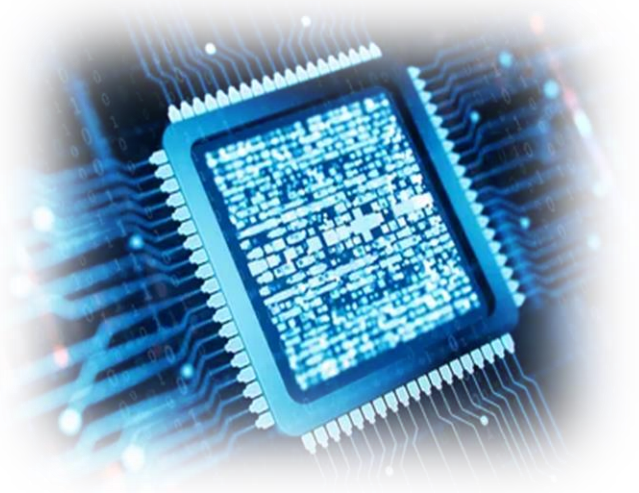


MICROPROCESSOR SYSTEMS

BLG212E

Burak Berk Üstündağ CRN: 11450

Gökhan İnce / Ayşe Yılmazer CRN: 11446

Faculty of Computer and Informatics Engineering
Istanbul Technical University

Week 3: Number Systems, Logical & Arithmetic Operations

Representing Data (Number) in Computers

Computers store numbers in binary form

One Byte = 8 bits is the unit for defining bit lengths

Assume 1 Byte of storage:

It can store unsigned numbers from

%0000 0000 to %1111 1111 or
$00 to $FF or
0 to 255

Representing Data (Number) in Computers

1 Byte of storage:

It can store signed numbers from

%0111 1111 to %1111 1111
Most Significant Bit, MSB is the sign bit,
0 is positive, 1 is negative

127 to -127

%1111 1111 -127
%1000 0001 -1
%0000 0000 0
%0000 0001 1
%0111 1111 127

Representing Data (Number) in Computers

Example: +7 0000 0111

Signed numbers : -7 -> 1000 0111

Complement numbers :

1's complement : 1111 1000

2's complement : 1111 1001 -> -7

Taking the 2s complement of the 2s complement restores the number

Representing Data (Number) in Computers

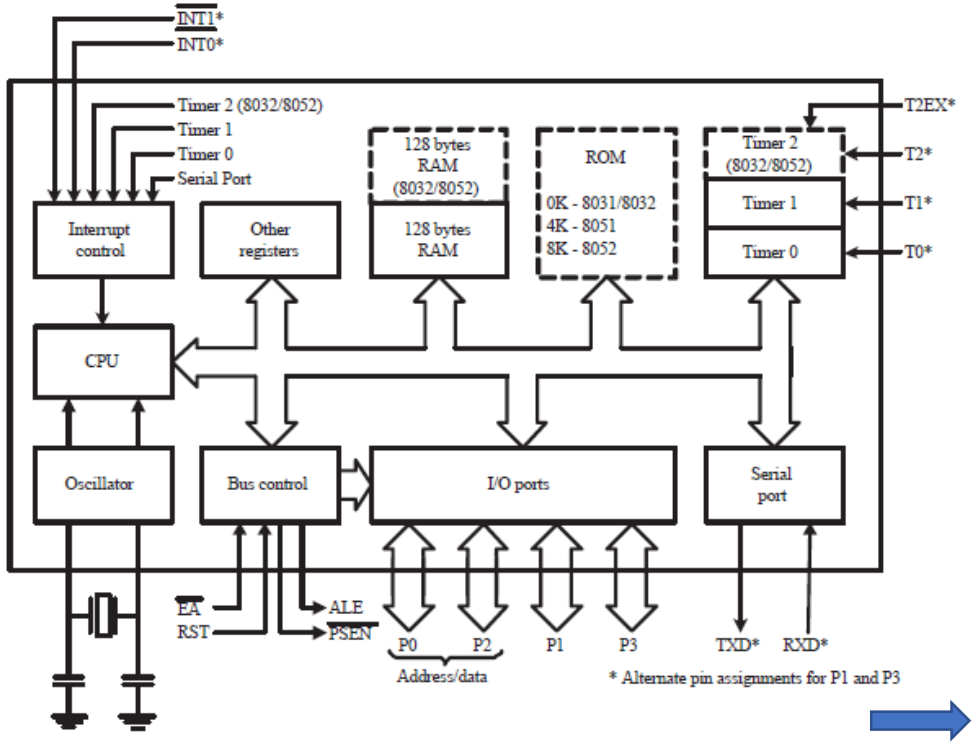
2's complement is an efficient method to represent signed numbers. For negative numbers, complement the number (replace 0s with 1s, 1s with 0s, then add 1)

