CS231 ระบบคอมพิวเตอร์และภาษาแอสแซมบลี

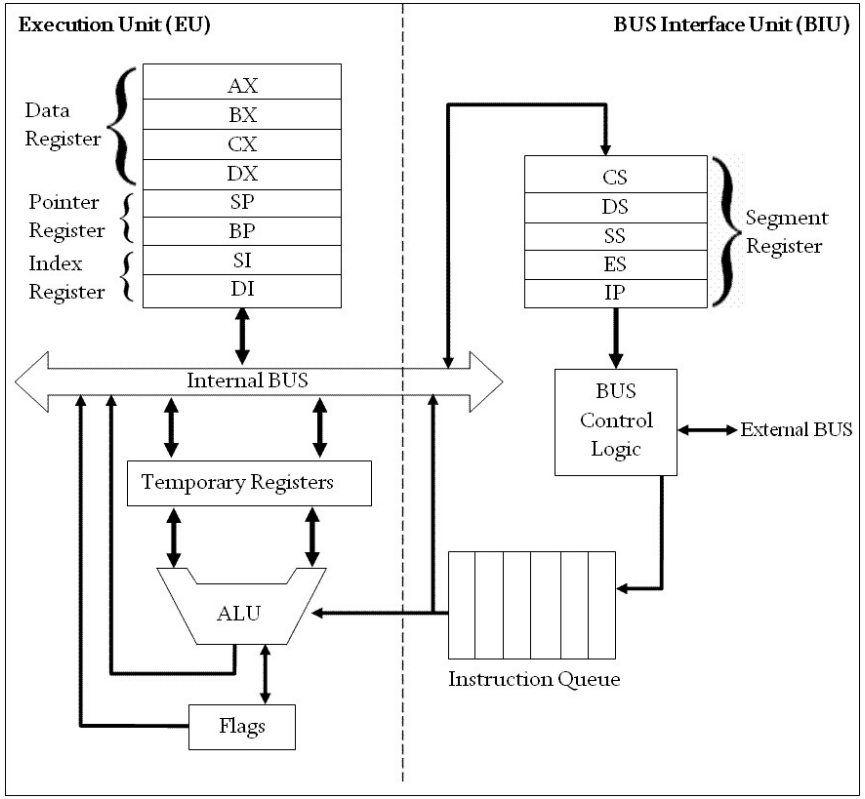
บทที่ 4: การประมวลผลทางคณิตศาสตร์

ผู้ช่วยศาสตราจารย์ ดร. ปวีณ เขื่อนแก้ว

สาขาวิชาวิทยาการคอมพิวเตอร์

คณะวิทยาศาสตร์ มหาวิทยาลัยแม่โจ้

การประมวลผลทางคณิตศาสตร์ จะกระทำที่วงจร ALU โดย input และ output ของการคำนวณจะพักไว้ที่ Temporary Register สถานะของคำตอบ ตัวอย่างเช่น ค่าล้น ติดลบ หรือคำนวณไม่ได้ จะแยกเก็บไว้ที่รีจิสเตอร์ Flags ซึ่งมีขนาด 16 บิต



Flags

บิตที่	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
ความหมาย					O	D	I	T	S	Z		AC		P		CY

บิต	ความหมาย	เป็น 1 เมื่อ
O	Overflow คำตอบมีค่าเกินขนาดของรีจิสเตอร์	ล้น
S	Sign คำตอบติดลบ (บิต MSB เป็น 1)	ติดลบ
Z	Zero คำตอบเป็นศูนย์	มีค่า 0
AC	Auxiliary Carry ใช้สำหรับรหัส BCD	บิตที่ 3 เป็น 1
P	Parity ภาวะคู่หรือคี่ของคำตอบ	คำตอบเป็นเลขคู่
CY	Carry มีการทดเลข	มีเลขทด

http://mrcetstudent.blogspot.com/2015/05/8086-microprocessor.html

OPCODE	OPERAND	EXPLANATION	EXAMPLE
ADD	D, S	D = D + S	ADD AX, [2050]
ADC	D, S	D = D + S + prev. carry	ADC AX, BX
INC	D	D = D + 1	INC AX
SUB	D, S	D = D – S	SUB AX, [SI]
SBB	D, S	D = D – S – prev. carry	SBB [2050], 0050
DEC	D	D = D – 1	DEC [2050]
MUL	8-bit register	AX = AL * 8-bit reg.	MUL BH
MUL	16-bit register	DX AX = AX * 16-bit reg.	MUL CX
IMUL	8 or 16 bit register	performs signed multiplication	IMUL CX
DIV	8-bit register	AX = AX / 8-bit reg. ; AL = quotient ; AH = remainder	DIV BL
DIV	16-bit register	DX AX / 16-bit reg. ; AX = quotient ; DX = remainder	DIV CX
IDIV	8 or 16 bit register	performs signed division	IDIV BL
CBW	none	converts signed byte to word	CBW
CWD	none	converts signed word to double word	CWD
NEG	D	D = 2's compliment of D	NEG AL
DAA	none	decimal adjust accumulator	DAA
DAS	none	decimal adjust accumulator after subtraction	DAS
AAA	none	ASCII adjust accumulator after addition	AAA
AAS	none	ASCII adjust accumulator after subtraction	AAS
AAM	none	ASCII adjust accumulator after multiplication	AAM
AAD	none	ASCII adjust accumulator after division	AAD

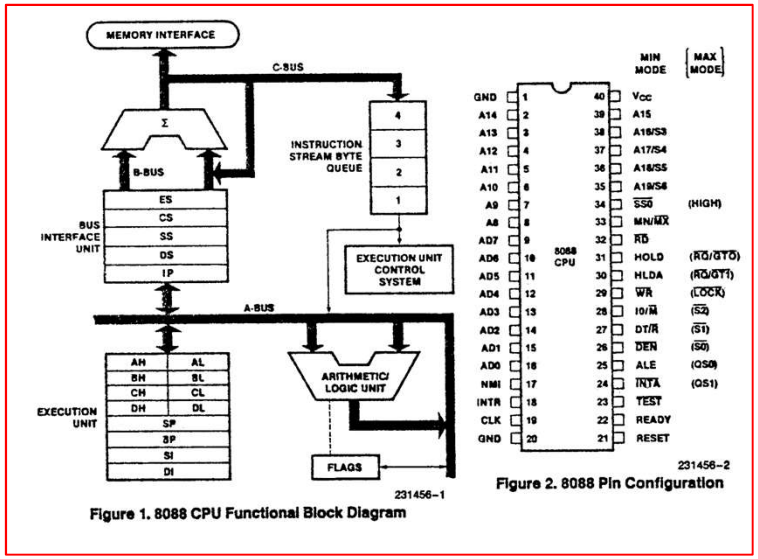
การแทนค่าจำนวนและนำเข้าสู่รีจิสเตอร์

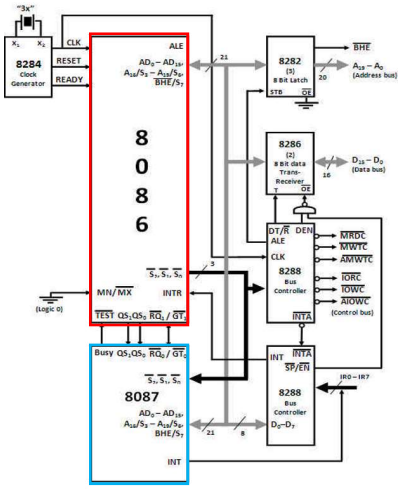
- การแทนจำนวนเต็ม ไม่คิดเครื่องหมาย (Unsigned Integer)
- การแทนจำนวนเต็ม คิดเครื่องหมาย (Signed Integer)

ALU

- เลขทศนิยม (Floating point)

FPU





จำนวนเต็มแบบไม่คิดเครื่องหมาย

ค่าต่ำสุดคือ 0 ค่าสูงสุดคือ $2^{\text{จำนวนบิต}}-1$

0	0
1	1

1 บิต
ค่าสูงสุดคือ 1

0	00
1	01
2	10
3	11

2บิต
ค่าสูงสุดคือ 3

0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

3 บิต
ค่าสูงสุดคือ 7

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

4 บิต ค่าสูงสุดคือ 15

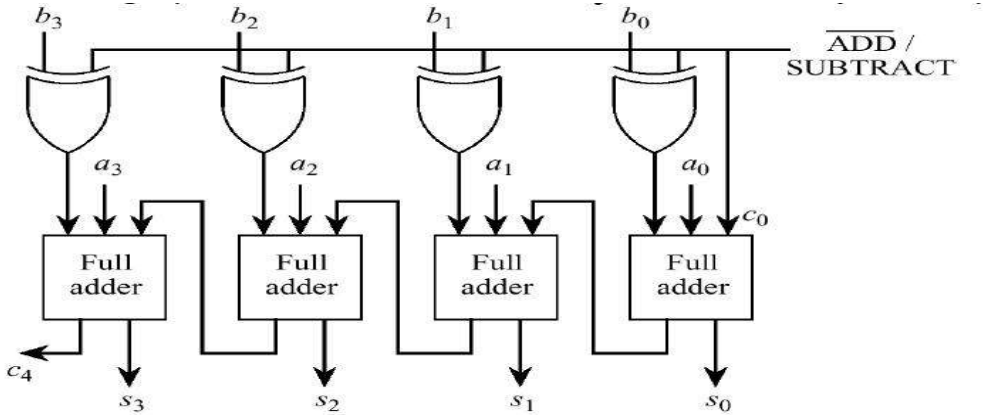
บิต	ค่าต่ำสุด	สูงสุด	เทียบกับภาษา C
8	0	254	unsigned char
16	0	65535	unsigned short

รองรับโดยวงจร ALU หากต้องการคำนวณเลขค่าสูงกว่า 16 บิต
ต้องคำนวณหลายครั้ง โดยใช้การทดหรือยืม
(ยกเว้นการคูณและหาร ที่รองรับเลข 32 บิต)

จำนวนเต็มแบบคิดเครื่องหมาย จะใช้บิต MSB เป็นตัวระบุเครื่องหมาย + , -

การแทนจำนวนติดลบ จะใช้เลขฐานสอง แบบ 2’s complement
เนื่องจากคุณสมบัติ A – B = A + (-B) ทำให้ ALU มีเพียงวงจรบวกเลขเท่านั้น การลบเลขจะใช้วงจรกลับเครื่องหมายก่อน แล้วป้อนเข้าวงจรบวก

ดังนั้นโปรแกรมเมอร์จึงต้องทำการแปลงเลขให้อยู่ในรูปแบบของ 2’s complement ก่อน แล้วค่อยนำมาใส่ใน source code หรือให้โปรแกรม Assembler ทำการแปลงให้ โดยใช้ operator



การแปลงเลขเป็น 2’s complement

1) กลับบิต
2) นำค่าที่ได้ + 1

จงแปลง 0 0 0 0 0 1 ให้เป็น 2’s complement

input	0 0 0 0 0 1	
		
กลับบิต	1 1 1 1 1 0	
		+
+ 1	0 0 0 0 0 1	
	1 1 1 1 1 1	ตอบ

การแปลงเลขเป็น 2’s complement

1) กลับบิต
2) นำค่าที่ได้ + 1

จงแปลง 1 1 1 1 1 1 ให้เป็น 2’s complement

input	1 1 1 1 1 1	
กลับบิต	0 0 0 0 0 0	+
+ 1	0 0 0 0 0 1	
	0 0 0 0 0 1	ตอบ

แปลง 2’s complement สองครั้งจะกลับไปเป็นค่าเดิม

การแปลงเลขเป็น 2’s complement

1) กลับบิต
2) นำค่าที่ได้ + 1

จงแปลง 1 0 1 0 1 1 ให้เป็น 2’s complement

input	1 0 1 0 1 1	
กลับบิต	0 1 0 1 0 0	+
+ 1	0 0 0 0 0 1	
	0 1 0 1 0 1	ตอบ

การแปลงเลขเป็น 2’s complement

1) กลับบิต
2) นำค่าที่ได้ + 1

จงแปลง 0 1 0 1 0 1 ให้เป็น 2’s complement

input	0 1 0 1 0 1	
กลับบิต	1 0 1 0 1 0	+
+ 1	0 0 0 0 0 1	
	1 0 1 0 1 1	ตอบ

การแปลงเลขเป็น 2’s complement

เลขฐาน 16

ใช้วิธีเดียวกับเลขฐาน 2 ก็ได้ แต่นี่คือวิธีลัด

1) นำ 15 (F) มาลบด้วยค่าแต่ละหลัก
2) นำค่าที่ได้ + 1

จงแปลง F 0 5 F ให้เป็น 2’s complement

นำ 15 (F) มาตั้งแล้วลบแต่ละหลัก F F F F

-

input F 0 5 F

0 F A 0

+

+ 1 0 0 0 1

0 F A 1 ตอบ

การแปลงเลขเป็น 2’s complement

เลขฐาน 16

1) นำ 15 (F) มาลบด้วยค่าแต่ละหลัก
2) นำค่าที่ได้ + 1

จงแปลง 0 F A 1 ให้เป็น 2’s complement

นำ 15 (F) มาตั้งแล้วลบแต่ละหลัก F F F F

-

input 0 F A 1

F 0 5 E

+ 1 +

0 0 0 1

F 0 5 F ตอบ

การแทนจำนวนแบบจำนวนเต็ม

บิต MSB เป็น 1 หมายถึงจำนวนติดลบ

บิต MSB เป็น 0 หมายถึงจำนวนเต็มบวก

แต่เราไม่สามารถกำหนดให้บิต MSB เป็น 1 แล้วข้อมูลนั้นจะติดลบทันที เพราะการกระทำเช่นนั้นจะทำให้จำนวนเปลี่ยนค่าไปเป็นเลขอื่น

การโหลดข้อมูลเข้าสู่ ALU นั้นมีขั้นตอนดังนี้

1) ให้แปลงเป็นเลขฐาน 2 โดยไม่คิดเครื่องหมาย
2) ถ้าเป็นจำนวนติดลบ ให้ทำ 2's complement

ต้องการโหลดจำนวน 100 (ฐาน10) เข้าสู่ ALU จะต้องโหลดด้วยค่าใด

แปลง 100 เป็นฐานสองได้ 0 1 1 0 0 1 0 0

เนื่องจากเป็นจำนวนเต็มบวกจึงโหลดค่า 0 1 1 0 0 1 0 0 เข้าสู่ ALU ได้เลย

ต้องการโหลดจำนวน -100 (ฐาน10) เข้าสู่ ALU จะต้องโหลดด้วยค่าใด

แปลง 100 เป็นฐานสองได้ 0 1 1 0 0 1 0 0

ค่าติดลบ จึงต้องทำ 2's complement 1 0 0 1 1 0 1 1

\+

0 0 0 0 0 0 0 1

1 0 0 1 1 1 0 0 ค่านี้คือค่าที่โหลดเข้าสู่ ALU

การแทนจำนวนแบบจำนวนเต็ม

ต้องการโหลดจำนวน 7ABC (ฐาน16) เข้าสู่ ALU จะต้องโหลดด้วยค่าใด

เนื่องจากเป็นจำนวนเต็มบวกฐาน 16 จึงโหลดค่า 7ABC เข้าสู่ ALU ได้เลย

ต้องการโหลดจำนวน -7ABC (ฐาน16) เข้าสู่ ALU จะต้องโหลดด้วยค่าใด

ค่าติดลบ จึงต้องทำ 2's complement

7 A B C

8 5 4 3

\+

0 0 0 1

8 5 4 4 ค่านี้คือค่าที่โหลดเข้าสู่ ALU

การตีความผลการคำนวณ ALU

1) ถ้าบิต MSB มีค่าเป็น 0 ให้นำคำตอบนั้นไปใช้ได้เลย
2) ถ้าบิต MSB มีค่าเป็น 1 แสดงจำนวนนั้นเป็น 2's complement ดังนั้นให้นำเอาจำนวนนั้นไปหา 2's complement ก่อนจึงนำไปใช้ได้

นอกจากสังเกตจาก MSB แล้ว ยังสังเกตจาก Flag ได้ เพราะ ALU จะทำการเปลี่ยนค่า Flag S เมื่อคำตอบติดลบ

หาก input ไม่มีเลข 0 นำหน้า ให้ดูขนาดรีจิสเตอร์ด้วย จะได้รู้ว่าเป็นบิต MSB หรือไม่

1
01
001
0000 0001

ทั้งหมดมีค่าเท่ากัน คือ "1" แต่จะรู้ได้อย่างไรว่า บิต MSB เป็นค่า 1 หรือ 0

การกำหนดขนาดของตัวแปร หรือการรู้ขนาดของรีจิสเตอร์เป็นสิ่งที่สำคัญมาก เพื่อให้สามารถระบุตำแหน่ง MSB ได้ วิธีการที่ปลอดภัยที่สุดคือ เติมเลข 0 ไปให้ครบทุกหลัก

d7 MSB	d6	d5	d4	d3	d2	d1	d0 LSB

ตัวอย่างรีจิสเตอร์ขนาด 8 บิต

การตีความผลการคำนวณ ALU

1) ถ้าบิต MSB มีค่าเป็น 0 ให้นำคำตอบนั้นไปใช้ได้เลย
2) ถ้าบิต MSB มีค่าเป็น 1 แสดงจำนวนนั้นเป็น 2's complement ดังนั้นให้นำเอาจำนวนนั้นไปหา 2's complement ก่อนจึงนำไปใช้ได้

นอกจากสังเกตจาก MSB แล้ว ยังสังเกตจาก Flag ได้ เพราะ ALU จะทำการเปลี่ยนค่า Flag S เมื่อคำตอบติดลบ

ค่า FCBA มีค่าเท่าใดเมื่อแปลงเป็นเลขฐาน 10 แบบคิดเครื่องหมาย กำหนดให้รีจิสเตอร์มีขนาด 16 บิต

F C B A

ที่มี MSB คือ F แปลงเป็นฐานสองคือ 1 1 1 1

1 1 1 1 บิต MSB เป็น 1 แสดงว่าค่าติดลบ ให้ทำ 2's complement

F F F F

F C B A -

0 3 4 5

0 0 0 1 +

0 3 4 6 ค่านี้คือค่าจริง และเป็นค่าติดลบ

-0346 แปลงเป็น ฐาน 10 คือ -838

การตีความผลการคำนวณ ALU

1) ถ้าบิต MSB มีค่าเป็น 0 ให้นำคำตอบนั้นไปใช้ได้เลย
2) ถ้าบิต MSB มีค่าเป็น 1 แสดงจำนวนนั้นเป็น 2's complement ดังนั้นให้นำเอาจำนวนนั้นไปหา 2's complement ก่อนจึงนำไปใช้ได้

นอกจากสังเกตจาก MSB แล้ว ยังสังเกตจาก Flag ได้ เพราะ ALU จะทำการเปลี่ยนค่า Flag S เมื่อคำตอบติดลบ

ค่า 0346 มีค่าเท่าใดเมื่อแปลงเป็นเลขฐาน 10 แบบคิดเครื่องหมาย กำหนดให้รีจิสเตอร์มีขนาด 16 บิต

0 3 4 6

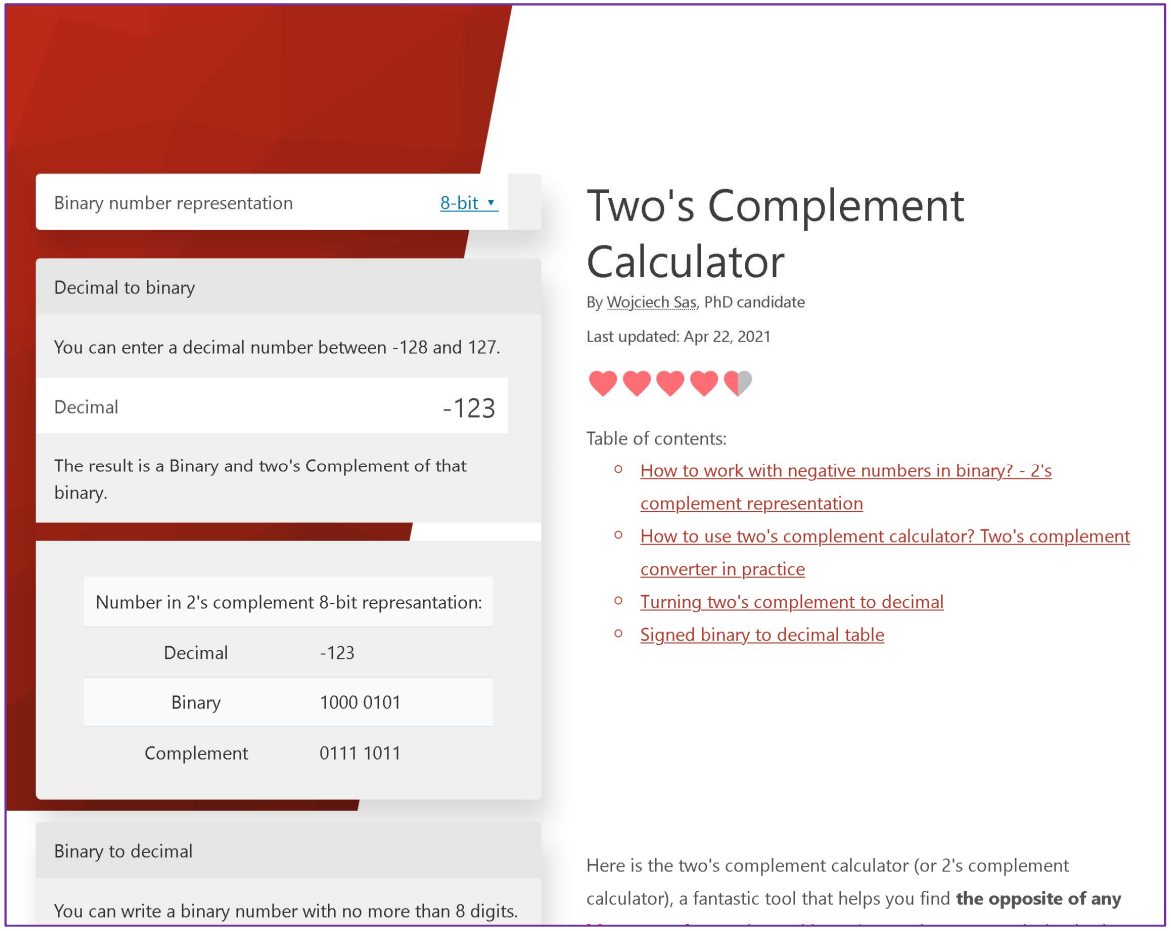
ที่มี MSB คือ 0 แปลงเป็นฐานสองคือ 0 0 0 0

0 0 0 0 บิต MSB เป็น 0 แสดงว่าค่าบวก จึงนำค่านี้มาใช้ได้ทันที

0 3 4 6 ตอบ

0346 แปลงเป็น ฐาน 10 คือ 838

ตัวอย่างเครื่องมือสำหรับฝึก แปลงเลข 2’s complement



https://www.omnicalculator.com/math/twos-complement

การตีความผลการคำนวณ ALU

0	0	0
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8
9	9	9
10	a	10
11	b	11
12	c	12
13	d	13
14	e	14
15	f	15
16	10	16
17	11	17
18	12	18
19	13	19
20	14	20
21	15	21
22	16	22
23	17	23
24	18	24
25	19	25
26	1a	26
27	1b	27
28	1c	28
29	1d	29
30	1e	30
31	1f	31
32	20	32
33	21	33
34	22	34
35	23	35
36	24	36
37	25	37
38	26	38
39	27	39
40	28	40
41	29	41
42	2a	42
43	2b	43
44	2c	44
45	2d	45
46	2e	46
47	2f	47
48	30	48
49	31	49
50	32	50
51	33	51
52	34	52
53	35	53
54	36	54
55	37	55
56	38	56
57	39	57
58	3a	58
59	3b	59
60	3c	60
61	3d	61
62	3e	62
63	3f	63

64	40	64
65	41	65
66	42	66
67	43	67
68	44	68
69	45	69
70	46	70
71	47	71
72	48	72
73	49	73
74	4a	74
75	4b	75
76	4c	76
77	4d	77
78	4e	78
79	4f	79
80	50	80
81	51	81
82	52	82
83	53	83
84	54	84
85	55	85
86	56	86
87	57	87
88	58	88
89	59	89
90	5a	90
91	5b	91
92	5c	92
93	5d	93
94	5e	94
95	5f	95
96	60	96
97	61	97
98	62	98
99	63	99
100	64	100
101	65	101
102	66	102
103	67	103
104	68	104
105	69	105
106	6a	106
107	6b	107
108	6c	108
109	6d	109
110	6e	110
111	6f	111
112	70	112
113	71	113
114	72	114
115	73	115
116	74	116
117	75	117
118	76	118
119	77	119
120	78	120
121	79	121
122	7a	122
123	7b	123
124	7c	124
125	7d	125
126	7e	126
127	7f	127

128	80	-128
129	81	-127
130	82	-126
131	83	-125
132	84	-124
133	85	-123
134	86	-122
135	87	-121
136	88	-120
137	89	-119
138	8a	-118
139	8b	-117
140	8c	-116
141	8d	-115
142	8e	-114
143	8f	-113
144	90	-112
145	91	-111
146	92	-110
147	93	-109
148	94	-108
149	95	-107
150	96	-106
151	97	-105
152	98	-104
153	99	-103
154	9a	-102
155	9b	-101
156	9c	-100
157	9d	-99
158	9e	-98
159	9f	-97
160	a0	-96
161	a1	-95
162	a2	-94
163	a3	-93
164	a4	-92
165	a5	-91
166	a6	-90
167	a7	-89
168	a8	-88
169	a9	-87
170	aa	-86
171	ab	-85
172	ac	-84
173	ad	-83
174	ae	-82
175	af	-81
176	b0	-80
177	b1	-79
178	b2	-78
179	b3	-77
180	b4	-76
181	b5	-75
182	b6	-74
183	b7	-73
184	b8	-72
185	b9	-71
186	ba	-70
187	bb	-69
188	bc	-68
189	bd	-67
190	be	-66
191	bf	-65

