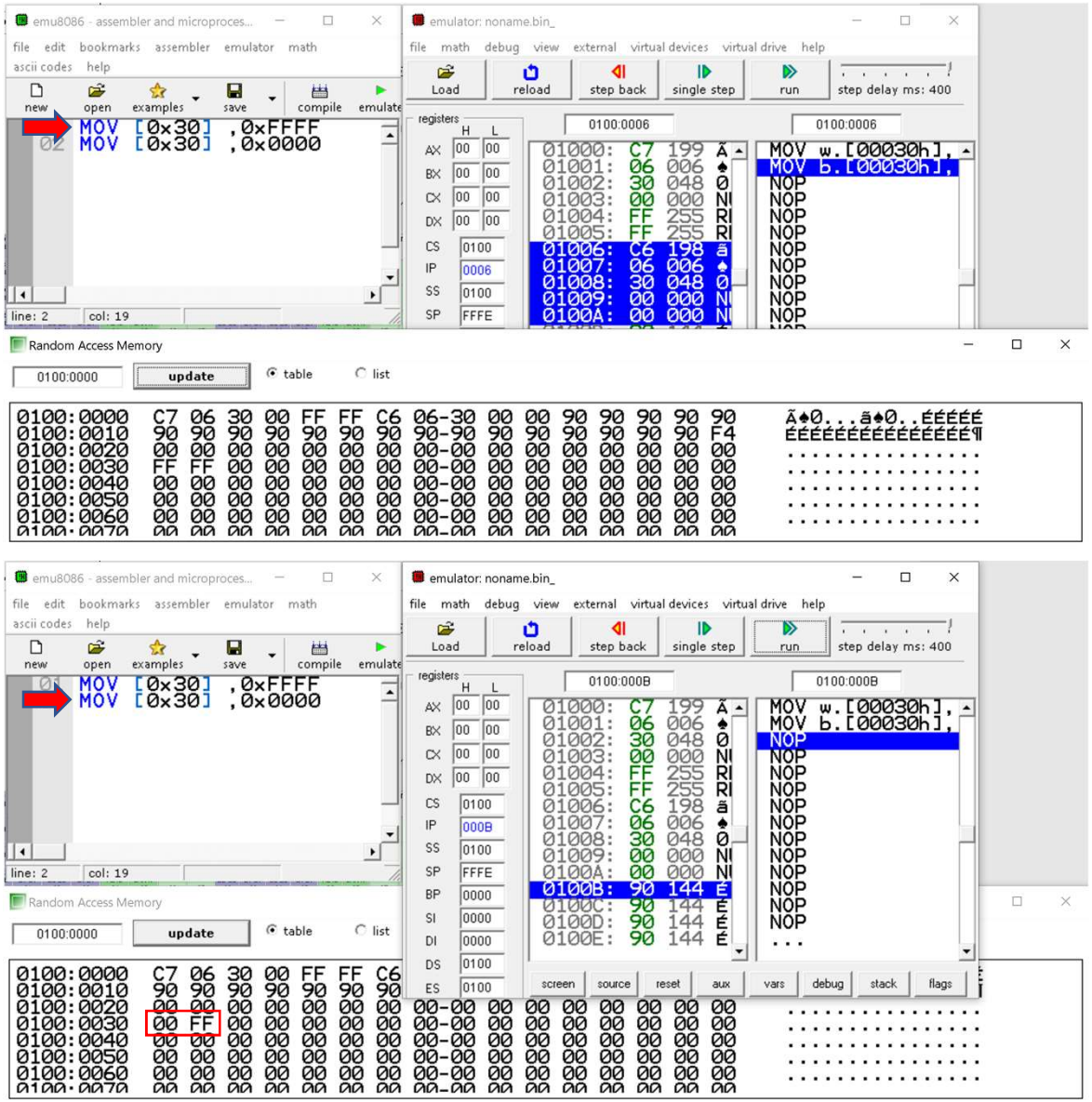
CS231 ระบบคอมพิวเตอร์และภาษาแอสแซมบลี

บทที่ 3: ข้อมูลและค่าคงที่

ผู้ช่วยศาสตราจารย์ ดร. ปวีณ เขื่อนแก้ว

สาขาวิชาวิทยาการคอมพิวเตอร์

คณะวิทยาศาสตร์ มหาวิทยาลัยแม่โจ้



เพราะอะไร หน่วยความจำจึงไม่บันทึก

00 00

ข้อมูลที่ไมโครโพรเซสเซอร์เข้าใจนั้นมีเพียงเลขฐาน 2 เท่านั้น เพื่อให้สามารถโปรแกรมได้ง่าย จึงได้รวมตัวเลขหลาย ๆ หลัก (บิต) เข้าไว้เป็นกลุ่ม

จำนวนบิต	ชื่อเรียก	หมายเหตุ
1	Bit	8086 ไม่รองรับ
8	Byte (b)	
16	Word (v, w)	
32	Doubleword	
64	Quadword	8086 ไม่รองรับ
80	Ten Byte	8086 ไม่รองรับ

ตาราง opcode ของไมโครโพรเซสเซอร์ X86

ADD Eb Gb 00	ADD Ev Gv 01	ADD Gb Eb 02	ADD Gv Ev 03	ADD AL Ib 04	ADD eAX Iv 05	PUSH ES 06	POP ES 07	OR Eb Gb 08	OR Ev Gv 09	OR Gb Eb 0A	OR Gv Ev 0B	OR AL Ib 0C	OR eAX Iv 0D	PUSH CS 0E	TWOBYTE 0F
ADC Eb Gb 10	ADC Ev Gv 11	ADC Gb Eb 12	ADC Gv Ev 13	ADC AL Ib 14	ADC eAX Iv 15	PUSH SS 16	POP SS 17	SBB Eb Gb 18	SBB Ev Gv 19	SBB Gb Eb 1A	SBB Gv Ev 1B	SBB AL Ib 1C	SBB eAX Iv 1D	PUSH DS 1E	POP DS 1F
AND Eb Gb 20	AND Ev Gv 21	AND Gb Eb 22	AND Gv Ev 23	AND AL Ib 24	AND eAX Iv 25	ES: 26	DAA 27	SUB Eb Gb 28	SUB Ev Gv 29	SUB Gb Eb 2A	SUB Gv Ev 2B	SUB AL Ib 2C	SUB eAX Iv 2D	CS: 2E	DAS 2F
XOR Eb Gb 30	XOR Ev Gv 31	XOR Gb Eb 32	XOR Gv Ev 33	XOR AL Ib 34	XOR eAX Iv 35	SS: 36	AAA 37	CMP Eb Gb 38	CMP Ev Gv 39	CMP Gb Eb 3A	CMP Gv Ev 3B	CMP AL Ib 3C	CMP eAX Iv 3D	DS: 3E	AAS 3F
INC eAX 40	INC eCX 41	INC eDX 42	INC eBX 43	INC eSP 44	INC eBP 45	INC eSI 46	INC eDI 47	DEC eAX 48	DEC eCX 49	DEC eDX 4A	DEC eBX 4B	DEC eSP 4C	DEC eBP 4D	DEC eSI 4E	DEC eDI 4F
PUSH eAX 50	PUSH eCX 51	PUSH eDX 52	PUSH eBX 53	PUSH eSP 54	PUSH eBP 55	PUSH eSI 56	PUSH eDI 57	POP eAX 58	POP eCX 59	POP eDX 5A	POP eBX 5B	POP eSP 5C	POP eBP 5D	POP eSI 5E	POP eDI 5F
PUSHA 60	POPA 61	BOUND Gv Ma 62	ARPL Ew Gw 63	FS: 64	GS: 65	OPSIZE: 66	ADSIZE: 67	PUSH Iv 68	IMUL Gv Ev Iv 69	PUSH Ib 6A	IMUL Gv Ev Ib 6B	INSB Yb DX 6C	INSW Yz DX 6D	OUTSB DX Xb 6E	OUTSW DX Xv 6F
JO Jb 70	JNO Jb 71	JB Jb 72	JNB Jb 73	JZ Jb 74	JNZ Jb 75	JBE Jb 76	JA Jb 77	JS Jb 78	JNS Jb 79	JP Jb 7A	JNP Jb 7B	JL Jb 7C	JNL Jb 7D	JLE Jb 7E	JNLE Jb 7F

ADD Eb Ib 80	ADD Ev Iv 81	SUB Eb Ib 82	SUB Ev Ib 83	TEST Eb Gb 84	TEST Ev Gv 85	XCHG Eb Gb 86	XCHG Ev Gv 87	MOV Eb Gb 88	MOV Ev Gv 89	MOV Gb Eb 8A	MOV Gv Ev 8B	MOV Ew Sw 8C	LEA Gv M 8D	MOV Sw Ew 8E	POP Ev 8F
NOP 90	XCHG eAX eCX 91	XCHG eAX eDX 92	XCHG eAX eBX 93	XCHG eAX eSP 94	XCHG eAX eBP 95	XCHG eAX eSI 96	XCHG eAX eDI 97	CBW 98	CWD 99	CALL Ap 9A	WAIT 9B	PUSHF Fv 9C	POPF Fv 9D	SAHF 9E	LAHF 9F
MOV AL Ob A0	MOV eAX Ov A1	MOV Ob AL A2	MOV Ov eAX A3	MOVSB Xb Yb A4	MOVSW Xv Yv A5	CMPSB Xb Yb A6	CMPSW Xv Yv A7	TEST AL Ib A8	TEST eAX Iv A9	STOSB Yb AL AA	STOSW Yv eAX AB	LODSB AL Xb AC	LODSW eAX Xv AD	SCASB AL Yb AE	SCASW eAX Yv AF
MOV AL Ib B0	MOV CL Ib B1	MOV DL Ib B2	MOV BL Ib B3	MOV AH Ib B4	MOV CH Ib B5	MOV DH Ib B6	MOV BH Ib B7	MOV eAX Iv B8	MOV eCX Iv B9	MOV eDX Iv BA	MOV eBX Iv BB	MOV eSP Iv BC	MOV eBP Iv BD	MOV eSI Iv BE	MOV eDI Iv BF
#2 Eb Ib C0	#2 Ev Ib C1	RETN Iw C2	RETN C3	LES Gv Mp C4	LDS Gv Mp C5	MOV Eb Ib C6	MOV Ev Iv C7	ENTER Iw Ib C8	LEAVE C9	RETF Iw CA	RETF CB	INT3 CC	INT Ib CD	INTO CE	IRET CF
#2 Eb 1 D0	#2 Ev 1 D1	#2 Eb CL D2	#2 Ev CL D3	AAM Ib D4	AAD Ib D5	SALC D6	XLAT D7	ESC 0 D8	ESC 1 D9	ESC 2 DA	ESC 3 DB	ESC 4 DC	ESC 5 DD	ESC 6 DE	ESC 7 DF
LOOPNZ Jb E0	LOOPZ Jb E1	LOOP Jb E2	JCXZ Jb E3	IN AL Ib E4	IN eAX Ib E5	OUT Ib AL E6	OUT Ib eAX E7	CALL Jz E8	JMP Jz E9	JMP Ap EA	JMP Jb EB	IN AL DX EC	IN eAX DX ED	OUT DX AL EE	OUT DX eAX EF
LOCK: F0	INT1 F1	REPNE: F2	REP: F3	HLT F4	CMC F5	#3 Eb F6	#3 Ev F7	CLC F8	STC F9	CLI FA	STI FB	CLD FC	STD FD	#4 INC/DEC FE	#5 INC/DEC FF