1 Byte of storage can store signed numbers from

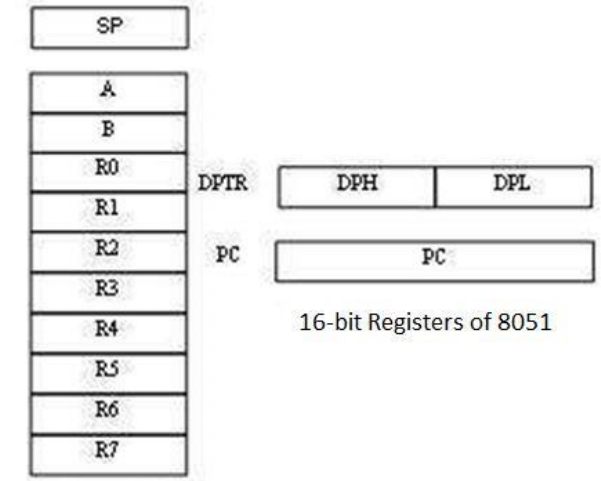
%0111 1111 to %1000 0000
Most Significant Bit, MSB is the sign bit,
0 is positive, 1 is negative

Numbers in a CPU: Registers & ALU :

A microcontroller case and historical comparison example



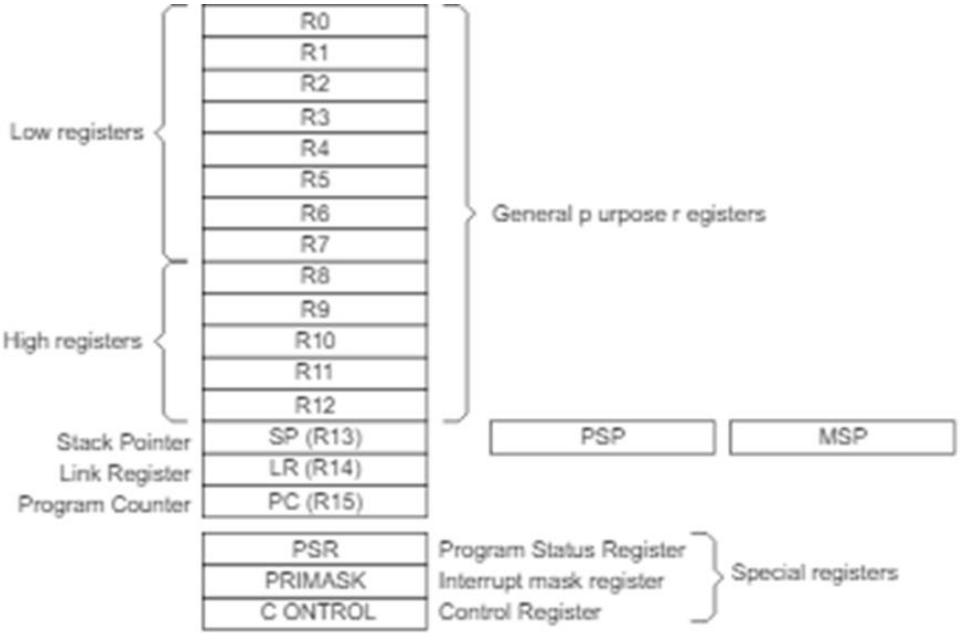
8x51 :



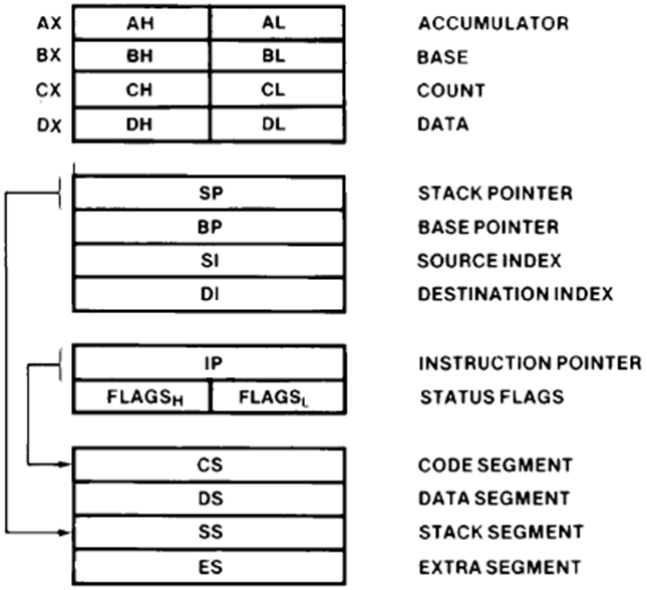
16-bit Registers of 8051

8-bit Registers of 8051

ARM Cortex M0



İ8086 :