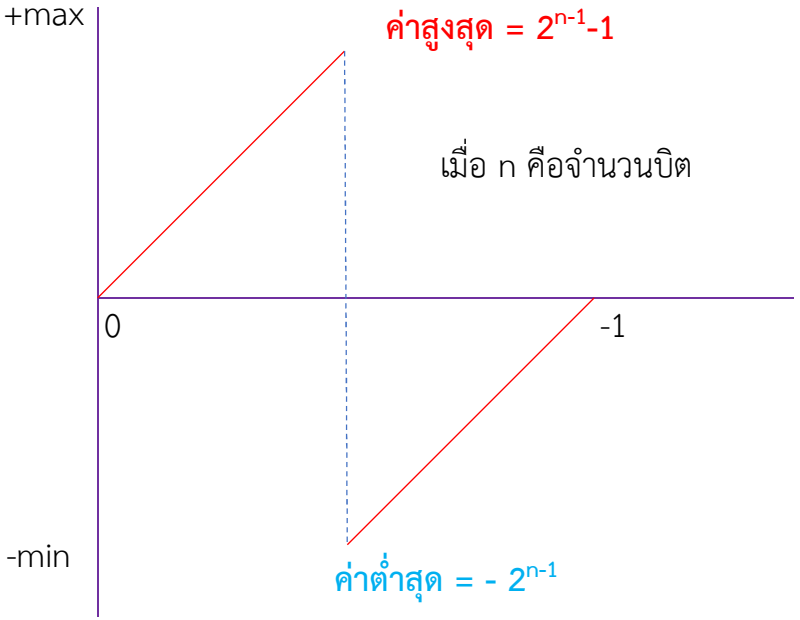
192	c0	-64
193	c1	-63
194	c2	-62
195	c3	-61
196	c4	-60
197	c5	-59
198	c6	-58
199	c7	-57
200	c8	-56
201	c9	-55
202	ca	-54
203	cb	-53
204	cc	-52
205	cd	-51
206	ce	-50
207	cf	-49
208	d0	-48
209	d1	-47
210	d2	-46
211	d3	-45
212	d4	-44
213	d5	-43
214	d6	-42
215	d7	-41
216	d8	-40
217	d9	-39
218	da	-38
219	db	-37
220	dc	-36
221	dd	-35
222	de	-34
223	df	-33
224	e0	-32
225	e1	-31
226	e2	-30
227	e3	-29
228	e4	-28
229	e5	-27
230	e6	-26
231	e7	-25
232	e8	-24
233	e9	-23
234	ea	-22
235	eb	-21
236	ec	-20
237	ed	-19
238	ee	-18
239	ef	-17
240	f0	-16
241	f1	-15
242	f2	-14
243	f3	-13
244	f4	-12
245	f5	-11
246	f6	-10
247	f7	-9
248	f8	-8
249	f9	-7
250	fa	-6
251	fb	-5
252	fc	-4
253	fd	-3
254	fe	-2
255	ff	-1

ตัวอย่างเส้นจำนวนของเลข 2's complement ขนาด 8 บิต

จำนวนตัวเลขทั้งหมด 256

ค่าสูงสุด 127

ค่าน้อยสุด -128



การป้อนข้อมูลตรวจสอบให้อยู่ในช่วง เพราะถ้าเกิดการล้น หรือ Overflow ข้อมูลอาจเปลี่ยนจาก 127 เป็น -128 ได้

ช่วงข้อมูลแบบไม่คิดเครื่องหมาย

บิต	ค่าต่ำสุด	สูงสุด	เทียบกับภาษา C
8	0	254	unsigned char
16	0	65535	unsigned short

ช่วงข้อมูลแบบคิดเครื่องหมาย

บิต	ค่าต่ำสุด	สูงสุด	เทียบกับภาษา C
8	-128	127	char
16	-32,768	32,767	short

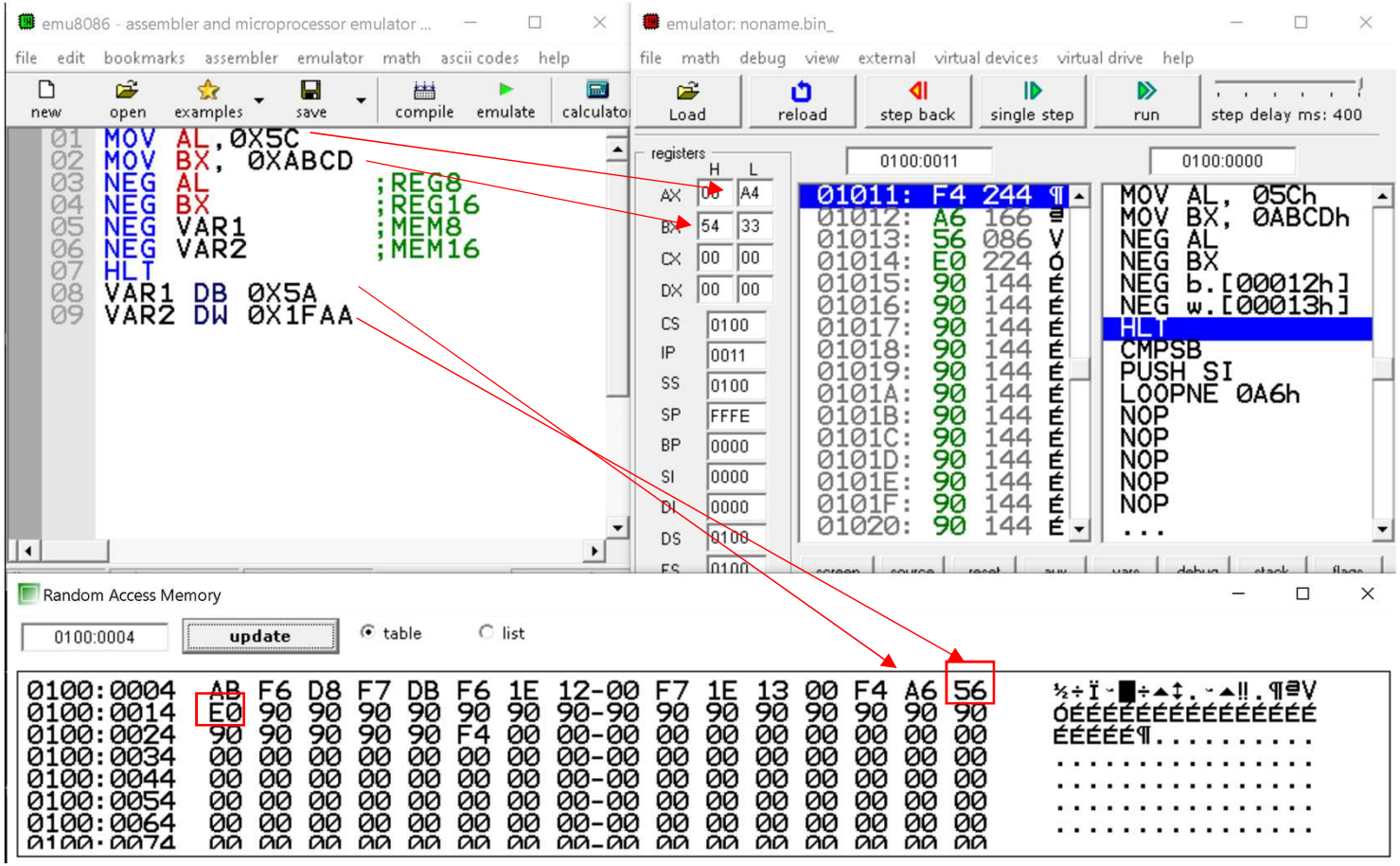
OPCODE	OPERAND	EXPLANATION	EXAMPLE
ADD	D, S	D = D + S	ADD AX, [2050]
ADC	D, S	D = D + S + prev. carry	ADC AX, BX
INC	D	D = D + 1	INC AX
SUB	D, S	D = D – S	SUB AX, [SI]
SBB	D, S	D = D – S – prev. carry	SBB [2050], 0050
DEC	D	D = D – 1	DEC [2050]
MUL	8-bit register	AX = AL * 8-bit reg.	MUL BH
MUL	16-bit register	DX AX = AX * 16-bit reg.	MUL CX
IMUL	8 or 16 bit register	performs signed multiplication	IMUL CX
DIV	8-bit register	AX = AX / 8-bit reg. ; AL = quotient ; AH = remainder	DIV BL
DIV	16-bit register	DX AX / 16-bit reg. ; AX = quotient ; DX = remainder	DIV CX
IDIV	8 or 16 bit register	performs signed division	IDIV BL
CBW	none	converts signed byte to word	CBW
CWD	none	converts signed word to double word	CWD
NEG	D	D = 2's compliment of D	NEG AL
DAA	none	decimal adjust accumulator	DAA
DAS	none	decimal adjust accumulator after subtraction	DAS
AAA	none	ASCII adjust accumulator after addition	AAA
AAS	none	ASCII adjust accumulator after subtraction	AAS
AAM	none	ASCII adjust accumulator after multiplication	AAM
AAD	none	ASCII adjust accumulator after division	AAD

NEG : Negate ใช้หา 2's complement ของ input

การใช้ : NEG operand

Operand : ที่ใช้ได้ reg8, reg16, mem8, mem16

Output: เก็บใน operand



การบวกเลข

OPCODE	OPERAND	EXPLANATION	EXAMPLE
ADD	D, S	D = D + S	ADD AX, [2050]
ADC	D, S	D = D + S + prev. carry	ADC AX, BX
INC	D	D = D + 1	INC AX
SUB	D, S	D = D – S	SUB AX, [SI]
SBB	D, S	D = D – S – prev. carry	SBB [2050], 0050
DEC	D	D = D – 1	DEC [2050]
MUL	8-bit register	AX = AL * 8-bit reg.	MUL BH
MUL	16-bit register	DX AX = AX * 16-bit reg.	MUL CX
IMUL	8 or 16 bit register	performs signed multiplication	IMUL CX
DIV	8-bit register	AX = AX / 8-bit reg. ; AL = quotient ; AH = remainder	DIV BL
DIV	16-bit register	DX AX / 16-bit reg. ; AX = quotient ; DX = remainder	DIV CX
IDIV	8 or 16 bit register	performs signed division	IDIV BL
CBW	none	converts signed byte to word	CBW
CWD	none	converts signed word to double word	CWD
NEG	D	D = 2's compliment of D	NEG AL
DAA	none	decimal adjust accumulator	DAA
DAS	none	decimal adjust accumulator after subtraction	DAS
AAA	none	ASCII adjust accumulator after addition	AAA
AAS	none	ASCII adjust accumulator after subtraction	AAS
AAM	none	ASCII adjust accumulator after multiplication	AAM
AAD	none	ASCII adjust accumulator after division	AAD

ADD : Addition ใช้บวกเลข

การใช้ : ADD D, S

Operand :

REG , memory

Memory ,REG

REG,REG

Memory. Immediate

REG, immediate

***REG: AX, BX, CX, DX, AL, BL, BH, CH, CL, DL, DI, SI, BP, SP**

Output: D

มีผลกับ Flag

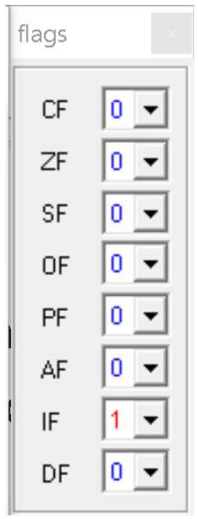
F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
				O	D	I	T	S	Z		AC		P		CY

บิต	ความหมาย	เป็น 1 เมื่อ
O	Overflow คำตอบมีค่าเกินขนาดของรีจิสเตอร์	ล้น
S	Sign คำตอบติดลบ (บิต MSB เป็น 1)	ติดลบ
Z	Zero คำตอบเป็นศูนย์	มีค่า 0
AC	Auxiliary Carry ใช้สำหรับรหัส BCD	บิตที่ 3 เป็น 1
P	Parity ภาวะคู่หรือคี่ของคำตอบ	คำตอบเป็นเลขคู่
CY	Carry มีการทดเลข	มีเลขทด

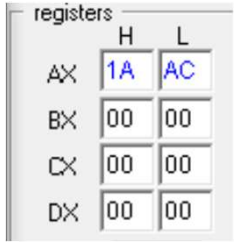
ADD : Addition ใช้บวกเลข

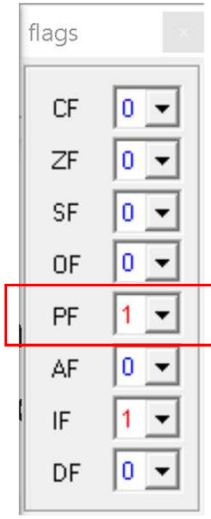
การใช้ : ADD D, S

MOV AX,0X1001
ADD AX,0X0AAA



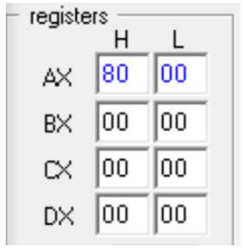
MOV AX,0X1002
ADD AX,0X0AAA

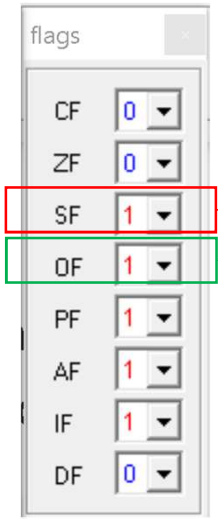




PF เป็น 1 เพราะคำตอบเป็นเลขคู่

MOV AX,0X7FFF
ADD AX,0X0001





SF เป็น 1 เพราะคำตอบติดลบ

OF เป็น 1 เพราะคำตอบเกินช่วง

ข้อนี้หากเป็นการบวกเลขไม่คิดเครื่องหมาย ตอบ 0x8000 ถูกต้องแล้ว

แต่หากเป็นแบบคิดเครื่องหมายต้องเอาไป 2's complement

จะได้คำตอบเป็น F F F F

8 0 0 0 –

7 F F F

1 +

-8 0 0 0 ซึ่งไม่ถูกต้อง

ดังนั้นหากเป็นการคำนวณแบบคิดเครื่องหมาย ถ้า OF เป็น 1 ห้ามนำคำตอบมาใช้เด็ดขาด

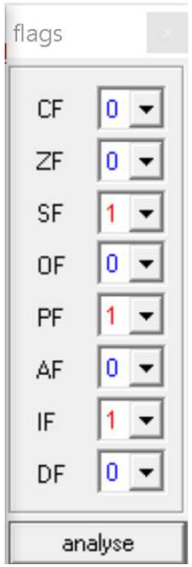
ADD : Addition ใช้บวกเลข

การใช้ : ADD D, S

FFFE คือ 2's complement ของ 0002

มีความหมายว่า -0002

```
MOV AX,0XFFFE
ADD AX,0X0001
```



ข้อนี้ OF ไม่ติด แสดงว่าคำตอบสามานำมาใช้ได้

SF เป็น 1 แสดงว่า คำตอบติดลบ ต้องนำคำตอบมาหา 2's complement

F F F F

F F F F –

0 0 0 0

0 0 0 1 +

0 0 0 1

ตอบ -0 0 0 1

โปรแกรมนี้เป็นการคำนวณ -2 + 1

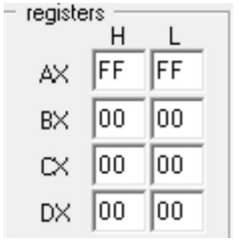
จึงตอบ -1

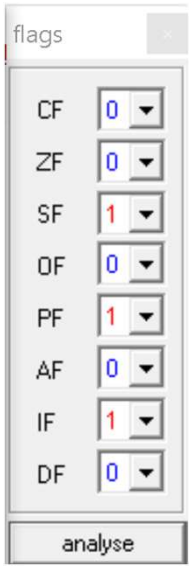
ADD : Addition ใช้บวกเลข

การใช้ : ADD D, S

```
MOV AX, 0x2
NEG AX
ADD AX,0X0001
```

หรือจะป้อนเป็นเลขจำนวนเต็มก่อนแล้วใช้ คำสั่ง NEG แปลงเป็นเลขติดลบ ให้ก็ได้คำตอบเท่ากัน แต่โปรแกรมจะยาวขึ้น 1 คำสั่ง





ข้อนี้ OF ไม่ติด แสดงว่าคำตอบสามานำมาใช้ได้

SF เป็น 1 แสดงว่า คำตอบติดลบ ต้องนำคำตอบมาหา 2's complement

F F F F

F F F F –

0 0 0 0

0 0 0 1 +

0 0 0 1

ตอบ -0 0 0 1

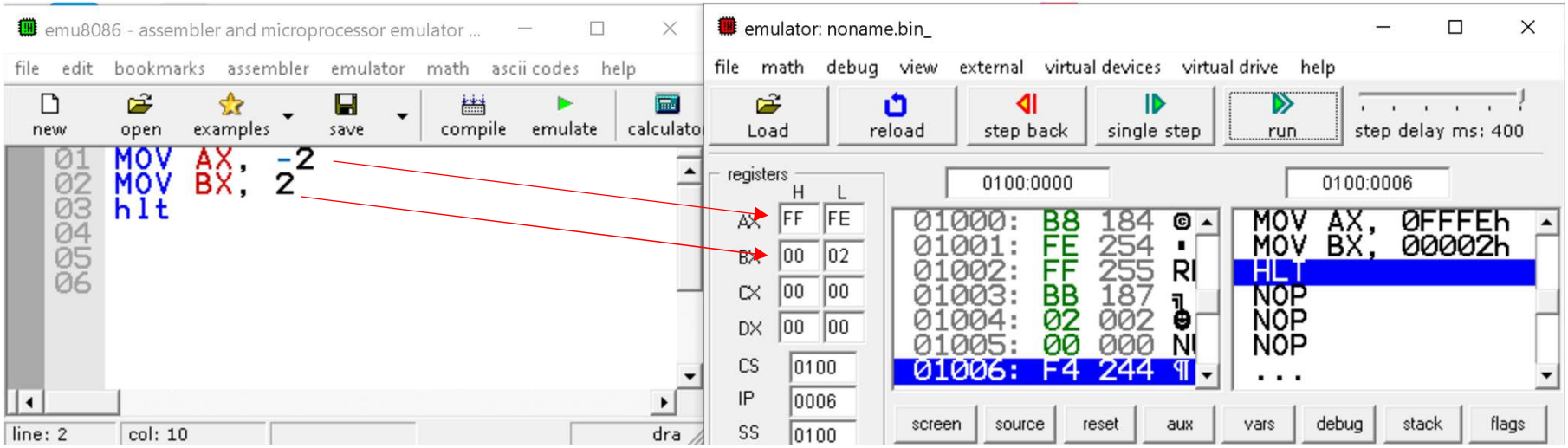
โปรแกรมนี้เป็นการคำนวณ -2 + 1

จึงตอบ -1

ADD : Addition ใช้บวกเลข

การใช้ : ADD D, S

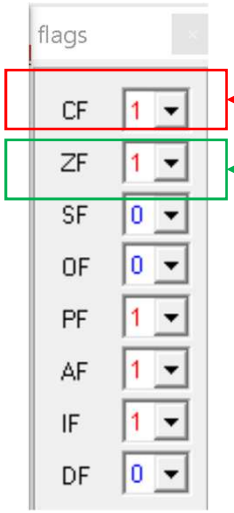
โปรแกรม Assembler ช่วยอำนวยความสะดวกให้โปรแกรมเมอร์โดยทำการแปลงเลขติดลบ เป็น 2's complement ให้ แต่หากเราเข้าใจระบบจำนวนของ ALU จะมีประโยชน์มากในการ Debug โปรแกรม



ADD : Addition ใช้บวกเลข

การใช้ : ADD D, S

```
MOV AX,0XFFFF
ADD AX,1
```



การบวกเลข 0xFFFF +1 จะมีค่าเป็น 0x10000 ซึ่งเกินขนาดของรีจิสเตอร์ที่มีขนาดเพียง 16 บิต จึงทำให้ค่ากลับมาเป็น 0 เรียกว่าการ Rollover กระณีนี้จะเกิดการทดเลข โปรแกรมเมอร์จึงต้องทำการตรวจสอบ CF ด้วย ว่าเกิด Carry หรือไม่ หากมี Carry คำตอบที่ได้จะขาดตัวเลขหลักที่ล้นรีจิสเตอร์ไป

CF เป็น 1 เพราะคำตอบเกินขนาดของรีจีสเตอร์

ZF เป็น 1 เพราะคำตอบมีค่า 0 (ทุกบิตเป็น 0)

ลองสังเกตดู การบวกเลขฐาน 2 ขนาด 4 บิตชุดนี้

```
  1 1 1 1
      +
  1
-------
  1 0 0 0 0
```

คำตอบที่อยู่ใน รีจิสเตอร์

บิตที่เกินมาจากรีจิสเตอร์จะเก็บไว้ที่ CY (CF ใน emulator)

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
				O	D	I	T	S	Z		AC		P		CY

เราสามารถใช้ CY ช่วยในการบวกเลขที่มีขนาดเกินกว่ารีจิสเตอร์ได้

การบวกเลข 1 0 0 0 1 1 1 1 + 0 0 0 0 0 0 0 1 โดยใช้รีจิสเตอร์ขนาด 4 บิต

แบ่งตัวเลขเป็น 2 ส่วนและเก็บในรีจิสเตอร์ 2 ตัว ทั้งตัวตั้งและตัวบวก

A = 1000

B = 1111

C = 0000

D = 0001

ADD B,D 1 1 1 1 +

0 0 0 1

B= 0 0 0 0 CY= 1

ADD C, CY 0 0 0 0

0 0 0 1

C= 0 0 0 1

ADD A, C 1 0 0 0 +

0 0 0 1

A= 1 0 0 1

นำคำตอบที่ค้างในรีจิสเตอร์มาเรียงใหม่เป็น A B จะได้คำตอบเป็น 1 0 0 1 0 0 0 0

ขนาดของรีจีสเตอร์ไม่ได้จำกัดจำนวนสูงสุดที่คอมพิวเตอร์สามารถคำนวณได้ แต่เป็นเพียงข้อจำกัดของการคำนวณตัวเลขสูงสุดใน 1 คำสั่ง

ข้อจำกัดจริงๆ คือพื้นที่เก็บข้อมูล ซึ่งในที่นี้คือขนาดของหน่วยความจำ

ถึงแม้ตัวอย่างนี้จะเป็นคอมพิวเตอร์ขนาดเพียง 4 บิต ก็สามารถใช้คำนวณตัวเลขขนาด 8 บิตได้

เพียงแต่เราต้องใช้รีจีสเตอร์ถึง 4 ตัว และมีคำสั่งเพิ่มขึ้น 2 คำสั่ง

หากจำนวนหลักมากกว่านี้ก็สามารถพ่วงคำสั่งต่อไปได้อีกเรื่อย ๆ

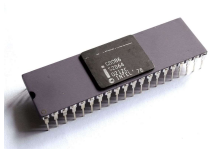
Primitive Data Types ของภาษา JAVA

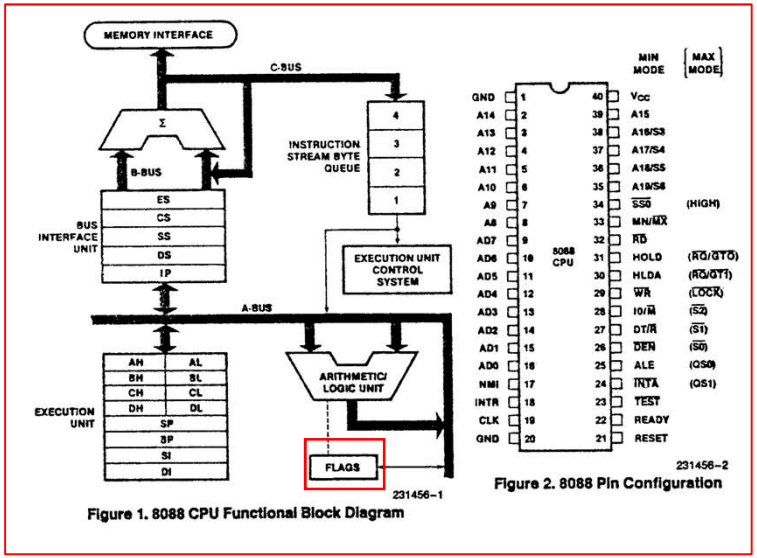
Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

หากโปรแกรมด้วย Assemble แม้จะโปรแกรมบน CPU ขนาดเพียง 4 บิต
จำนวนที่มีขนาดใหญ่กว่านี้ก็ยังสามารถคำนวณได้

รีจิสเตอร์ของหน่วยประมวลผลกลาง 8086 มีทั้งหมด 14 ตัว

- สำหรับใช้งานทั่วไป 8 ตัว
- สำหรับระบุ เซกเมนต์ (ตำแหน่งหน่วยความจำภายนอก) 4 ตัว
- รีจิสเตอร์พิเศษ 2 ตัว





AL	AL
BH	BL
CH	CL
DH	DL
SP	
BP	
SI	
DI	
CS	
DS	
SS	
ES	
IP	
FLAGS	

รีจิสเตอร์ A, B, C, D
4 ตัวพอดี
บังเอิญหรือไม่ ?

