PART 2: Embedded Programming STM32

Numbers representation (2’s complement), shift operation, basic architecture,

ISEN ALL IS DIGITAL! OUEST | yncréa

Unsigned Numbers Representation

Unsigned numbers are represented naturally

The number X is coded on n bits

n bits	$0 \leq X \leq 2^n-1$
8 bits	$0 \leq X \leq 255$
16 bits	$0 \leq X \leq 65,535$
24 bits	$0 \leq X \leq 16,777,215$
32 bits	$0 \leq X \leq 4,294,967,295$

Signed Numbers Representation

Signed: Main Significant Bit (MSB)

0 => positive number 1=> negative number

Dynamic of a number represented with n bits

n bits	$-2^{n-1} \leq X \leq 2^{n-1} - 1$
8 bits	$-128 \leq X \leq 127$
16 bits	$-16{,}384 \leq X \leq 16{,}383$
24 bits	$-8{,}388{,}608 \leq X \leq 8{,}388{,}607$
32 bits	$-2{,}147{,}483{,}648 \leq X \leq -2{,}147{,}483{,}647$

ISEN | école d'ingénieurs
ALL IS DIGITAL!

Compléments à 2

Addition des nombres binaires

Principe :

- On additionne deux nombres binaires de même taille.
- bit retenue=1 si on additionne au moins deux bits à 1 (retenue entrante comprise)
- bit résultat=1 si on additionne 1 ou 3 bits à 1 (retenue entrante comprise)
- La retenue sortante au poids fort est stockée séparément du résultat, dans l'indicateur C (Carry).

Exemple :

bit	(C)	b_7	b_6	b_5	b_4	b_3	b_2	b_2	b_0
retenue									0
		1	0	1	0	1	0	1	1
+		1	0	1	1	0	0	1	1
résultat									

Addition est XOR

Table de Vérité de la logique XOR (OU Exclusif)

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

L’addition entre 2 nombres positifs est donc une opération logique Xor bit à bit

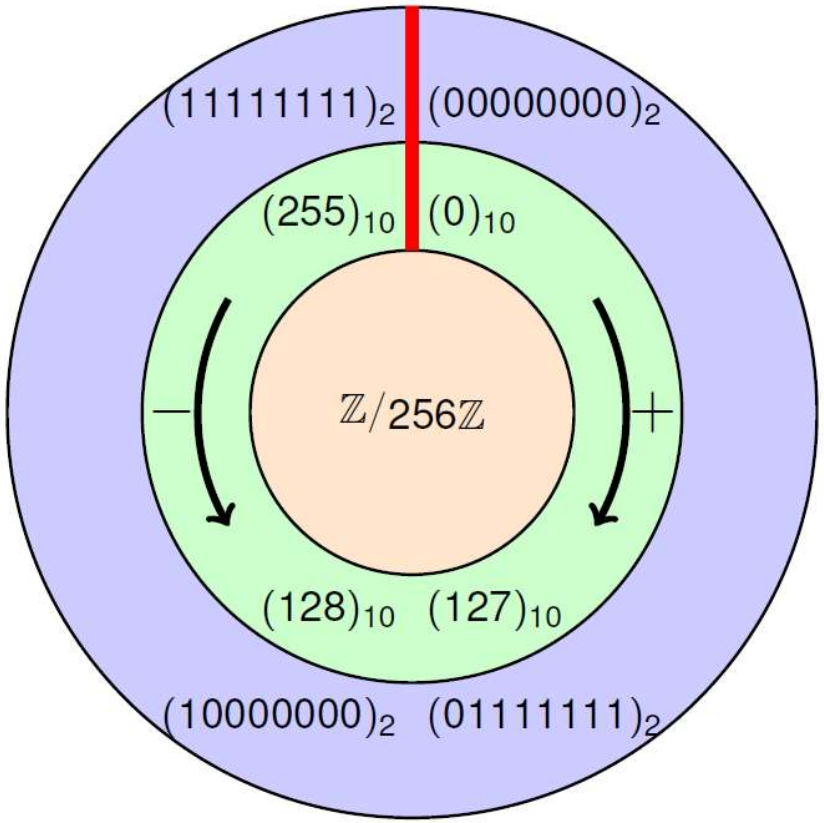
Limite représentation des nombres

Les nombres entiers représentés en machine (ici sur 8 bits) ne sont pas des nombres entiers mathématiques (ensemble $\mathbb{N}$) : ils se comportent de manière cyclique comme l'ensemble $\mathbb{Z}/256\mathbb{Z}$.