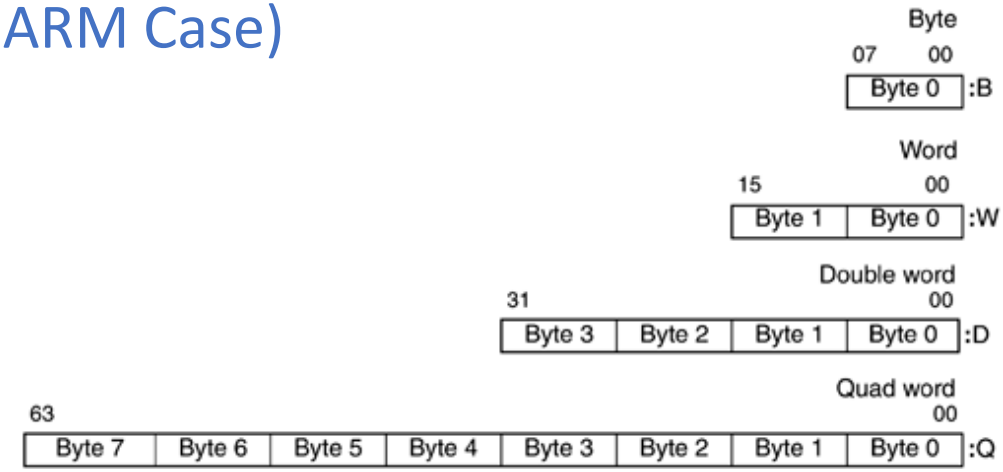
ACORN COMPUTER

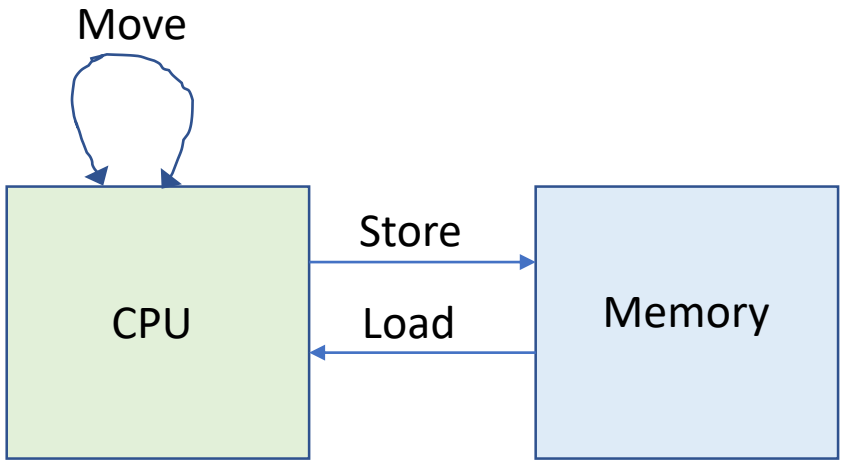
Acorn RISC Machine: ARM (1983-1985 / ARM1)
Advanced RISC Machine: ARM (1990)

1968 intel® 2006 intel

8051: 1981
8086: 1978

Data in Registers & the Memory (ARM Case)





MOV, MOVN, ….

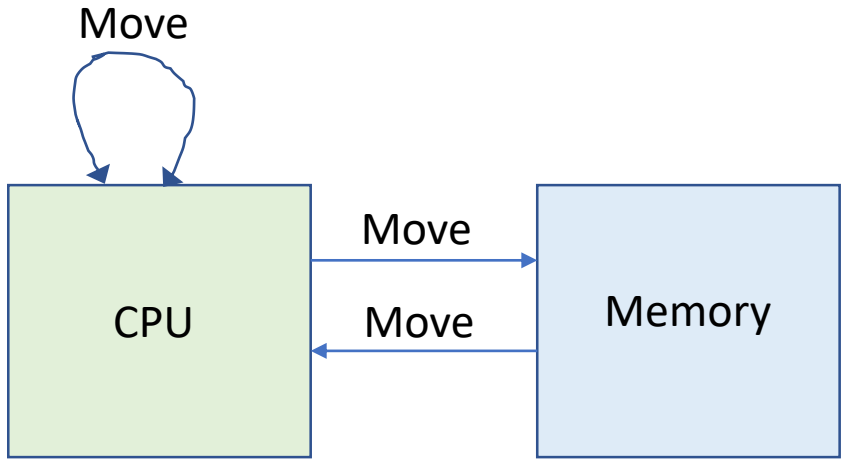
Allows complex instructions:

MOV R1, R2, LSL #2 ; R1 = R2 << 2

ldr = Load Word
ldrh = Load unsigned Half Word
ldrsh = Load signed Half Word
ldrb = Load unsigned Byte
ldrsb = Load signed Bytes

str = Store Word
strh = Store unsigned Half Word
strsh = Store signed Half Word
strb = Store unsigned Byte
strsb = Store signed Byte

8051 Case:



Allows simple instructions:

MOV A,@R0 ; A=[R0]

(Intel 8x51 Microcontrollers)

MOV A,#0FAH ; A=$(FA)_{16}$

MOV A,#11111010B ; A=$(11111010)_2$

MOV A,#250D ; A=$(250)_{10}$

74H FAH

Number is Data

Complement CPL(A):

A=$(a_7\ a_6\ a_5\ a_4\ a_3\ a_2\ a_1\ a_0)_2$

$CPL(A) \rightarrow A = \overline{A}$ $\overline{A}=(\overline{a_7}\ \overline{a_6}\ \overline{a_5}\ \overline{a_4}\ \overline{a_3}\ \overline{a_2}\ \overline{a_1}\ \overline{a_0})_2$

