration register is a register with programmable switches.

General-Purpose Input/Output Run Mode Clock Gating Control (RCGCGPIO)

Base 0x400F.E000
Offset 0x608
Type RW, reset 0x0000.0000

Address Indexing = Base Address + Offset = [Base Address, Offset]

- Turn on or Turn off
- Enable or Disable
- Input or Output

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved										R5	R4	R3	R2	R1	R0
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Port F ← Port A

Address Indexing Methods

Address Indexing = Base Address + Offset = [Base Address, Offset]

In both types of instruction, the addressing mode is formed from two parts:

- the base register
- the offset.

The base register can be any one of the general-purpose registers (including the PC, which allows PC-relative addressing for position-independent code).

The offset takes one of three formats:

- Preindexing without writeback
- Preindexing with writeback
- Postindexing

Immediate

The offset is an unsigned number that can be added to or subtracted from the base register. Immediate offset addressing is useful for accessing data elements that are a fixed distance from the start of the data object, such as structure fields, stack offsets and input/output registers.

For the word and unsigned byte instructions, the immediate offset is a 12-bit number. For the halfword and signed byte instructions, it is an 8-bit number.

Register

The offset is a general-purpose register (not the PC), that can be added to or subtracted from the base register. Register offsets are useful for accessing arrays or blocks of data.

Scaled register

The offset is a general-purpose register (not the PC) shifted by an immediate value, then added to or subtracted from the base register. The same shift operations used for data-processing instructions can be used (Logical Shift Left, Logical Shift Right, Arithmetic Shift Right and Rotate Right), but Logical Shift Left is the most useful as it allows an array indexed to be scaled by the size of each array element.

Scaled register offsets are only available for the word and unsigned byte instructions.

- Shift Left: Scale up
- Shift Right: Scale down

Examples of Doing Address Indexing …

- ***Preindex:****without writeback* Use [Base+Offset] and Do not do post-update of base address
  - data: *mem[base+offset]*
  - Base address register: *not updated*

mem32[address] = data

```
LDR r0,[r1, #4]        ; r0 = mem32[r1+4]
```

- ***Postindex*:** → Use [base] and Do post-update of base address
  - data: *mem[base]*
  - Base address register: *base + offset*

```
LDR r0,[r1], #4        ; r0 = mem32[r1], then r1 = r1 + 4
```

- ***Preindex with writeback* (also called *auto-indexing*)**
  - Data: *mem[base+offset]*
  - Base address register: *base + offset*

```
LDR r0, [r1,#4]!       ; r0 = mem32[r1+4], then r1 = r1+4
```

→ Use [Base+Offset] and Do post-update of base address

More Examples of Determining Addresses …

Offset	The base register and offset are added or subtracted to form the memory address.
Pre-indexed	The base register and offset are added or subtracted to form the memory address. The base register is then updated with this new address, to allow automatic indexing through an array or memory block.
Post-indexed	The value of the base register alone is used as the memory address. The base register and offset are added or subtracted and this value is stored back in the base register, to allow automatic indexing through an array or memory block.

- Example 1: Pre-indexed Address without writeback

```
LDR     R11, [R1, R2]          ; Load R11 from the address in R1 + R2
STRB    R10, [R7, -R4]         ; Store byte from R10 to addr in R7 - R4
```

- Example 2: Pre-indexed or Pre-indexed with writeback

```
LDR     R11, [R3, R5, LSL #2]  ; Load R11 from R3 + (R5 x 4)
LDR     R1, [R0, #4]!          ; Load R1 from R0 + 4, then R0 = R0 + 4
STRB    R7, [R6, #-1]!         ; Store byte from R7 to R6 - 1,
                               ;  then R6 = R6 - 1
```

Examples of LDR Using Different Addressing Modes

More

(Summary)

	Instruction	R0 =	R1 +=
Preindex with writeback	LDR r0, [r1, #0x4]!	mem32[r1+0x4]	0x4
	LDR r0, [r1,r2]!	mem32[r1+r2]	r2
	LDR r0,[r1, r2, LSR#0x4]!	mem32[r1+(r2 LSR 0x4)]	(r2 LSR 0x4)
Preindex Without writeback	LDR r0, [r1, #0x4]	mem32[r1+0x4]	not updated
	LDR r0, [r1, r2]	mem32[r1+r2]	not updated
	LDR r0, [r1, -r2, LSR #0x4]	Mem32[r1-(r2 LSR 0x4)]	not updated
Postindex	LDR r0, [r1], #0x4	mem32[r1]	0x4
	LDR r0, [r1], r2	Mem32[r1]	r2
	LDR r0, [r1], r2 LSR #0x4	mem32[r1]	(r2 LSR 0x4)

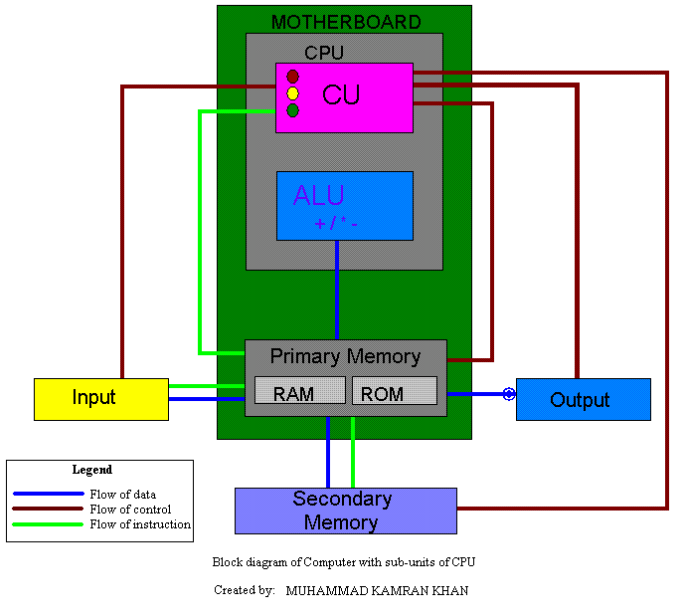
Single-Register Load-Store - Addressing Mode