แต่ Flag มองไม่เห็นจากโปรแกรม

การบวกเลขแบบคิดตัวทด

OPCODE	OPERAND	EXPLANATION	EXAMPLE
ADD	D, S	D = D + S	ADD AX, [2050]
ADC	D, S	D = D + S + prev. carry	ADC AX, BX
INC	D	D = D + 1	INC AX
SUB	D, S	D = D – S	SUB AX, [SI]
SBB	D, S	D = D – S – prev. carry	SBB [2050], 0050
DEC	D	D = D – 1	DEC [2050]
MUL	8-bit register	AX = AL * 8-bit reg.	MUL BH
MUL	16-bit register	DX AX = AX * 16-bit reg.	MUL CX
IMUL	8 or 16 bit register	performs signed multiplication	IMUL CX
DIV	8-bit register	AX = AX / 8-bit reg. ; AL = quotient ; AH = remainder	DIV BL
DIV	16-bit register	DX AX / 16-bit reg. ; AX = quotient ; DX = remainder	DIV CX
IDIV	8 or 16 bit register	performs signed division	IDIV BL
CBW	none	converts signed byte to word	CBW
CWD	none	converts signed word to double word	CWD
NEG	D	D = 2's compliment of D	NEG AL
DAA	none	decimal adjust accumulator	DAA
DAS	none	decimal adjust accumulator after subtraction	DAS
AAA	none	ASCII adjust accumulator after addition	AAA
AAS	none	ASCII adjust accumulator after subtraction	AAS
AAM	none	ASCII adjust accumulator after multiplication	AAM
AAD	none	ASCII adjust accumulator after division	AAD

ADC : Add with Carry บวกเลขแบบคิดตัวทด

การใช้ : ADC D, S

การทำงาน D = D + S + CY

Operand ที่ใช้ได้:

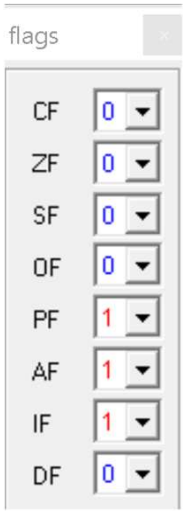
- **REG, memory**
- **memory, REG**
- **REG, REG**
- **memory, immediate**
- **REG, immediate**

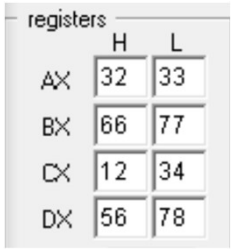
ตัวอย่างการใช้งาน บวกเลขขนาด 32 บิต

0x1FFF 0FFF + 0x1234 5678

AX BX CX DX

```
MOV AX,0X1FFF
MOV BX,0X0FFF
MOV CX,0X1234
MOV DX,0X5678

ADD BX,DX
ADC AX,CX
```





คำตอบสุดท้ายอยู่ใน AX BX

คือ 0x3233 6677

เบื่อเลขฐาน 16 หรือยัง

แทนเลขฐาน 16 จำนวน 1 หลัก ใช้พื้นที่ 4 บิต

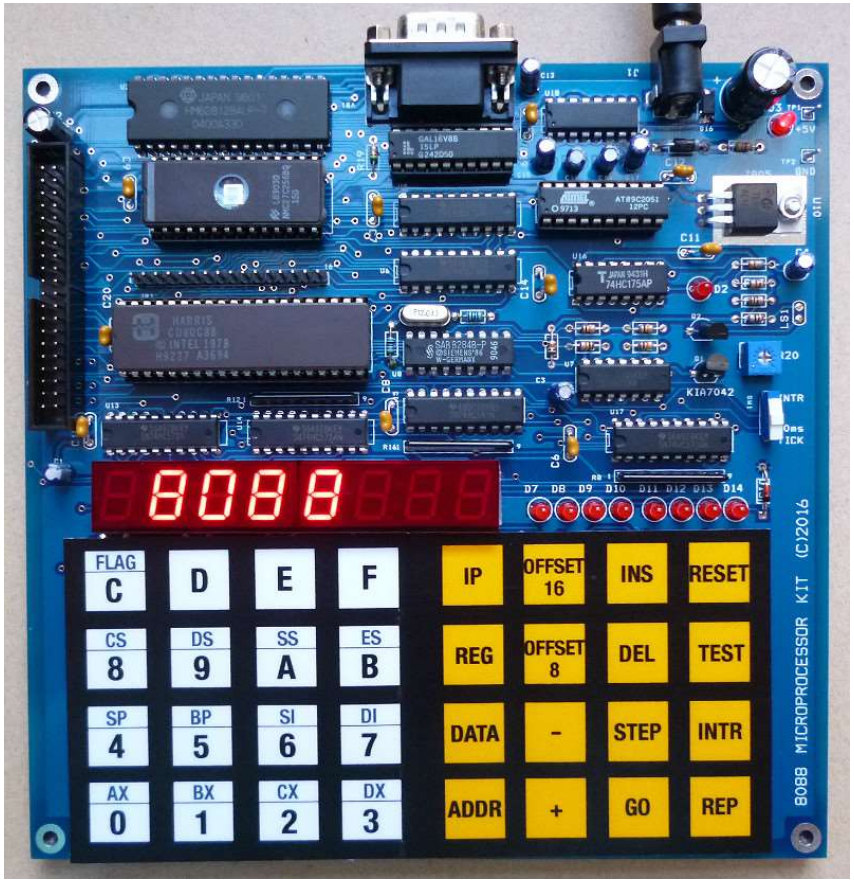
แทนเลขฐาน 10 จำนวน 1 หลัก ก็ใช้พื้นที่ 4 บิต

จะโปรแกรมโดยใช้เลขฐาน 10 ล้วน ๆได้ไหม

ใช้แค่ 0 – 9

ไม่ใช่ A - F

ไมโครคอมพิวเตอร์ในยุกต์แรกๆ มักจะมีหน้าจอตัวเลขแสดงค่าจากโปรแกรมได้ ซึ่งจะเป็นตัวเลขฐาน 16 และมีโปรแกรมเล็ก ๆ ให้ผู้ใช้สามารถส่งค่าจากรีจิสเตอร์ต่าง ๆ ออกมาแสดงได้ เรียกว่า โปรแกรมมอนิเตอร์



การแสดงเลขฐาน 10 บนหน้าจอที่แสดงเลขฐาน 16 สามารถทำได้ โดยการเข้ารหัสข้อมูลตัวเลขแบบ BCD8421
จากนี้ไปขอเรียก BCD8421 ว่า BCD หรือ (Binary-coded Decimal) เป็นการเขียนเลขฐานสอง เพื่อแทนเลขฐาน 10

จำนวน ฐาน 10	แปลงเป็นฐาน 2	รหัส BCD			
		8	4	2	1
0	0000	0	0	0	0
1	0001	0	0	0	1
2	0010	0	0	1	0
3	0011	0	0	1	1
4	0100	0	1	0	0
5	0101	0	1	0	1
6	0110	0	1	1	0
7	0111	0	1	1	1
8	1000	1	0	0	0
9	1001	1	0	0	1
10	1010	ไม่มี			
11	1011				
12	1100				
13	1101				
14	1110				
15	1111				

รหัส BCD คือเลขฐานสอง ของ 0 – 9
ส่วนที่เกินเลข 9 ขึ้นมา จะไม่นิยาม
เลขฐาน 2 เหมือนกับรหัส BCD เฉพาะแบบ BCD8421 เท่านั้น
รหัส BCD มีการเรียงบิตหลายแบบ
แน่ในวิชานี้จะใช้ BCD แบบ 8421
ซึ่งสามารถแปลงเลขฐาน 10 เป็น ฐาน 2 ใช้ได้เลย

ถ้าจำนวนเกิน 9 จะเข้ารหัสเป็น BCD อย่างไร ?

หากจำนวนมีค่าเกิน 9 ก็สามารถเข้ารหัส BCD ในหลักต่อไปได้เลย เหมือนกับการแปลงตัวอักษรเป็นรหัส 1 BYTE จะแทนตัวเลขได้ 2 หลัก

จำนวน ฐาน 10	แปลงเป็น ฐาน 2	รหัส BCD			
		8	4	2	1
0	0000	0	0	0	0
1	0001	0	0	0	1
2	0010	0	0	1	0
3	0011	0	0	1	1
4	0100	0	1	0	0
5	0101	0	1	0	1
6	0110	0	1	1	0
7	0111	0	1	1	1
8	1000	1	0	0	0
9	1001	1	0	0	1
10	1010	ไม่มี			
11	1011				
12	1100				
13	1101				
14	1110				
15	1111				

123987 แปลงเป็นเลขฐานสองได้เป็น 0001 1110 0100 0101 0011

123987 แปลงเป็นBCDได้เป็น 0001 0010 0010 1001 1000 0111

การแปลงเลขฐาน 10 เป็นรหัส BCD ทำได้ง่าย ถึงแปลงเลขฐาน 2 ไม่เป็น ก็สามารถใช้วิธีเปิดตารางแปลงได้

ข้อดีคือ เข้ารหัสง่าย ถอดรหัสง่าย

ข้อเสียคือ ใช้พื้นที่เก็บมากกว่าการใช้เลขฐาน 2 แทนเลขฐาน 16

การแปลงโดยใช้ 4 บิตแทนตัวเลขฐาน 10 จำนวน 1 หลัก พื้นที่ 1 Byte จะเก็บได้ 2 หลัก เรียกว่า Packed BCD

การแปลงโดยใช้ 8 บิตแทนตัวเลขฐาน 10 จำนวน 1 หลัก พื้นที่ 1 Byte จะเก็บได้ 1 หลัก (4 บิตบนไม่ถูกใช้งาน) เรียกว่า Unpacked BCD

จำนวน ฐาน 10	แปลงเป็น ฐาน 2	รหัส BCD			
		8	4	2	1
0	0000	0	0	0	0
1	0001	0	0	0	1
2	0010	0	0	1	0
3	0011	0	0	1	1
4	0100	0	1	0	0
5	0101	0	1	0	1
6	0110	0	1	1	0
7	0111	0	1	1	1
8	1000	1	0	0	0
9	1001	1	0	0	1
10	1010	ไม่มี			
11	1011				
12	1100				
13	1101				
14	1110				
15	1111				

23 แปลงเป็น Packed BCD จะได้เป็น 0010 0011 หรือ 0x23

23 แปลงเป็น Unpacked BCD จะได้เป็น 00000010 00000011 หรือ 0x0203

ในบางกรณี เราสามารถป้อนรหัส BCD เพื่อให้ไมโครโพรเซสเซอร์คำนวณ แล้วอ่านคำตอบจากหน้าจอที่แสดงผลเป็นเลข ฐาน 16 ได้ทันทีหากคำตอบไม่เกินช่วงของรหัส BCD

ตัวอย่างเช่น 1234 + 1111 ป้อนเป็นรหัส BCD ได้ดังนี้

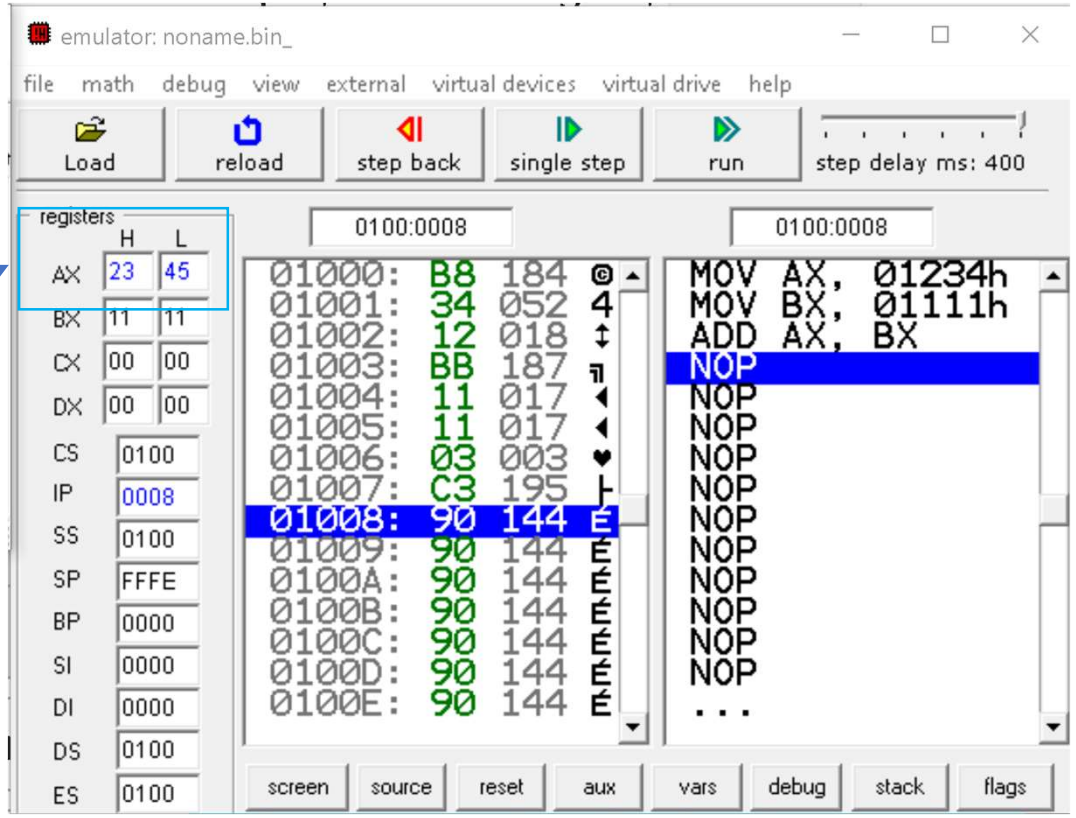
1 2 3 4

ป้อน 1234 เลยก็ได้ แต่ต้องมี 0x นำหน้าเพราะจริง ๆเป็นเลขฐาน 16

MOV AX, 0001001000110100B

MOV BX, 0001000100010001B

ADD AX,BX



คำตอบคือ 2345

อยู่ในรีจิสเตอร์ AX

หน้าจอแสดงเลขฐาน 16

แต่อ่านเป็นฐาน 10 ได้เลย

เพราะข้อมูลเป็นรหัส BCD

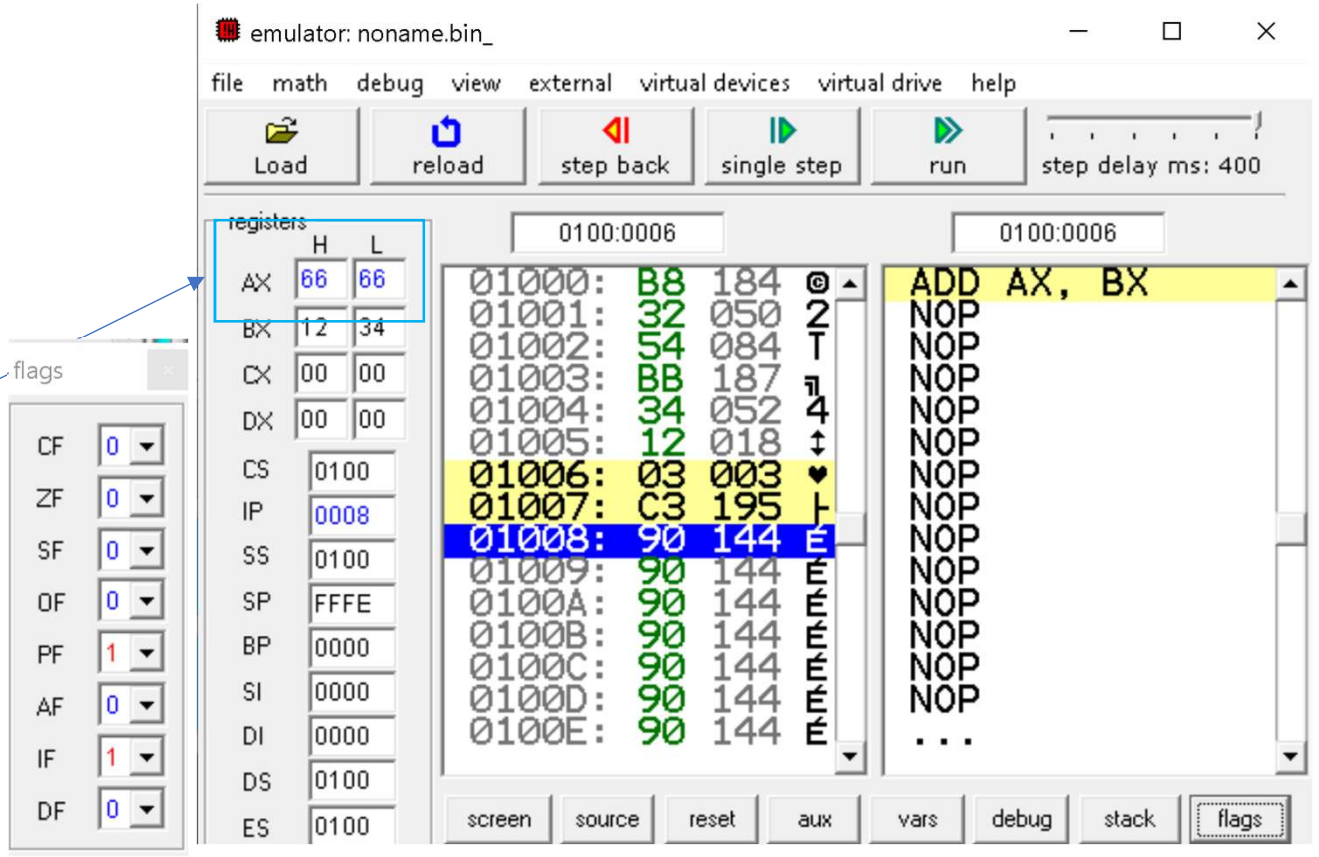
ในบางกรณี เราสามารถป้อนรหัส BCD เพื่อให้ไมโครโพรเซสเซอร์คำนวณ แล้วอ่านคำตอบจากหน้าจอที่แสดงผลเป็นเลข ฐาน 16 ได้ทันทีหากคำตอบไม่เกินช่วงของรหัส BCD

ตัวอย่างเช่น 5432 + 1234 ป้อนเป็นรหัส BCD ได้ดังนี้

```
MOV AX, 0x5432
MOV BX, 0x1234
ADD AX,BX
```

คำตอบคือ 6666
อยู่ในรีจิสเตอร์ AX

หน้าจอแสดงเลขฐาน 16
แต่อ่านเป็นฐาน 10 ได้เลย
เพราะข้อมูลเป็นรหัส BCD



แต่การบวกเลขแบบนี้ไม่สามารถใช้ได้ในทุกกรณี
ตัวอย่างเช่น 6789 + 1234 ป้อนเป็นรหัส BCD ได้ดังนี้

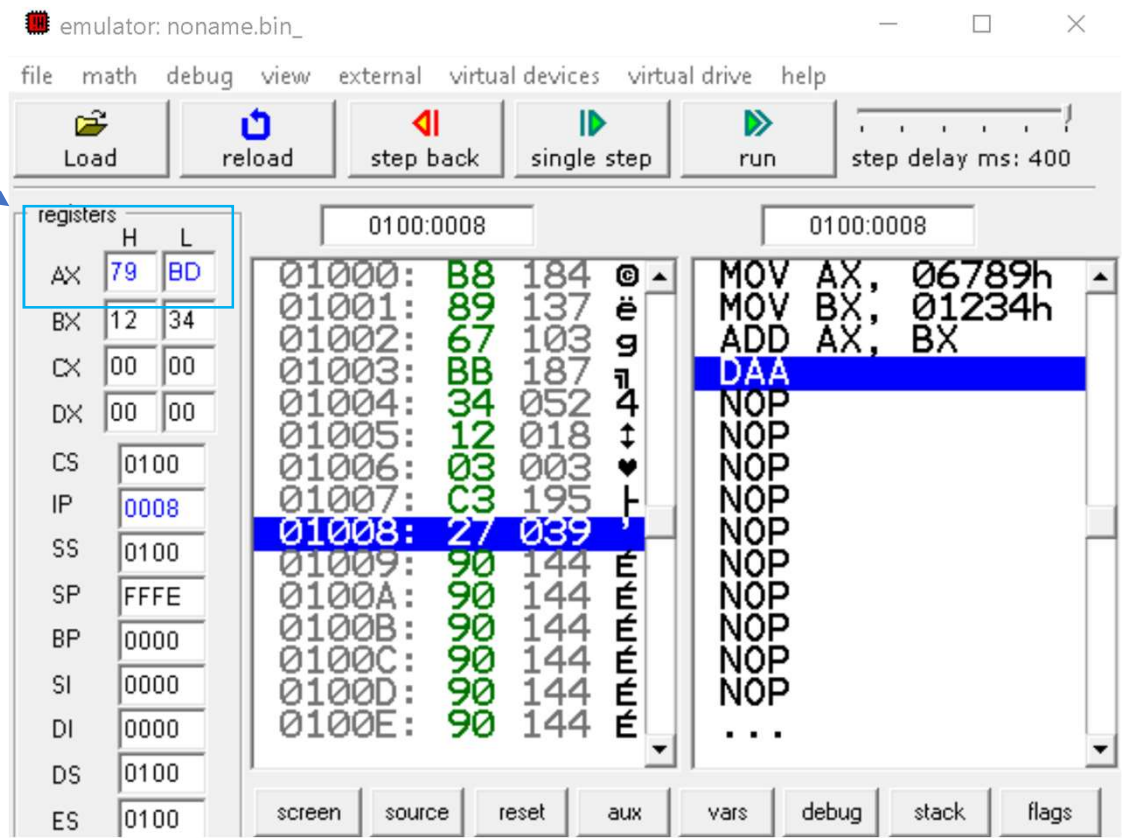
คำตอบจริงคือ

8023

ค่าเกินช่วงรหัส BCD แสดงเป็นเลข ฐาน 16
และคำตอบไม่ถูกต้อง
หากตัวเลขล้น ต้องนำคำตอบหลักที่เกินไปบวก
6

```
7 9 B  D      หลักสุดท้าย D ค่าเกิน 79BD + ไป 6 เก็บหลักสุดท้าย
      6  +
7 9 C 3       หลักสุดท้าย C ค่าเกิน 79C + ไป 6 เก็บหลักสุดท้าย
    6    +
7 A 2         หลักสุดท้าย A ค่าเกิน 7A + ไป 6 เก็บหลักสุดท้าย
  6      +
8 0           ไม่มีค่าเกินจึงหยุดการบวก

8 0 2 3  ตอบ
```

```
MOV AX, 0x6789
MOV BX, 0x1234
ADD AX,BX
```



แต่การบวกเลขแบบนี้ไม่สามารถใช้ได้ในทุกกรณี
ตัวอย่างเช่น 6789 + 1 ป้อนเป็นรหัส BCD ได้ดังนี้

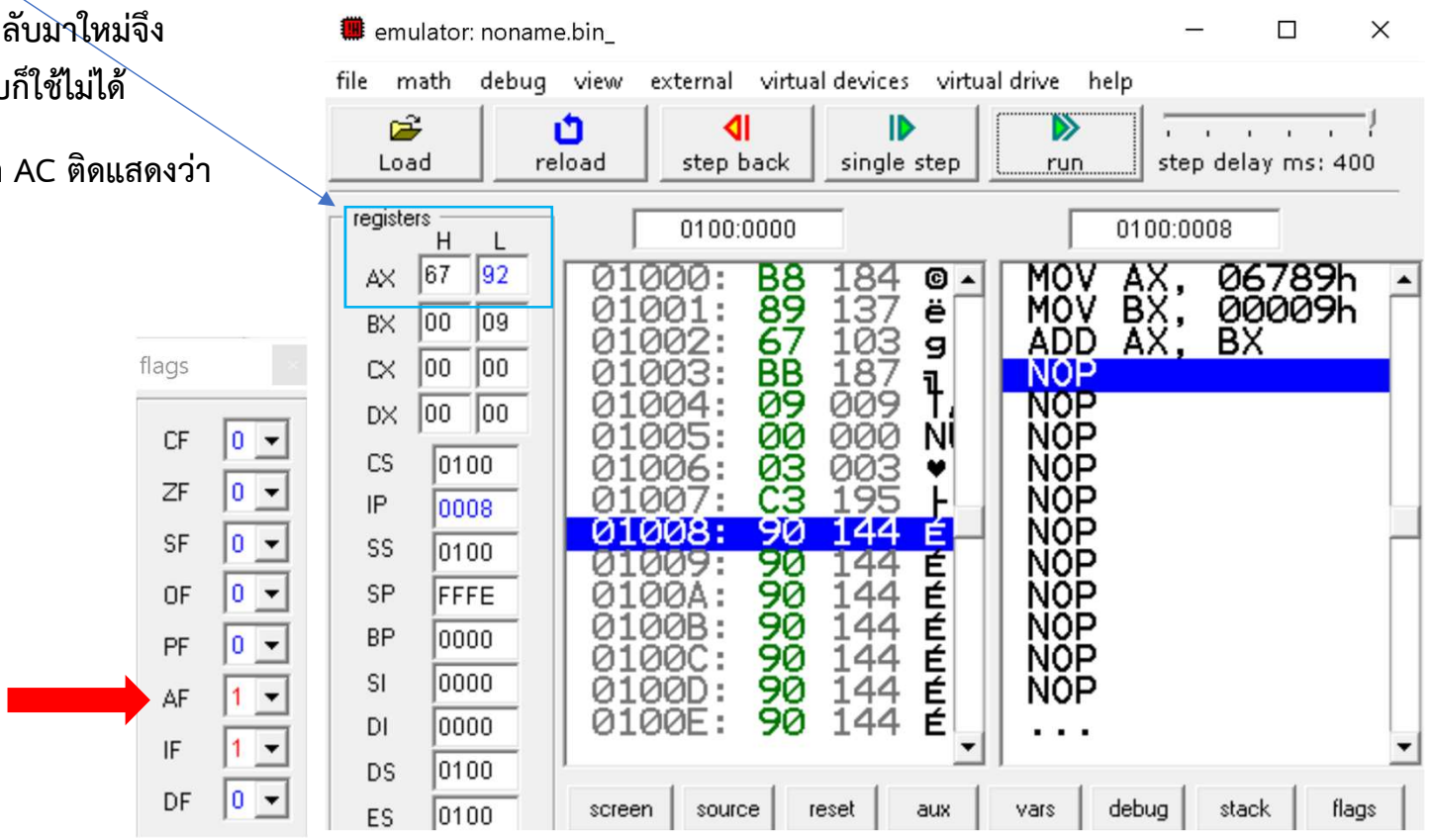
คำตอบจริงคือ

6798

ค่าเกินช่วงรหัส BCD และวนกลับมาใหม่จึง
จึงไม่ติดเลขฐาน 16 แต่คำตอบก็ใช้ไม่ได้

กรณีนี้ให้สังเกต Flag AC หาก AC ติดแสดงว่า
คำตอบนั้น ยังใช้ไม่ได้
ต้องนำมาบวก 6 ก่อน
6792 + 6
= 6798

```
MOV AX, 0X6789
MOV BX, 0x0009
ADD AX,BX
```



OPCODE	OPERAND	EXPLANATION	EXAMPLE
ADD	D, S	D = D + S	ADD AX, [2050]
ADC	D, S	D = D + S + prev. carry	ADC AX, BX
INC	D	D = D + 1	INC AX
SUB	D, S	D = D – S	SUB AX, [SI]
SBB	D, S	D = D – S – prev. carry	SBB [2050], 0050
DEC	D	D = D – 1	DEC [2050]
MUL	8-bit register	AX = AL * 8-bit reg.	MUL BH
MUL	16-bit register	DX AX = AX * 16-bit reg.	MUL CX
IMUL	8 or 16 bit register	performs signed multiplication	IMUL CX
DIV	8-bit register	AX = AX / 8-bit reg. ; AL = quotient ; AH = remainder	DIV BL
DIV	16-bit register	DX AX / 16-bit reg. ; AX = quotient ; DX = remainder	DIV CX
IDIV	8 or 16 bit register	performs signed division	IDIV BL
CBW	none	converts signed byte to word	CBW
CWD	none	converts signed word to double word	CWD
NEG	D	D = 2's compliment of D	NEG AL
DAA	none	decimal adjust accumulator	DAA
DAS	none	decimal adjust accumulator after subtraction	DAS
AAA	none	ASCII adjust accumulator after addition	AAA
AAS	none	ASCII adjust accumulator after subtraction	AAS
AAM	none	ASCII adjust accumulator after multiplication	AAM
AAD	none	ASCII adjust accumulator after division	AAD

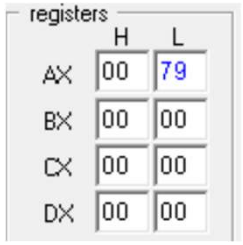
DAA : Decimal Adjust AL after Addition : ปรับค่าใน AL ในกรณีที่ค่าเกินช่วงจากการบวกเลข BCD

การใช้ : DAA

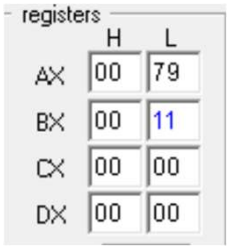
ไม่มี Operand

คำสั่งนี้ทำงานที่รีจิสเตอร์ 8 บิต

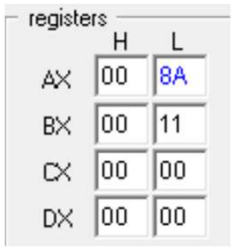
```
MOV AL,0x79
MOV BL,0x11
ADD AL,BL
DAA
```



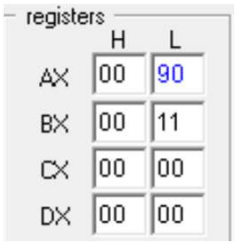
MOV AL,0x79



MOV BL,0x11



ADD AL,BL



DAA

หากต้องการบวกเลข BCD ที่มีขนาดเกิน 8 บิต จะทำอย่างไร ?