MOV [0x30], 0xFFFF ถูก Assembler แปลเป็น opcode C7
เพราะ 0xFFFF มีขนาด 16 บิต

MOV [0x30], 0x0 ถูก Assembler แปลเป็น opcode C6
เพราะ 0 สามารถเก็บด้วยพื้นที่เพียง 8 บิตได้

Eb = ตำแหน่งหน่วยความจำของ Byte

Ib = Immediate Byte

*char ในภาษา C

Ev = ตำแหน่งหน่วยความจำของ Word

Iv = Immediate Word

*short ในภาษา C

เราสามารถบังคับ Assembler ให้แปล

MOV [0x30], 0x0

ด้วย opcode C7 ได้

โดยการใช้ คำสั่งชี้แนะ กับโปรแกรม Assembler

คำสั่งชี้แนะ หรือ Directive คือคำสั่งที่ Assembler เข้าใจ ใช้กับ Assembler เท่านั้น

ไมโครโพรเซสเซอร์ ไม่สามารถเข้าใจ Directive ได้

Directive ≠ Opcode

8086 Assembler Directive

(a) The DB directive
(b) The DW directive
(c) The DD directive
(d) The STRUCT (or STRUC) and ENDS directives
(counted as one)
(e)The EQU Directive
(f)The COMMENT directive
(g)ASSUME
(h) EXTERN
(i) GLOBAL
(j) SEGMENT
(k)OFFSET
(l) PROC
(m)GROUP
(n) INCLUDE

8086 Assembler Directive

(a) The DB directive
(b) The DW directive
(c) The DD directive
(d) The STRUCT (or STRUC) and ENDS directives (counted as one)
(e)The EQU Directive
(f)The COMMENT directive
(g)ASSUME
(h) EXTERN
(i) GLOBAL
(j) SEGMENT
(k)OFFSET
(l) PROC
(m)GROUP
(n) INCLUDE

8086 Assembler Directive

การนิยามข้อมูลลงไปในโปรแกรมโดยตรง สามารถทำได้โดย

DB = Byte

DW = Word

DD = Double word

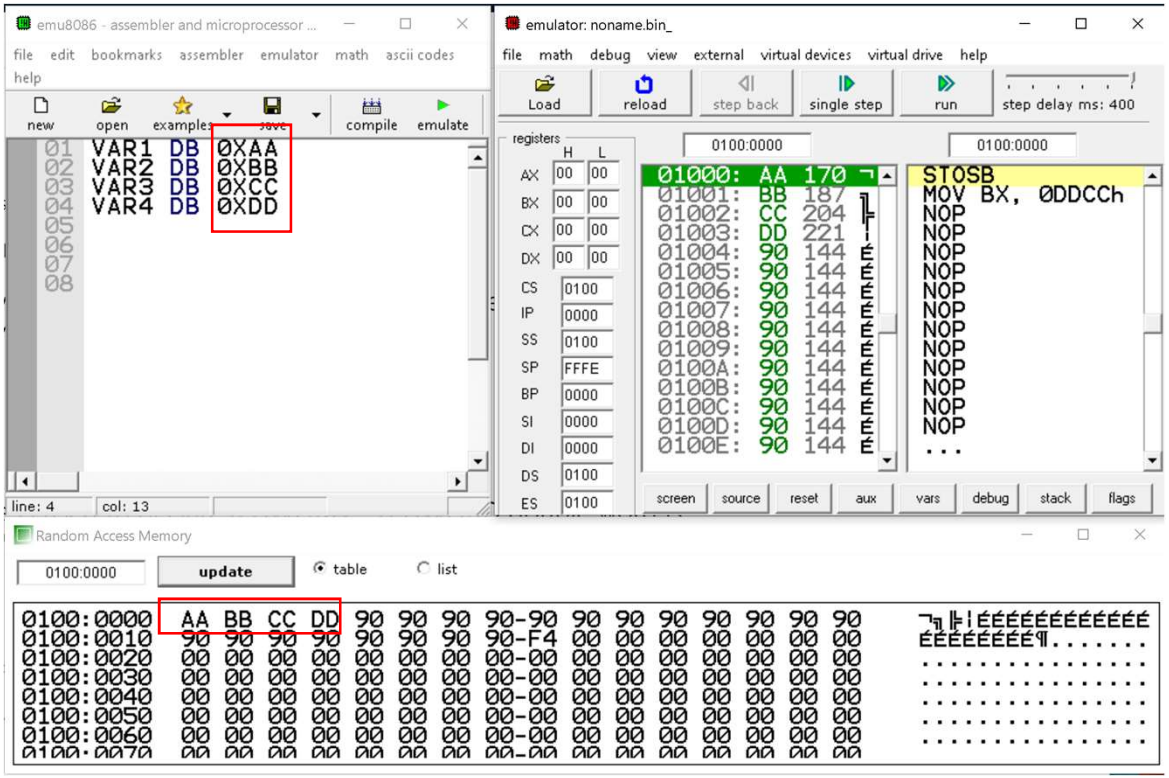
Directive นี้สามารถทำงานได้เหมือนตัวแปรตัวอย่างเช่น

การใช้งาน

Label Directive Data

ตัวอย่างเช่น

VAR1 DB 0xAA



ข้อมูลจะอยู่ในหน่วยความจำทันที โดยไม่ต้องรันโปรแกรม

เมื่อมีการอ้างถึง Label จะได้ข้อมูลตำแหน่งในหน่วยความจำ

8086 Assembler Directive

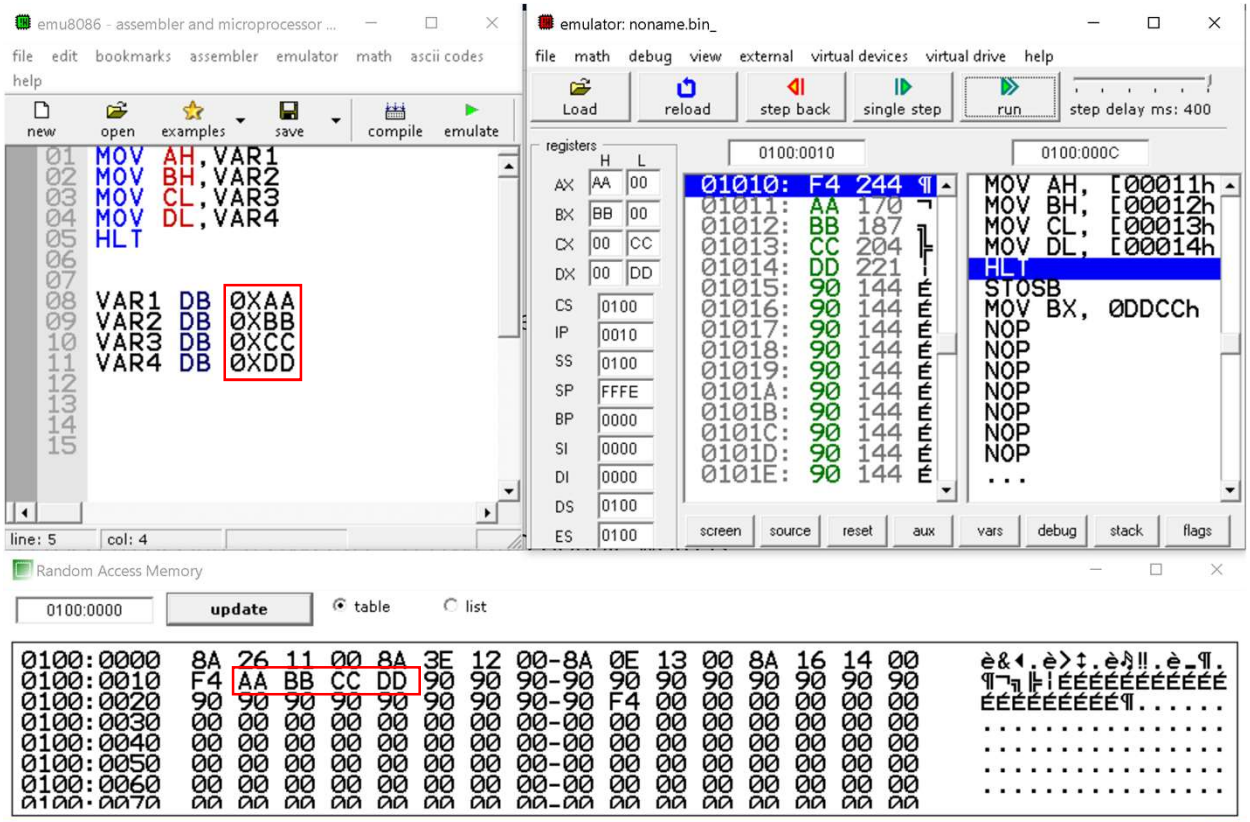
การนิยามข้อมูลลงไปในโปรแกรมโดยตรง สามารถทำได้โดย

DB = Byte

DW = Word

DD = Double word

เนื่องจากข้อมูลที่นิยาม ไม่ใช่โปรแกรมจึงต้องนิยามไว้ด้านล่างโปรแกรม และใช้คำสั่ง HLT เพื่อให้โปรแกรมหยุดทำงาน ก่อนโปรแกรมจะวิ่งมาถึงส่วนที่เป็นข้อมูล