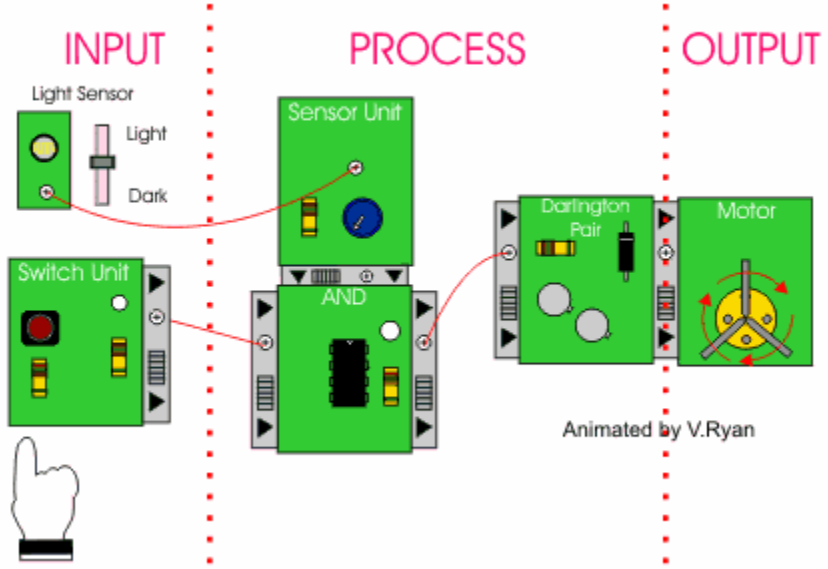
(Summary)

Addressing mode and index method	Addressing syntax
Preindex with immediate offset	[Rn, #+/-offset_10]
Preindex with register offset	[Rn, +/-Rm]
Preindex with scaled register offset	[Rn, +/-Rm, shift #shift_imm]
Preindex writeback with immediate offset	[Rn, #+/-offset_10]!
Preindex writeback with register offset	[Rn, +/-Rm]!
Preindex writeback with scaled register offset	[Rn, +/-Rm, shift #shift_imm]
Immediate postindexed	[Rn], #+/-offset_10]
Register postindexed	[Rn], +/-Rm!
Scaled register postindexed	[Rn], +/-Rm, shift #shift_imm

Outline

- Nature of Data Input and Output
- Memory-Mapped I/O Devices
- Basics of Address Indexing
- ARM Instructions for Data I/O
- Maskable Data Registers in ARM





Questions to focus on …

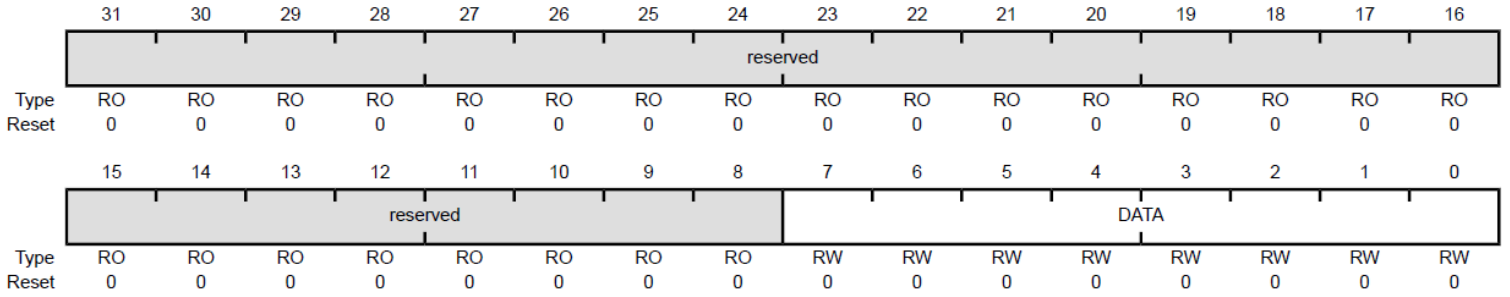
1. What is the address of port for getting input data?
2. What is the address of port for sending output data?
3. What are the instructions for doing input and output?

GPIO Data (GPIODATA)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
GPIO Port F (APB) base: 0x4002.5000
GPIO Port F (AHB) base: 0x4005.D000
Offset 0x000
Type RW, reset 0x0000.0000

Pay Attention:

The first 12 bits are all zeros. Hence, no pin is being selected for input and/or output.