Instructions	OpCode	Bytes	Cycles	Flags
MOV @R0,*#data*	0x76	2	1	None
MOV @R1,*#data*	0x77	2	1	None
MOV @R0,A	0xF6	1	1	None
MOV @R1,A	0xF7	1	1	None
MOV @R0,*iram addr*	0xA6	2	2	None
MOV @R1,*iram addr*	0xA7	2	2	None
MOV A,*#data*	0x74	2	1	None
MOV A,@R0	0xE6	1	1	None
MOV A,@R1	0xE7	1	1	None
MOV A,R0	0xE8	1	1	None
MOV A,R1	0xE9	1	1	None
MOV A,R2	0xEA	1	1	None
MOV A,R3	0xEB	1	1	None
MOV A,R4	0xEC	1	1	None
MOV A,R5	0xED	1	1	None
MOV A,R6	0xEE	1	1	None
MOV A,R7	0xEF	1	1	None
MOV A,*iram addr*	0xE5	2	1	None
MOV C,*bit addr*	0xA2	2	1	C
MOV DPTR,*#data16*	0x90	3	2	None

MOV A,#0FAH ; A=$(FA)_{16}$

CPL A ; A=CPL(A)

; A=$(00000101)_2$=$(05)_{16}$=05H

Instructions	OpCode	Bytes	Cycles	Flags
CPL A	0xF4	1	1	None
CPL C	0xB3	1	1	C
CPL *bit addr*	0xB2	2	1	None

Logical OR:

a	b	$a \vee b$
0	0	0
0	1	1
1	0	1
1	1	1

$A=(a_7\ a_6\ a_5\ a_4\ a_3\ a_2\ a_1\ a_0)_2$

$R=(r_7\ r_6\ r_5\ r_4\ r_3\ r_2\ r_1\ r_0)_2$

ORL A,R → A = A V R

$AVR=(a_7 Vr_7\ a_6 Vr_6\ a_5 Vr_5\ a_4 Vr_4\ a_3 Vr_3\ a_2 Vr_2\ a_1 Vr_1\ a_0 Vr_0)_2$

MOV A,#0A2H ; $A=(A2)_{16}$

ORL A,#0F0H ; $A=AV(11110000)_2$

; $A=(101000010)_2 V(11110000)_2$

; $A=(11110010)_2=(0F2)_{16}$ =F2H

MOV A,#11D ; $A=(11)_{10}=(00001011)_2$

MOV R0,#00110011B ; $R0=(00110011)_2$

ORL A,R0 ; A=AVR0

; $A=(00001011)_2 V(00110011)_2$

; $A=(00111011)_2=(3B)_{16}$ =3BH

Instructions	OpCode	Bytes	Cycles	Flags
ORL *iram addr*,A	0x42	2	1	None
ORL *iram addr*,*#data*	0x43	3	2	None
ORL A,*#data*	0x44	2	1	None
ORL A,*iram addr*	0x45	2	1	None
ORL A,@R0	0x46	1	1	None
ORL A,@R1	0x47	1	1	None
ORL A,R0	0x48	1	1	None
ORL A,R1	0x49	1	1	None
ORL A,R2	0x4A	1	1	None
ORL A,R3	0x4B	1	1	None
ORL A,R4	0x4C	1	1	None
ORL A,R5	0x4D	1	1	None
ORL A,R6	0x4E	1	1	None
ORL A,R7	0x4F	1	1	None
ORL C,*bit addr*	0x72	2	2	C
ORL C,/*bit addr*	0xA0	2	1	C

8086: OR

ARM: ORR

Logical AND:

a	b	$a \wedge b$
0	0	0
0	1	0
1	0	0
1	1	1

$A=(a_7\ a_6\ a_5\ a_4\ a_3\ a_2\ a_1\ a_0)_2$

$R=(r_7\ r_6\ r_5\ r_4\ r_3\ r_2\ r_1\ r_0)_2$

ANL A,R → $A=A \wedge R$

$A \wedge R=(a_7 \wedge r_7\ a_6 \wedge r_6\ a_5 \wedge r_5\ a_4 \wedge r_4\ a_3 \wedge r_3\ a_2 \wedge r_2\ a_1 \wedge r_1\ a_0 \wedge r_0)_2$

```
MOV   A,#0A2H    ; A=(A2)16
ANL   A,#0F0H    ;  A=A∧(11110000)2
                 ; A=(101000010)2∧(11110000)2
                 ; A=(10100000)2=(0A0)16 =A0H
```

```
MOV   A,#11D              ;  A=(11)10=(00001011)2
MOV   R0,#00110011B       ;  R0=(00110011)2
ANL   A,R0                ;  A=A∧R0
                          ; A=(00001011)2∧(00110011)2
                          ; A=(00000011)2=(03)16 =03H
```

Instructions	OpCode	Bytes	Cycles	Flags
ANL iram addr,A	0x52	2	1	None
ANL iram addr,#data	0x53	3	2	None
ANL A,#data	0x54	2	1	None
ANL A,iram addr	0x55	2	1	None
ANL A,@R0	0x56	1	1	None
ANL A,@R1	0x57	1	1	None
ANL A,R0	0x58	1	1	None
ANL A,R1	0x59	1	1	None
ANL A,R2	0x5A	1	1	None
ANL A,R3	0x5B	1	1	None
ANL A,R4	0x5C	1	1	None
ANL A,R5	0x5D	1	1	None
ANL A,R6	0x5E	1	1	None
ANL A,R7	0x5F	1	1	None
ANL C,bit addr	0x82	2	1	C
ANL C,/bit addr	0xB0	2	1	C

8086: AND

ARM: AND

Exclusive OR:

$A=(a_7\ a_6\ a_5\ a_4\ a_3\ a_2\ a_1\ a_0)_2$

$R=(r_7\ r_6\ r_5\ r_4\ r_3\ r_2\ r_1\ r_0)_2$

a	b	$a \underline{\vee} b$
0	0	0
0	1	1
1	0	1
1	1	0

XRL A,R → A = A ⊕ R

$A \oplus R=(a_7 \oplus r_7\ a_6 \oplus r_6\ a_5 \oplus r_5\ a_4 \oplus r_4\ a_3 \oplus r_3\ a_2 \oplus r_2\ a_1 \oplus r_1\ a_0 \oplus r_0)_2$

```
MOV   A,#0A2H    ; A=(A2)16
XRL   A,#0F0H    ; A=A⊕(11110000)2
                 ; A=(101000010)2⊕(11110000)2
                 ; A=(01010010)2=(52)16 =52H
```

```
MOV   A,#11D              ; A=(11)10=(00001011)2
MOV   R0,#00110011B       ; R0=(00110011)2
XRL   A,R0                ; A=A⊕R0
                          ; A=(00001011)2⊕(00110011)2
                          ; A=(00111000)2=(38)16 =38H
```

Instructions	OpCode	Bytes	Cycles	Flags
XRL *iram addr*,A	0x62	2	1	None
XRL *iram addr*,#*data*	0x63	3	2	None
XRL A,#*data*	0x64	2	1	None
XRL A,*iram addr*	0x65	2	1	None
XRL A,@R0	0x66	1	1	None
XRL A,@R1	0x67	1	1	None
XRL A,R0	0x68	1	1	None
XRL A,R1	0x69	1	1	None
XRL A,R2	0x6A	1	1	None
XRL A,R3	0x6B	1	1	None
XRL A,R4	0x6C	1	1	None
XRL A,R5	0x6D	1	1	None
XRL A,R6	0x6E	1	1	None
XRL A,R7	0x6F	1	1	None

8086: XOR

ARM: EOR

8051:

```
MOV  R0,#01101001B
MOV  R1,#11000111B

MOV  A,R0     ; A=R0
ORL  A,R0     ; A=A&R1=(11101111)2
```

ARM:

r0: 01101001
r1: 11000111

```
ORR r3, r0,r1        ; r3: 11101111
AND r3,r0,r1         ; r3: 01000001
EOR r3,r0,r1         ; r3: 10101110
BIC r3, r0, r1       ; r3: 00101000
```

Subtraction in Unsigned numbers

Adding 2s complement to another unsigned number is subtraction of unsigned numbers

5-2=3

```
5:    0000 0101                        0000 0101
2:    0000 0010   2's complement   +    1111 1110
3:                                   1 0000 0011
```

End carry: discard it, the result is positive

2-5=-3

```
2:    0000 0010                        0000 0010
5:    0000 0101 2's complement     +   1111 1011
-3:                                    1111 1101
```

No end carry: result is negative-> take 2's complement to find the magnitude

Addition and subtraction in signed numbers

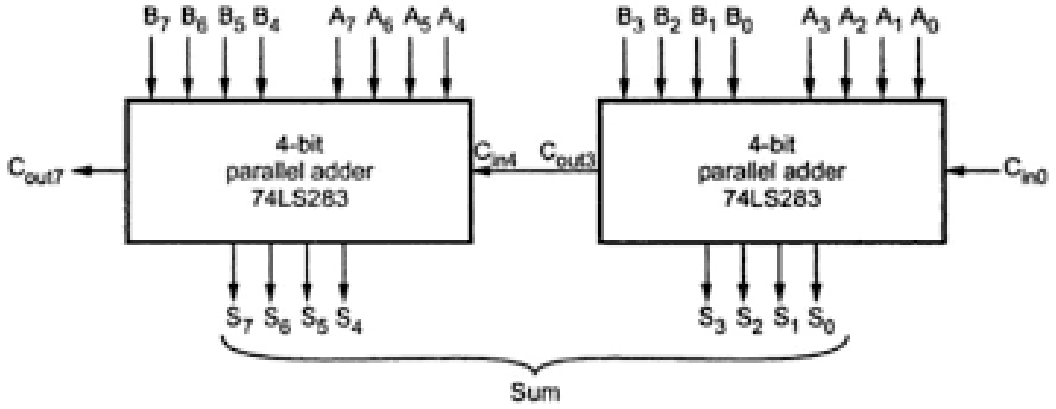
Add two numbers including their sign bits, then discard the carry bit. Negative numbers are represented in 2s complement.

+6 : 0000 0110
+13: + 0000 1101
+19: 0001 0011

-6 : 1111 1010
+13: + 0000 1101
+7: 0000 0111

+6 : 0000 0110
-13: + 1111 0011
-7 : 1111 1001

-6 : 1111 1010
-13: + 1111 0011
-19: 1110 1101



Overflow condition and detection

Overflow occurs if the resulting number exceeds the length of the register word. (For example, addition result of two 8-bit numbers do not fit into 8-bits)

While adding two unsigned numbers: Overflow is detected from the end carry-out bit of the MSB

While adding two signed numbers: When two signed numbers are added, the sign bit is treated as part of the number. End carry does not indicate overflow. **Overflow cannot occur if one number is positive and the other is negative**. Overflow occurs if the sign bit changes for the addition of two positive, or two negative number.

```
carries:     0 -> 1
  70:    0 100 0110
  80: + 0 101 0000
 150:   1 001 0110
```

```
carries:     1 -> 0
 -70:    1 011 1010
 -80: +  1 011 0000
-150:    0 110 1010
```

Overflow cases

Computer Arithmetics

- **Carry**: In the case of unsigned addition carry occurs when the correct result is too large to be represented.