คำถาม Label ที่เป็น operand ของคำสั่ง MOV คือ operand ในข้อใด

1 REG

2 SREG

3 IMMEDIATE

4 MEMORY

8086 Assembler Directive

การนิยามข้อมูลลงไปในโปรแกรมโดยตรง สามารถทำได้โดย

DB = Byte

DW = Word

DD = Double word

คำถาม Label ที่เป็น operand ของคำสั่ง MOV คือ operand ในข้อใด

1 REG

2 SREG

3 IMMEDIATE

4 MEMORY

ลองเขียนดู โปรแกรมนี้จะรันไม่ได้

```
MOV [0x30],VAR1
HLT
VAR1 DB 0XAA
```

เพราะ 8086 ไม่มี Opcode คำสั่ง MOV MEMORY , MEMORY

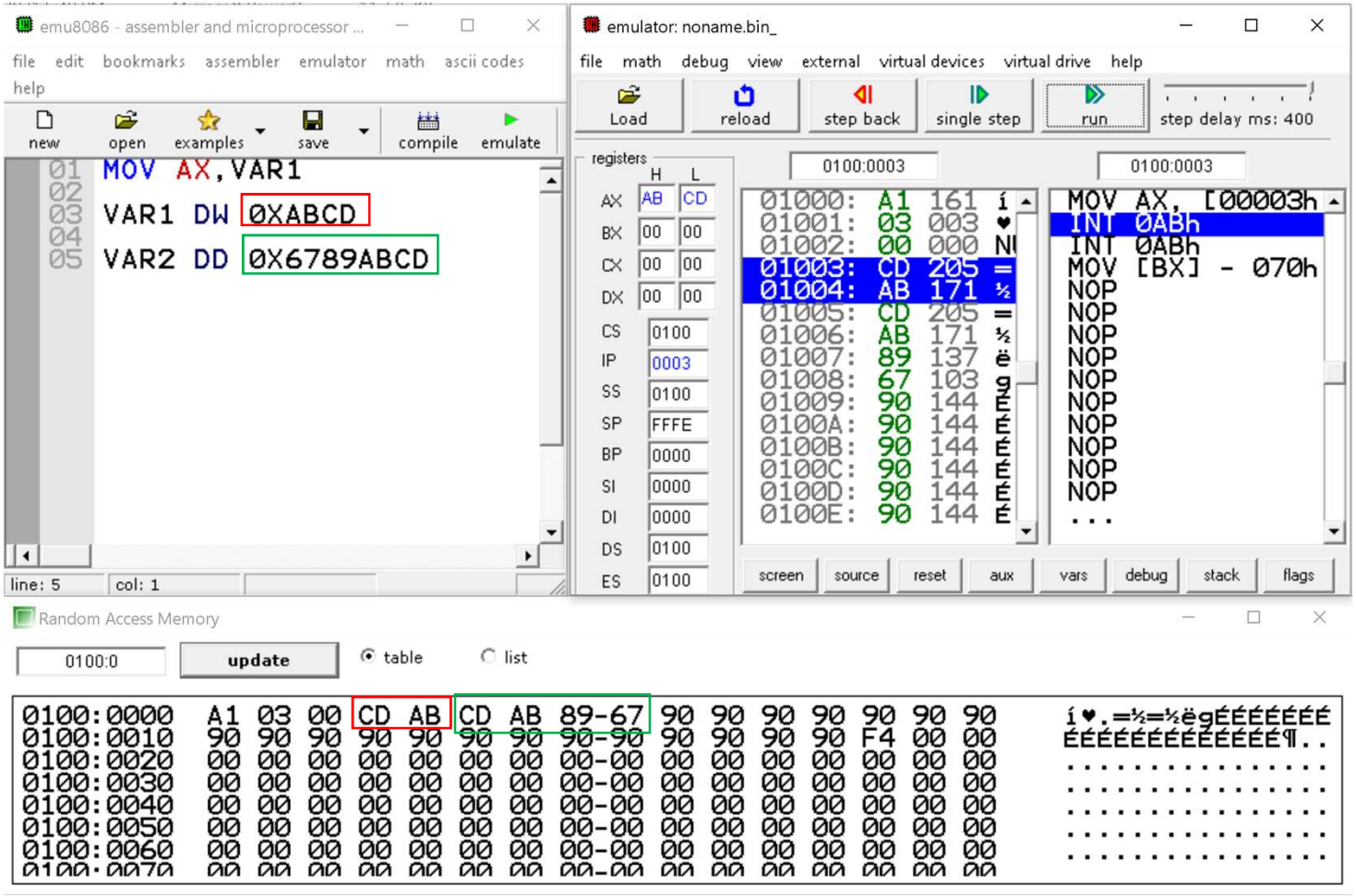
8086 Assembler Directive

การนิยามข้อมูลลงไปในโปรแกรมโดยตรง สามารถทำได้โดย

DB = Byte

DW = Word

DD = Double word



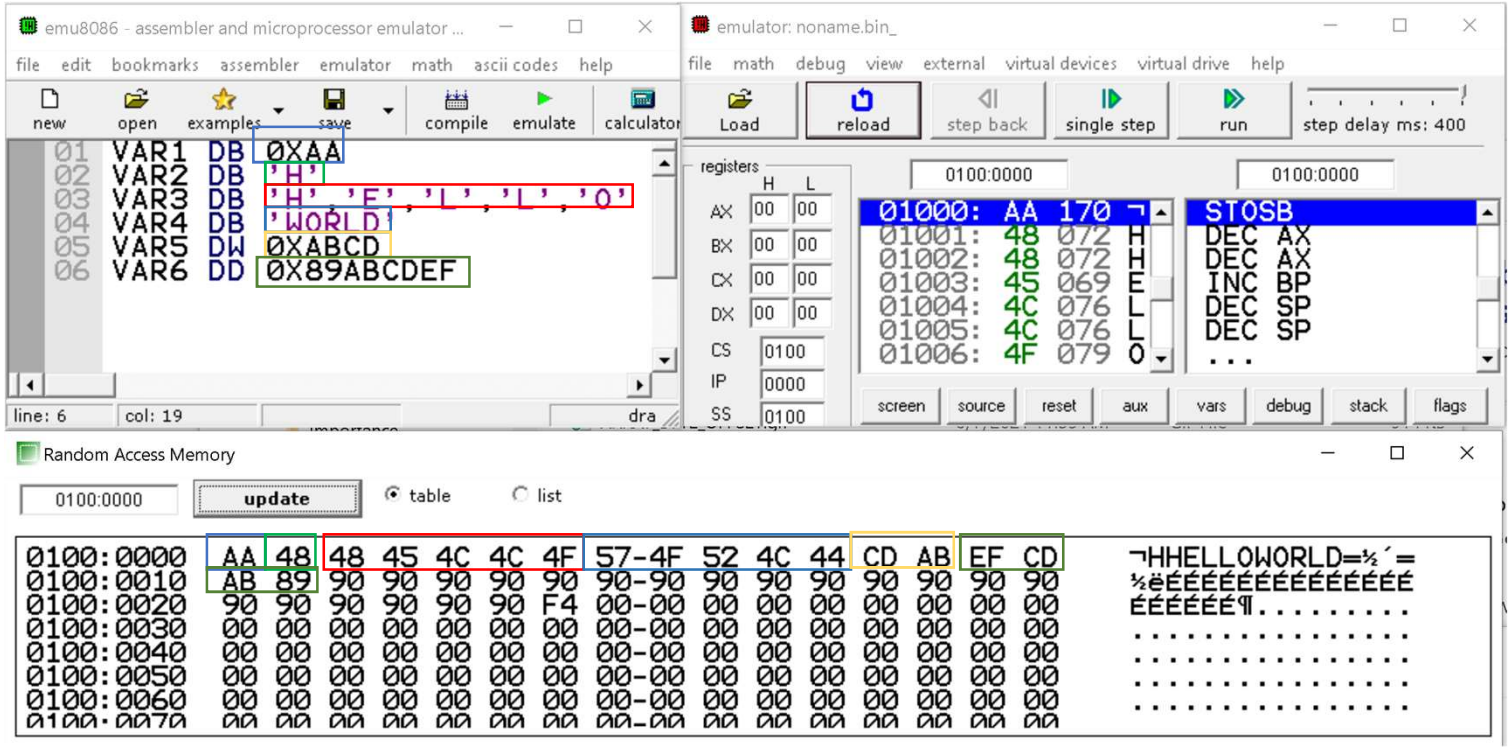
8086 Assembler Directive

การนิยามข้อมูลลงไปในโปรแกรมโดยตรง สามารถทำได้โดย

DB = Byte

DW = Word

DD = Double word



8086 Assembler Directive

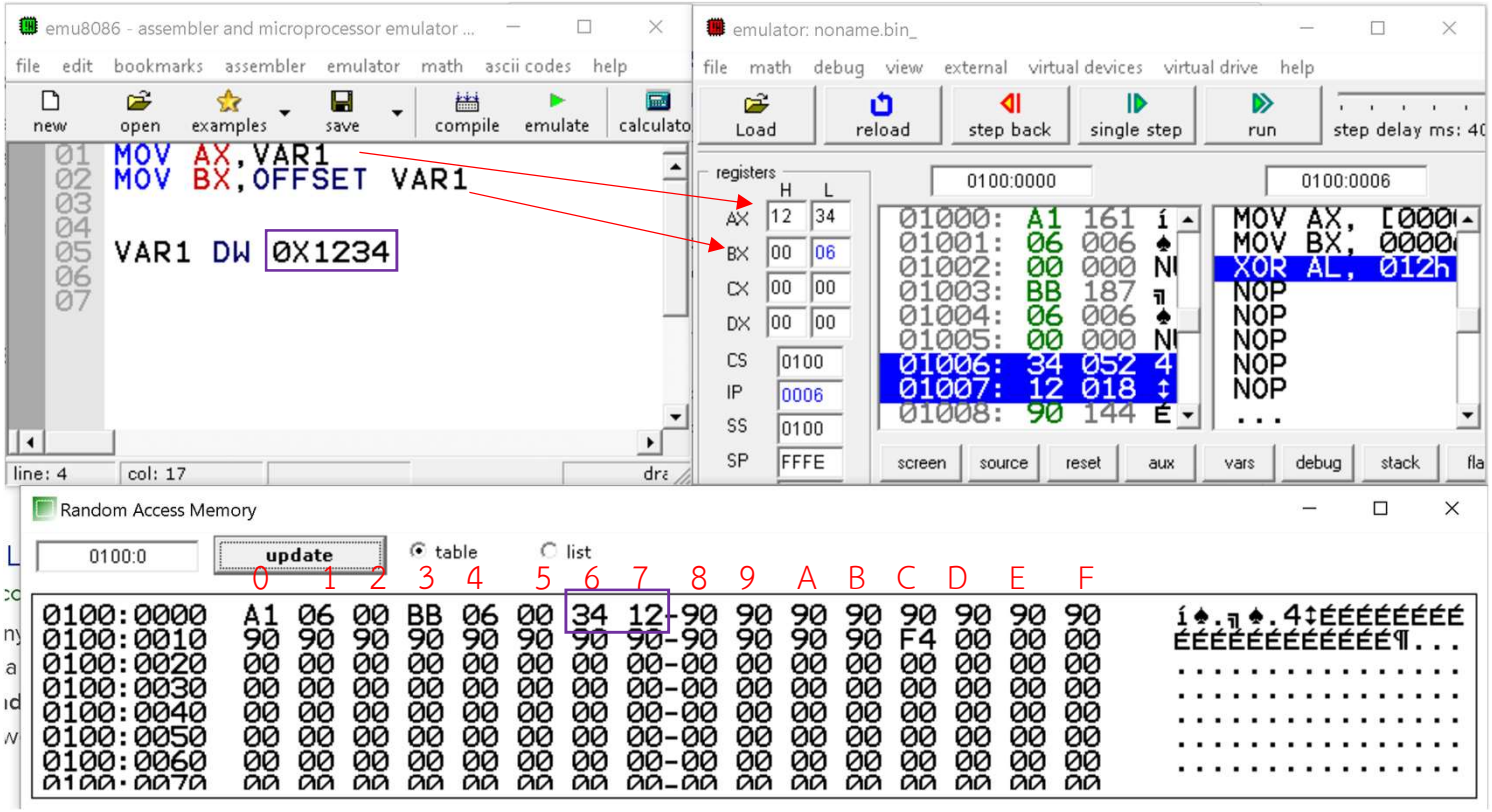
(a) The DB directive
(b) The DW directive
(c) The DD directive
(d) The STRUCT (or STRUC) and ENDS directives
(counted as one)
(e)The EQU Directive
(f)The COMMENT directive
(g)ASSUME
(h) EXTERN
(i) GLOBAL
(j) SEGMENT
(k)OFFSET
(l) PROC
(m)GROUP
(n) INCLUDE