บวกเลข BCD ที่มีขนาด 16 บิต

```
MOV CX , 0x6789
MOV DX , 0x1234
```

ป้อนตัวเลขที่ต้องการบวก ลงในรีจิสเตอร์ CX และ DX

```
MOV AL,CL
ADD AL,DL
DAA
MOV CL,AL
```

นำ byte ล่างของ CX เข้าสู่ AL และทำการบวกกับ DL

ปรับคำตอบเป็น BCD แล้วสำเนาค่ากลับไปไว้ที่ byte ล่างของ CX

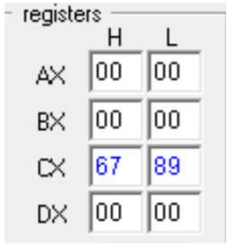
```
MOV AL,CH
ADC AL,DH
DAA
MOV CH,AL
```

นำ byte บนของ CX เข้าสู่ AL และทำการบวกกับ DH แบบนำตัวทดจากการบวกก่อนหน้ามารวมด้วย

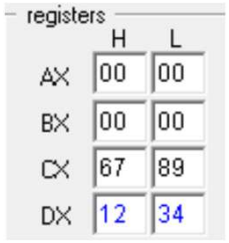
ปรับคำตอบเป็น BCD แล้วสำเนาค่ากลับไปไว้ที่ byte บนของ CX

บวกเลข BCD ที่มีขนาด 16 บิต

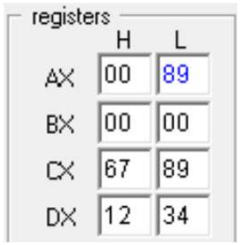
```
MOV CX , 0x6789
MOV DX , 0x1234

MOV AL,CL
ADD AL,DL
DAA
MOV CL,AL

MOV AL,CH
ADC AL,DH
DAA
MOV CH,AL
```



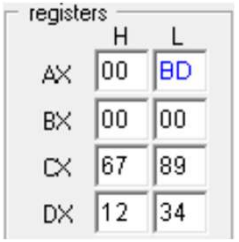
MOV CX , 0x6789



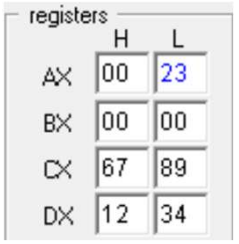
MOV DX , 0x1234



MOV AL,CL

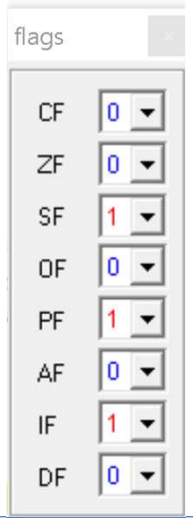


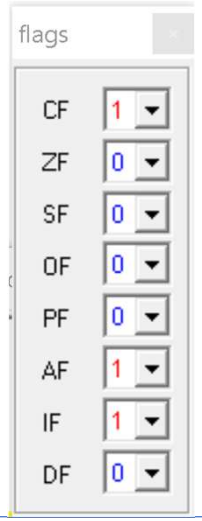
ADD AL,DL



DAA

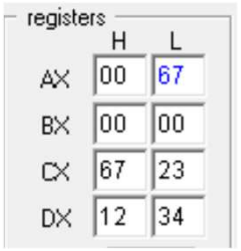
MOV CL,AL



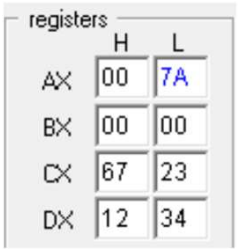


บวกเลข BCD ที่มีขนาด 16 บิต

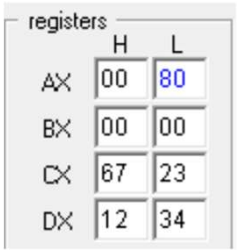
```
MOV CX , 0x6789
MOV DX , 0x1234

MOV AL,CL
ADD AL,DL
DAA
MOV CL,AL

MOV AL,CH
ADC AL,DH
DAA
MOV CH,AL
```



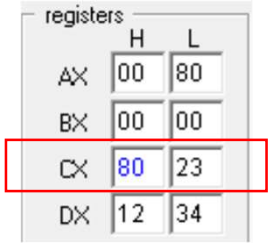
MOV AL,CH



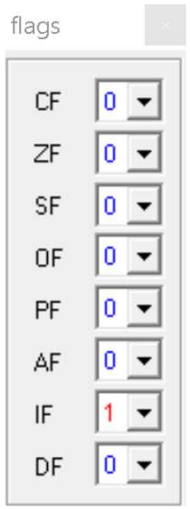
ADD AL,DH

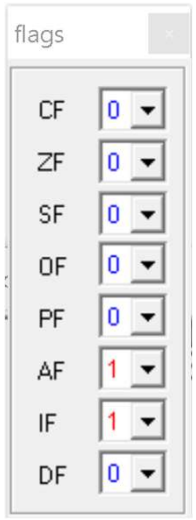


DAA



MOV CH,AL





OPCODE	OPERAND	EXPLANATION	EXAMPLE
ADD	D, S	D = D + S	ADD AX, [2050]
ADC	D, S	D = D + S + prev. carry	ADC AX, BX
INC	D	D = D + 1	INC AX
SUB	D, S	D = D – S	SUB AX, [SI]
SBB	D, S	D = D – S – prev. carry	SBB [2050], 0050
DEC	D	D = D – 1	DEC [2050]
MUL	8-bit register	AX = AL * 8-bit reg.	MUL BH
MUL	16-bit register	DX AX = AX * 16-bit reg.	MUL CX
IMUL	8 or 16 bit register	performs signed multiplication	IMUL CX
DIV	8-bit register	AX = AX / 8-bit reg. ; AL = quotient ; AH = remainder	DIV BL
DIV	16-bit register	DX AX / 16-bit reg. ; AX = quotient ; DX = remainder	DIV CX
IDIV	8 or 16 bit register	performs signed division	IDIV BL
CBW	none	converts signed byte to word	CBW
CWD	none	converts signed word to double word	CWD
NEG	D	D = 2's compliment of D	NEG AL
DAA	none	decimal adjust accumulator	DAA
DAS	none	decimal adjust accumulator after subtraction	DAS
AAA	none	ASCII adjust accumulator after addition	AAA
AAS	none	ASCII adjust accumulator after subtraction	AAS
AAM	none	ASCII adjust accumulator after multiplication	AAM
AAD	none	ASCII adjust accumulator after division	AAD

AAA : ASCII adjust after addition ชื่อคำสั่งไม่เกี่ยวอะไรกับรหัส ASCII คำสั่งนี้คล้าย DAA แต่รับ input จาก AL แล้วแปลงเป็น Unpacked BCD โดยผลการแปลงจะหลักสิบจะอยู่ที่ AH และหลักหน่วยจะอยู่ที่ AL

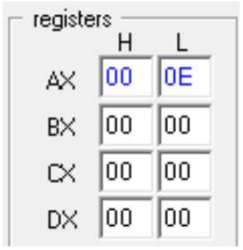
การใช้ : AAA

ไม่มี Operand

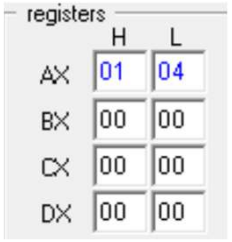
ใช้บวกเลขฐาน 10 หลักเดียวเท่านั้น มีประโยชน์เมื่อนำไปใช้กับ Hardware ที่รับ input แบบ Unpacked BCD หรือหน่วยประมวลผลที่รับข้อมูลได้แค่ครั้งละ 4 บิต เท่านั้น

```
MOV AX,0X5
ADD AX,0X9
AAA
```

ก่อนคำสั่ง AAA



หลังคำสั่ง AAA

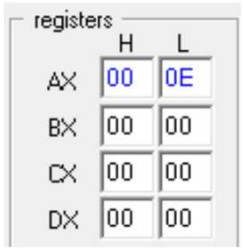


อ่านค่าเฉพาะ 4 บิตล่างของ AH และ AL

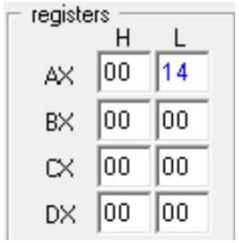
ตอบ 14

```
MOV AX,0X5
ADD AX,0X9
DAA
```

ก่อนคำสั่ง DAA



หลังคำสั่ง DAA

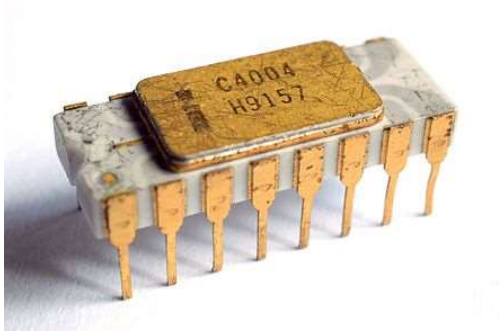


อ่านคำตอบจาก AL ทั้ง 8 บิต

ตอบ 14

หากเครื่องคอมพิวเตอร์สามารถแสดงข้อมูลได้ทั้ง 16 บิต ใช้ DAA จะดีกว่า

ไมโครโพรเซสเซอร์ 8086 รองรับการคำนวณแบบ BCD เนื่องจากมันถูกพัฒนาต่อมาจาก โมโครโพรเซสเซอร์ 4004



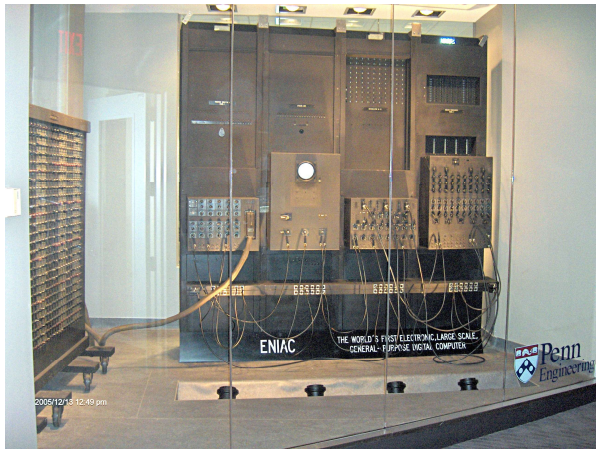
Max. CPU clock rate	740-750 kHz
Data width	4 bits
Address width	12 bits (multiplexed)

The Unicom 141P and the NCR 18-36 were OEM versions of the Busicom 141-PF

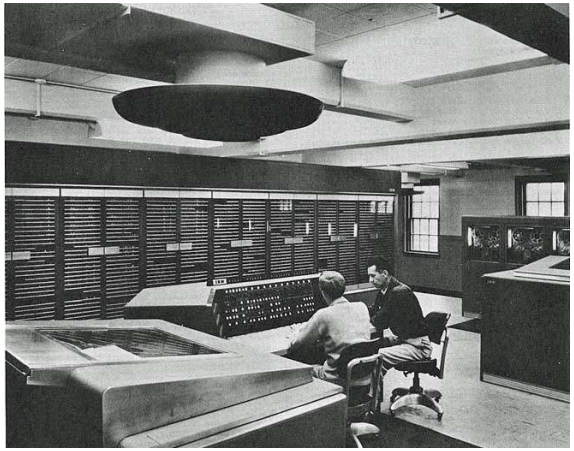
เป็นไปได้หรือไม่ที่จะสร้างไมโครโพรเซสเซอร์ ที่ประมวลผลด้วยรหัส BCD ล้วน ๆ แล้วโปรแกรมข้อมูล และ ตำแหน่งด้วยรหัส BCD ทั้งหมด ซึ่งแปลงเป็นเลขฐาน 10 ได้ง่ายมาก ๆ

คำสั่ง ADD ทำงานกับ BCD โดยตรง
ไม่จำเป็นต้องแปลงด้วยคำสั่ง DAA
?

จริง ๆแล้วคอมพิวเตอร์ในยุกต์แรก ๆ เป็นแบบเลขฐาน 10 เรียกว่า Decimal Computer และใช้รหัส BCD ตัวอย่างเช่น ENIAC, IBM NORC, IBM 650, IBM 1620, IBM 7070, UNIVAC Solid State 80



ENIAC



IBM Naval Ordnance Research Calculator

IBM 650

ความนิยมของ Decimal Computer ค่อย ๆ ลดลงเนื่องจาก การป้อนข้อมูลโดยตรงกับรีจิสเตอร์ หรือ การอ่านผลการคำนวณจากสัญญาณไฟฟ้า ถูกแทนที่ด้วย User interface ของระบบปฏิบัติการ การรับหรือแสดงผลเลขฐาน 10 ถูกแปลงด้วย software แทน ทุกวันนี้จึงไม่มี Decimal Computer เหลืออยู่แล้ว

ที่เหลืออยู่ก็เพียง opcode สำหรับแปลง BCD และ 1 บิตใน flag ของ CPU X86 เท่านั้น

INC : Increment เพิ่มค่าไป 1

การใช้ : INC D

การทำงาน D = D +1

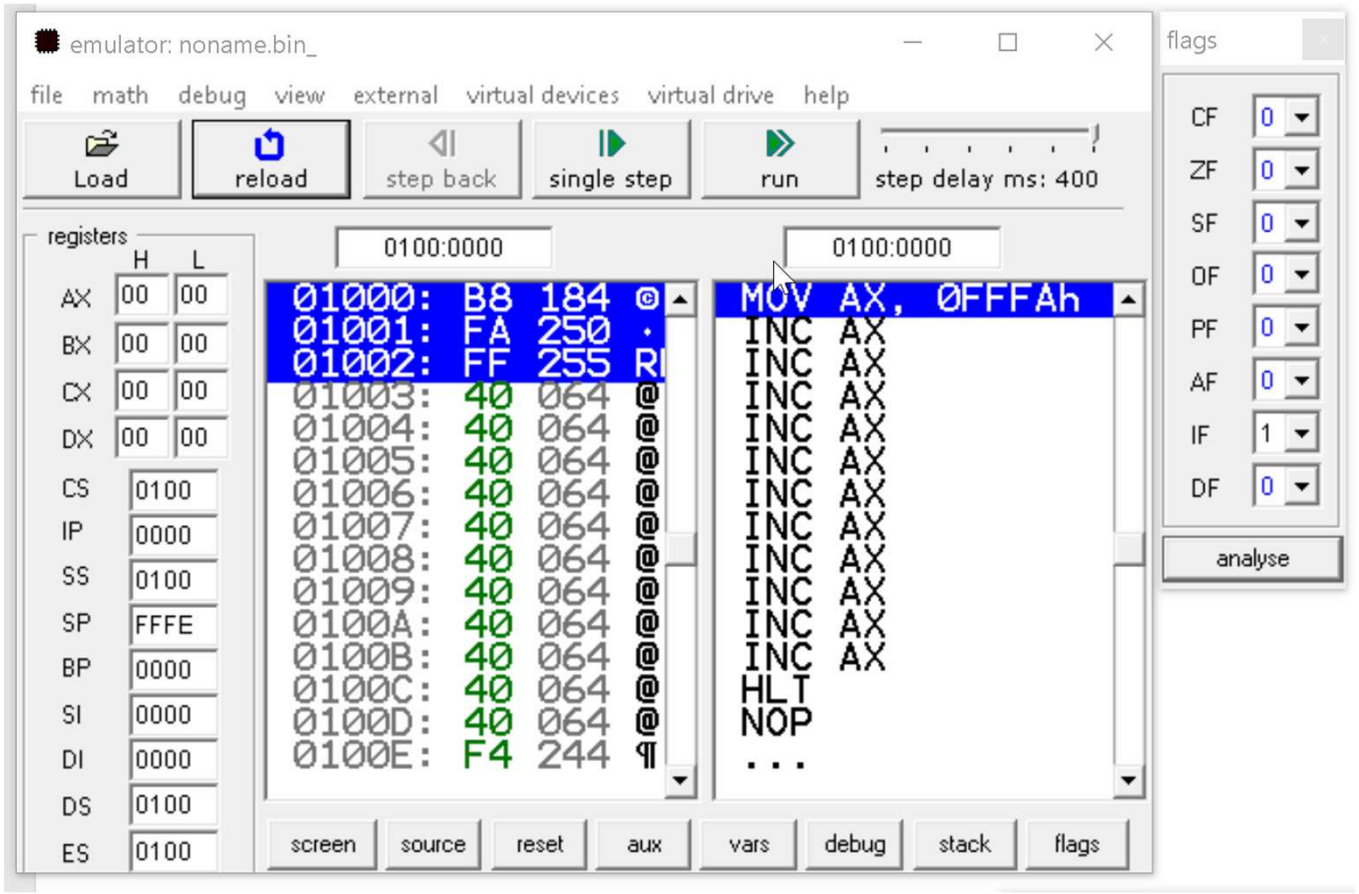
Operand ที่ใช้ได้:

REG

memory

มีผลกับ Flag:

ZF, SF, OF, PF, AF,




```
MOV AX,0xFFFA
INC AX
INC AX
INC AX
INC AX
INC AX
INC AX
INC AX
INC AX
INC AX
INC AX
INC AX
HLT
```

OPCODE	OPERAND	EXPLANATION	EXAMPLE
ADD	D, S	D = D + S	ADD AX, [2050]
ADC	D, S	D = D + S + prev. carry	ADC AX, BX
INC	D	D = D + 1	INC AX
SUB	D, S	D = D – S	SUB AX, [SI]
SBB	D, S	D = D – S – prev. carry	SBB [2050], 0050
DEC	D	D = D – 1	DEC [2050]
MUL	8-bit register	AX = AL * 8-bit reg.	MUL BH
MUL	16-bit register	DX AX = AX * 16-bit reg.	MUL CX
IMUL	8 or 16 bit register	performs signed multiplication	IMUL CX
DIV	8-bit register	AX = AX / 8-bit reg. ; AL = quotient ; AH = remainder	DIV BL
DIV	16-bit register	DX AX / 16-bit reg. ; AX = quotient ; DX = remainder	DIV CX
IDIV	8 or 16 bit register	performs signed division	IDIV BL
CBW	none	converts signed byte to word	CBW
CWD	none	converts signed word to double word	CWD
NEG	D	D = 2's compliment of D	NEG AL
DAA	none	decimal adjust accumulator	DAA
DAS	none	decimal adjust accumulator after subtraction	DAS
AAA	none	ASCII adjust accumulator after addition	AAA
AAS	none	ASCII adjust accumulator after subtraction	AAS
AAM	none	ASCII adjust accumulator after multiplication	AAM
AAD	none	ASCII adjust accumulator after division	AAD

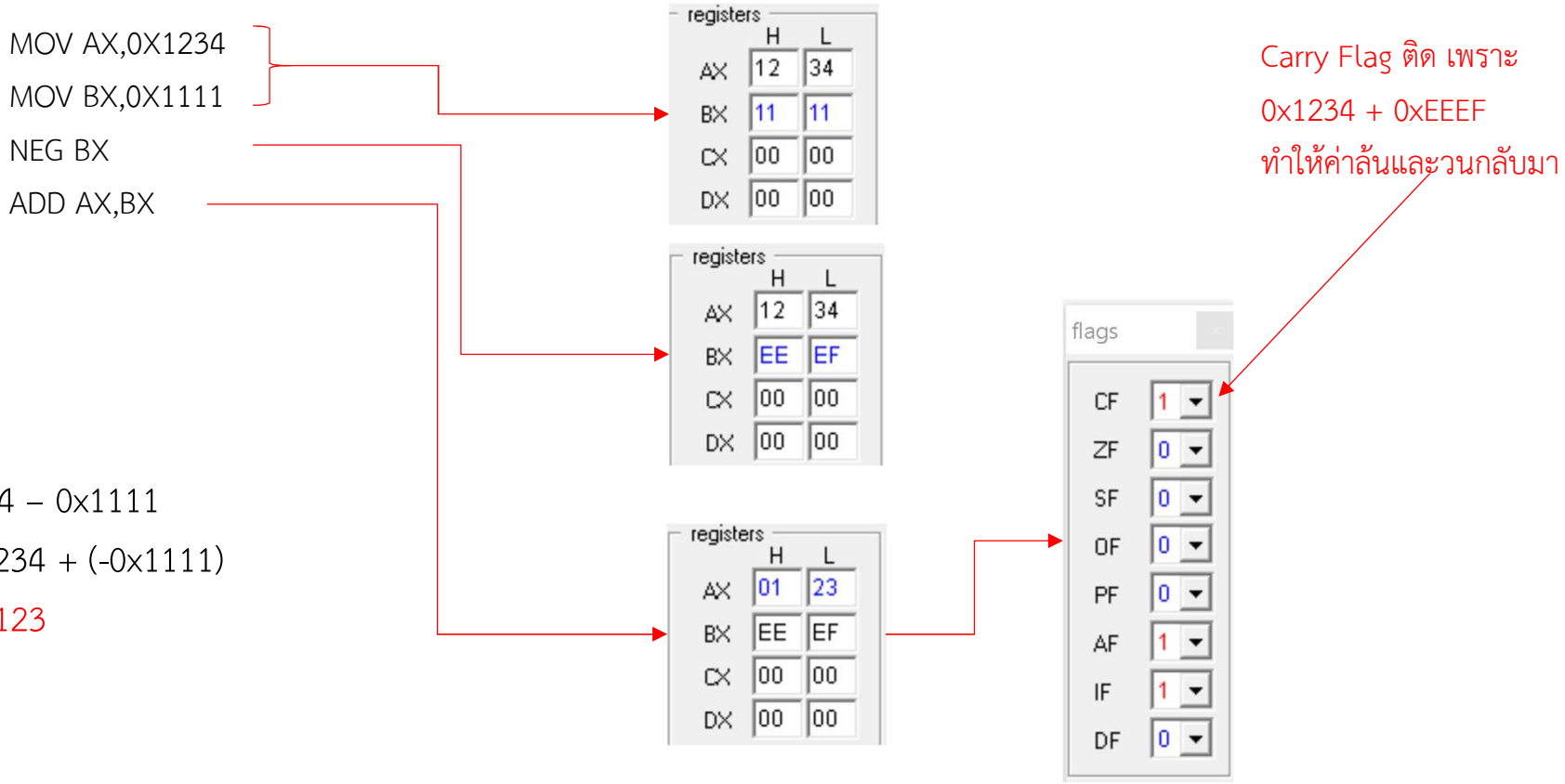
การลบเลข

ถ้าเราอาศัยคุณสมบัติ

A – B = A + (-B)