Débordement :

- indicateur Carry

Et si on veut représenter les nombres négatifs ?



Nombres négatives: Approche naïve

Une représentation naïve pourrait utiliser ce bit de poids fort comme un marqueur de signe, les autres signes donnant une valeur absolue:

0000 0010 = +2 en décimal
1000 0010 = -2 en décimal

Deux inconvénients majeurs à cette représentation:

- ✓ Possède deux représentations pour le nombre zéro: 0000 0000 et 1000 0000
- ✓ Cette représentation impose de modifier l'algorithme d'addition

Erreur avec Approche naïve

Si un des nombres est négatif, l'addition binaire usuelle donne un résultat incorrect. Par exemple 3 + (-4)

	3	0000 0011
+	-4	1000 0100
=	-1	1000 0111

Or $(1000\ 0111)_2$ = -7 en décimal au lieu de -1

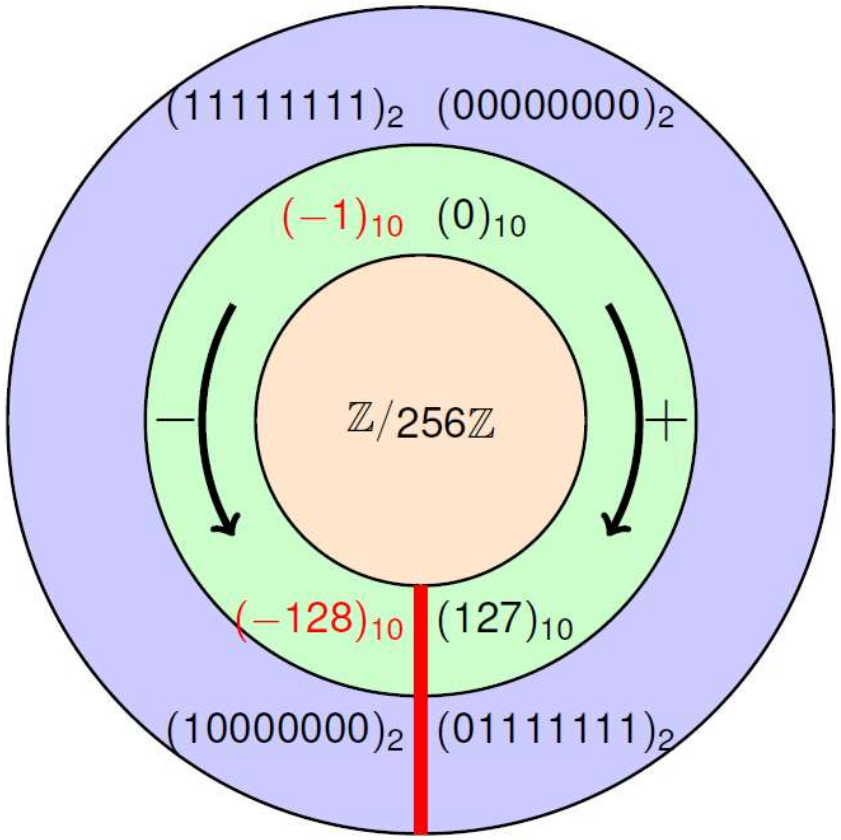
Représentation des Nombres Négatifs

Pour représenter les nombres négatifs, on choisit de donner au bit de poids fort un poids négatif (-128) : il devient le bit de signe. On parle alors de nombres signés.