- **Borrow**: In the case of unsigned subtraction borrow occurs when the value being subtracted is larger than the value it is subtracted from. If subtraction is performed by negating and adding , there is a borrow if the addition does not produce a carry.

- **Overflow**: In the case of signed addition and subtraction, overflow occurs when the correct result is either more positive than the largest possible positive value ore more negative than the smallest possible negative one. It is indicated by a result having incorrect sign.

  There is overflow if:

  pos + pos $\rightarrow$neg pos – neg$\rightarrow$neg

  neg + neg $\rightarrow$pos neg – pos$\rightarrow$pos

Floating Point number representation

Mantissa and Exponent defines a very large scale of scientific numbers

Example: 6132.789 DEC can be represented as +0.6132789*10^+04

Similarly, %1001.11 -> $(0.1001110)_2 * 2^{000100}$

Mantissa: 01001110, Radix: 2, Exponent: 000100

The Radix and Radix point position of the mantissa is known

The mantissa and exponent can be signed numbers resulting in a very large positive and negative range and resolution

Arithmetic operations with floating point numbers require more complicated hardware, but it is necessary for scientific computation

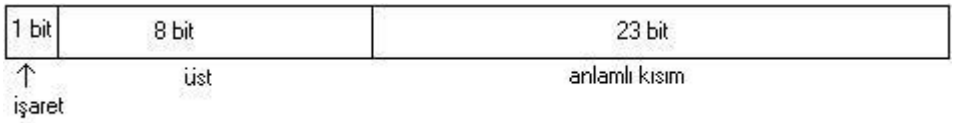
Capable microprocessors have floating point operation feature

Standard for Floating-Point Arithmetic, 1985

IEEE 754

Name	Common name	Base	Significant bits or digits	Decimal digits
binary16	Half precision	2	11	3.31
binary32	Single precision	2	24	7.22
binary64	Double precision	2	53	15.95
binary128	Quadruple precision	2	113	34.02

Single precision:



$(6{,}375)_{10}=(\ ?\ )_2$

0,375 × 2 = 0,75
0,75 × 2 = 1,5
0,5 × 2 = 1,0

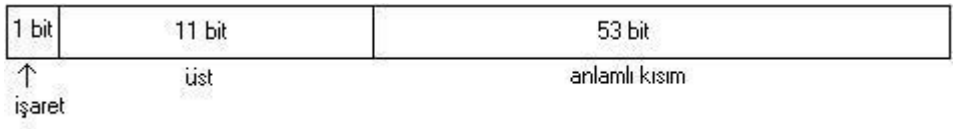
$=(-1)^{\text{sign bit}} * (1+\text{fraction}) * 2^{\text{exponent - bias}}$

$0{,}375 = (0{,}011)_2 \rightarrow 6{,}375 = (110{,}011)_2$

Bias is 127 for the exp, 129-127=2

0 10000001 10011000000000000000000

Double precision:



Convert the IEEE 754 number (single-precision) **1 10000001 10110011001100110011010** *into a floating-point decimal value.*

1- **Put the bits in three groups.**
Bit **31** (the leftmost bit) show the sign of the number.
Bits **23-30** (the next 8 bits) are the exponent.
Bits **0-22** (on the right) give the fraction

2- Look at the sign bit.
If this bit is a 1, the number is negative.
If it is 0, the number is positive.
This bit is 1, so the number is negative.

3- Get the exponent and the correct bias.
The exponent is simply a positive binary number.
$10000001_{bin} = 129_{ten}$

Bias must be subtracted from this exponent to find the power of 2. Since this is a single-precision number, the bias is 127.

4. Convert the fraction string into base ten.
The binary string represents a fraction.
Binary fractions :
$0.1 = (1/2) = 2^{-1}$
$0.01 = (1/4) = 2^{-2}$
$0.001 = (1/8) = 2^{-3}$

each digit must be multiplied by the corresponding power of 2:

$0.10110011001100110011010_{bin} = \mathbf{1*2^{-1}} + 0*2^{-2} + \mathbf{1*2^{-3}} + \mathbf{1*2^{-4}} + 0*2^{-5} + 0 * 2^{-6} + ...$
$= 1/2 + 1/8 + 1/16 + ...$
Note that this number is just an approximation on some decimal number. There will most likely be some error. In this case, the fraction is about 0.7000000476837158.