8086 Assembler Directive

OFFSET ใช้กับ Label เพื่อบอก Assembler ให้นำค่าของ ตำแหน่งหน่วยความจำ (Address) ของ Label นั้นมาใช้ หากไม่ระบุ จะหมายถึง ค่าข้อมูล (Value)

MOV AX, Var1 มีค่าเหมือน AX = *Var1 ในภาษา C

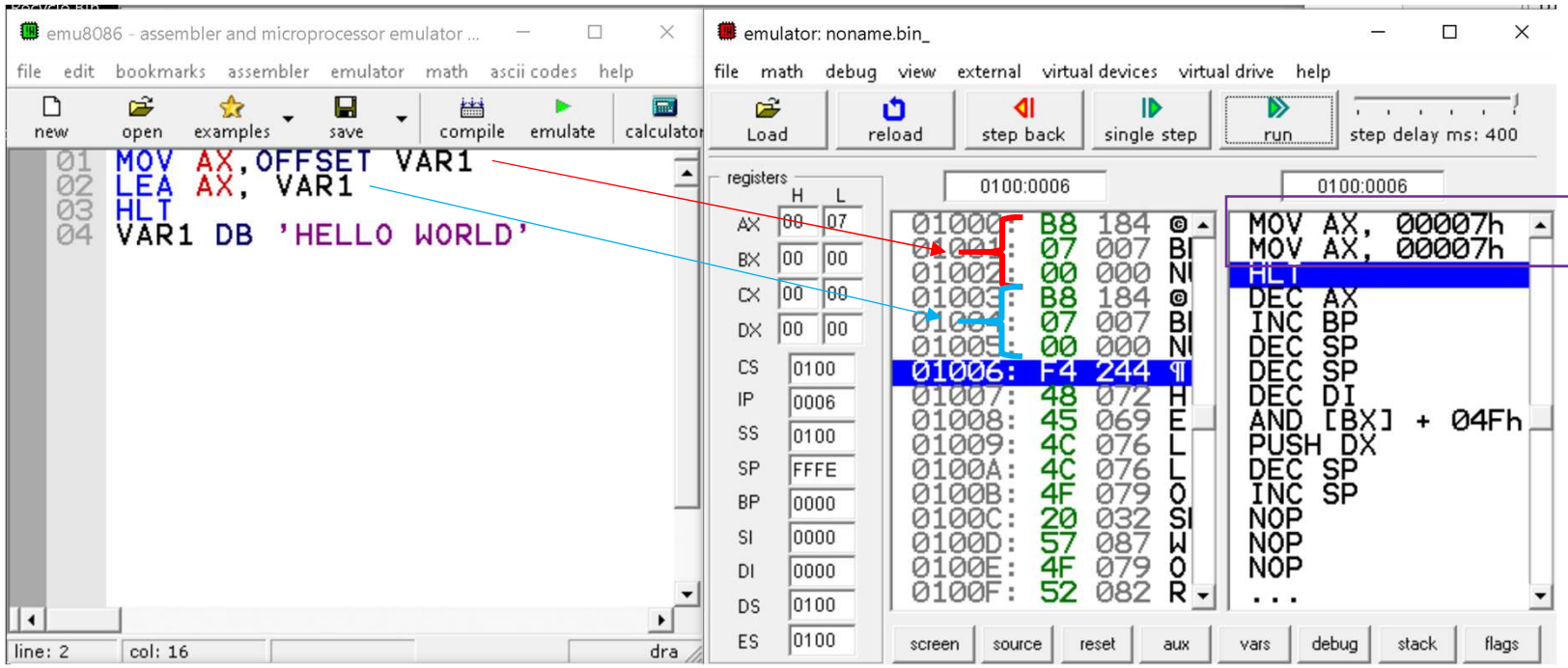
MOV AX, OFFSET Var1 มีค่าเหมือน AX= &Var ในภาษา C



8086 Assembler Directive

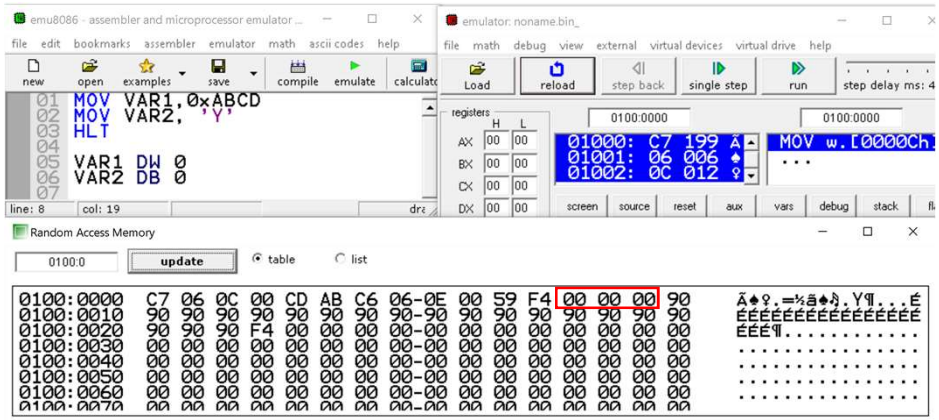
OFFSET ใช้กับ Label เพื่อบอก Assembler ให้นำค่าของ ตำแหน่งหน่วยความจำ (Address) ของ Label นั้นมาใช้
หากไม่ระบุ จะหมายถึง ค่าข้อมูล (Value)

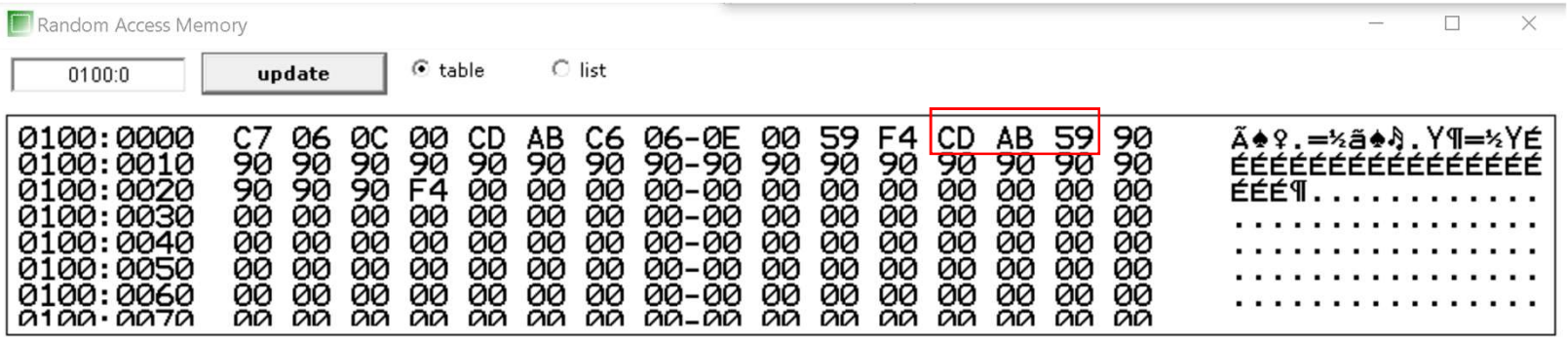
Opcode สำหรับโหลดตำแหน่งของ Label คือ LEA หรือ Load Effective Address
วิธีใช้ LEA รีจิสเตอร์, Label