Encoding of Instructions in ARM

Encoding ≠ Formatting

- S=1: signed
- S=0: unsigned

- S=1: update CPSR
- S=0: not update CPSR

	31 30 29 28	27 26 25	24 23 22 21	20	19 18 17 16	15 14 13 12	11 10 9 8	7	6 5	4	3 2 1 0
Data processing immediate shift	cond [1]	0 0 0	opcode	S	Rn	Rd	shift amount		shift	0	Rm
Miscellaneous instructions: See Figure A3-4	cond [1]	0 0 0	1 0 x x	0	x x x x x x x x x x x x				x x	0	x x x x
Data processing register shift [2]	cond [1]	0 0 0	opcode	S	Rn	Rd	Rs	0	shift	1	Rm
Miscellaneous instructions: See Figure A3-4	cond [1]	0 0 0	1 0 x x	0	x x x x x x x x x x x x			0	x x	1	x x x x
Multiplies: See Figure A3-3 Extra load/stores: See Figure A3-5	cond [1]	0 0 0	x x x x x x x x x x x x x x x x					1	x x	1	x x x x
Data processing immediate [2]	cond [1]	0 0 1	opcode	S	Rn	Rd	rotate	immediate			
Undefined instruction	cond [1]	0 0 1	1 0 x	0 0	x x x x x x x x x x x x x x x x						
Move immediate to status register	cond [1]	0 0 1	1 0 R	1 0	Mask	SBO	rotate	immediate			
Load/store immediate offset	cond [1]	0 1 0	P U B W L		Rn	Rd	immediate				
Load/store register offset	cond [1]	0 1 1	P U B W L		Rn	Rd	shift amount		shift	0	Rm
Media instructions [4]: See Figure A3-2	cond [1]	0 1 1	x x x x x x x x x x x x x x x x							1	x x x x
Architecturally undefined	cond [1]	0 1 1	1 1 1 1 1	x x x x x x x x x x x x				1 1 1 1			x x x x
Load/store multiple	cond [1]	1 0 0	P U S W L		Rn	register list					
Branch and branch with link	cond [1]	1 0 1	L	24-bit offset							
Coprocessor load/store and double register transfers	cond [3]	1 1 0	P U N W L		Rn	CRd	cp_num	8-bit offset			
Coprocessor data processing	cond [3]	1 1 1 0	opcode1		CRn	CRd	cp_num	opcode2		0	CRm
Coprocessor register transfers	cond [3]	1 1 1 0	opcode1	L	CRn	Rd	cp_num	opcode2		1	CRm
Software interrupt	cond [1]	1 1 1 1	swi number								
Unconditional instructions: See Figure A3-6	1 1 1 1	x x x x x x x x x x x x x x x x									

LDR and STR Instructions for Data I/O

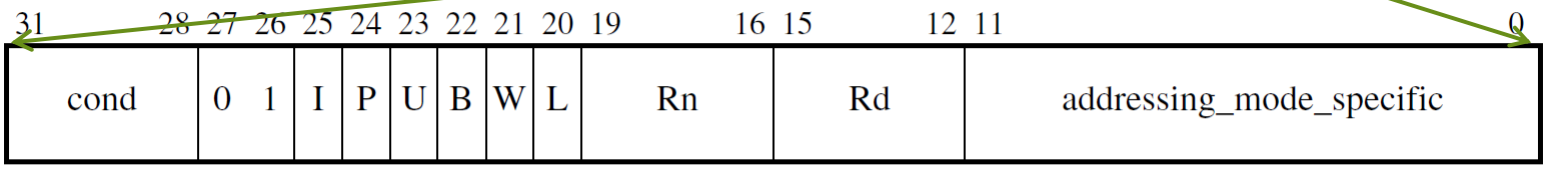
Load instructions load a single value from memory and write it to a general-purpose register.

Store instructions read a value from a general-purpose register and store it to memory.

- Load (input to ALU) and store (output from ALU) words and unsigned bytes

This is the value at the address from register PC

```
LDR|STR{<cond>}{B}{T} Rd, <addressing_mode>
```



I, P, U, W Are bits that distinguish between different types of <addressing_mode>.

L bit Distinguishes between a Load (L==1) and a Store instruction (L==0).

B bit Distinguishes between an unsigned byte (B==1) and a word (B==0) access.

Rn Specifies the base register used by <addressing_mode>.

Rd Specifies the register whose contents are to be loaded or stored.

Examples of Using LDR Instructions …

- **r0 = 0x00000000, r1 = 0x00009000,** ← Initial values inside r0 and r1
- **Content of:**
  - mem32[0x00009000] = 0x01010101 ← Set 0x01010101 to 0x00009000
  - mem32[0x00009004] = 0x02020202 ← Set 0x02020202 to 0x00009004

Case 1

- ***Preindexing:* without writeback** — **LDR r0, [r1, #4]** → Get value from 0x00009004
  - r0 = 0x02020202, r1=0x00009000

Case 2

- ***Postindexing:*** — **LDR r0, [r1], #4** → **Post-update r1** → Get value from 0x00009000
  - r0 = 0x01010101, r1=0x00009004

Case 3

- ***Preindexing with writeback:*** — **LDR r0, [r1, #4]!** → **Post-update r1** → Get value from 0x00009004
  - r0 = 0x02020202, r1=0x00009004

Examples of Using STR Instructions …

Pre-indexing Without Writeback

```
LDR     R1, [R0]                ; Load R1 from the address in R0
LDR     R8, [R3, #4]            ; Load R8 from the address in R3 + 4
LDR     R12, [R13, #-4]         ; Load R12 from R13 - 4
STR     R2, [R1, #0x100]        ; Store R2 to the address in R1 + 0x100

LDRB    R5, [R9]                ; Load byte into R5 from R9
                                ;  (zero top 3 bytes)
LDRB    R3, [R8, #3]            ; Load byte to R3 from R8 + 3
                                ;  (zero top 3 bytes)
STRB    R4, [R10, #0x200]       ; Store byte from R4 to R10 + 0x200

LDR     R11, [R1, R2]           ; Load R11 from the address in R1 + R2
STRB    R10, [R7, -R4]          ; Store byte from R10 to addr in R7 - R4

LDR     R11, [R3, R5, LSL #2]   ; Load R11 from R3 + (R5 x 4)
```