- $(0xxxxxxx)_2$: nombre positif
- $(1xxxxxxx)_2$: nombre négatif

Débordement :

- indicateur oVerflow



Complément à 2

Principe du complément à 2 :

- passer de la représentation binaire de N à celle de $-N$
- remarque sur le complément à 1 :

$$(xxxxxxxx)_2 + (\overline{xxxxxxxx})_2 = (11111111)_2 = -1$$

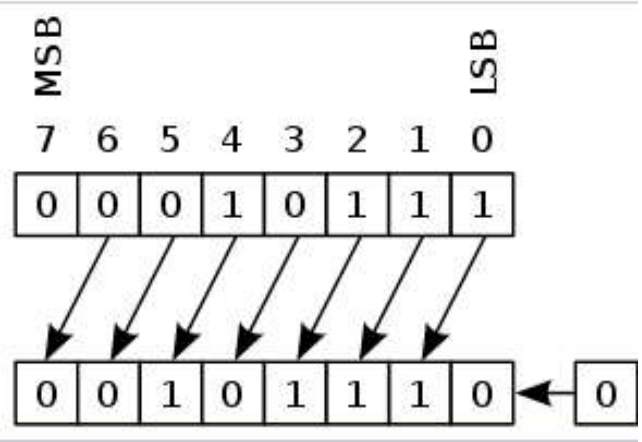
- d'où (en ignorant la retenue car nombres signés) :

$$(xxxxxxxx)_2 + ((\overline{xxxxxxxx})_2 + 1) = (00000000)_2 = 0$$

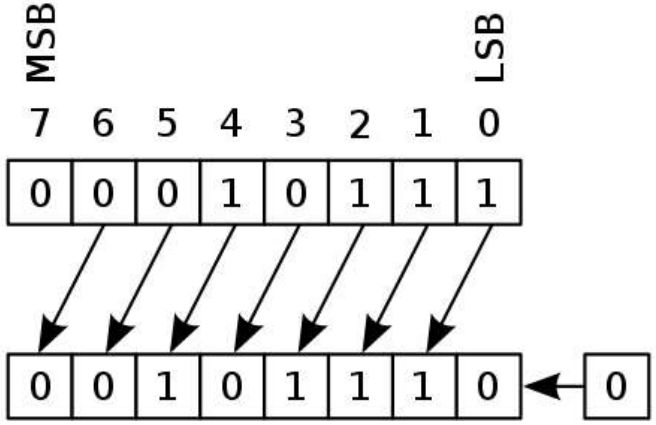
- donc si $N = (xxxxxxxx)_2$, alors $-N = (\overline{xxxxxxxx})_2 + 1$

On appelle $(\overline{xxxxxxxx})_2 + 1$ le complément à 2 de $(xxxxxxxx)_2$.