Opcode สำหรับลบเลขก็ไม่จำเป็น

เราสามารถคำนวณได้ด้วย Opcode คำนวณ 2's complement และ Opcode บวกเลข



0x1234 – 0x1111

= 0x1234 + (-0x1111)

= 0x0123

OPCODE	OPERAND	EXPLANATION	EXAMPLE
ADD	D, S	D = D + S	ADD AX, [2050]
ADC	D, S	D = D + S + prev. carry	ADC AX, BX
INC	D	D = D + 1	INC AX
SUB	D, S	D = D – S	SUB AX, [SI]
SBB	D, S	D = D – S – prev. carry	SBB [2050], 0050
DEC	D	D = D – 1	DEC [2050]
MUL	8-bit register	AX = AL * 8-bit reg.	MUL BH
MUL	16-bit register	DX AX = AX * 16-bit reg.	MUL CX
IMUL	8 or 16 bit register	performs signed multiplication	IMUL CX
DIV	8-bit register	AX = AX / 8-bit reg. ; AL = quotient ; AH = remainder	DIV BL
DIV	16-bit register	DX AX / 16-bit reg. ; AX = quotient ; DX = remainder	DIV CX
IDIV	8 or 16 bit register	performs signed division	IDIV BL
CBW	none	converts signed byte to word	CBW
CWD	none	converts signed word to double word	CWD
NEG	D	D = 2's compliment of D	NEG AL
DAA	none	decimal adjust accumulator	DAA
DAS	none	decimal adjust accumulator after subtraction	DAS
AAA	none	ASCII adjust accumulator after addition	AAA
AAS	none	ASCII adjust accumulator after subtraction	AAS
AAM	none	ASCII adjust accumulator after multiplication	AAM
AAD	none	ASCII adjust accumulator after division	AAD

SUB : Subtraction ใช้ลบเลข

การใช้ : SUB D, S

Operand :

REG , memory

Memory ,REG

REG,REG

Memory. Immediate

REG, immediate

***REG: AX, BX, CX, DX, AL, BL, BH, CH, CL, DL, DI, SI, BP, SP**

Output: D

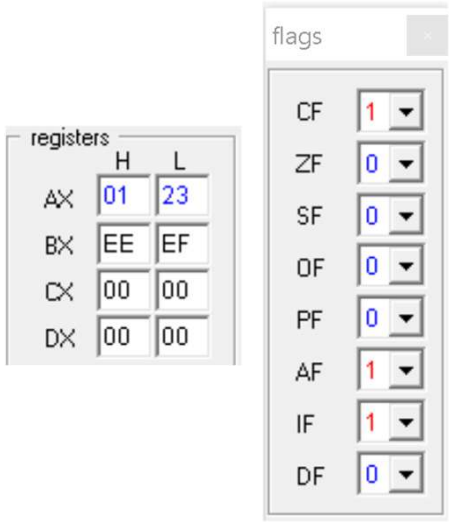
มีผลกับ Flag

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
				O	D	I	T	S	Z		AC		P		CY

SUB กับ ADD นั้นมี Operand และมีผลกับ Flag เหมือนกันทุกประการ

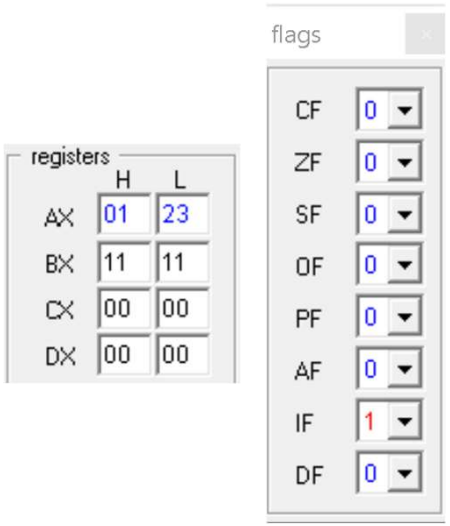
บิต	ความหมาย	เป็น 1 เมื่อ
O	Overflow คำตอบมีค่าเกินขนาดของรีจิสเตอร์	ล้น
S	Sign คำตอบติดลบ (บิต MSB เป็น 1)	ติดลบ
Z	Zero คำตอบเป็นศูนย์	มีค่า 0
AC	Auxiliary Carry ใช้สำหรับรหัส BCD	บิตที่ 3 เป็น 1
P	Parity ภาวะคู่หรือคี่ของคำตอบ	คำตอบเป็นเลขคู่
CY	Carry มีการทดเลข	มีเลขทด

```
MOV AX,0X1234
MOV BX,0X1111
NEG BX
ADD AX,BX
```

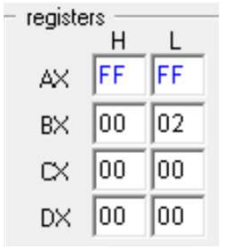


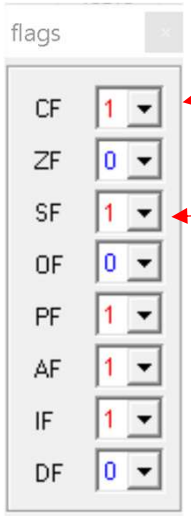

```
MOV AX,0X1234
MOV BX,0X1111
SUB AX,BX
```

Carry Flag ไม่ติด เพราะ
0x1234 - 0x1111
คำตอบไม่ล้น



```
MOV AX,0X0001
MOV BX,0X0002
SUB AX,BX
```





มีการยืม

คำตอบติดลบ

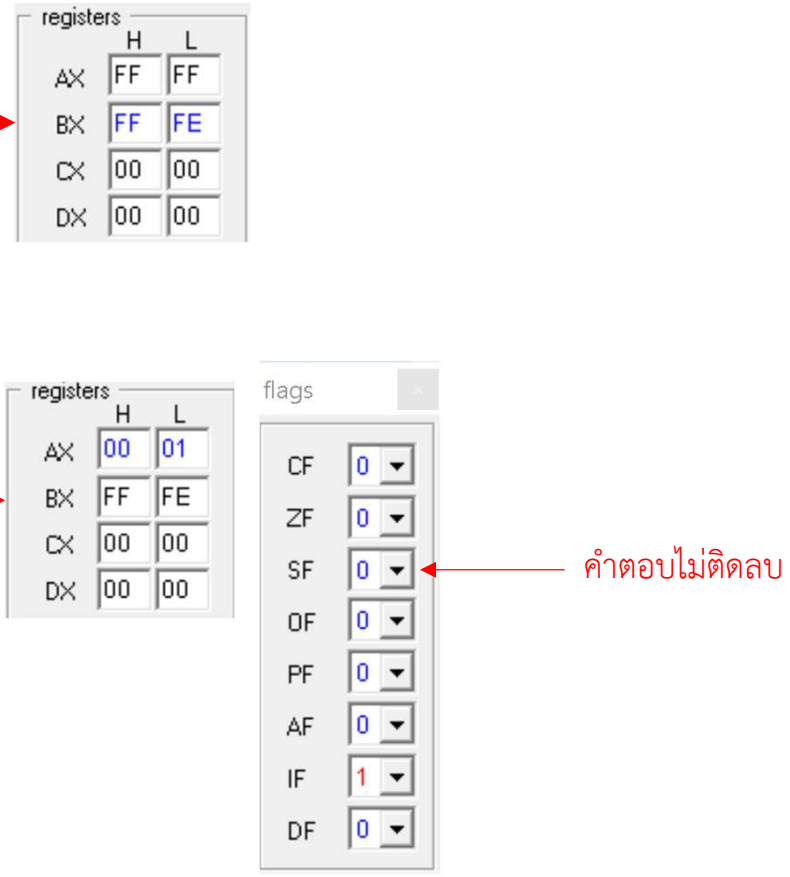
หากตัวตั้งน้อยกว่าตัวลบ
คำตอบจะติดลบ
และมีการยืมค่าจากหลักที่สูงกว่า
คำตอบจึงเป็น FFFF
Sign Flag มีค่าเป็น 1 แสดงว่าติดลบ
จึงต้องนำคำตอบมาหา 2’s complement ก่อน
ซึ่ง 2’s complement ของ FFFF คือ 0001
ดังนั้นคำตอบคือ -1

การป้อนเลขติดลบ

MOV AX,-0X0001

MOV BX,-0X0002

SUB AX,BX



OPCODE	OPERAND	EXPLANATION	EXAMPLE
ADD	D, S	D = D + S	ADD AX, [2050]
ADC	D, S	D = D + S + prev. carry	ADC AX, BX
INC	D	D = D + 1	INC AX
SUB	D, S	D = D – S	SUB AX, [SI]
SBB	D, S	D = D – S – prev. carry	SBB [2050], 0050
DEC	D	D = D – 1	DEC [2050]
MUL	8-bit register	AX = AL * 8-bit reg.	MUL BH
MUL	16-bit register	DX AX = AX * 16-bit reg.	MUL CX
IMUL	8 or 16 bit register	performs signed multiplication	IMUL CX
DIV	8-bit register	AX = AX / 8-bit reg. ; AL = quotient ; AH = remainder	DIV BL
DIV	16-bit register	DX AX / 16-bit reg. ; AX = quotient ; DX = remainder	DIV CX
IDIV	8 or 16 bit register	performs signed division	IDIV BL
CBW	none	converts signed byte to word	CBW
CWD	none	converts signed word to double word	CWD
NEG	D	D = 2's compliment of D	NEG AL
DAA	none	decimal adjust accumulator	DAA
DAS	none	decimal adjust accumulator after subtraction	DAS
AAA	none	ASCII adjust accumulator after addition	AAA
AAS	none	ASCII adjust accumulator after subtraction	AAS
AAM	none	ASCII adjust accumulator after multiplication	AAM
AAD	none	ASCII adjust accumulator after division	AAD

SBB : Subtraction with Borrow ลบเลขแบบคิดตัวยืม

การใช้ : SBB D, S

การทำงาน D = D - S - CY

Operand ที่ใช้ได้:

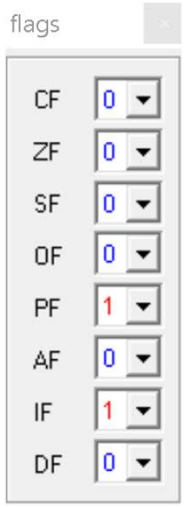
- **REG, memory**
- **memory, REG**
- **REG, REG**
- **memory, immediate**
- **REG, immediate**

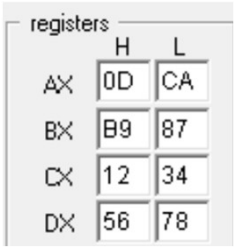
ตัวอย่างการใช้งาน ลบเลขขนาด 32 บิต

0x1FFF 0FFF - 0x1234 5678

AX BX CX DX

```
MOV AX,0X1FFF
MOV BX,0X0FFF
MOV CX,0X1234
MOV DX,0X5678

SUB BX,DX
SBB AX,CX
```



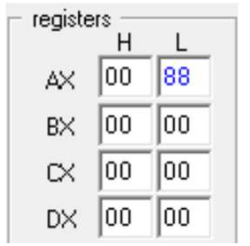


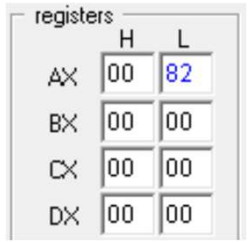
คำตอบสุดท้ายอยู่ใน AX BX

คือ 0x0DCA B987

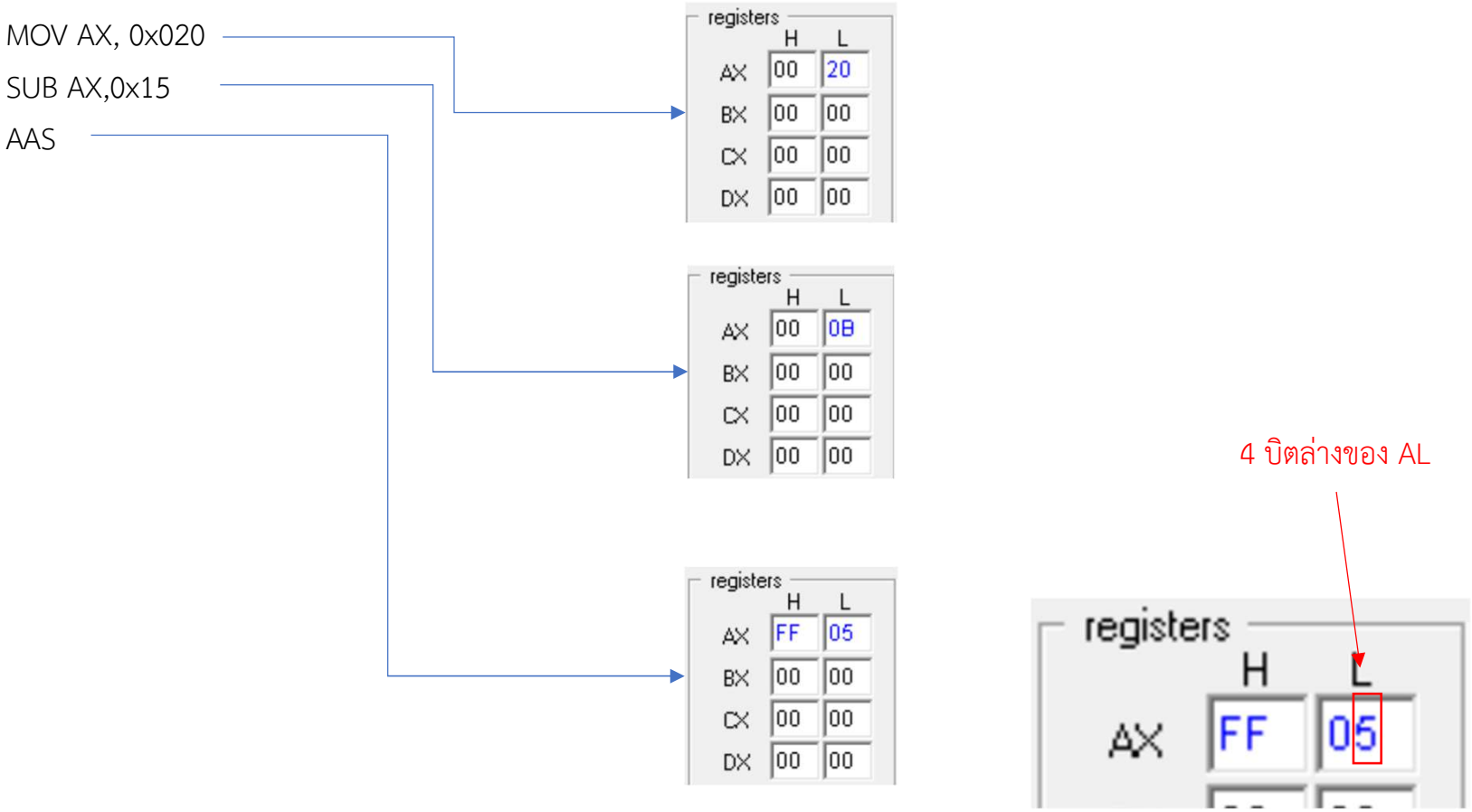
การลบเลขด้วยรหัส BCD ก็สามารถทำได้เหมือนการบวกต่างกันที่ Opcode ปรับค่าให้เป็น BCD สำหรับการลบจะใช้ DAS : **Decimal Adjust AL after Subtraction คำสั่งนี้ทำงานเฉพาะ AL เหมือนกับ DAA**

```
MOV AX,0X0091
SUB  AX,0X0009
DAS
```





การแปลงลงลบเป็น Unpacked-BCD ทำได้ด้วยคำสั่ง AAS โดยคำตอบต้องอยู่ในช่วง 0 – 9 โดยผลรันอยู่ที่ 4 บิตล่างของ AL
ตัวอย่างเช่น 20 - 15



DEC : Decrement ลดค่าไป 1

การใช้ : DEC D

การทำงาน D = D -1

Operand ที่ใช้ได้:

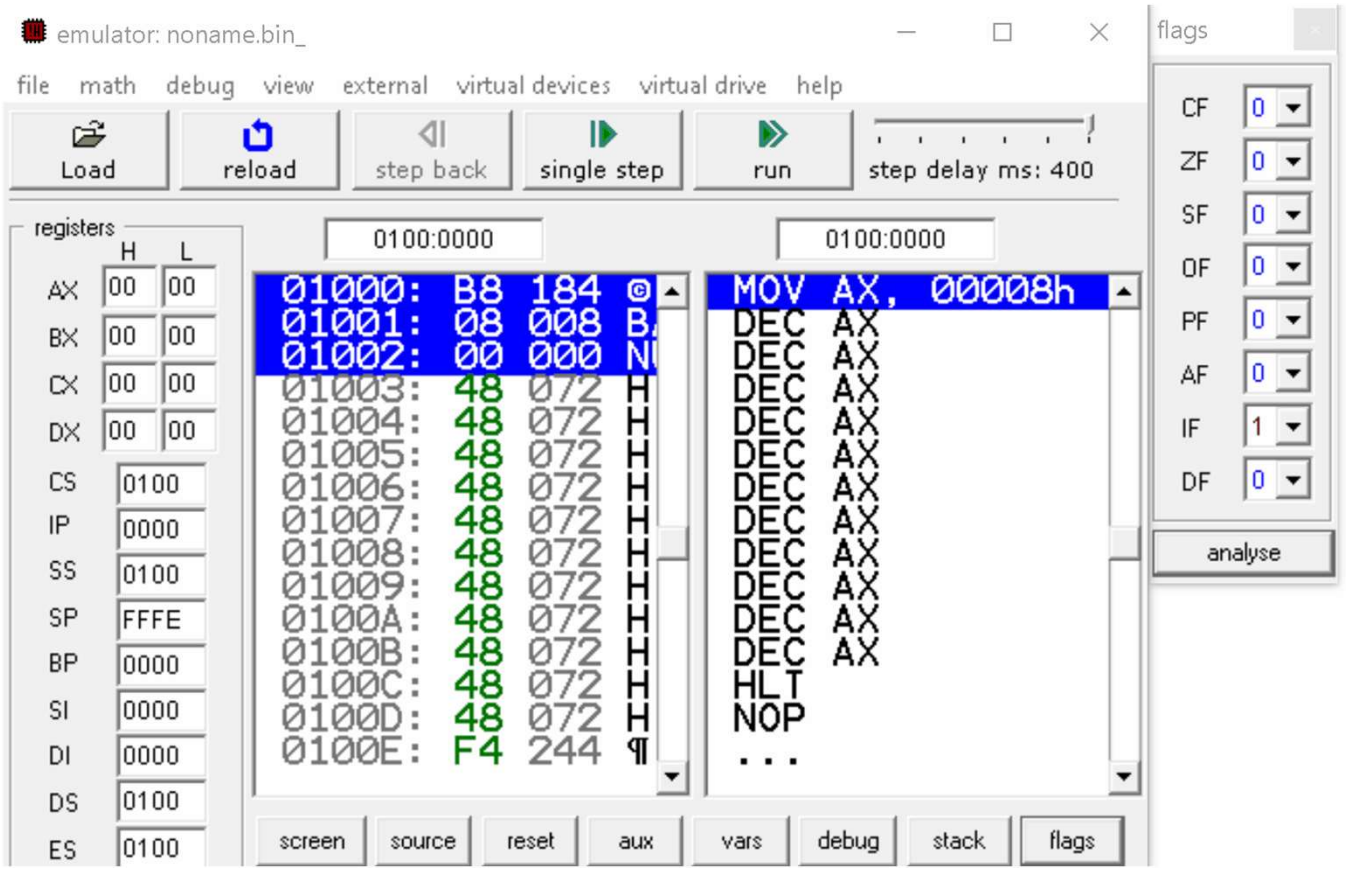
REG

memory

มีผลกับ Flag:

ZF, SF, OF, PF, AF,

```
MOV AX,0x0008
DEC AX
DEC AX
DEC AX
DEC AX
DEC AX
DEC AX
DEC AX
DEC AX
DEC AX
DEC AX
DEC AX
DEC AX
HLT
```



การคูณเลข

OPCODE	OPERAND	EXPLANATION	EXAMPLE
ADD	D, S	D = D + S	ADD AX, [2050]
ADC	D, S	D = D + S + prev. carry	ADC AX, BX
INC	D	D = D + 1	INC AX
SUB	D, S	D = D – S	SUB AX, [SI]
SBB	D, S	D = D – S – prev. carry	SBB [2050], 0050
DEC	D	D = D – 1	DEC [2050]
MUL	8-bit register	AX = AL * 8-bit reg.	MUL BH
MUL	16-bit register	DX AX = AX * 16-bit reg.	MUL CX
IMUL	8 or 16 bit register	performs signed multiplication	IMUL CX
DIV	8-bit register	AX = AX / 8-bit reg. ; AL = quotient ; AH = remainder	DIV BL
DIV	16-bit register	DX AX / 16-bit reg. ; AX = quotient ; DX = remainder	DIV CX
IDIV	8 or 16 bit register	performs signed division	IDIV BL
CBW	none	converts signed byte to word	CBW
CWD	none	converts signed word to double word	CWD
NEG	D	D = 2's compliment of D	NEG AL
DAA	none	decimal adjust accumulator	DAA
DAS	none	decimal adjust accumulator after subtraction	DAS
AAA	none	ASCII adjust accumulator after addition	AAA
AAS	none	ASCII adjust accumulator after subtraction	AAS
AAM	none	ASCII adjust accumulator after multiplication	AAM
AAD	none	ASCII adjust accumulator after division	AAD

การคูณเลขมี 2 แบบ คือ แบบ 8 บิต และ 16 บิต
ทั้งสองแบบใช้ Mnemonic เดียวกันคือ

MUL

MUL : Multiply คูณเลขแบบ 8 บิต

การใช้ : MUL operand

การทำงาน AX = AL x operand

Operand ที่ใช้ได้:

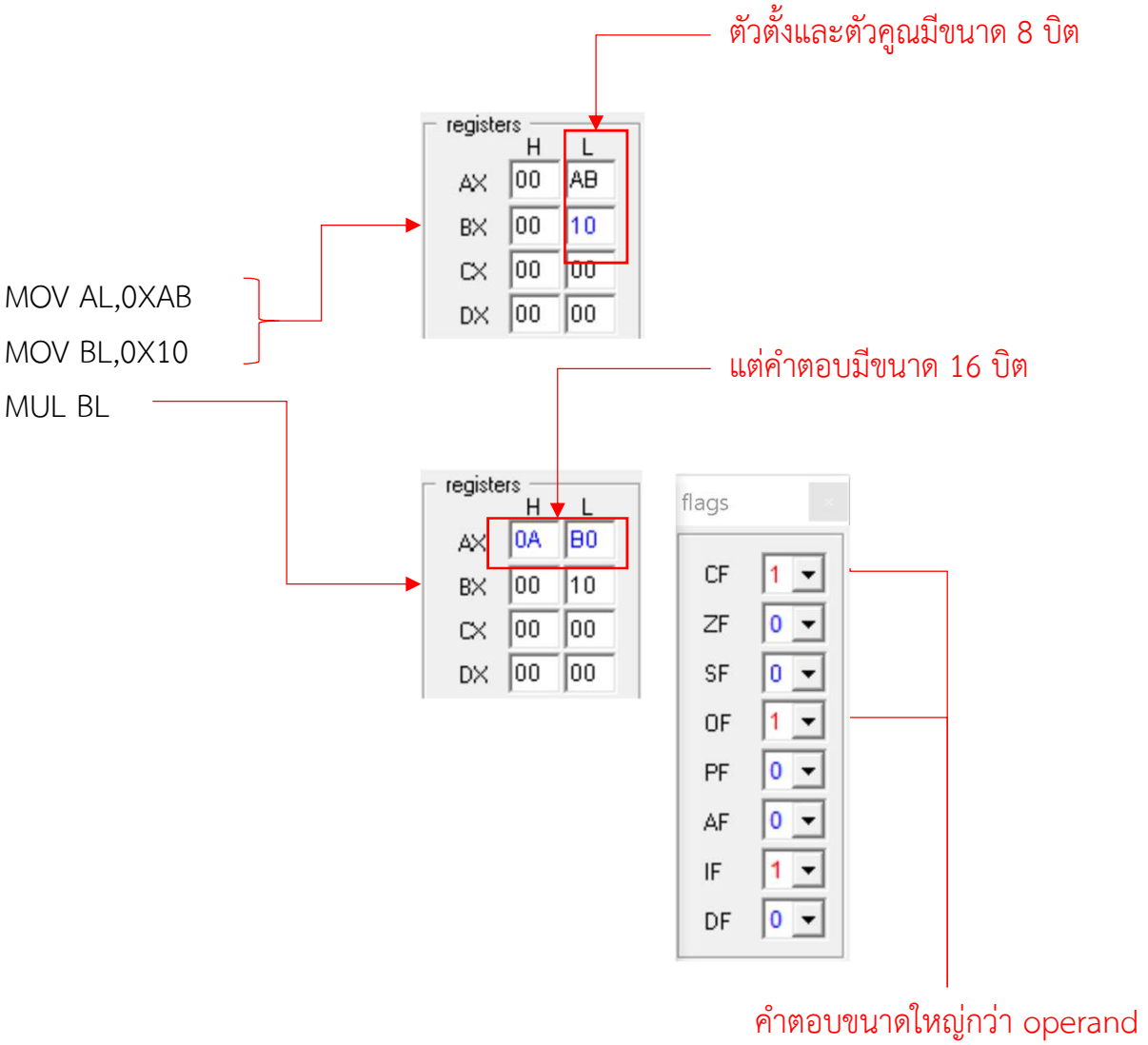
- REG8 บิต
- Memory

มีผลกับ Flag:

- CF, OF

หากคำตอบมีขนาดเกิน operand CF และ OF จะเป็น 1

คำสั่งนี้ไม่พิจารณาเลขติดลบ จึงไม่มีผลกับ SF



MUL : Multiply คูณเลขแบบ 8 บิต

การใช้ : MUL operand

การทำงาน AX = AL x operand

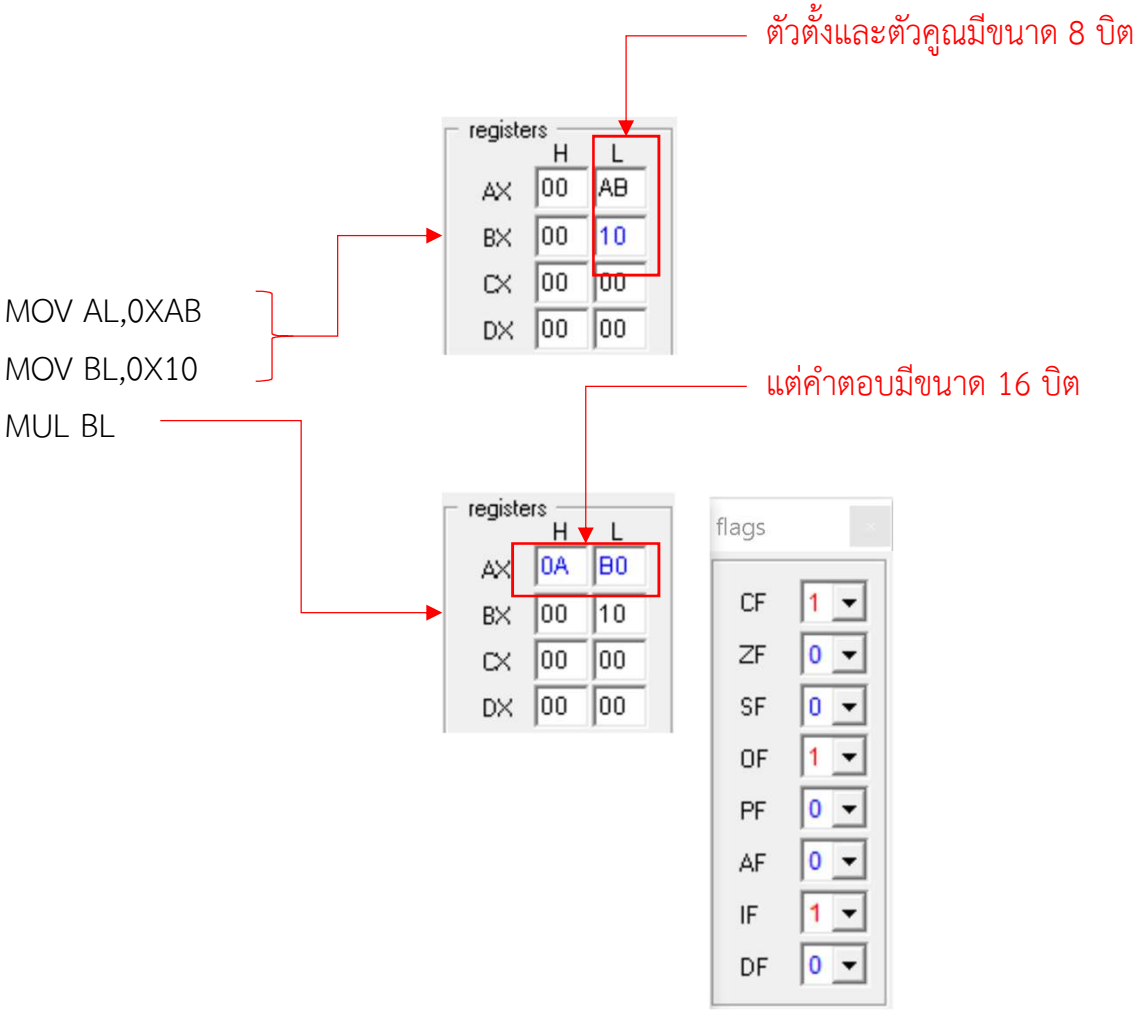
Operand ที่ใช้ได้:

- **REG8 บิต**
- **Memory**

มีผลกับ Flag:

- **CF, OF**

คำสั่งนี้ไม่พิจารณาเลขติดลบ จึงไม่มีผลกับ SF



MUL : Multiply คูณเลขแบบ 8 บิต

การใช้ : MUL operand

การทำงาน AX = AL x operand

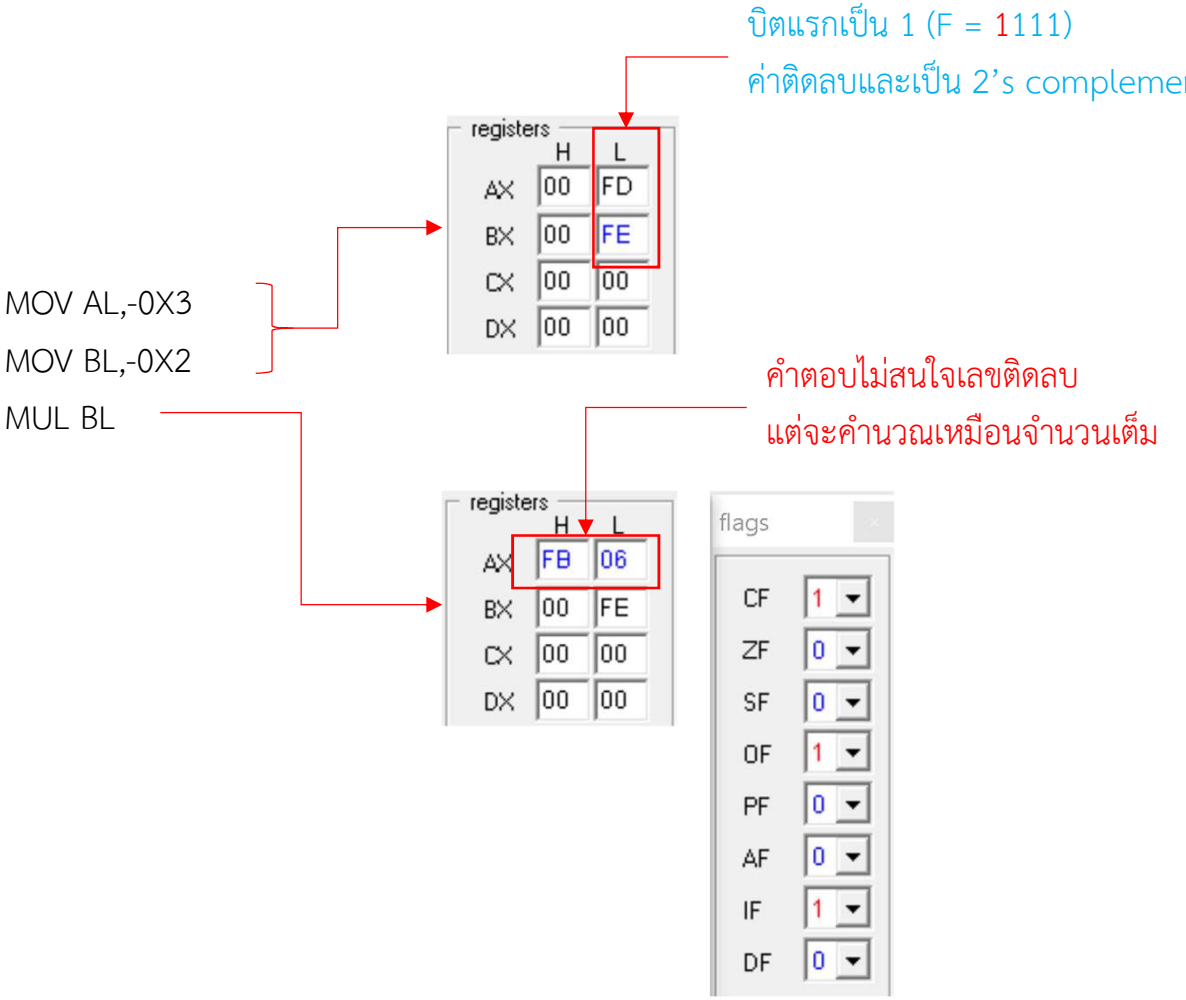
Operand ที่ใช้ได้:

- **REG8 บิต**
- **Memory**

มีผลกับ Flag:

- **CF, OF**

คำสั่งนี้ไม่พิจารณาเลขติดลบ จึงไม่มีผลกับ SF



MUL : Multiply คูณเลขแบบ 8 บิต

การใช้ : MUL operand

การทำงาน AX = AL x operand

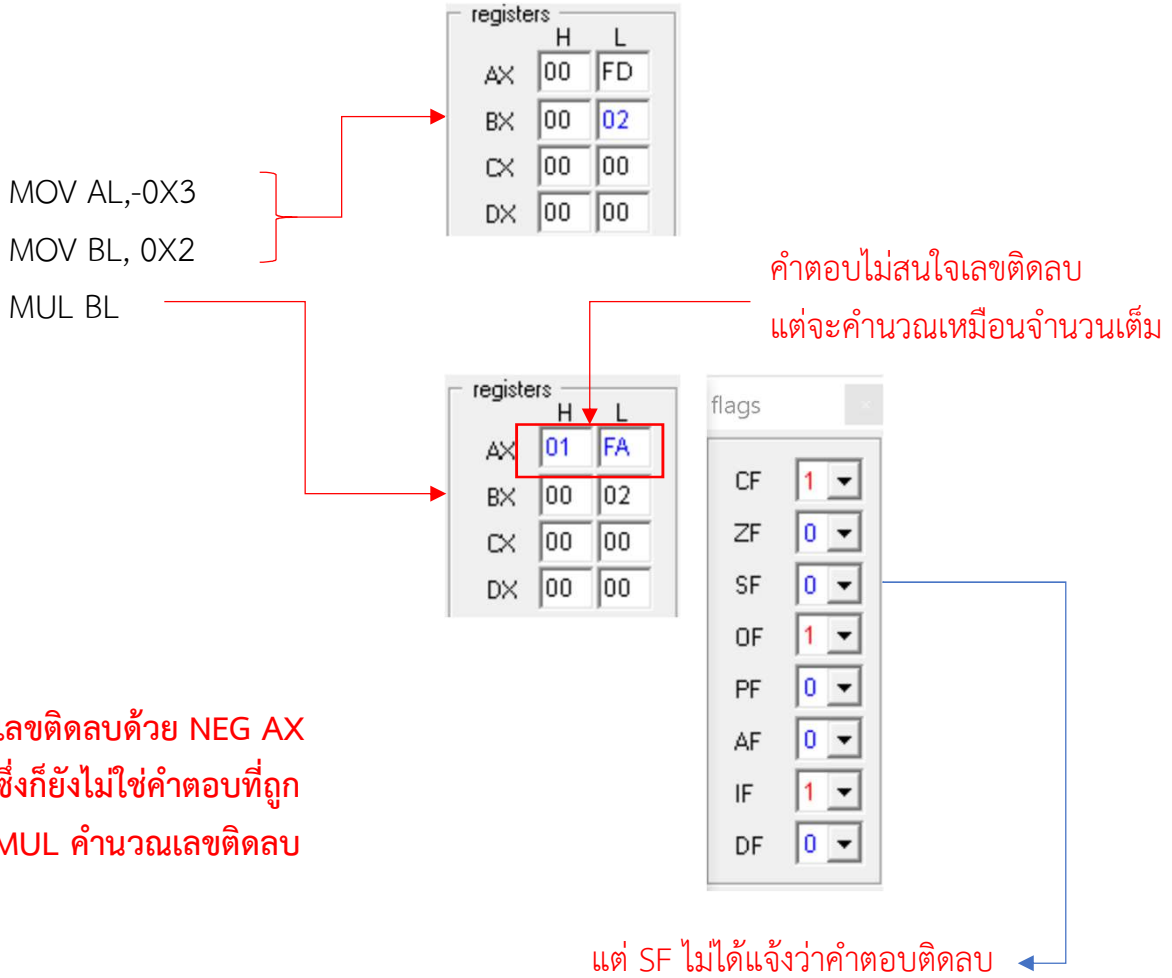
Operand ที่ใช้ได้:

- **REG8 บิต**
- **Memory**

มีผลกับ Flag:

- **CF, OF**

คำสั่งนี้ไม่พิจารณาเลขติดลบ จึงไม่มีผลกับ SF



MUL : Multiply คูณเลขแบบ 16 บิต

การใช้ : MUL operand

การทำงาน DX AX = AX x operand

Operand ที่ใช้ได้:

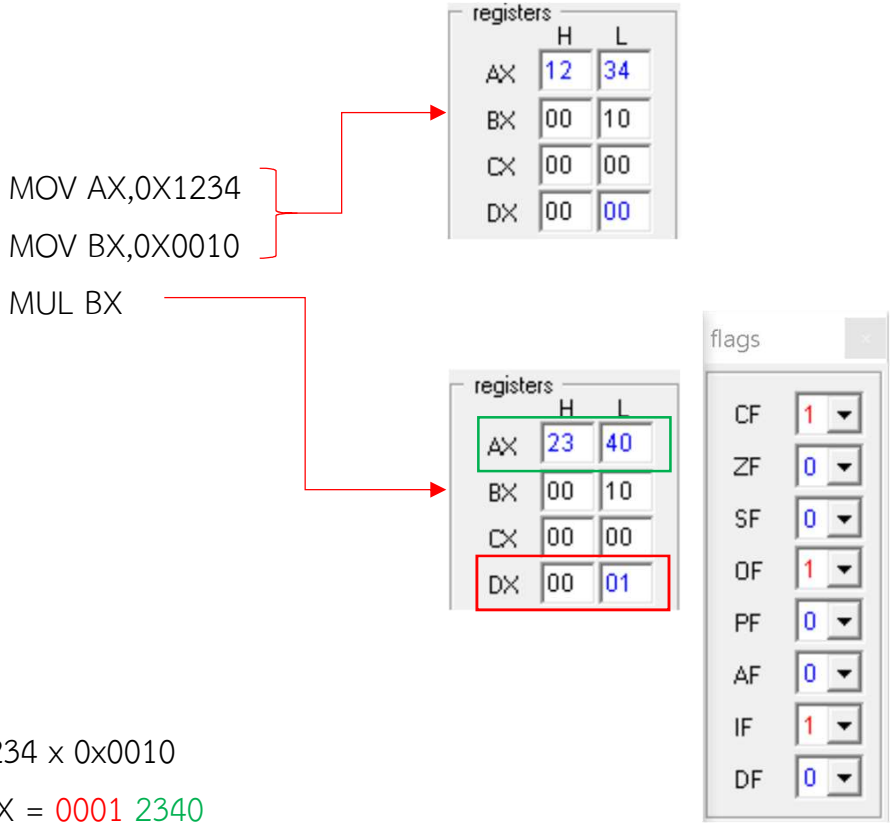
- **REG16 บิต**
- **Memory**

มีผลกับ Flag:

- **CF, OF**

หากคำตอบมีขนาดเกิน operand CF และ OF จะเป็น 1

คำสั่งนี้ไม่พิจารณาเลขติดลบ จึงไม่มีผลกับ SF

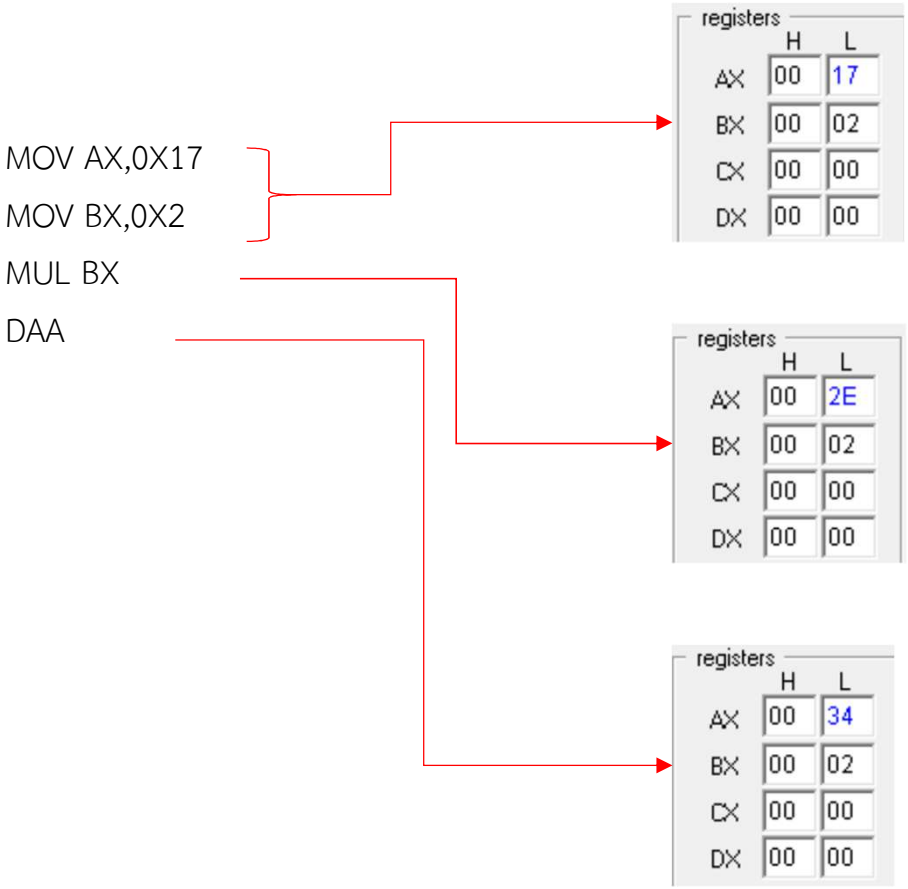


คำตอบของ 0x1234 x 0x0010

อยู่ที่ DX และ AX = 0001 2340

มีขนาด 32 บิต

การคูณเลขด้วยรหัส BCD ก็สามารถทำได้ด้วย คำสั่ง MUL เช่นกัน และทำการปรับตัวเลขด้วยคำสั่ง DAA แต่ได้แค่ 8 บิต



การคูณเลขแบบคิดเครื่องหมายจะใช้ Mnemonic

IMUL

โดยมีการทำงานเหมือนกับ MUL แต่ต่างกันที่ IMUL จะรองรับจำนวน

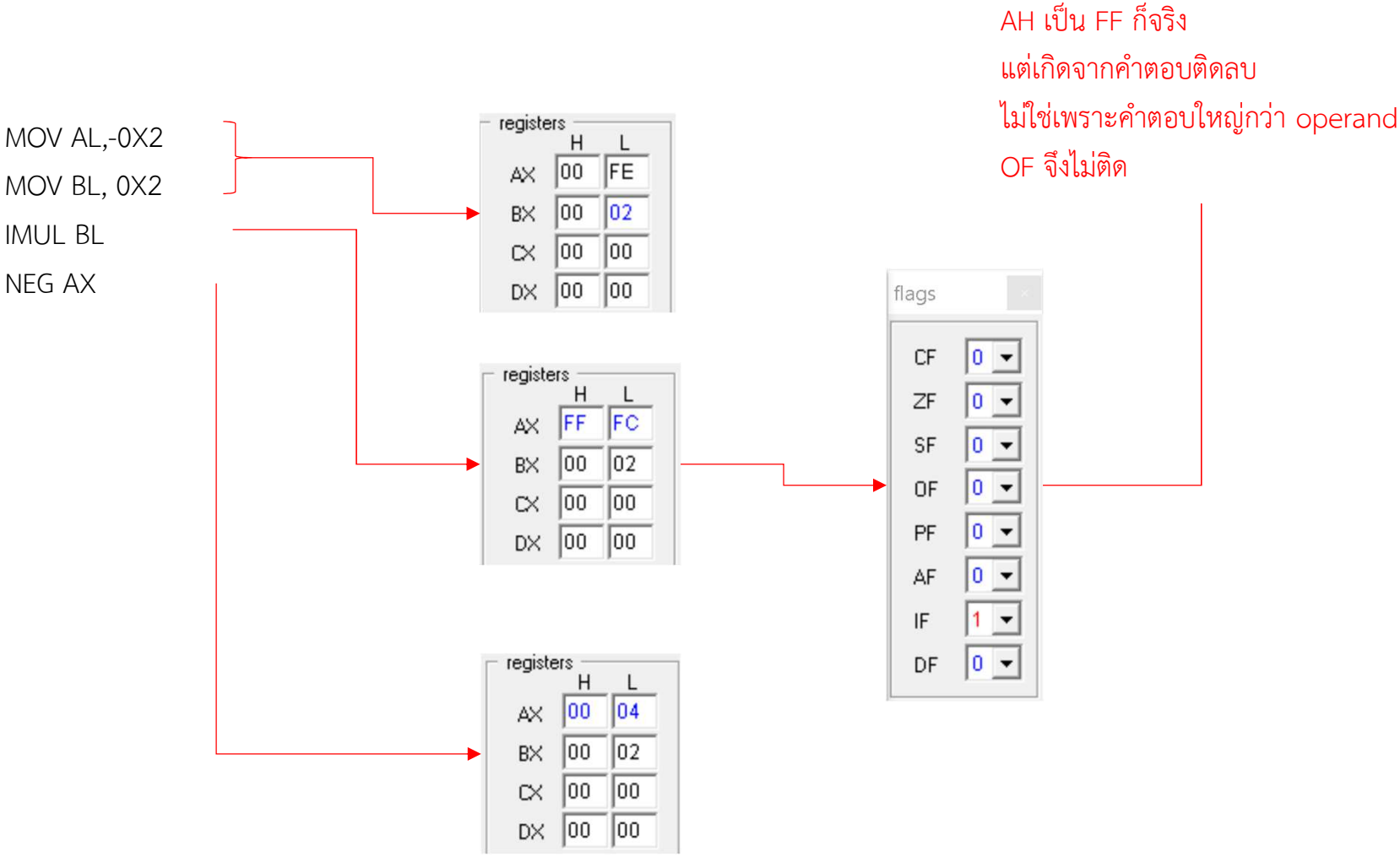
2’s complement ทำให้คำนวณเลขติดลบได้

แต่ IMUL จะไม่มีผลกับ flag SF ทำให้

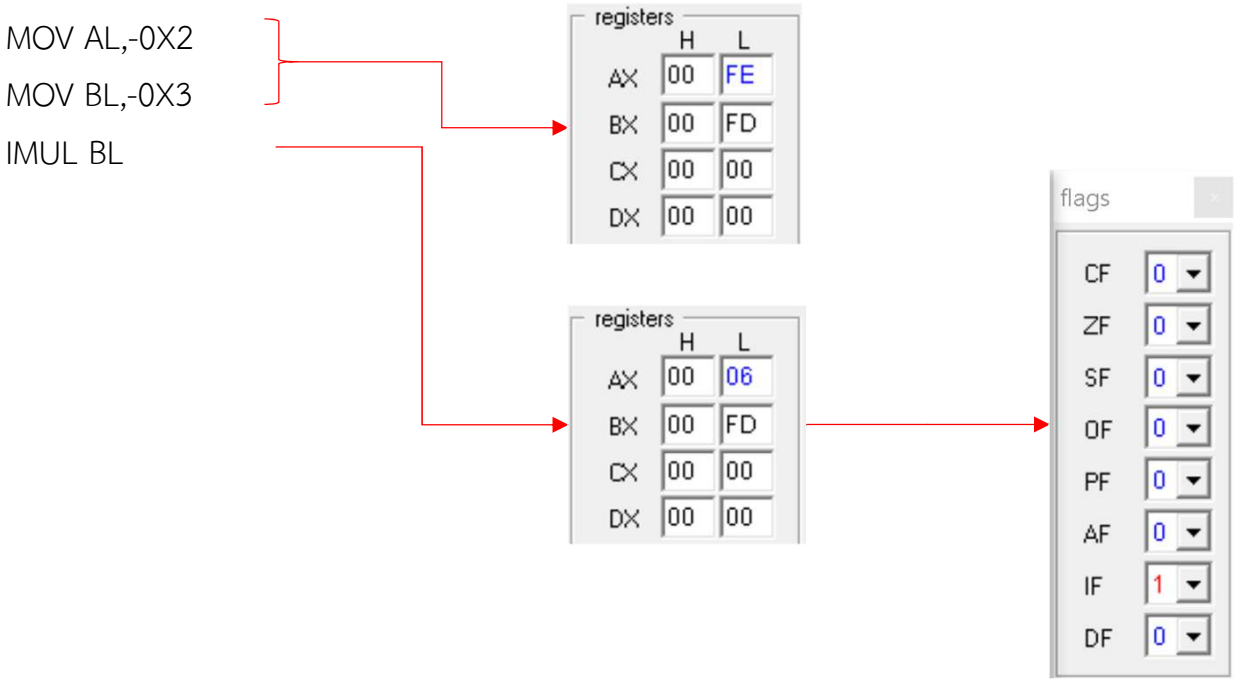
โปรแกรมเมอร์ต้องเช็คบิตแรกของคำตอบเองว่าเป็น 1 หรือไม่