Pre-indexing With Writeback

```
LDR     R1, [R0, #4]!           ; Load R1 from R0 + 4, then R0 = R0 + 4
STRB    R7, [R6, #-1]!          ; Store byte from R7 to R6 - 1,
                                ;  then R6 = R6 - 1
```

Post-indexing

```
LDR     R3, [R9], #4            ; Load R3 from R9, then R9 = R9 + 4
STR     R2, [R5], #8            ; Store R2 to R5, then R5 = R5 + 8
```

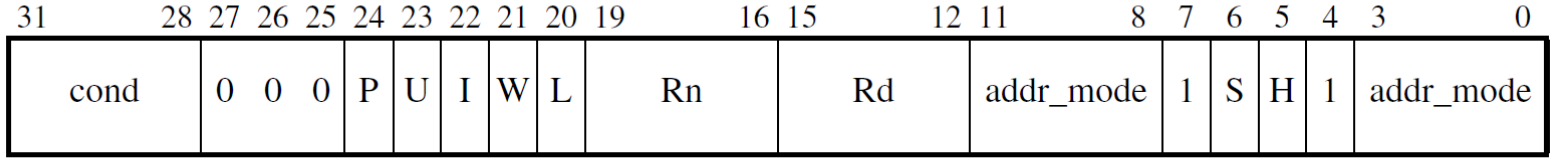
LDR and STR Instructions for Data I/O

Load instructions load a single value from memory and write it to a general-purpose register.

Store instructions read a value from a general-purpose register and store it to memory.

- Load (input to ALU) and store (output from ALU) halfwords or doublewords, and unsigned bytes

```
LDR|STR{<cond>}D|H|SH|SB  Rd, <addressing_mode>
```



addr_mode Are addressing-mode-specific bits.

I, P, U, W Are bits that specify the type of addressing mode

- S=1: signed
- S=0: unsigned

L, S, H These bits combine to specify signed or unsigned loads or stores, and doubleword, halfword, or byte accesses.

Rn Specifies the base register used by the addressing mode.

Rd Specifies the register whose contents are to be loaded or stored.

Example of Using LDR and STR …

```
LDR     R0, [PC,  #0x40]    ; Load R0 from PC + 0x40 (= address of
                            ;  the LDR instruction + 0x40)
LDR     R0, [R1], R2        ; Load R0 from R1, then R1 = R1 + R2

LDRH    R1, [R0]            ; Load halfword to R1 from R0
                            ;  (zero top 2 bytes)
LDRH    R8, [R3, #2]        ; Load halfword into R8 from R3 + 2
LDRH    R12, [R13, #-6]     ; Load halfword into R12 from R13 - 6
STRH    R2, [R1, #0x80]     ; Store halfword from R2 to R1 + 0x80
```

Pre-indexing Without Writeback

```
LDRSH   R5, [R9]            ; Load signed halfword to R5 from R9
LDRSB   R3, [R8, #3]        ; Load signed byte to R3 from R8 + 3
LDRSB   R4, [R10, #0xC1]    ; Load signed byte to R4 from R10 + 0xC1

LDRH    R11, [R1, R2]       ; Load halfword into R11 from address
                            ;  in R1 + R2
STRH    R10, [R7, -R4]      ; Store halfword from R10 to R7 - R4
```

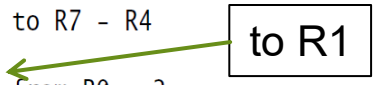
Pre-indexing With Writeback

```
LDRSH   R1, [R0, #2]!       ; Load signed halfword R1 from R0 + 2,
                            ;  then R0 = R0 + 2

LDRSB   R7, [R6, #-1]!      ; Load signed byte to R7 from R6 - 1,
                            ;  then R6 = R6 - 1
```

Post-indexing

```
LDRH    R3, [R9], #2        ; Load halfword to R3 from R9,
                            ;  then R9 = R9 + 2
STRH    R2, [R5], #8        ; Store halfword from R2 to R5,
                            ;  then R5 = R5 + 8
LDRD    R4, [R9]            ; Load word into R4 from
                            ;  the address in R9
                            ; Load word into R5 from
                            ;  the address in R9 + 4
STRD    R8, [R2, #0x2C]     ; Store R8 at the address in
                            ;    R2 + 0x2C
                            ; Store R9 at the address in
                            ;    R2 + 0x2C+4
```



MOV Instruction

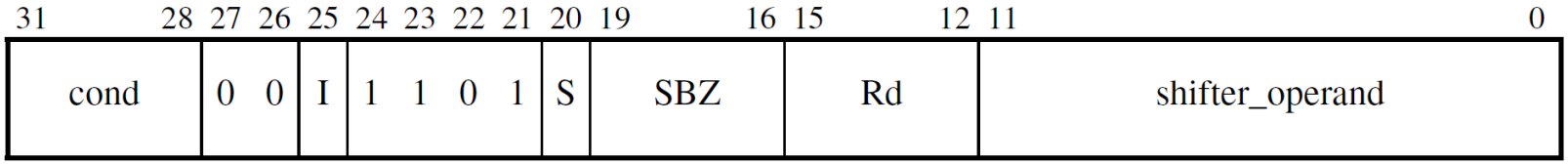
- S=1: update CPSR
- S=0: not update CPSR

- Purpose:

MOV (Move) writes a value to the destination register. The value can be either an immediate value or a value from a register, and can be shifted before the write.