Binary Logic / Left Shift

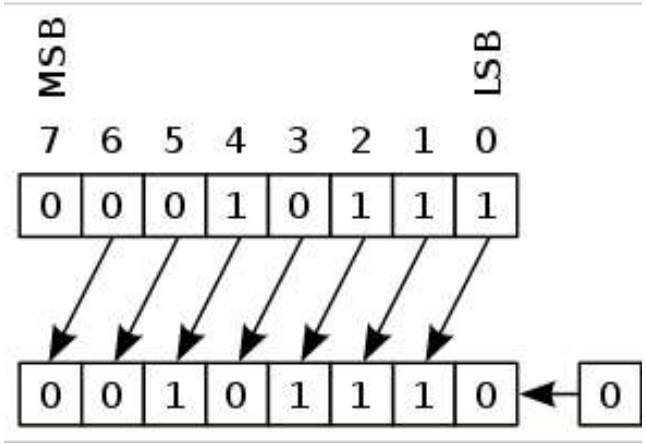


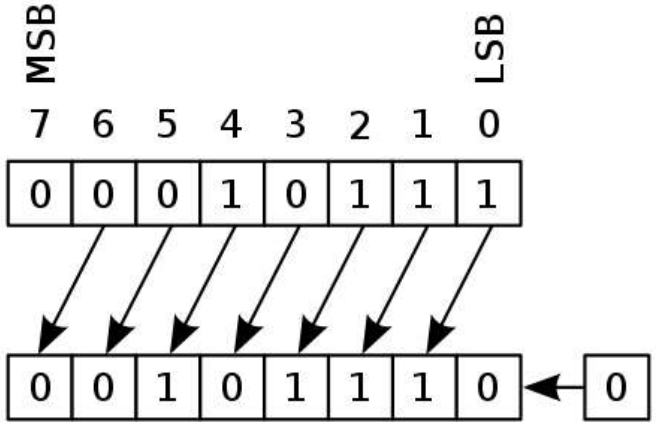
Left arithmetic shift of a binary by 1. The empty position in the LSB is filled with a zero



Left Logical shift of a binary by 1. The empty position in the LSB is filled with a zero

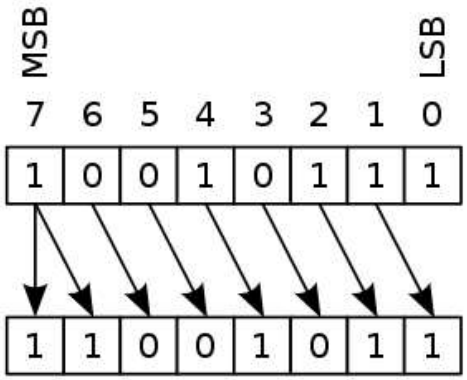
Left Shift / Multiplication by 2



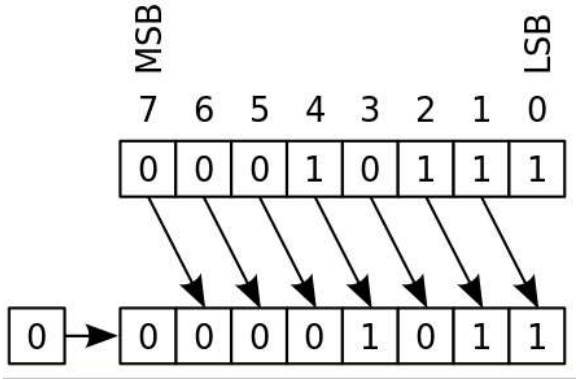


Left Shift of a binary by 1 is a multiplication by 2.
23 = 0b1011 and 46 = 0b101110

Binary Logic / Right Shift

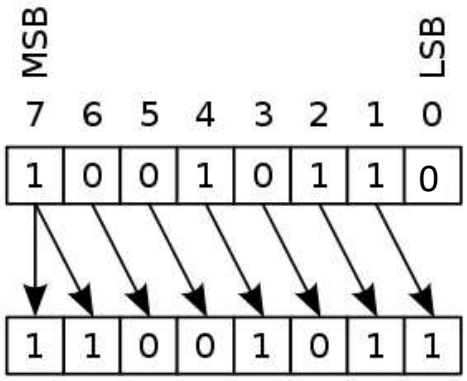


Right Arithmetic shift of a binary by 1. The empty position in the MSB is filled with a copy of the original MSB

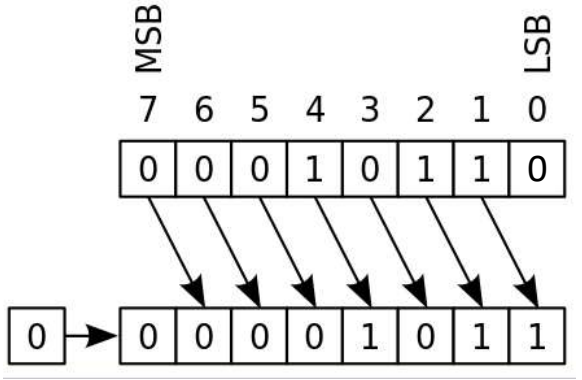


Right Logic shift of a binary by 1. The empty position in the MSB is filled with a copy of the original MSB