5. We can put these numbers in the expression:

$(-1)^{\text{sign bit}} * (1+\text{fraction}) * 2^{\text{exponent} - \text{bias}}$
$= (-1)^{1} * (1.7000000476837158) * 2^{129-127}$
= -6.8 (approximately)

Other representation systems in computers

ASCII

a method to represent alphanumeric characters

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

8051 Instruction Set Summary

Rn Register R7-R0 of the currently selected Register Bank.
Data 8-bit internal data location's address. This could be an internal Data RAM location (0-127) or a SFR [i.e. I/O port, control register, status register, etc. (128-255)].
@Ri 8-bit Internal Data RAM location (0-255) addressed indirectly through register R1 or R0.
#data 8-bit constant included in instruction.
#data16 16-bit constant included in instruction.
addr16 16-bit destination address. Used by LCALL and LJMP. A branch can be anywhere within the 64k byte Program Memory address space.
addr11 11-bit destination address. Used by ACALL and AJMP. The branch will be within the same 2k byte page of Program Memory as the first byte of the following instruction.
rel Signed (two's component) 8-bit offset byte. Used by SJMP and all conditional jumps. Range is –128 to +127 bytes relative to first byte of the following instruction.
bit Direct Addressed bit in Internal Data RAM or Special Function Register.

Instruction	Flag			Instruction	Flag		
	C	OV	AC		C	OV	AC
ADD	X	X	X	CLR C	0		
ADDC	X	X	X	CPL C	X		
SUBB	X	X	X	ANL C,bit	X		
MUL	0	X		ANL C,/bit	X		
DIV	0	X		ORL C,bit	X		
DA	X			ORL C,/bit	X		
RRC	X			MOV C,bit	X		
RLC	X			CJNE	X		
SETB C	1						

Note that operations on SFR byte address 208 or bit addresses 209-215 (i.e. the PSW or bits in the PSW) will also affect flag settings.

Mnemonic		Description	Byte	Cycle
Arithmetic operations				
ADD	A,Rn	Add register to accumulator	1	1
ADD	A,direct	Add direct byte to accumulator	2	1
ADD	A,@Ri	Add indirect RAM to accumulator	1	1
ADD	A,#data	Add immediate data to accumulator	2	1
ADDC	A,Rn	Add register to accumulator with carry flag	1	1
ADDC	A,direct	Add direct byte to A with carry flag	2	1
ADDC	A,@Ri	Add indirect RAM to A with carry flag	1	1
ADDC	A,#data	Add immediate data to A with carry flag	2	1
SUBB	A,Rn	Subtract register to accumulator with borrow	1	1
SUBB	A,direct	Subtract direct byte to A with carry borrow	2	1
SUBB	A,@Ri	Subtract indirect RAM to A with carry borrow	1	1
SUBB	A,#data	Subtract immediate data to A with carry borrow	2	1
INC	A	Increment accumulator	1	1
INC	Rn	Increment register	1	1
INC	direct	Increment direct byte	2	1
INC	@Ri	Increment indirect RAM	1	1
DEC	A	Decrement accumulator	1	1
DEC	Rn	Decrement register	1	1
DEC	direct	Decrement direct byte	2	1
DEC	@Ri	Decrement indirect RAM	1	1
INC	DPTR	Increment data pointer	1	2
MUL	AB	Multiply A and B -> [B hi]:[A lo]	1	4
DIV	AB	Divide A by B -> A=result, B=remainder	1	4
DA	A	Decimal adjust accumulator	1	1
CLR	A	Clear accumulator	1	1
CPL	A	Complement accumulator	1	1
RL	A	Rotate accumulator left	1	1
RLC	A	Rotate accumulator left through carry	1	1
RR	A	Rotate accumulator right	1	1
RRC	A	Rotate accumulator right through carry	1	1
SWAP	A	Swap nibbles within the accumulator	1	1
Logic operations				
ANL	A,Rn	AND register to accumulator	1	1
ANL	A,direct	AND direct byte to accumulator	2	1
ANL	A,@Ri	AND indirect RAM to accumulator	1	1
ANL	A,#data	AND immediate data to accumulator	2	1
ANL	direct,A	AND accumulator to direct byte	2	1
ANL	direct,#data	AND immediate data to direct byte	3	2
ORL	A,Rn	OR register to accumulator	1	1
ORL	A,direct	OR direct byte to accumulator	2	1
ORL	A,@Ri	OR indirect RAM to accumulator	1	1
ORL	A,#data	OR immediate data to accumulator	2	1
ORL	direct,A	OR accumulator to direct byte	2	1
ORL	direct,#data	OR immediate data to direct byte	3	2
XRL	A,Rn	Exclusive OR register to accumulator	1	1
XRL	A,direct	Exclusive OR direct byte to accumulator	2	1
XRL	A,@Ri	Exclusive OR indirect RAM to accumulator	1	1
XRL	A,#data	Exclusive OR immediate data to accumulator	2	1
XRL	direct,A	Exclusive OR accumulator to direct byte	2	1
XRL	direct,#data	Exclusive OR immediate data to direct byte	3	2
Boolean variable manipulation				
CLR	C	Clear carry flag	1	1
CLR	bit	Clear direct bit	2	1
SETB	C	Set carry flag	1	1
SETB	bit	Set direct bit	2	1
CPL	C	Complement carry flag	1	1
CPL	bit	Complement direct bit	2	1
ANL	C,bit	AND direct bit to carry flag	2	2
ANL	C,/bit	AND complement of direct bit to carry	2	2
ORL	C,bit	OR direct bit to carry flag	2	2
ORL	C,/bit	OR complement of direct bit to carry	2	2
MOV	C,bit	Move direct bit to carry flag	2	1
MOV	bit,C	Move carry flag to direct bit	2	2
Program and machine control				
ACALL	addr11	Absolute subroutine call	2	2
LCALL	addr16	Long subroutine call	3	2
RET		Return from subroutine	1	2
RETI		Return from interrupt	1	2
AJMP	addr11	Absolute jump	2	2
LJMP	addr16	Long jump	3	2
SJMP	rel	Short jump (relative address)	2	2
JMP	@A+DPTR	Jump indirect relative to the DPTR	1	2
JZ	rel	Jump if accumulator is zero	2	2
JNZ	rel	Jump if accumulator is not zero	2	2
JC	rel	Jump if carry flag is set	2	2
JNC	rel	Jump if carry flag is not set	2	2
JB	bit,rel	Jump if bit is set	3	2
JNB	bit,rel	Jump if bit is not set	3	2
JBC	bit,rel	Jump if direct bit is set and clear bit	3	2
CJNE	A,direct,rel	Compare direct byte to A and jump if not equal	3	2
CJNE	A,#data,rel	Compare immediate to A and jump if not equal	3	2
CJNE	Rn,#data,rel	Compare immed. to reg. and jump if not equal	3	2
CJNE	@Rn,#data,rel	Compare immed. to ind. and jump if not equal	3	2
DJNZ	Rn,rel	Decrement register and jump in not zero	2	2
DJNZ	direct,rel	Decrement direct byte and jump in not zero	3	2
NOP		No operation	1	1
Data transfer				
MOV	A,Rn	Move register to accumulator	1	1
MOV	A,direct*)	Move direct byte to accumulator	2	1
MOV	A,@Ri	Move indirect RAM to accumulator	1	1
MOV	A,#data	Move immediate data to accumulator	2	1
MOV	Rn,A	Move accumulator to register	1	1
MOV	Rn,direct	Move direct byte to register	2	2
MOV	Rn,#data	Move immediate data to register	2	1
MOV	direct,A	Move accumulator to direct byte	2	1
MOV	direct,Rn	Move register to direct byte	2	2
MOV	direct,direct	Move direct byte to direct byte	3	2
MOV	direct,@Ri	Move indirect RAM to direct byte	2	2
MOV	direct,#data	Move immediate data to direct byte	3	2
MOV	@Ri,A	Move accumulator to indirect RAM	1	1
MOV	@Ri,direct	Move direct byte to indirect RAM	2	2
MOV	@Ri,#data	Move immediate data to indirect RAM	2	1
MOV	DPTR,#data16	Load data pointer with a 16-bit constant	3	2
MOVC	A,@A+DPTR	Move code byte relative to DPTR to accumulator	1	2
MOVC	A,@A+PC	Move code byte relative to PC to accumulator	1	2
MOVX	A,@Ri	Move external RAM (8-bit addr.) to A	1	2
MOVX	A,@DPTR	Move external RAM (16-bit addr.) to A	1	2
MOVX	@Ri,A	Move A to external RAM (8-bit addr.)	1	2
MOVX	@DPTR,A	Move A to external RAM (16-bit addr.)	1	2
PUSH	direct	Push direct byte onto stack	2	2
POP	direct	Pop direct byte from stack	2	2
XCH	A,Rn	Exchange register to accumulator	1	1
XCH	A,direct	Exchange direct byte to accumulator	2	1
XCH	A,@Ri	Exchange indirect RAM to accumulator	1	1
XCHD	A,@Ri	Exchange low-order nibble indir. RAM with A	1	1