MOV can optionally update the condition code flags, based on the result.

- Coding:



- Syntax: `MOV{<cond>}{S} <Rd>, <shifter_operand>`

S	Sets the S bit (bit[20]) in the instruction to 1 and specifies that the instruction updates the CPSR. If S is omitted, the S bit is set to 0 and the CPSR is not changed by the instruction.

Example

- MOV: move a 32-bit word (i.e., four bytes).
- MOVW: move a 16-bit halfword into the lower bits [15, 0] of register.
- MOVT: move a 16-bit halfword into the upper bits [31, 16] of register.

- What is the value of r1 at the end of the following code?

```
3242ba66    f6454118    movw    r1, 0x5c18
3242ba6a        466f    mov     r7, sp
3242ba6c    f6c0415a    movt    r1, 0xc5a
3242ba70    f2460002    movw    r0, 0x6002
3242ba74    f6c0405a    movt    r0, 0xc5a
3242ba78        4479    add     r1, pc
3242ba7a        4478    add     r0, pc
3242ba7c        6809    ldr     r1, [r1, #0]
```

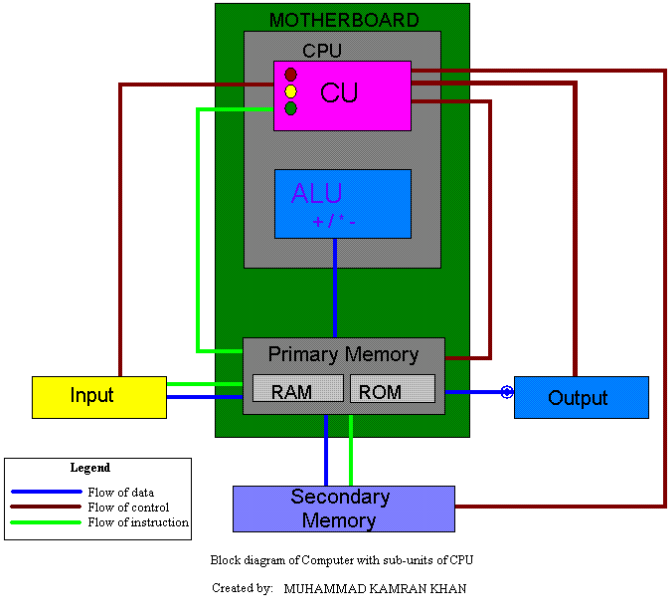
Answer:

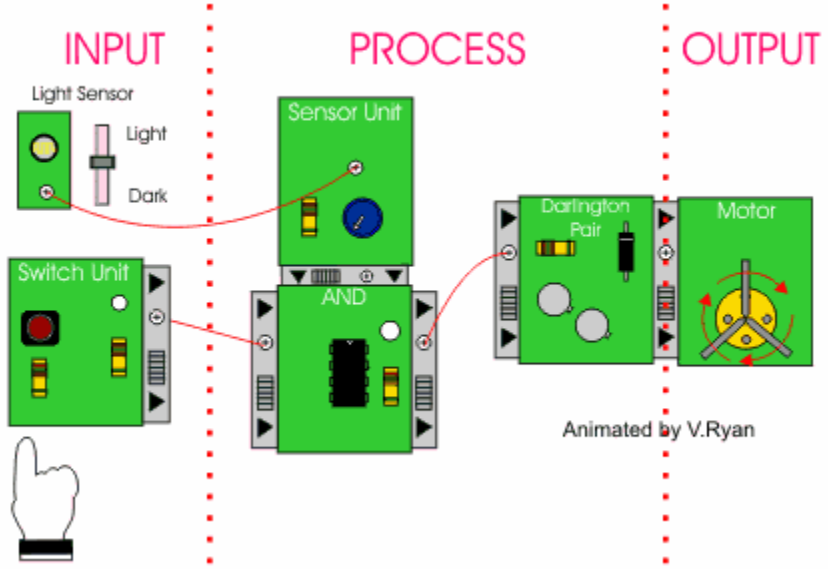
The value of r1 is the value at the following address:

r1=0x0c5a.5c18+0x3242.ba78

Outline

- Nature of Data Input and Output
- Memory-Mapped I/O Devices
- Basics of Address Indexing
- ARM Instructions for Data I/O
- Maskable Data Registers in ARM