Right Shift / Division by 2



Right Arithmetic shift:
-106 = 0b1001 0110
-53 = 0b1100 1011

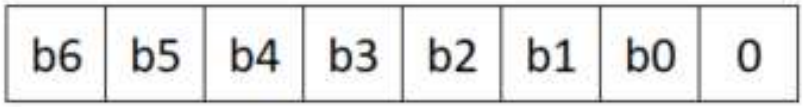


Right Logic shift:
22 = 0b0001 0110
11 = 0b0001 011

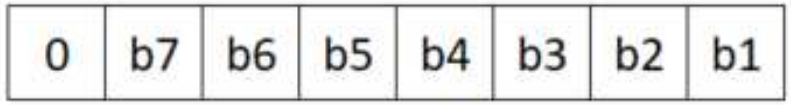
Arithmetic shifting is prefered to logic shifting when operations are performed

Shifting Summary

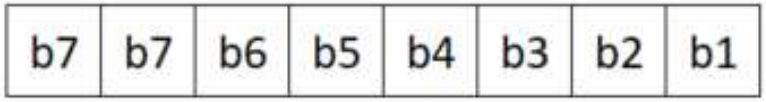
Left Arithmetic and Logic shift



Right Logic Shift



Right Arithmetic Shift



Shifting C Language

Left Shift (<<)

e.i.: 0b00100111 << 2 = 0b10011100

Right Shift (>>)

e.i.: 0b01010011 >> 2 = 0b00010100

In C language, in a program, we prefer to write:

0b0000 0110 = 1<<1 | 1<<2

to be more readable

ISEN ALL IS DIGITAL! OUEST | yncréa

Characters Representation / ASCII

–ASCII stands for American Standard Code for Information Interchange

–Codage of each character with 8 bits : A is coded as 0x41

*	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	TAB	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SPACE	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

ISEN ALL IS DIGITAL! OUEST | yncréa

Code ASCII Etendu

–Codage of each character with 8 bits

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	Ç	ü	é	â	ä	à	å	ç	ê	ë	è	ï	î	ì	Ä	Å
9	É	æ	Æ	ô	ö	ò	û	ù	ÿ	Ö	Ü	¢	£	¥	₧	ƒ
A	á	í	ó	ú	ñ	Ñ	ª	º	¿	⌐	¬	½	¼	¡	«	»
B	▒	▓	▒	┃	┤	╡	╢	╖	╕	╣	╗	╝	╜	╛	┐	└
C	┴	┬	├	─	┼	╞	╟	╚	╔	╩	╦	╠	═	╬	╧	╨
D	╤	╥	╙	╘	╒	╓	╫	╪	┘	┌	▒	▄	▌	▐	▀	
E	α	ß	Γ	π	Σ	σ	µ	τ	Φ	Θ	Ω	δ	∞	ø	∈	∩
F	≡	±	≥	≤	⌠	⌡	÷	≈	°	∙	·	√	ⁿ	²	■	

Working with Microcontrollers

Microcontrollers need a computer to be programmed

- ✓ Develop program with IDE (Integrated Development Environment)
  - C/C++ (very much similar)
  - Assembly language
- ✓ Convert the program in Hexadecimal file
- ✓ From the PC, burning the program inside the IC