8086 Assembler Directive

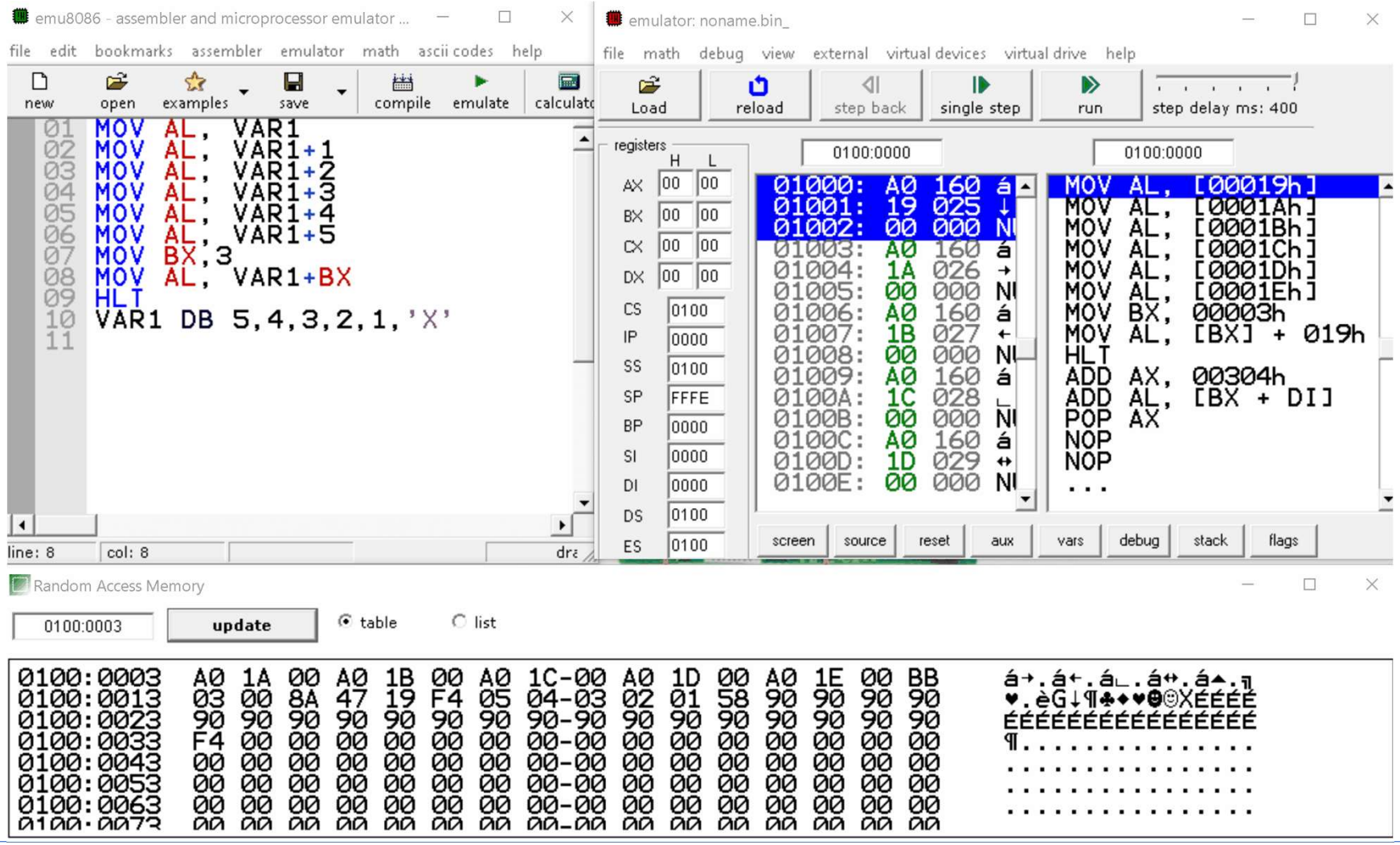
Label ตำแหน่งหน่วยความจำ สามารถใช้เป็นตัวแปรได้





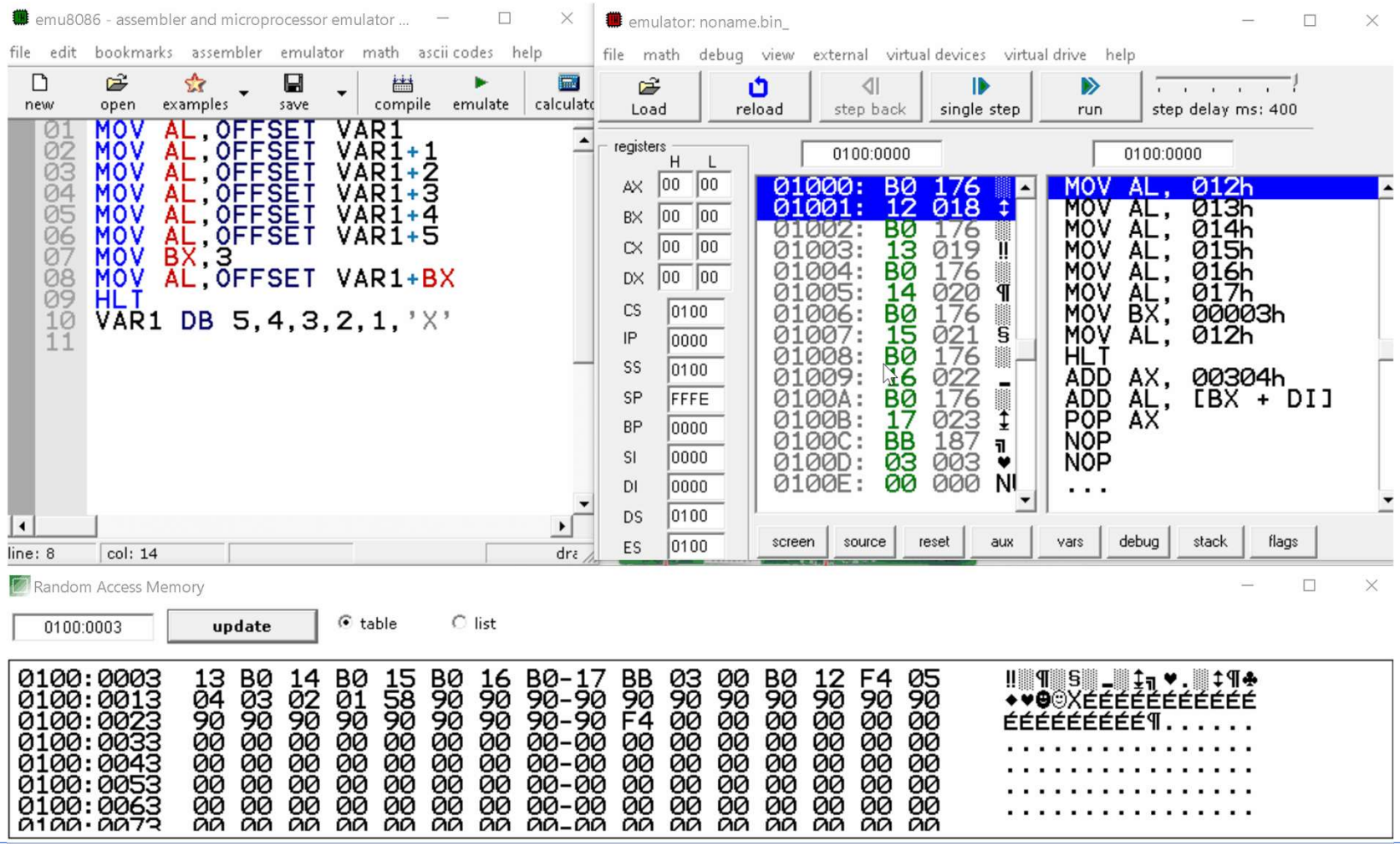
8086 Assembler Directive

สามารถใช้เป็น Array ได้ แต่ต้องคำนวณตำแหน่งถัดไปเอง



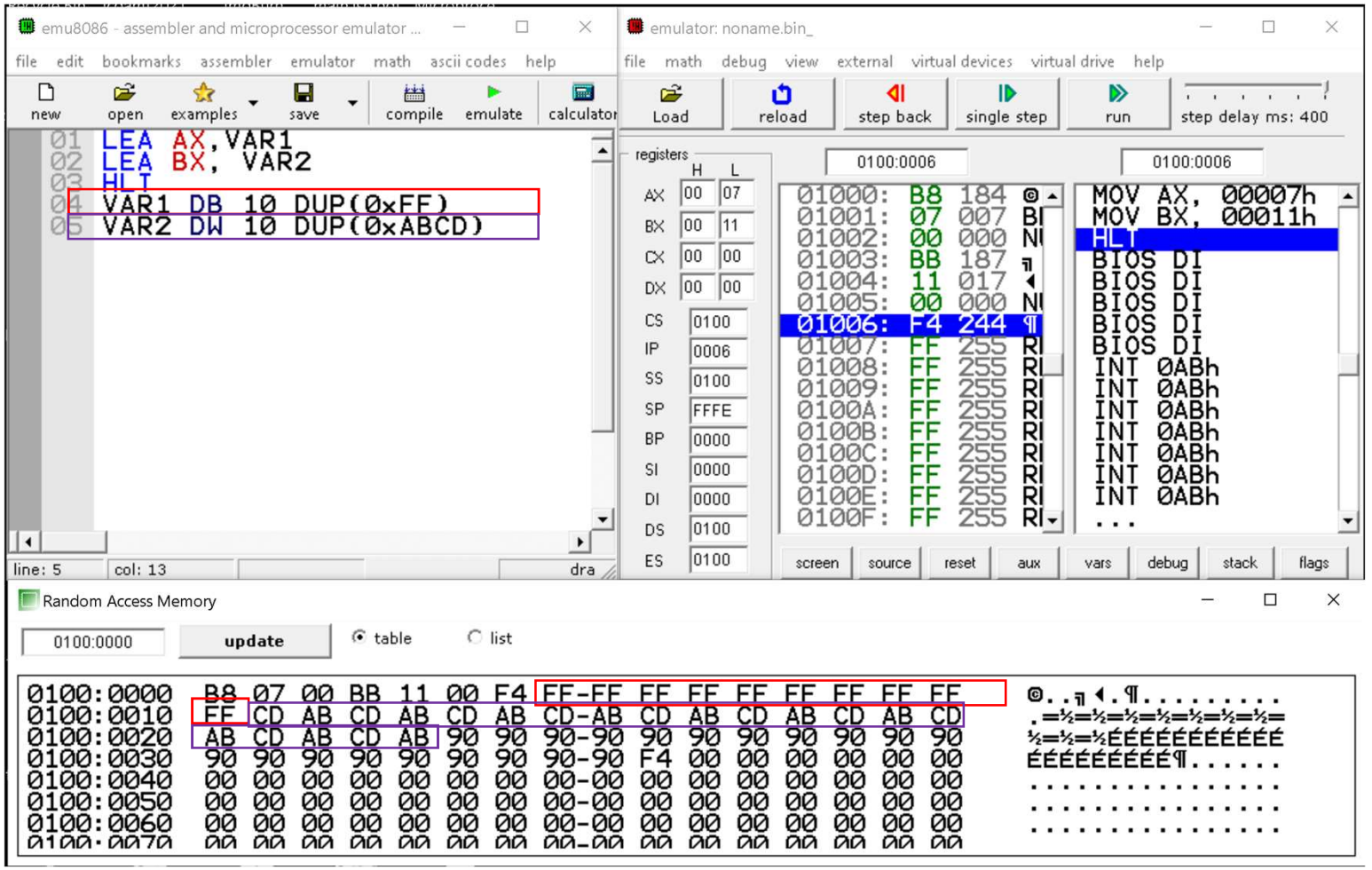
8086 Assembler Directive

สามารถใช้เป็น Array ได้ แต่ต้องคำนวณตำแหน่งถัดไปเอง



การประกาศ Array สามารถกำหนดค่าเริ่มต้นซ้ำ ๆกันได้ โดยใช้ Dup Operator (Duplicate operator)

การใช้งาน จำนวนครั้ง DUP (ค่าที่ต้องการทำซ้ำ)