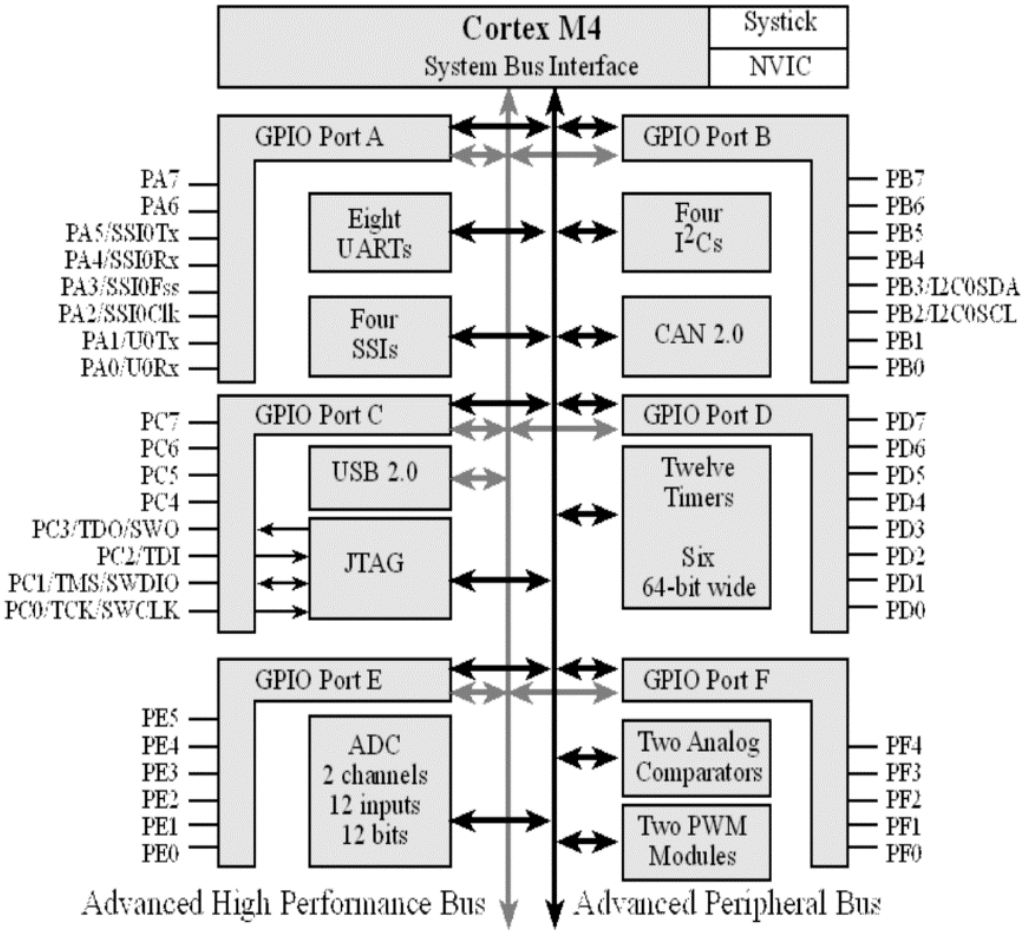




Question to focus on …

- All the data registers in ARM Cortex-M4 are maskable.

- Question:
  - What is maskable data register?
  - Where to add and retrieve masks imposed to data flow?



What is Maskable Data Register?

- A data register, which has a pin selection mask, is called Maskable data register.
- Example: Inside the first 12 bits, Bit 9 to Bit 2 are for pin selection.

GPIO_PORTF_DIR_R	EQU 0x40025 400	GPIO Direction
GPIO_PORTF_AFSEL_R	EQU 0x40025 420	GPIO Alt function select
GPIO_PORTF_PUR_R	EQU 0x40025 510	GPIO Pull up select
GPIO_PORTF_DEN_R	EQU 0x40025 51C	GPIO Digital enable
GPIO_PORTF_AMSEL_R	EQU 0x40025 528	GPIO Analog Mode Select
GPIO_PORTF_PCTL_R	EQU 0x40025 52C	GPIO Port Control
PF1	EQU 0x40025 008	GPIO "port" address
PF2	EQU 0x40025 010	GPIO "port" address
PF3	EQU 0x40025 020	GPIO "port" address
PF4	EQU 0x40025 040	GPIO "port" address
PFA	EQU 0x40025 038	- address for 3 ports PF[1-3]
SYSCTL_RCGCGPIO_R	EQU 0x400FE608	Run Mode Clock Gating Ctrl

Port's Address with Pin Selection Mask

Why to have maskable data registers?

Security in Control

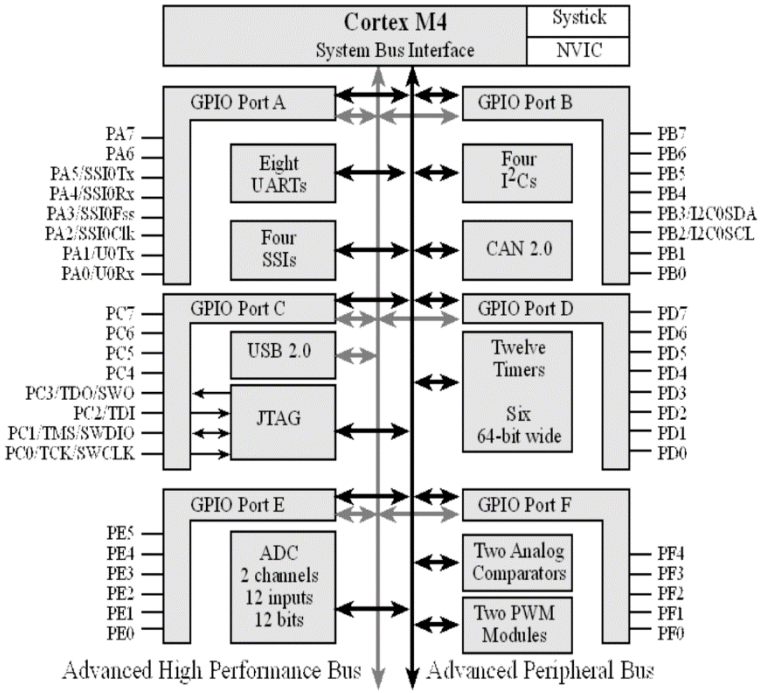
- We do not want to allow any incident such as:
  - Intention is to turn on one motor.
  - Actual effort is that several other motors have been turned on.