Programming Layers Arduino

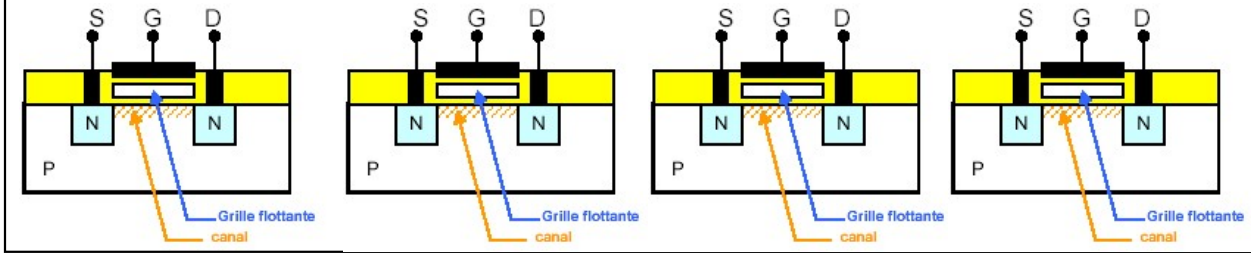
Application

Arduino Programming Interface

C / C++

Assembly Language

µController Instructions



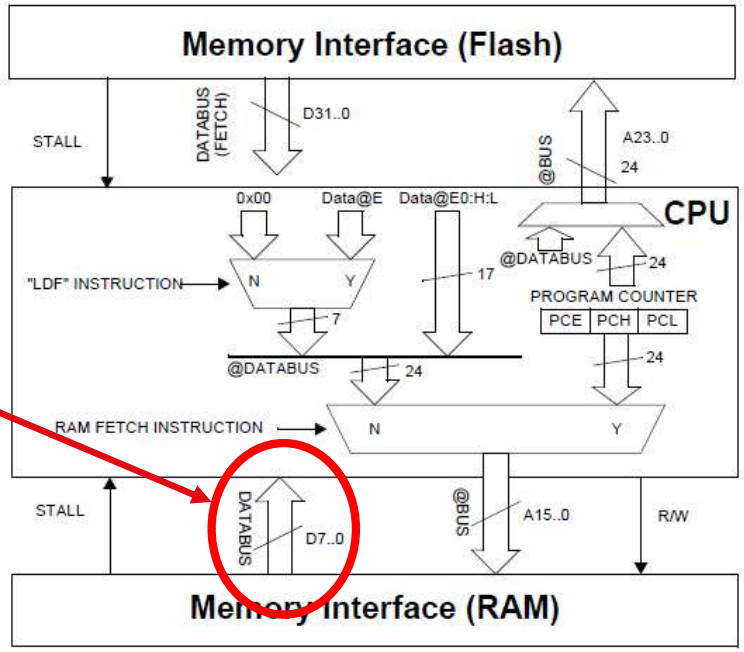
Why Low Level languages

When using a high-level language, many parameters are not controlled

- ✓ Timing execution (i.e. in an airbag application, Android (Java Vs C))
- ✓ Memory size (code optimization)
- ✓ Current consumption (i.e. in IoT many boards)
- ✓ Bugs (stack overflow for instance)
- ✓ Security

What is n-bit architecture

- ✓ The wider a data bus is, the µController can perform faster and more complex instructions
- ✓ The bus width determines the n-bit architecture
- ✓ The STM8 is an 8-bit architecture (but program memory is 32-bit wide)
- ✓ The STM32 is a 32-bit architecture
- ✓ Program memory bus is 32-bit wide



Types de données élémentaires

En C classique	En EMBARQUE
char variable sur 8 bits pouvant prendre des attributs supplémentaire signed ou unsigned souvent implicites.	**uint8_t** variable sur 8 bits non signée **int8_t** variable sur 8 bits signée
int variable sur 16 bits pouvant prendre des attributs supplémentaire signed ou unsigned souvent implicites	**uint16_t** variable sur 16 bits non signée **int16_t** variable sur 16 bits signée
Long variable sur 32 bits pouvant prendre des attributs supplémentaire signed ou unsigned souvent implicites	**uint32_t** variable sur 32 bits non signée **int32_t** variable sur 32b signée