*) MOV A,ACC is not a valid instruction

```
jne  A,#data,@          cjne   A,#data,@
 (jump if A ! = data)

je   A, #data,@         add    A,#low(–data)     or    cjne   A,#(data),ne
 (jump if A == data)    jz     @                       jmp    @
                                                ne:    ...

ja, jnbe  A,#data,@     add    A,#low(–data–1)   or    cjne   A,#(data+1),ne
 (jump if A > data)     jc     @                 ne:   jnc    @

jae, jnb  A,#data,@     add    A,#low(–data)     or    cjne   A,#(data),ne
 (jump if A >= data)    jc     @                 ne:   jnc    @

jb, jnae  A,#data,@     add    A,#low,(–data)    or    cjne   A,#(data),ne
 (jump if A < data)     jnc    @                 ne:   jc     @

jbe, jna  A,#data,@     add    A,#low(–data–1)   or    cjne   A,#(data+1),ne
 (jump if A <= data)    jnc    @                 ne:   jc     @

switch  A <,==,> #data  cjne   A,#data,ne
 (no A modification)    ...                             ; execute code if A==data
                  ne:   jc     is_below                 ; jump if A<data
                        jnc    is_above                 ; jump if A>data or exec. code
```

References

- Lecture Slides: Dr. Şule Gündüz Öğüdücü
- Lecture Slides: Dr. Erdem Matoğlu
- Lecture Slides: Dr. Feza Buzluca
- Lecture Slides: Dr. Bassel Soudan
- Lecture Slides: Dr. Gökhan İnce
- Lecture Slides: Dr. B.Berk Üstündağ