Security in Communication

- We do not want others to interpret the intercepted data in communication network. This is possible because:
  - Intercepted data is not filtered by any specific mask.
  - Received data is filtered by any specific mask.

ARM Cortex M4 has Six Maskable Data Registers …

- Data Registers for Input and Output



Pin Selection

GPIO Data (GPIODATA)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
GPIO Port F (APB) base: 0x4002.5000
GPIO Port F (AHB) base: 0x4005.D000
Offset 0x000
Type RW, reset 0x0000.0000

Note: Last 12 bits are not used for address.

They are used to mask Input/output data!!!

By default, choose offset = 0xFFF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved								DATA							
Type	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit/Field	Name	Type	Reset	Description
31:8	reserved	RO	0x0000.00	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
7:0	DATA	RW	0x00	GPIO Data This register is virtually mapped to 256 locations in the address space. To facilitate the reading and writing of data to these registers by independent drivers, the data read from and written to the registers are masked by the eight address lines [9:2]. Reads from this register return its current state. Writes to this register only affect bits that are not masked by ADDR[9:2] and are configured as outputs. See "Data Register Operation" on page 654 for examples of reads and writes.

Bits 2 to 9 are used for mask

Figure 10-3. GPIODATA Write Example

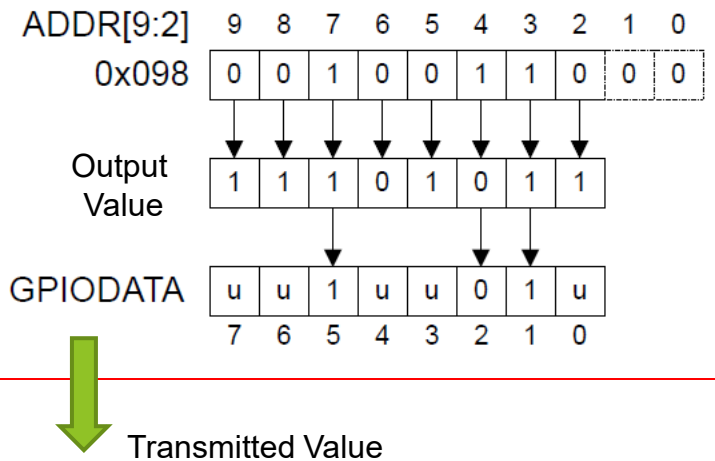
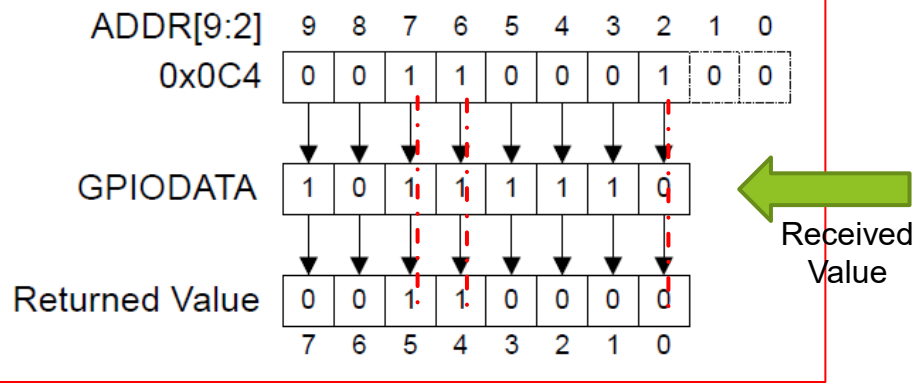
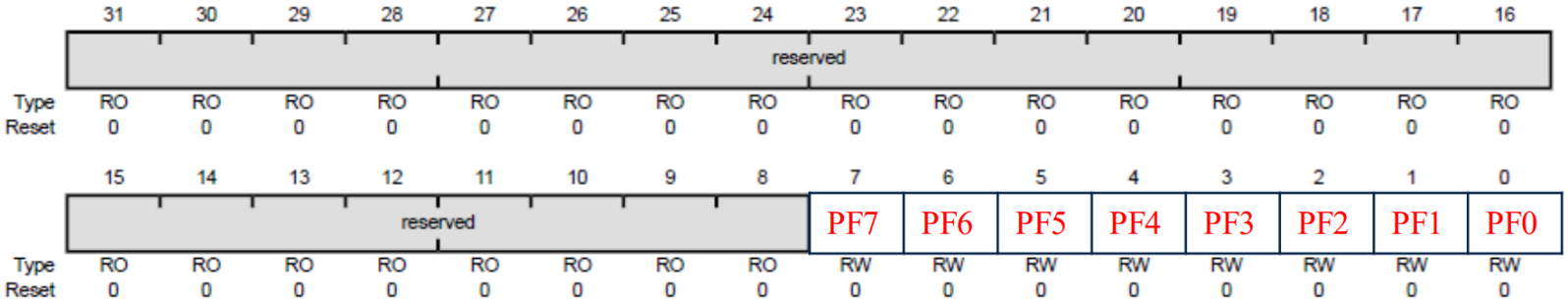


Figure 10-4. GPIODATA Read Example



Example with Port F

Name: GPIO Data (Port F) Address: 0x4002.5000

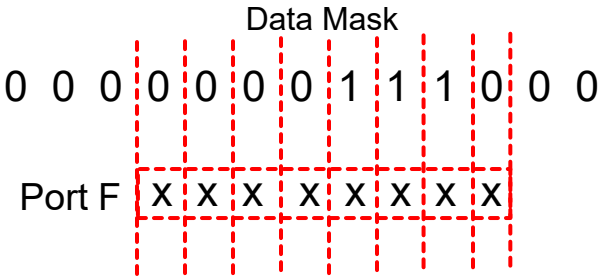


This register is virtually mapped to 256 locations in the address space. To facilitate the reading and writing of data to these registers by independent drivers, the data read from and written to the registers are masked by the eight address lines [9:2]. Reads from this register return its current state. Writes to this register only affect bits that are not masked by ADDR[9:2] and are configured as outputs.