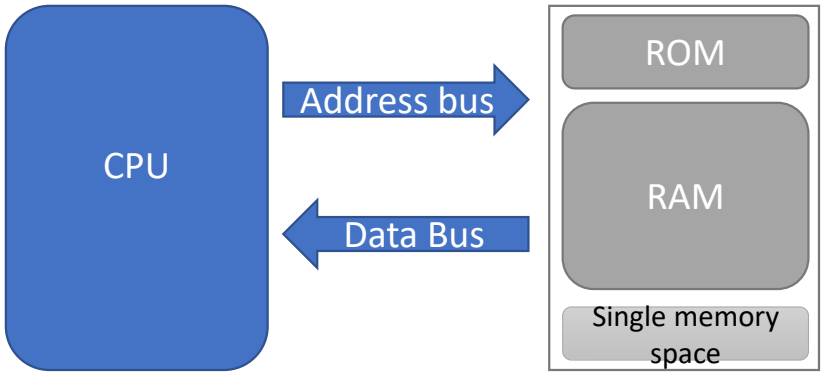
ISEN ALL IS DIGITAL! OUEST | yncréa

Types of µController

Two different memory architectures exist:

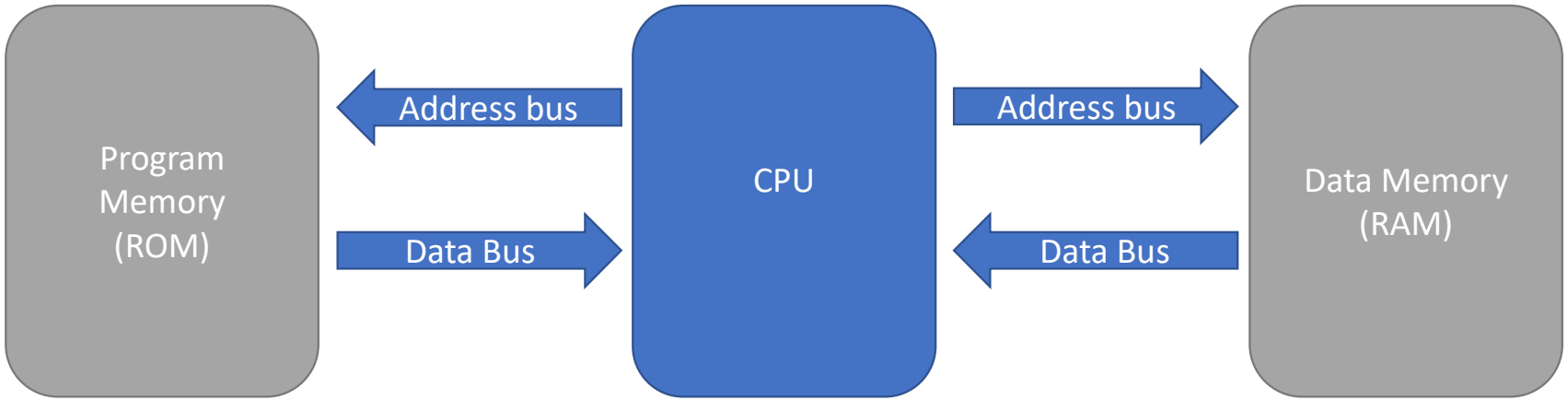
- ✓ The Harvard Architecture
- ✓ The Von Neumann architecture

The Von Neumann architecture



- ✓ Same memory and bus are used to store both data and instruction
- ✓ Cannot access program memory and data memory simultaneously
- ✓ Von Neumann architecture is susceptible to bottlenecks and system performance is affected

The Harvard architecture



- ✓ Machine instructions and data in separate memory units connected by different busses
- ✓ At least two memory address spaces to work with
- ✓ Able to run a program and access data independently, and simultaneously

Harvard Vs Von Neumann

Von Neumann Architecture	Harvard Architecture
It uses same physical memory address for instructions and data	It uses separate memory addresses for instructions and data
Processor needs two clock cycles to execute an instruction	Processor needs one cycle to complete an instruction
Simpler control unit design and development of one is cheaper and faster	Control Unit for two buses is more complicated which adds to the development cost
Data transfers and instruction fetches cannot be performed simultaneously	Data transfers and instruction fetches can be performed at the same time
Used in PC (Intel)	Used in microcontrollers