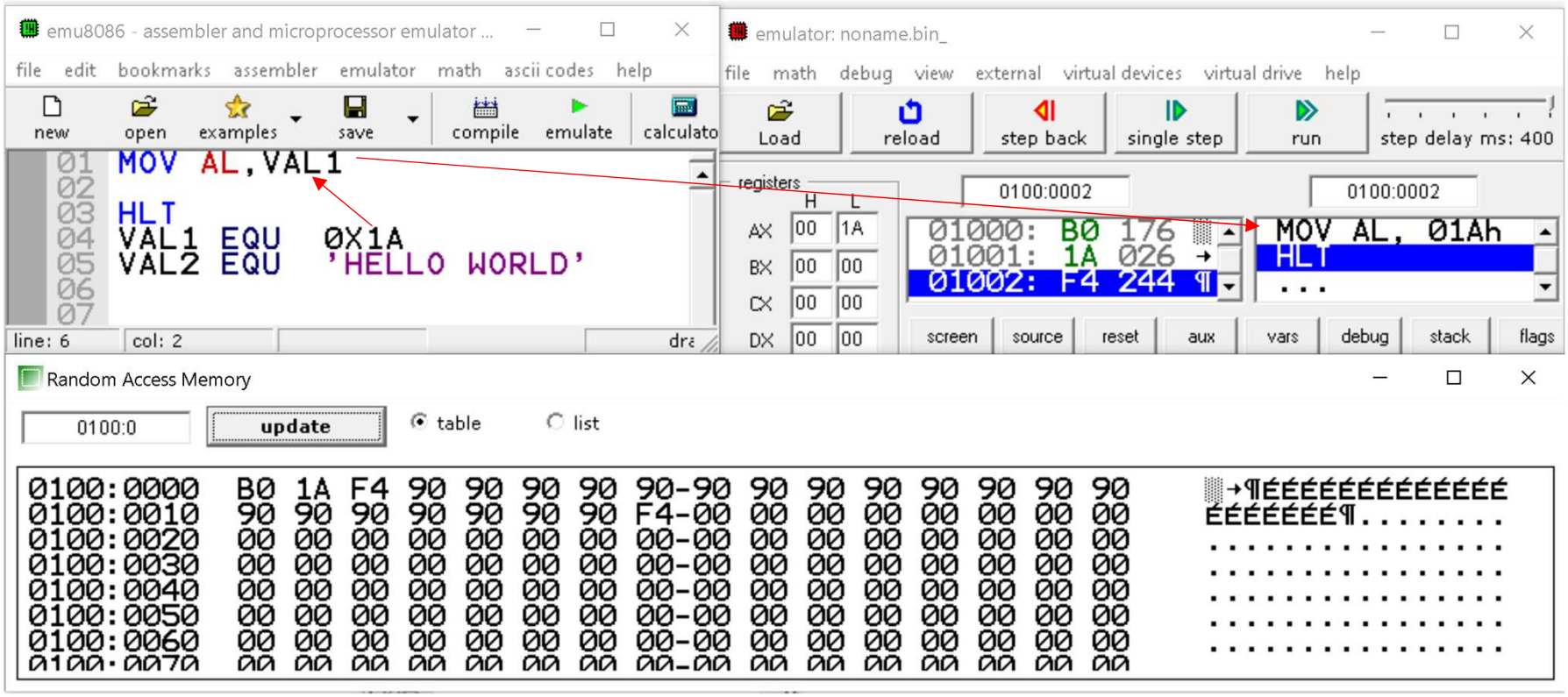


8086 Assembler Directive

(a) The DB directive
(b) The DW directive
(c) The DD directive
(d) The STRUCT (or STRUC) and ENDS directives
(counted as one)
(e)The EQU Directive
(f)The COMMENT directive
(g)ASSUME
(h) EXTERN
(i) GLOBAL
(j) SEGMENT
(k)OFFSET
(l) PROC
(m)GROUP
(n) INCLUDE

EQU หรือ Equivalent ใช้สำหรับประกาศค่าคงที่

8086 Assembler Directive

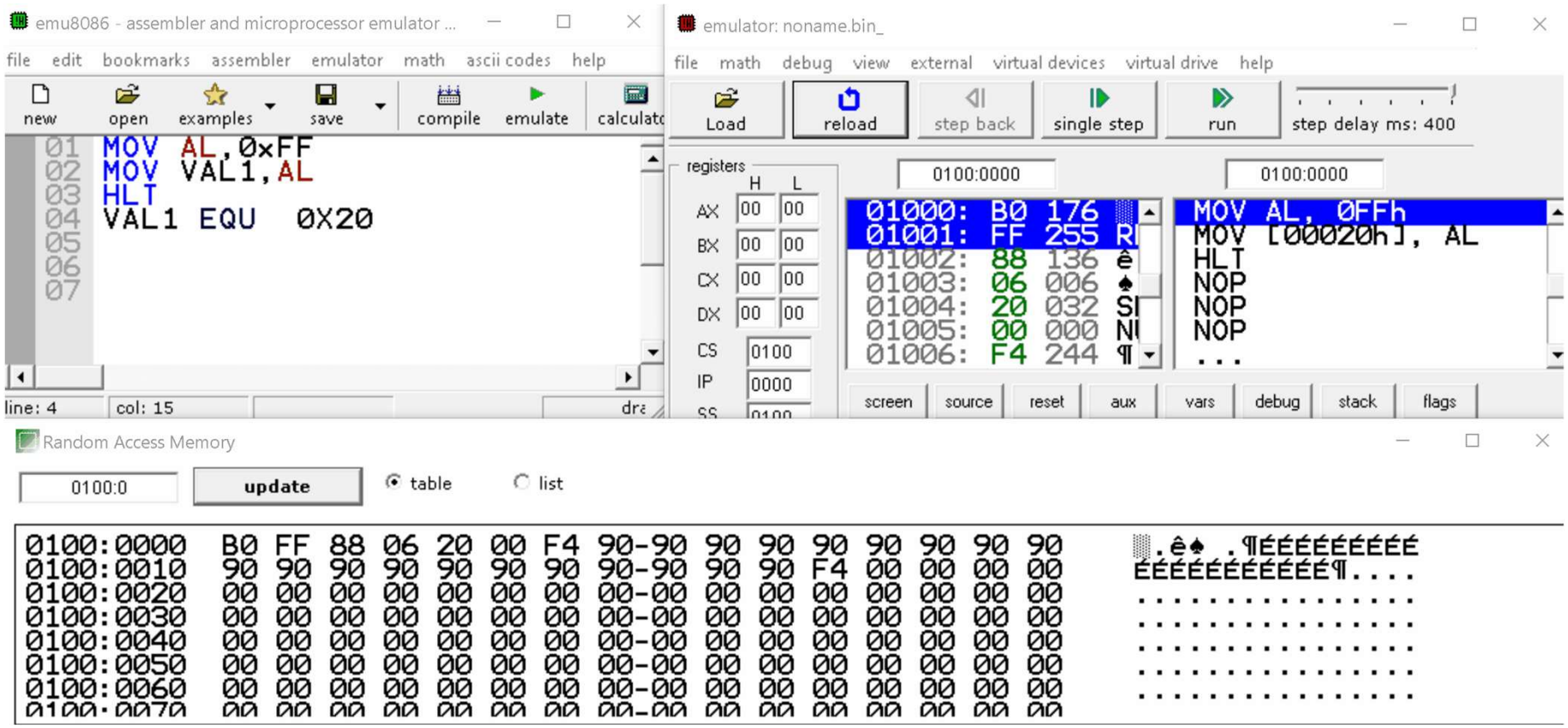


EQU แตกต่างกับ DB ตรงที่ ค่าคงที่ที่นิยามไว้ จะไม่ถูกเขียนไว้ในแฟ้มไบนารี่และไม่อยู่ในหน่วยความจำ แต่จะใช้การนำค่าที่นิยามไว้มาแทน Label ในขนะคอมไพล์ แทน

* VAL2 ไม่ถูกนำมาใช้ จึงไม่ปรากฏ HELLO WORLD อยู่ในหน่วยความจำ

8086 Assembler Directive

EQU หากใช้เป็น operand ตัวแรกของ MOV จะหมายถึงตำแหน่งหน่วยความจำ



8086 สามารถอ้างตำแหน่งหน่วยความจำได้ถึง 1MB หรือ 1,048,576 Bytes
แต่รีจิสเตอร์มีความกว้างแค่ 16 บิต ดังนั้นจำนวนสูงสุดที่รีจิสเตอร์สามารถแสดงได้คือ
0xFFFF หรือ 65535 ในเลขฐาน 10
การอ้างอิงตำแหน่งของหน่วยความจำขนาด 1MB จะต้องใช้รีจิสเตอร์ขนาด 20 บิต
0x00000 ถึง 0xFFFFF หรือ 0 – 1,048,575