PF0	EQU 0x40025004	ADR[9:0] = 0 0000 0100
PF1	EQU 0x40025008	ADR[9:0] = 0 0000 1000
PF2	EQU 0x40025010	ADR[9:0] = 0 0001 0000
PF3	EQU 0x40025020	ADR[9:0] = 0 0010 0000

+

PFA[1:3] EQU 0x40025038 ← 038h = 0000 0011 1000



Example with Port F (continued)

Name: GPIO Data (Port F) **Address: 0x40025 000 (0x4002.5000)**

The data read from and written to the registers are masked by the eight address lines [9:2]. Reads from this register return its current state. Writes to this register only affect bits that are not masked by ADDR[9:2] and are configured as outputs.

Address: 0x40025 000

Pin Selection

PF0	EQU 0x40025 004	PF1	EQU 0x40025 008
PF2	EQU 0x40025 010	PF3	EQU 0x40025 020
PFA[1:3]	EQU 0x40025 038		

Port F register at 0x40025 000

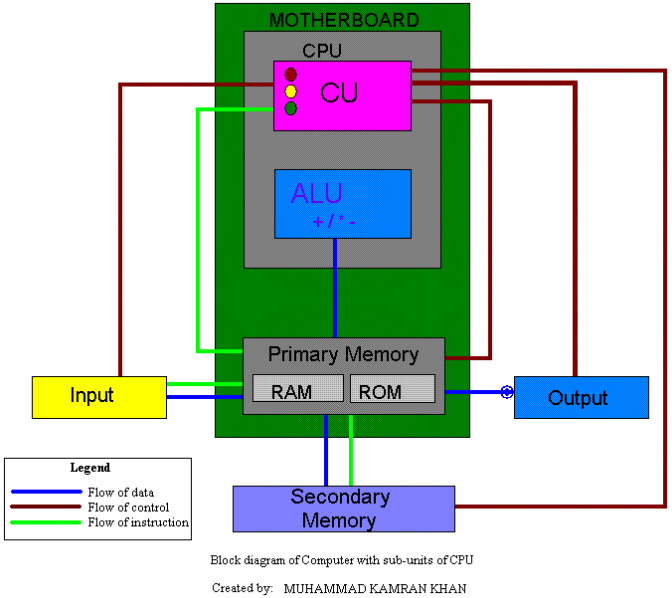
PF7	PF6	PF5	PF4	PF3	PF2	PF1	PF0

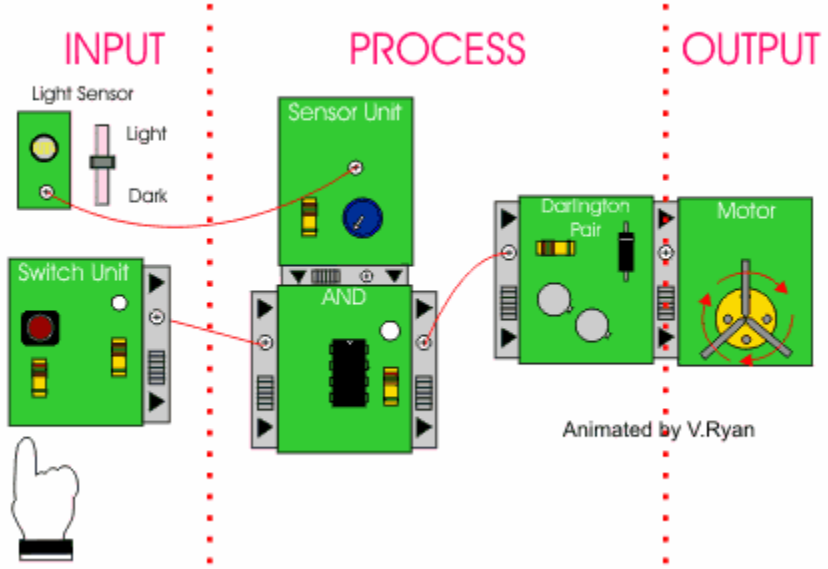
	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
0x004	0	0	0	0	0	0	0	0	0	1	x	x
0x008	0	0	0	0	0	0	0	0	1	0	x	x
0x010	0	0	0	0	0	0	0	1	0	0	x	x
0x020	0	0	0	0	0	0	1	0	0	0	x	x

0x38

Summary

- Nature of Data Input and Output
- Memory-Mapped I/O Devices
- Basics of Address Indexing
- ARM Instructions for Data I/O
- Maskable Data Registers in ARM





"Ask not what your country can do for you – ask what you can do for your country," - John F. Kennedy

"Do not think that you are needy – think that you are needed in the world", - Manis Friedman

"Study will make you knowledgeable, resourceful, and hence more needed", - Xie Ming

Thank You for Listening!