ถ้าเป็น 1 ต้องทำ 2’s complement เพื่อให้ได้คำตอบสุดท้าย

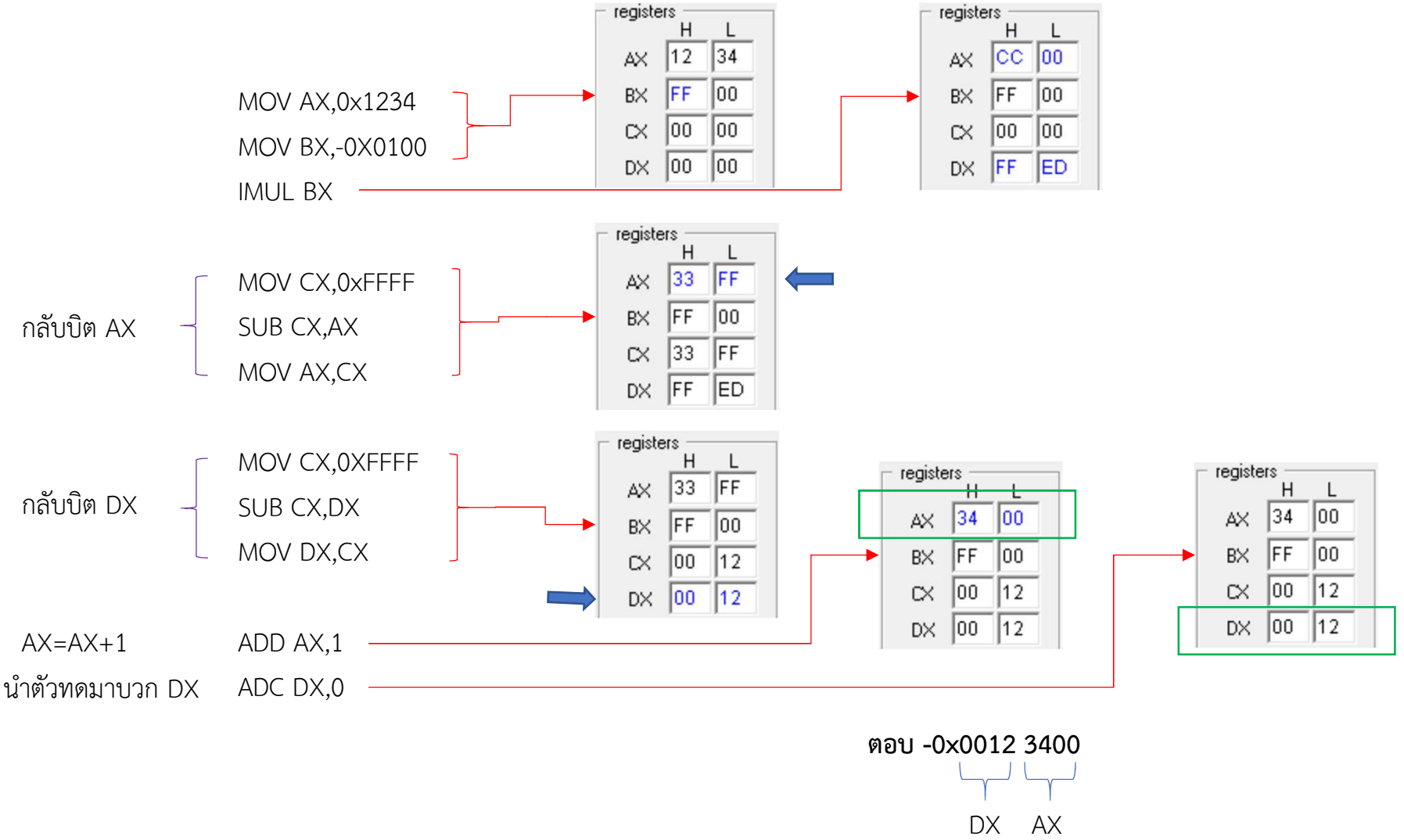
OPCODE	OPERAND	EXPLANATION	EXAMPLE
ADD	D, S	D = D + S	ADD AX, [2050]
ADC	D, S	D = D + S + prev. carry	ADC AX, BX
INC	D	D = D + 1	INC AX
SUB	D, S	D = D – S	SUB AX, [SI]
SBB	D, S	D = D – S – prev. carry	SBB [2050], 0050
DEC	D	D = D – 1	DEC [2050]
MUL	8-bit register	AX = AL * 8-bit reg.	MUL BH
MUL	16-bit register	DX AX = AX * 16-bit reg.	MUL CX
IMUL	8 or 16 bit register	performs signed multiplication	IMUL CX
DIV	8-bit register	AX = AX / 8-bit reg. ; AL = quotient ; AH = remainder	DIV BL
DIV	16-bit register	DX AX / 16-bit reg. ; AX = quotient ; DX = remainder	DIV CX
IDIV	8 or 16 bit register	performs signed division	IDIV BL
CBW	none	converts signed byte to word	CBW
CWD	none	converts signed word to double word	CWD
NEG	D	D = 2's compliment of D	NEG AL
DAA	none	decimal adjust accumulator	DAA
DAS	none	decimal adjust accumulator after subtraction	DAS
AAA	none	ASCII adjust accumulator after addition	AAA
AAS	none	ASCII adjust accumulator after subtraction	AAS
AAM	none	ASCII adjust accumulator after multiplication	AAM
AAD	none	ASCII adjust accumulator after division	AAD



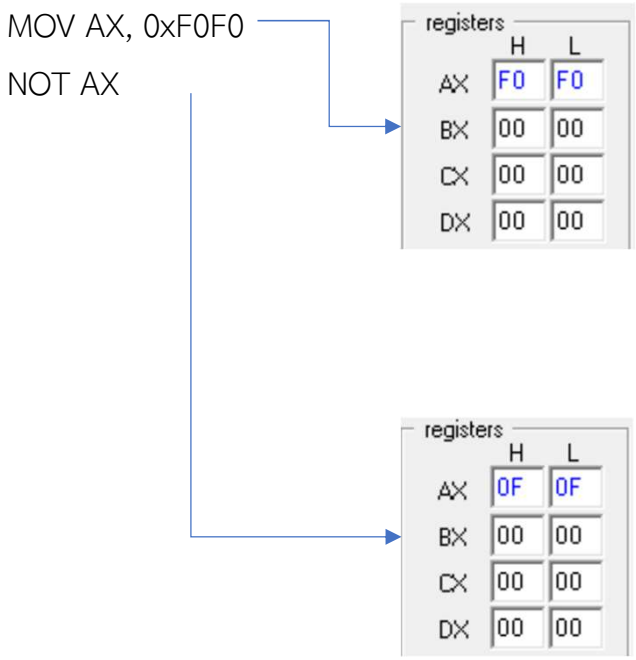
ถ้าคูณด้วย MUL คำตอบที่ AX จะมีค่าเป็น 01FC หลังจากกลับ 2's complement แล้วจะได้คำตอบเป็น FE04 ซึ่งไม่ถูกต้อง


MOV AL,-0X2
MOV BL,-0X3
IMUL BL
registers
H L
AX 00 FE
BX 00 FD
CX 00 00
DX 00 00
registers
H L
AX 00 06
BX 00 FD
CX 00 00
DX 00 00
flags
CF 0
ZF 0
SF 0
OF 0
PF 0
AF 0
IF 1
DF 0

การคูณเลข 16 บิตแบบคิดเครื่องหมาย คำตอบจะมีขนาด 32 บิต กระจายอยู่ในรีจิสเตอร์ AX และ DX หากคำตอบติดลบต้องทำ 2's complement แต่คำสั่ง NEG ทำงานได้แค่ 16 บิต ดังนั้น ต้องเขียนโปรแกรม กลับบิตแล้วบวก 1 เอง



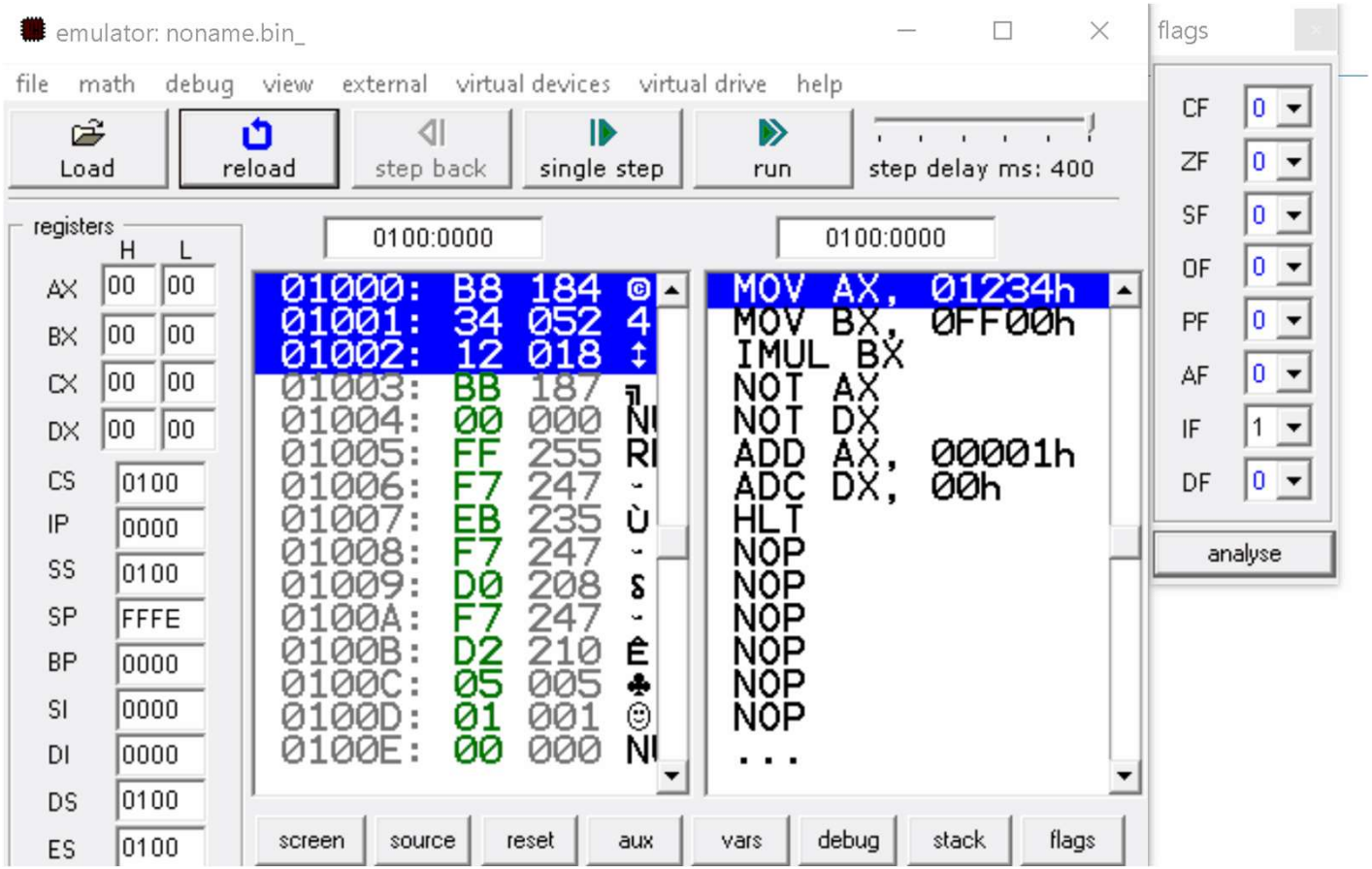
การกลับบิตสามารถใช้คำสั่ง NOT ได้ ช่วยให้โปรแกรมสั้นลง



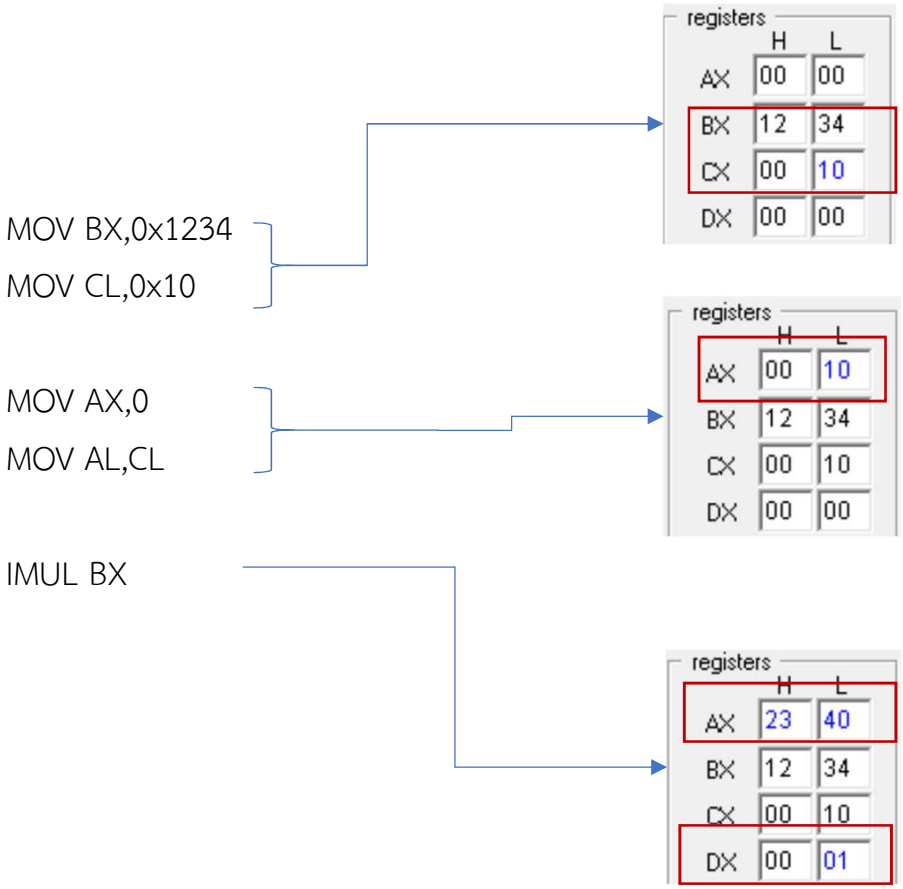

```
                MOV AX,0x1234
                MOV BX,-0X0100
                IMUL BX

กลับบิต AX      MOV CX,0xFFFF
                SUB CX,AX          NOT AX
                MOV AX,CX

กลับบิต DX      MOV CX,0XFFFF
                SUB CX,DX          NOT CX
                MOV DX,CX

AX=AX+1         ADD AX,1
นำตัวทดมาบวก DX  ADC DX,0
```

การกลับบิตสามารถใช้คำสั่ง NOT ได้ ช่วยให้โปรแกรมสั้นลง

```
MOV AX,0x1234
MOV BX,-0X0100
IMUL BX

NOT AX
NOT DX

ADD AX,1
ADC DX,0
```



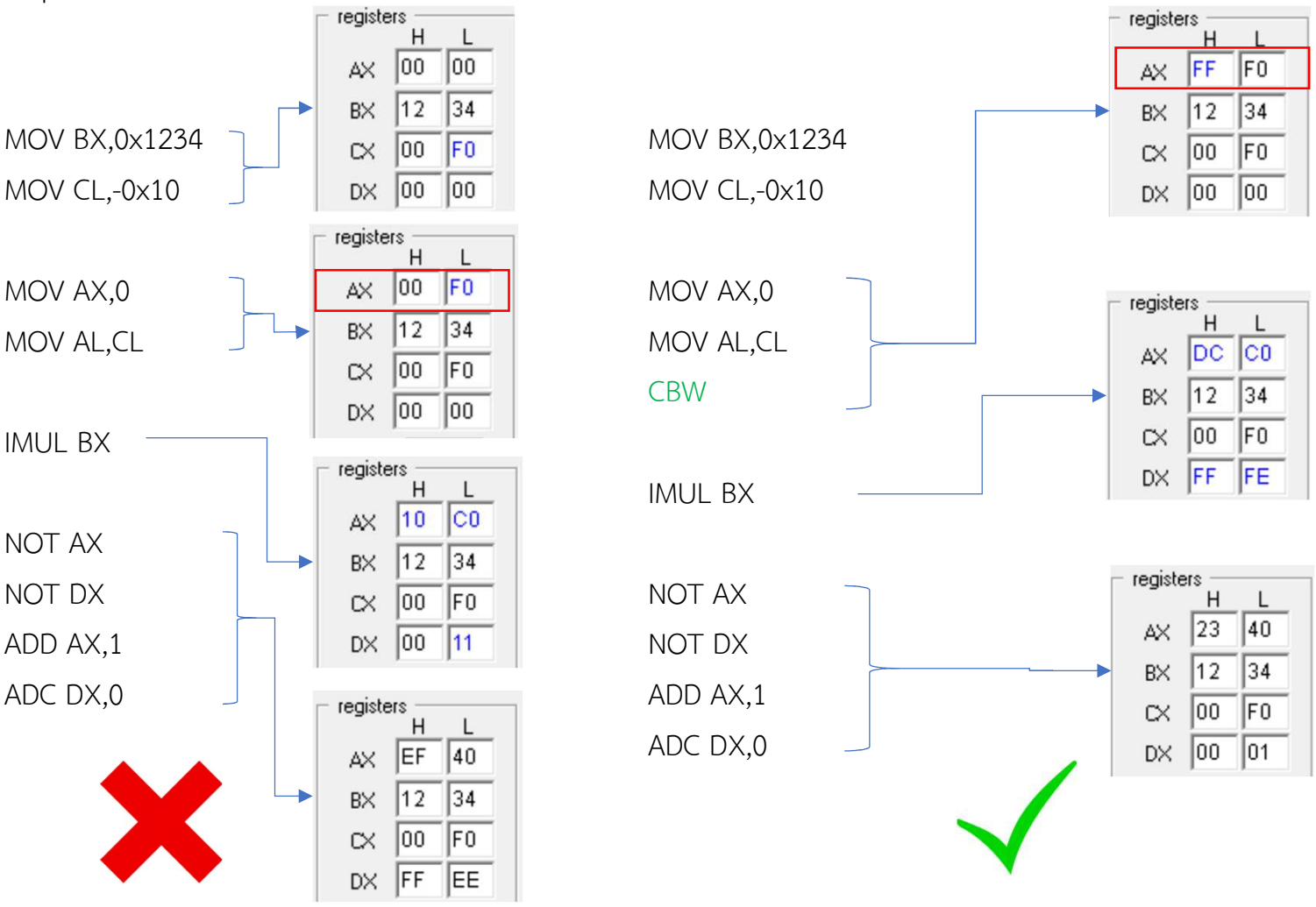
8086 ไม่มี Opcode ในการคูณเลขระหว่างรีจิสเตอร์ 16 บิตและ 8 แต่เราสามารถคูณได้ โดยการย้ายข้อมูล 8 บิตไปใส่ในรีจิสเตอร์ 16 บิตก่อน ตัวอย่างเช่น ต้องการคูณ BX และ CL



ถ้าจำนวนที่เก็บในรีจิสเตอร์ CL เป็นเลขติดลบ จะทำอย่างไร ? ตัวอย่างเช่น FF หรือ -1 หากย้ายขึ้น AX จะกลายเป็น 00FF ซึ่งจะมีความหมายเป็น 255 ถ้าต้องการให้รีจีสเตอร์ AX มีค่าเป็น -1 ต้องป้อน FFFF

OPCODE	OPERAND	EXPLANATION	EXAMPLE
ADD	D, S	D = D + S	ADD AX, [2050]
ADC	D, S	D = D + S + prev. carry	ADC AX, BX
INC	D	D = D + 1	INC AX
SUB	D, S	D = D – S	SUB AX, [SI]
SBB	D, S	D = D – S – prev. carry	SBB [2050], 0050
DEC	D	D = D – 1	DEC [2050]
MUL	8-bit register	AX = AL * 8-bit reg.	MUL BH
MUL	16-bit register	DX AX = AX * 16-bit reg.	MUL CX
IMUL	8 or 16 bit register	performs signed multiplication	IMUL CX
DIV	8-bit register	AX = AX / 8-bit reg. ; AL = quotient ; AH = remainder	DIV BL
DIV	16-bit register	DX AX / 16-bit reg. ; AX = quotient ; DX = remainder	DIV CX
IDIV	8 or 16 bit register	performs signed division	IDIV BL
CBW	none	converts signed byte to word	CBW
CWD	none	converts signed word to double word	CWD
NEG	D	D = 2's compliment of D	NEG AL
DAA	none	decimal adjust accumulator	DAA
DAS	none	decimal adjust accumulator after subtraction	DAS
AAA	none	ASCII adjust accumulator after addition	AAA
AAS	none	ASCII adjust accumulator after subtraction	AAS
AAM	none	ASCII adjust accumulator after multiplication	AAM
AAD	none	ASCII adjust accumulator after division	AAD

สั่ง CBW : converts signed byte to word ใช้ขยายตัวเลขแบบคิดเครื่องหมาย 8 บิตให้เป็นตัวเลข 16บิต
ทำงานกับรีจีสเตอร์ AX เท่านั้น โดยจะขยายจำนวนใน AL ให้เต็ม AX
ไม่มี Operand



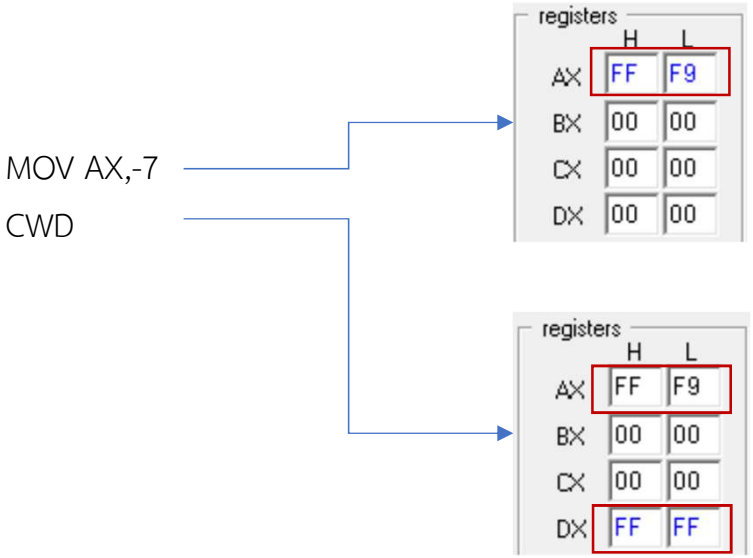
OPCODE	OPERAND	EXPLANATION	EXAMPLE
ADD	D, S	D = D + S	ADD AX, [2050]
ADC	D, S	D = D + S + prev. carry	ADC AX, BX
INC	D	D = D + 1	INC AX
SUB	D, S	D = D – S	SUB AX, [SI]
SBB	D, S	D = D – S – prev. carry	SBB [2050], 0050
DEC	D	D = D – 1	DEC [2050]
MUL	8-bit register	AX = AL * 8-bit reg.	MUL BH
MUL	16-bit register	DX AX = AX * 16-bit reg.	MUL CX
IMUL	8 or 16 bit register	performs signed multiplication	IMUL CX
DIV	8-bit register	AX = AX / 8-bit reg. ; AL = quotient ; AH = remainder	DIV BL
DIV	16-bit register	DX AX / 16-bit reg. ; AX = quotient ; DX = remainder	DIV CX
IDIV	8 or 16 bit register	performs signed division	IDIV BL
CBW	none	converts signed byte to word	CBW
CWD	none	converts signed word to double word	CWD
NEG	D	D = 2's compliment of D	NEG AL
DAA	none	decimal adjust accumulator	DAA
DAS	none	decimal adjust accumulator after subtraction	DAS
AAA	none	ASCII adjust accumulator after addition	AAA
AAS	none	ASCII adjust accumulator after subtraction	AAS
AAM	none	ASCII adjust accumulator after multiplication	AAM
AAD	none	ASCII adjust accumulator after division	AAD

หากต้องการขยายเลขแบบคิดเครื่องหมาย 16 บิต เป็น 32 บิต สามารถทำได้โดยใช้คำสั่ง

CWD : converts signed word to double word

คำสั่งนี้ไม่มี operand

การทำงานจะขยายจำนวน 16 บิตในรีจิสเตอร์ AX ไปยัง DX



FFF9 ถูกขยายเป็น FFFF FFF9

AX DX AX

การหารเลข

การหารเลขก็เหมือนกับการคูณเลข

มี 2 แบบ คือ แบบ 8 บิต และ 16 บิต

ทั้งสองแบบใช้ Mnemonic เดียวกันคือ

DIV สำหรับหารแบบไม่คิดเครื่องหมาย

IDIV สำหรับการหารแบบคิดเครื่องหมาย

DIV : Divide หารแบบ 8 บิต

การใช้ : DIV operand

การทำงาน AL = AX / operand

เศษที่เหลือจากการหารจะเก็บที่ AH

Operand ที่ใช้ได้:

REG8 บิต

Memory

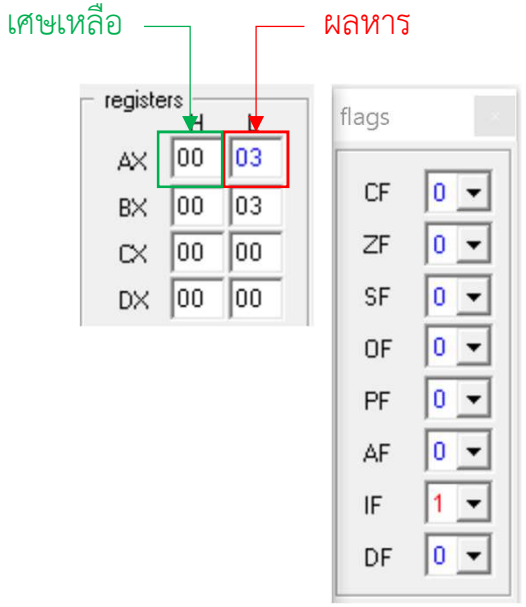
มีผลกับ Flag:

CF, OF

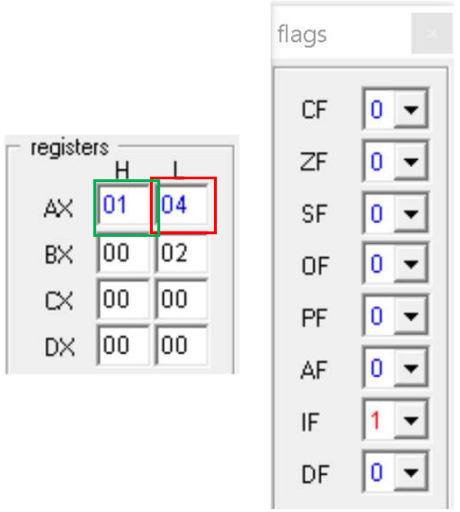
หากคำตอบมีขนาดเกิน operand CF และ OF จะเป็น 1

คำสั่งนี้ไม่พิจารณาเลขติดลบ จึงไม่มีผลกับ SF

```
MOV AL,0X09
MOV BL,0X03
DIV BL
```




```
MOV AL,0X09
MOV BL,0X02
DIV BL
```