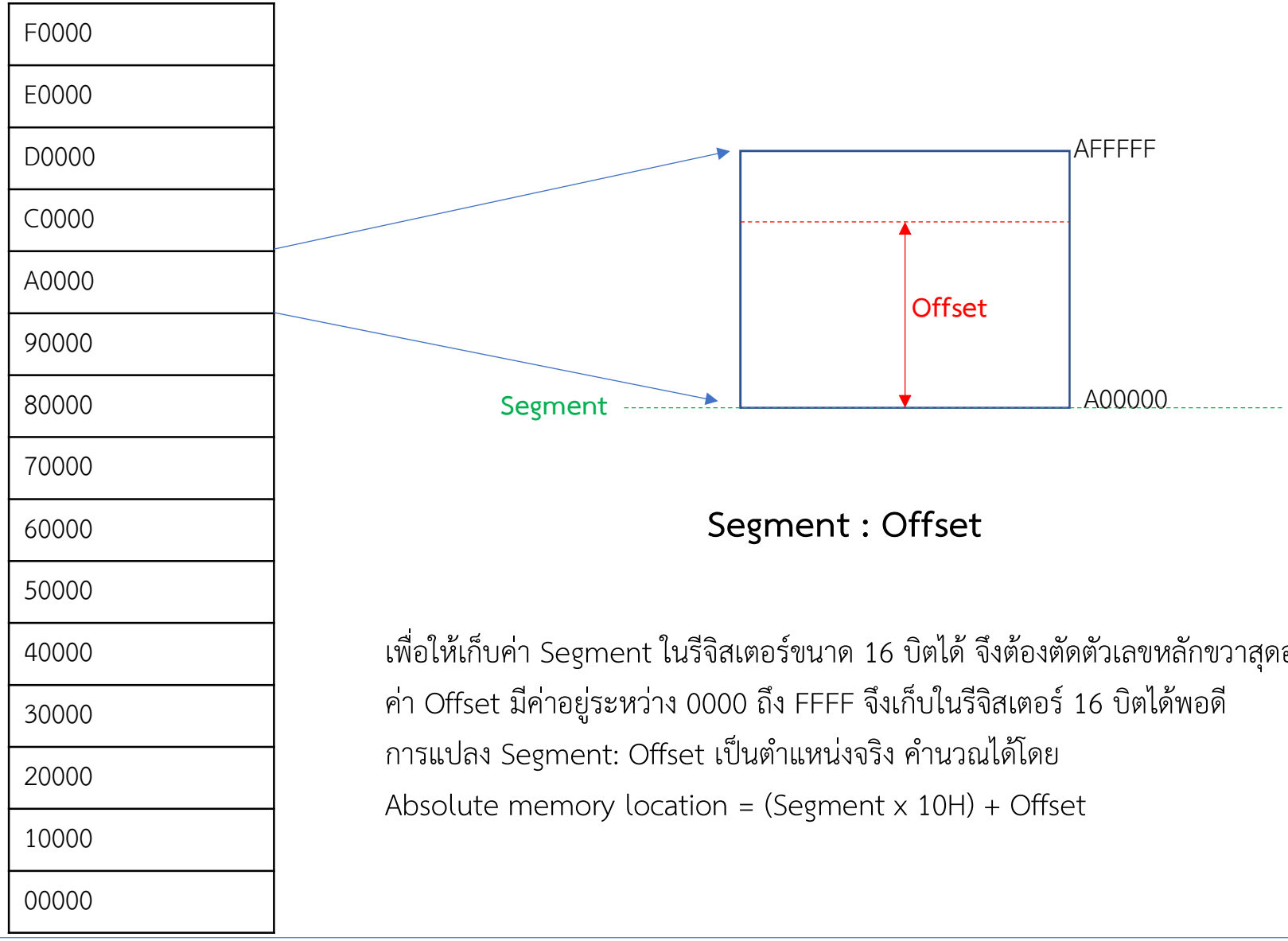
ไมโครโพรเซสเซอร์เบอร์นี้เป็นชนิด 16 บิต ดังนั้นการอ้างตำแหน่งจริง ๆ จึงทำไม่ได้
แต่ Intel สามารถทำให้รองรับหน่วยความจำขนาด 1MB ได้โดยการ
แบ่งหน่วยความจำออกเป็นหลายๆ Segment แต่ละ Segment มีขนาด 64kB
และใช้รีจิสเตอร์ Segment ในการระบุ Segment
ดังนั้นการอ้างตำแหน่งหน่วยความจำจริง ๆ จะต้องใช้รีจิสเตอร์ 2 ตัว
การอ้างตำแหน่งหน่วยความจำในรูปแบบนี้เรียกว่า
Real-Address Mode

ไมโครโพรเซสเซอร์รุ่นถัดมาคือ **80386** มีการขยายให้เป็น 32 บิต จึงสามารถอ้างตำแหน่งได้ถึง 4GB หรือ 4,294,967,295 Bytes จึงไม่จำเป็นต้องแบ่งหน่วยความจำออกเป็น Segment โหมดการอ้างอิงตำแหน่งหน่วยความจำโดยใช้รีจิสเตอร์ตัวเดียวคือ Flat Segmentation Mode

ไมโครโพรเซสเซอร์ตระกูล X64 รุ่นแรก คือ **AMD64** เนื่องจากรีจิสเตอร์มีความกว่างถึง 64 บิต จึงอ้างตำแหน่งได้มากถึง 18,446,744,073,709,551,616 Bytes และเข้าถึงหน่วยความจำได้เฉพาะแบบ Flat Segmentation Mode เท่านั้น หากต้องการใช้ Real-Address mode ต้องเปลี่ยนโหมดไปทำงานแบบ X86

โหมดการทำงานแบบ Flat นี้ เราจะศึกษากันหลังสอบกลางภาค

8086 Real-Address Mode



Segment : Offset

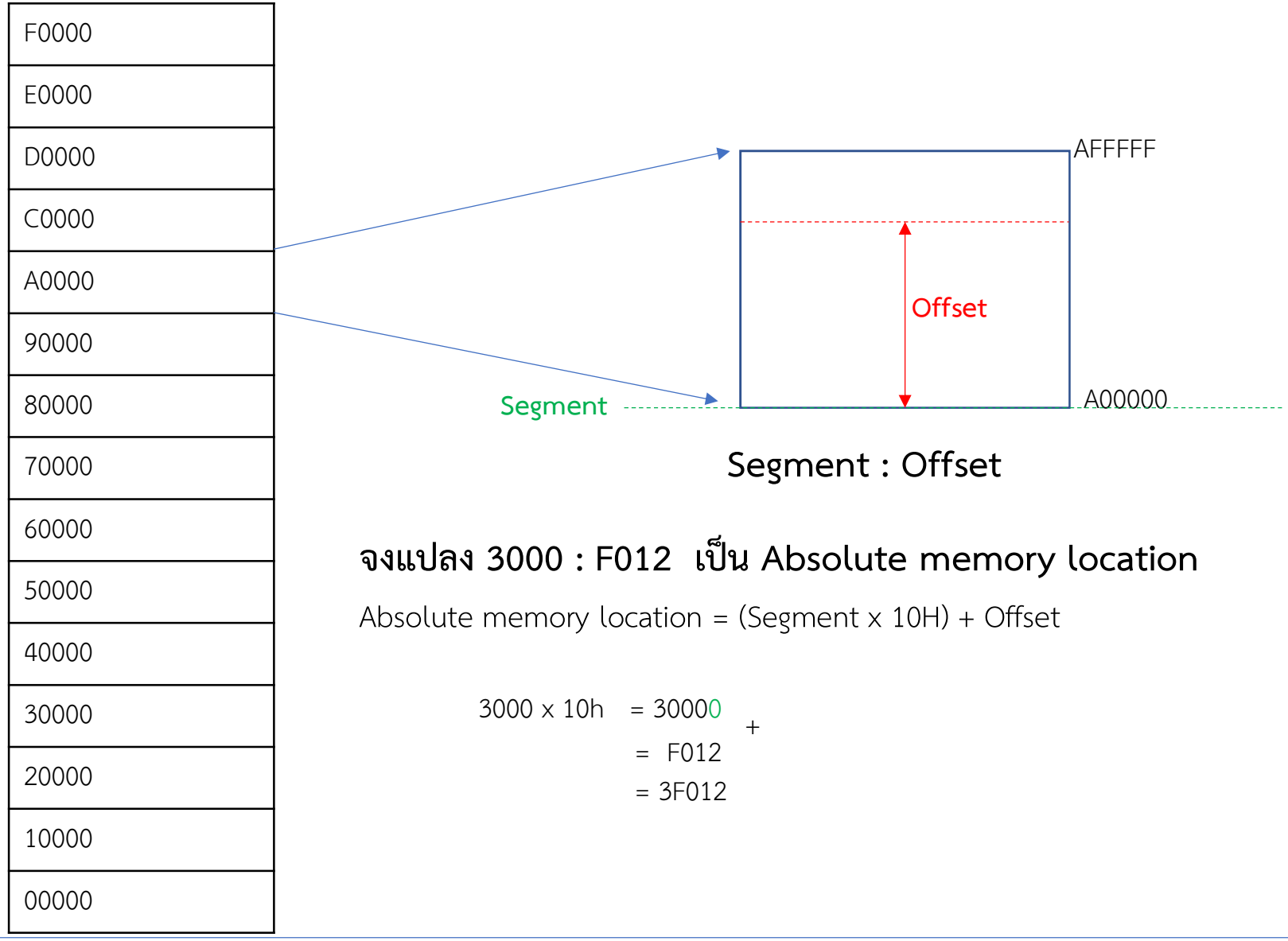
เพื่อให้เก็บค่า Segment ในรีจิสเตอร์ขนาด 16 บิตได้ จึงต้องตัดตัวเลขหลักขวาสุดออก