DIV : Divide หารแบบ 16 บิต

การใช้ : DIV operand

การทำงาน AX = AX / operand

เศษที่เหลือจากการหารจะเก็บที่ DX

Operand ที่ใช้ได้:

REG16 บิต

Memory

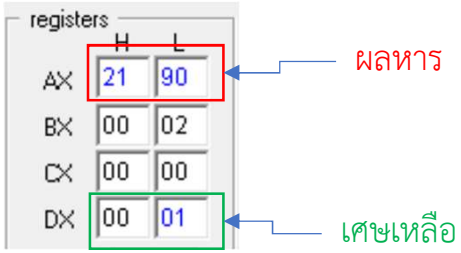
มีผลกับ Flag:

CF, OF

หากคำตอบมีขนาดเกิน operand CF และ OF จะเป็น 1

คำสั่งนี้ไม่พิจารณาเลขติดลบ จึงไม่มีผลกับ SF

```
MOV AX,0X4321
MOV BX,0X2
DIV BX
```



DIV : Divide หาร 32 บิต ด้วย 16 บิต

การใช้ : DIV operand

การทำงาน AX = DX:AX / operand

เศษที่เหลือจากการหารจะเก็บที่ DX

Operand ที่ใช้ได้:

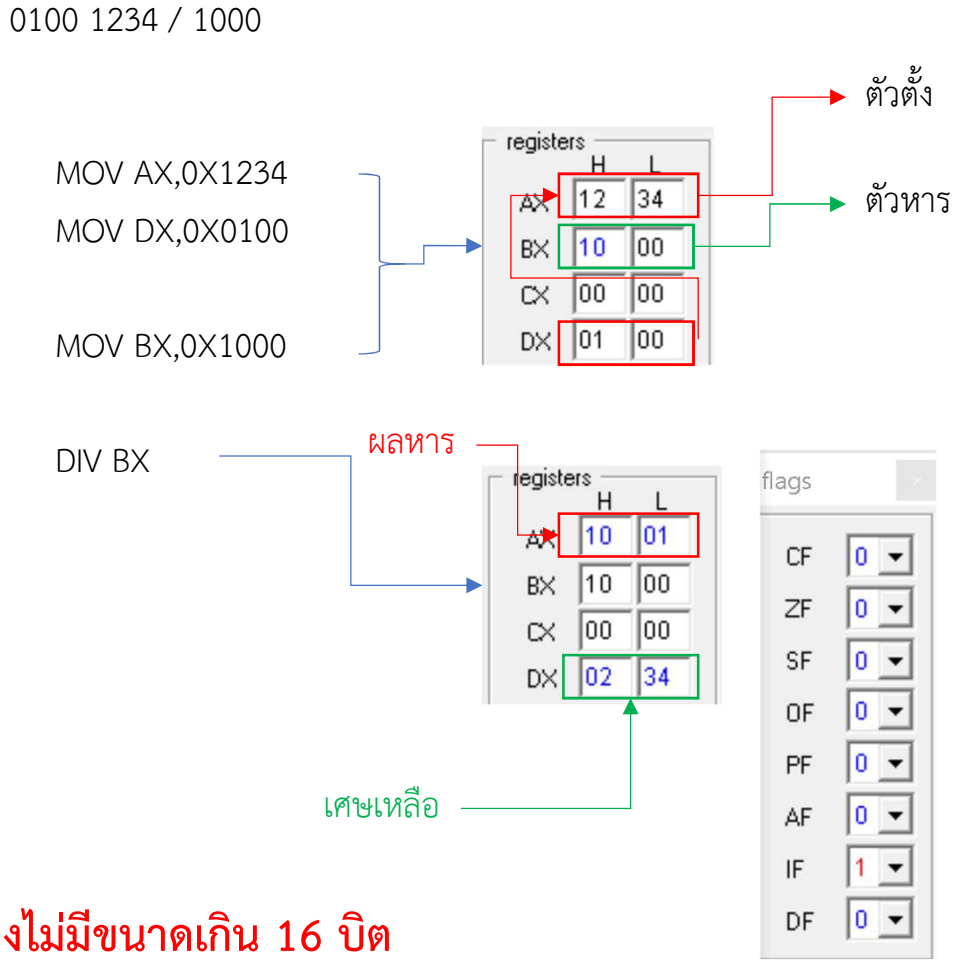
- REG16 บิต
- Memory

มีผลกับ Flag:

- CF, OF

หากคำตอบมีขนาดเกิน operand CF และ OF จะเป็น 1

คำสั่งนี้ไม่พิจารณาเลขติดลบ จึงไม่มีผลกับ SF



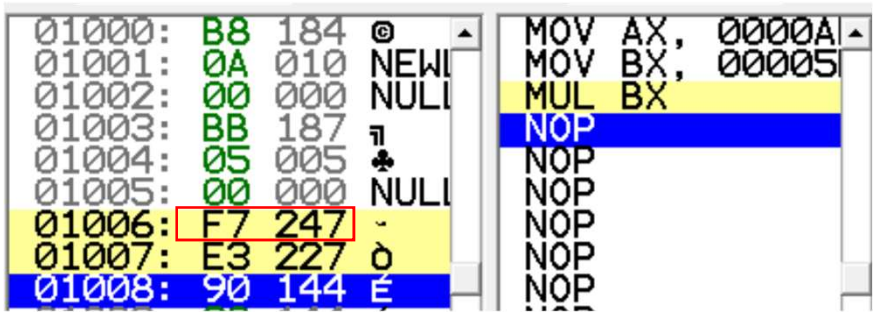
ผลหารต้องไม่มีขนาดเกิน 16 บิต

แล้วไมโครโพรเซสเซอร์รู้ได้อย่างไรว่า เราจะหารเลขเลขที่มีตัวตั้งเป็น 16 บิต หรือ 32 บิต
จะใช้เพียง AX เป็นตัวตั้งเพียงตัวเดียว หรือ ต้องใช้ทั้ง DX และ AX ?

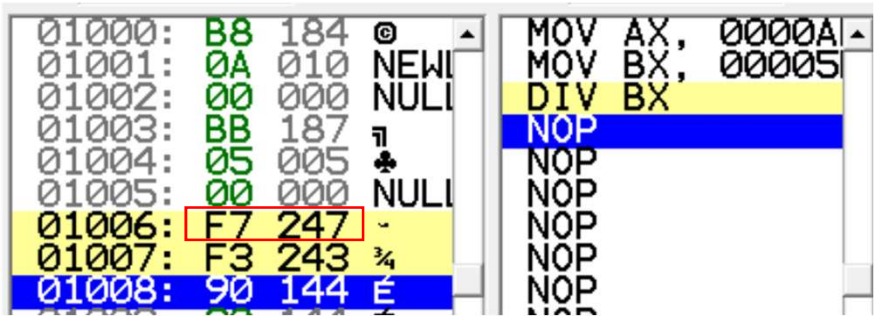
ไมโครโพรเซสเซอร์ ไม่รู้ เพราะทั้งสองใช้ Opcode เดียวกัน
จริง ๆแล้ว ทั้งคูณและหาร ใช้ Opcode เดียวกัน มีเพียงการแปล operand เป็นภาษาเครื่องแตกต่างกัน
ดังนั้นก่อนการหารเลขทุกครั้ง
ต้องทำให้ DX เป็น 0 ก่อนทุกครั้ง

MOV DX, 0

MOV AX,10
MOV BX,5
MUL BX



MOV AX,10
MOV BX,5
DIV BX

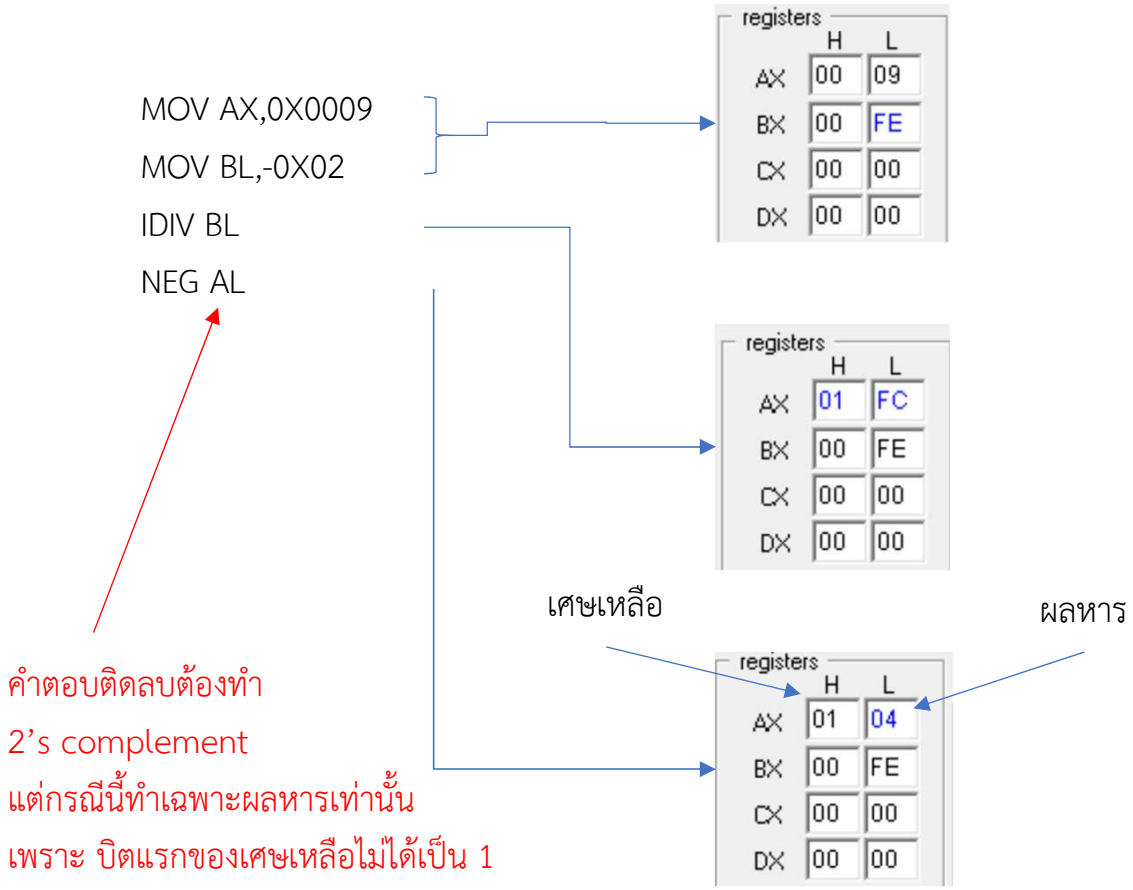


การหารเลขแบบคิดเครื่องหมายจะมีหลักแบบเดียวกับการคูณเลขคิดเครื่องหมาย

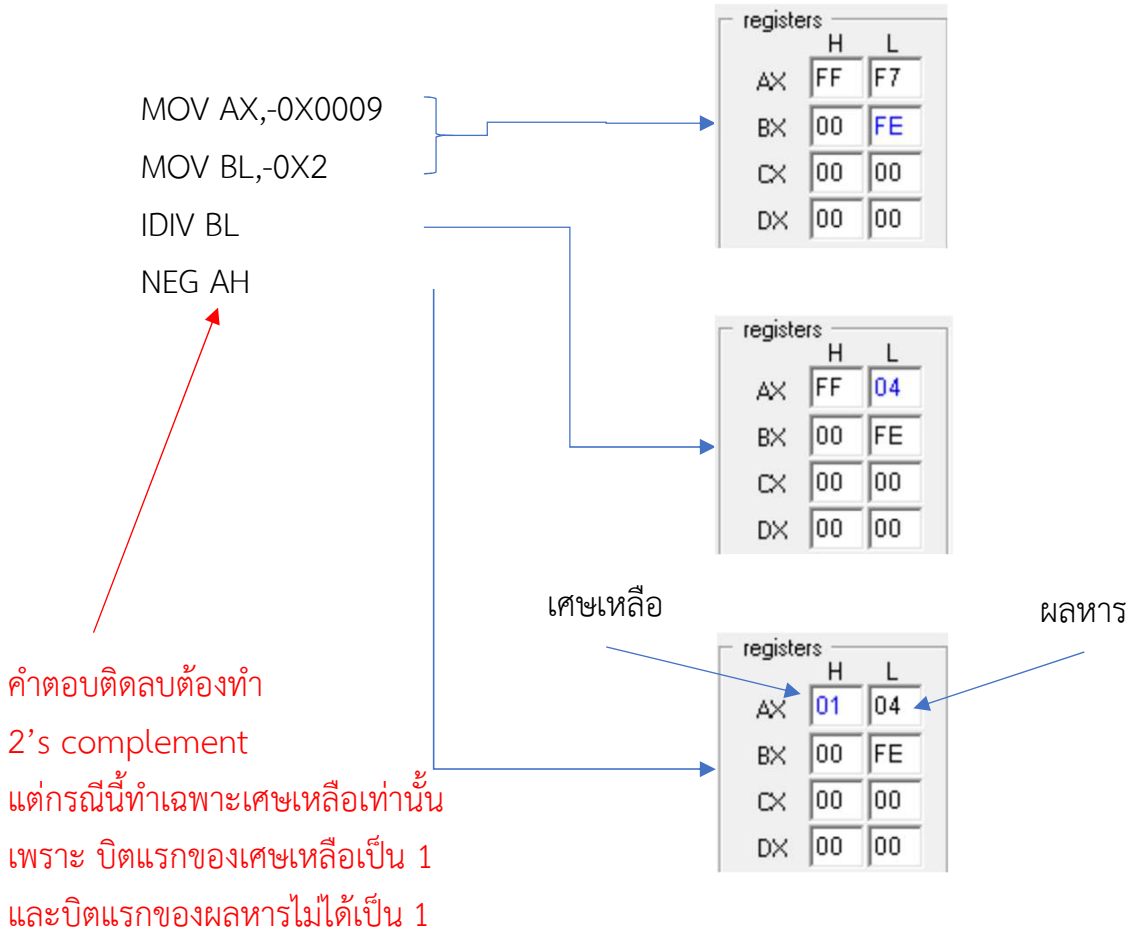
การหารแบบคิด เครื่องหมายใช้คำสั่ง

IDIV

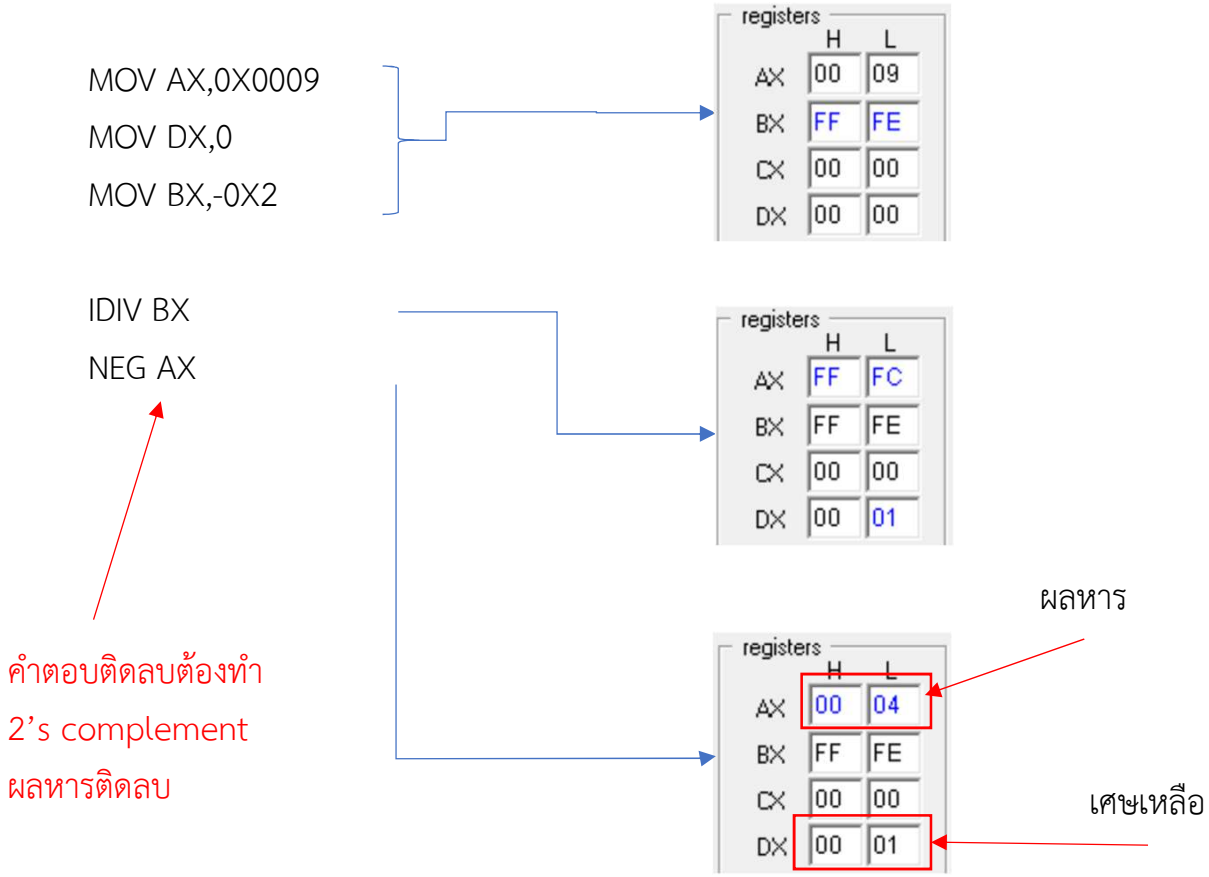
การหารเลข 0x0009 ที่มีขนาด 16 บิต ด้วย -0x02 ที่มีขนาด 8 บิต



การหารเลข -0x0009 ที่มีขนาด 16 บิต ด้วย -0x02 ที่มีขนาด 8 บิต

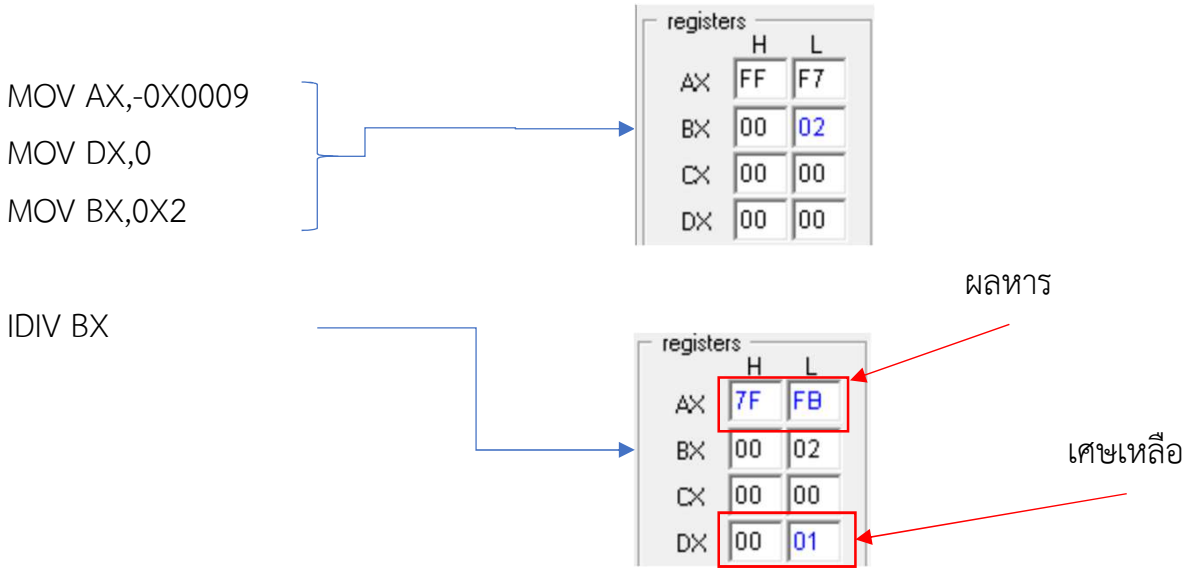


การหารเลข 0x0009 ที่มีขนาด 16 บิต ด้วย -0x02 ที่มีขนาด 16 บิต



จริง ๆโปรแกรมนี้มีตัวตั้งขนาด 32 บิต (รวมDX ด้วย) แต่ที่คำตอบถูกเพราะ ตัวตั้งไม่ติดลบ DX จึงมีค่าเป็น 0 พอดี

การหารเลข -0x0009 ที่มีขนาด 16 บิต ด้วย 0x02 ที่มีขนาด 16 บิต



โปรแกรมนี้ให้คำตอบเป็น 7FFB เศษ 1 ไม่ติดลบ เป็นคำตอบที่ไม่ถูกต้อง

เพราะไมโครโพรเซสเซอร์ เข้าใจว่าตัวตั้งเป็นเลข 32 บิต จึงนำ DX มารวมกับตัวตั้งเป็น 0000 FFF7 ซึ่งมีค่าเท่ากับ 65527 ในระบบเลขฐาน 10

ดังนั้นหากตัวหารมีขนาด 16 บิต ตั้งทำการขยายตัวตั้งให้เป็นเลข 32 บิตด้วย

การขยายเลขแบบคิดเครื่องหมายจาก 16 บิตเป็น 32 บิตจะใช้คำสั่ง

CWD: converts signed word to double word

การทำงานจะขยายค่าจาก AX ไปยัง DX:AX

คำสั่งนี้ไม่มี operand

การหารเลข -0x0009 ที่มีขนาด 16 บิต

ด้วย 0x02 ที่มีขนาด 16 บิต

MOV AX,-0X0009

MOV DX,0

MOV BX,0X2

CWD

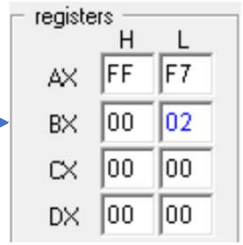
IDIV BX

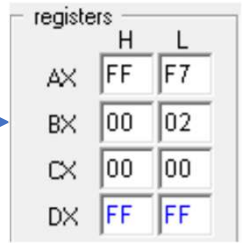
NEG AX

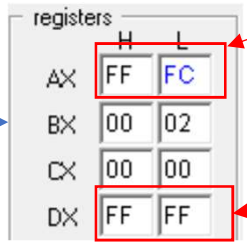
NEG DX

ติดลบทั้งผลหารและเศษเหลือ

ต้องทำ 2's complement ทั้งคู่

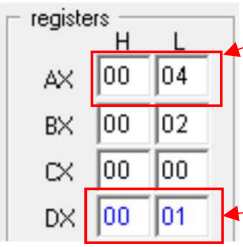






ผลหาร ติดลบ

เศษเหลือ ติดลบ



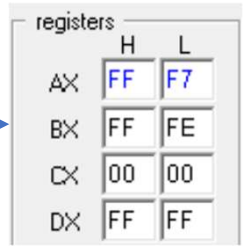
ผลหาร สุดท้าย

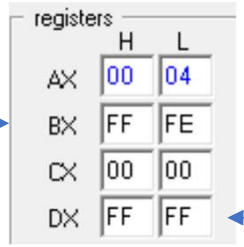
เศษเหลือ สุดท้าย

การหารเลข -0x0009 ที่มีขนาด 16 บิต
ด้วย -0x02 ที่มีขนาด 16 บิต

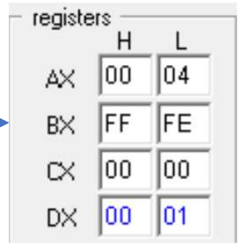
MOV AX,-0X0009
MOV DX,0
MOV BX,-0X2
CWD

IDIV BX
NEG DX





เศษเหลือติดลบ ต้องทำ 2's complement



OPCODE	OPERAND	EXPLANATION	EXAMPLE
ADD	D, S	D = D + S	ADD AX, [2050]
ADC	D, S	D = D + S + prev. carry	ADC AX, BX
INC	D	D = D + 1	INC AX
SUB	D, S	D = D – S	SUB AX, [SI]
SBB	D, S	D = D – S – prev. carry	SBB [2050], 0050
DEC	D	D = D – 1	DEC [2050]
MUL	8-bit register	AX = AL * 8-bit reg.	MUL BH
MUL	16-bit register	DX AX = AX * 16-bit reg.	MUL CX
IMUL	8 or 16 bit register	performs signed multiplication	IMUL CX
DIV	8-bit register	AX = AX / 8-bit reg. ; AL = quotient ; AH = remainder	DIV BL
DIV	16-bit register	DX AX / 16-bit reg. ; AX = quotient ; DX = remainder	DIV CX
IDIV	8 or 16 bit register	performs signed division	IDIV BL
CBW	none	converts signed byte to word	CBW
CWD	none	converts signed word to double word	CWD
NEG	D	D = 2's compliment of D	NEG AL
DAA	none	decimal adjust accumulator	DAA
DAS	none	decimal adjust accumulator after subtraction	DAS
AAA	none	ASCII adjust accumulator after addition	AAA
AAS	none	ASCII adjust accumulator after subtraction	AAS
AAM	none	ASCII adjust accumulator after multiplication	AAM
AAD	none	ASCII adjust accumulator after division	AAD

ในบทนี้เป็นการใช้ไมโครโพรเซสเซอร์ทำงานเป็นเครื่องคิดเลข เหมือนเป็นการป้อนคำสั่งให้ ทำงานทีละคำสั่ง
ผู้ใช้ต้องทำการเช็ค Flag ดูว่ามีตัวทดตัวยืม ติดลบ หรือค่าล้น
ในกรณีของการคูณและหาร ต้องคอยตรวจสอบบิตแรกเอง ว่าต้องทำ 2's complement หรือไม่
ในบทต่อไปเราจะศึกษาส่วนของการเปรียบเทียบทางตรรกศาสตร์ และการกระทำตามเงื่อนไข
เพื่อกลับมาโปรแกรมเรื่องคิดเลขให้ทำงานได้อย่างอัตโนมัติโดยสมบูรณ์