ค่า Offset มีค่าอยู่ระหว่าง 0000 ถึง FFFF จึงเก็บในรีจิสเตอร์ 16 บิตได้พอดี

การแปลง Segment: Offset เป็นตำแหน่งจริง คำนวณได้โดย

Absolute memory location = (Segment x 10H) + Offset

8086 Real-Address Mode



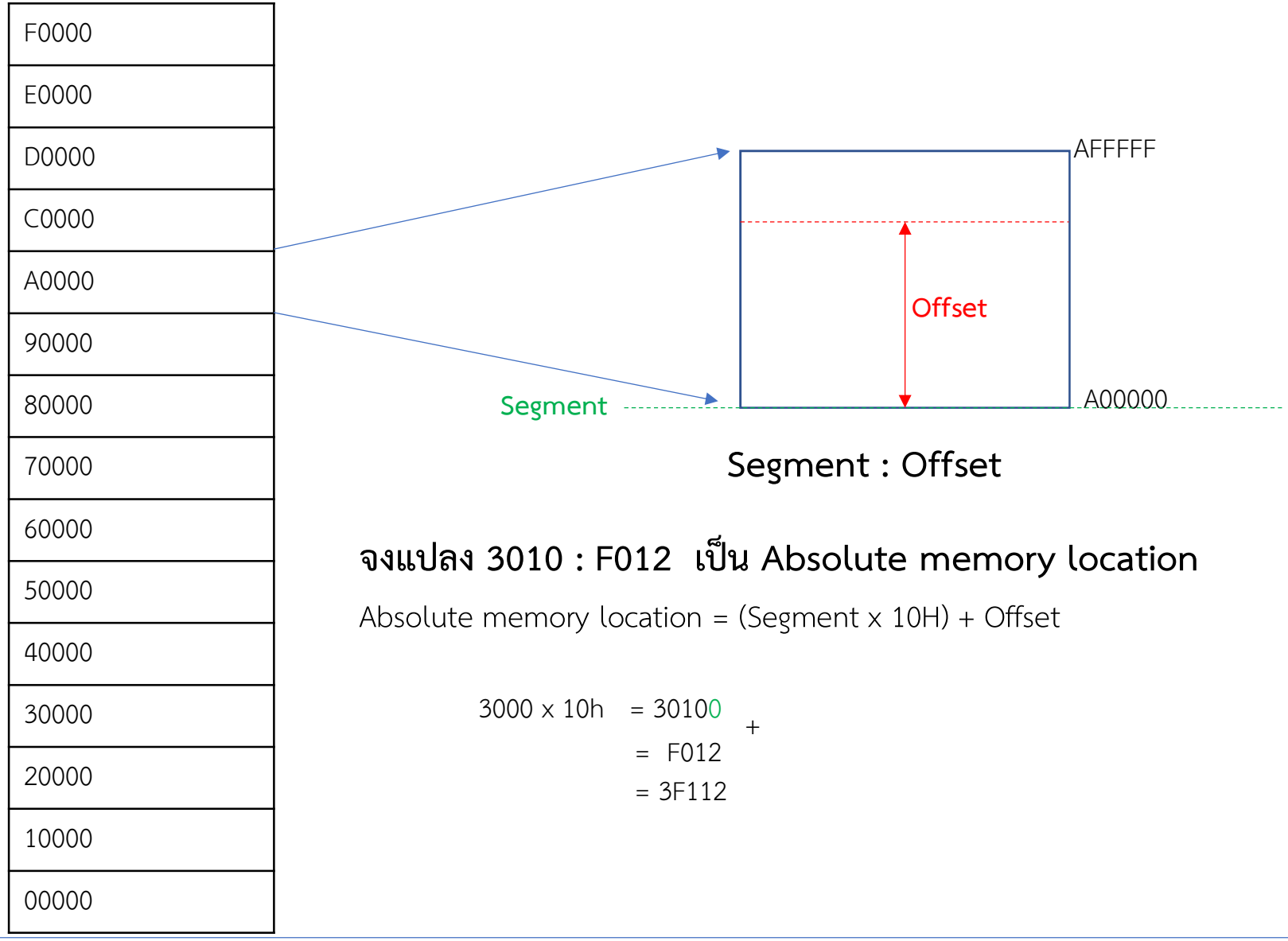
Segment : Offset

จงแปลง 3000 : F012 เป็น Absolute memory location

Absolute memory location = (Segment x 10H) + Offset

3000 x 10h = 30000 +
= F012
= 3F012

8086 Real-Address Mode



Segment : Offset

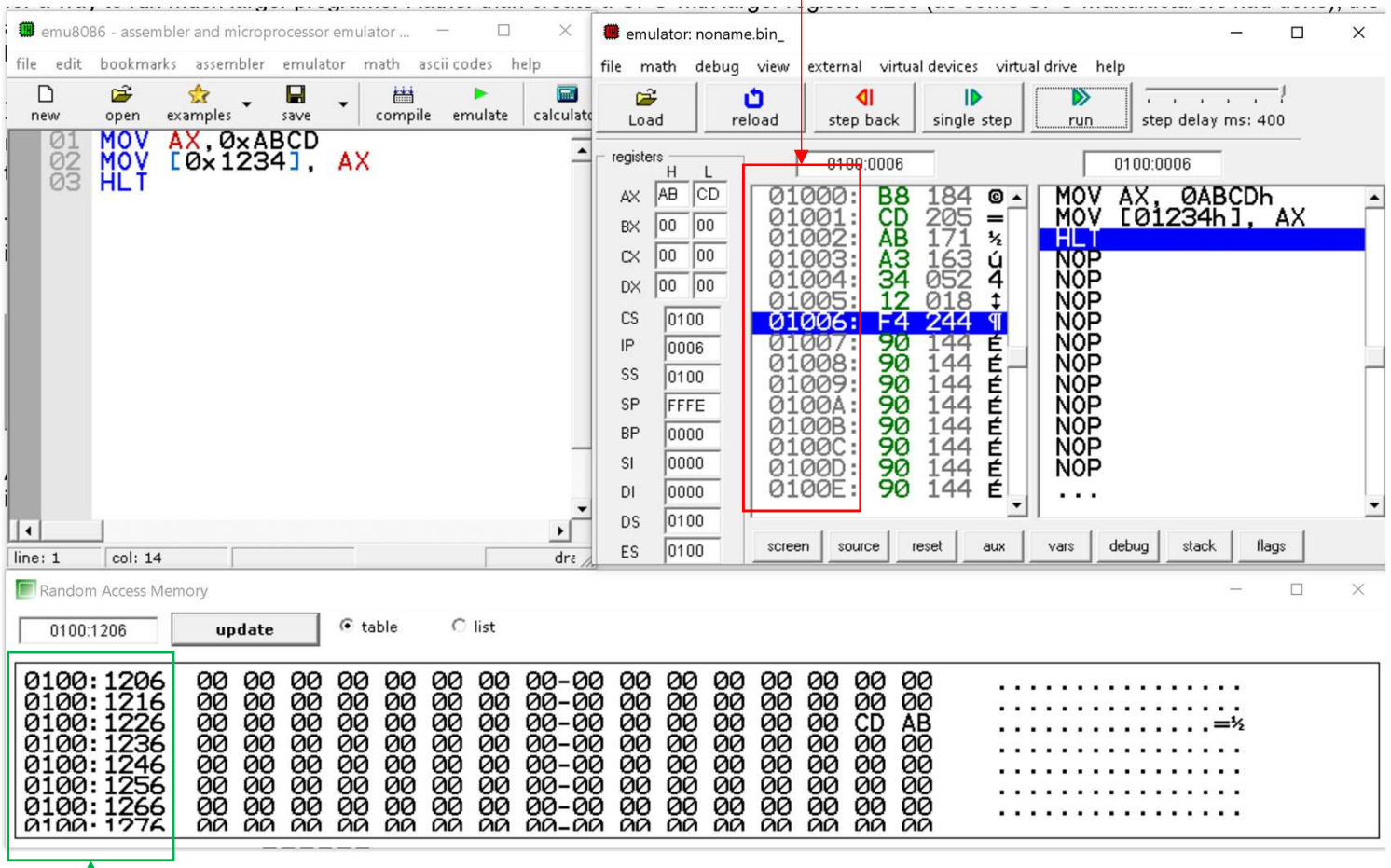
จงแปลง 3010 : F012 เป็น Absolute memory location

Absolute memory location = (Segment x 10H) + Offset

3000 x 10h = 30100 +
= F012
= 3F112

8086 Real-Address Mode

Absolute Memory Location, Flat address, Linear Address



Segment Memory Map, Real-Address mode

8086 Real-Address Mode

F0000
E0000
D0000
C0000
A0000
90000
80000
70000
60000
50000
40000
30000
20000
10000
00000

Segment : Offset

การขยับ Segment จะกระทำได้ผ่านรีจิสเตอร์ Segment เท่านั้น (SREG)

CS คือ Code Segment ใช้กำหนด Segment เริ่มต้นของโปรแกรม

DS คือ Data Segment ใช้กำหนด Segment เริ่มต้นของข้อมูล

SS คือ Stack Segment ใช้กำหนดจุดเริ่มต้นของ Stack

ES คือ Extra Segment ใช้ชี้ Segment ตำแหน่งที่สอง (สำหรับบาง opcode)

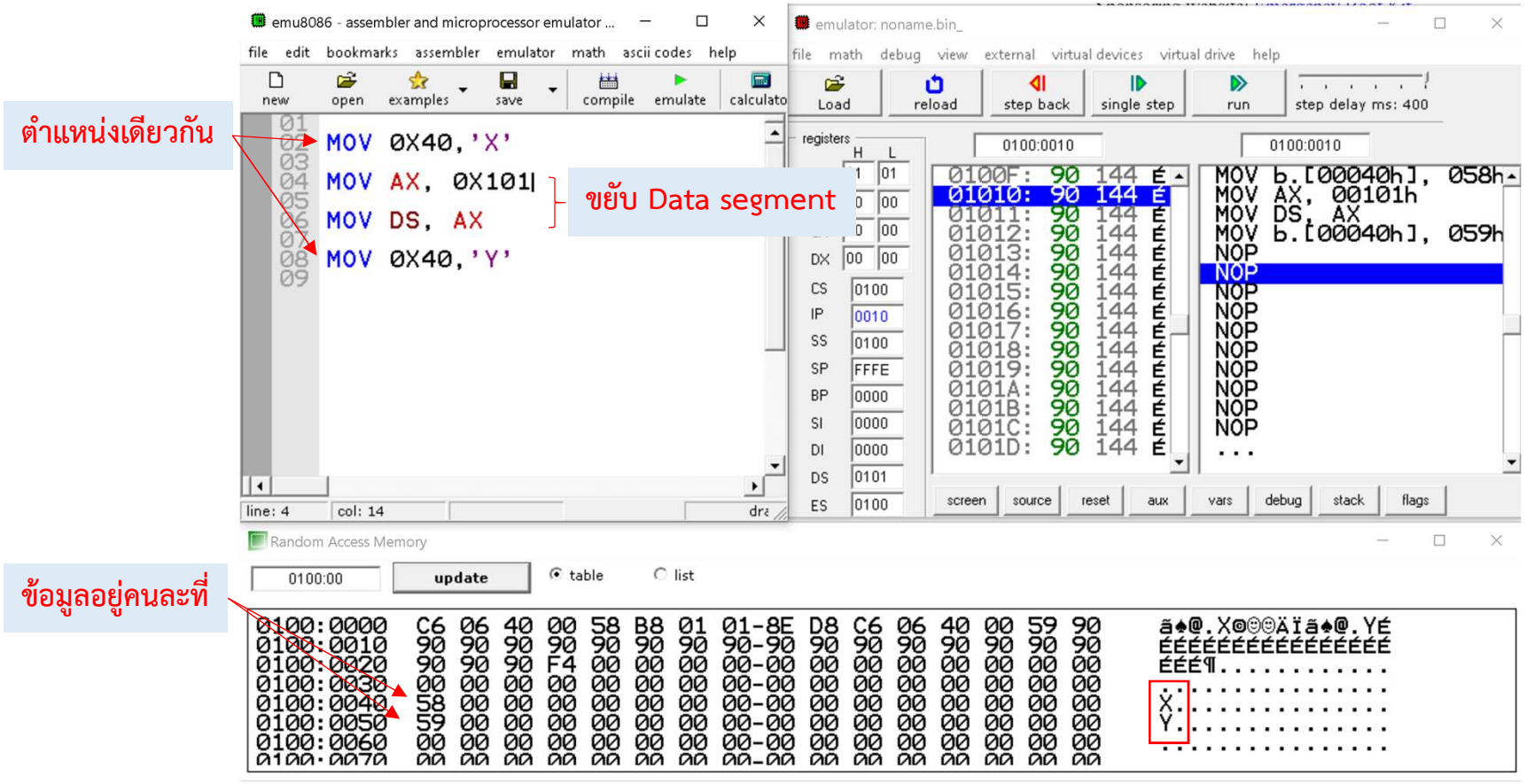
8086 Real-Address Mode

คำสั่ง MOV MEMORY, IMMEDIATE

MOV MEMORY, REG

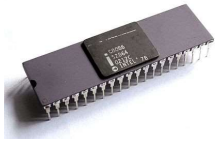
MOV MEMORY, SREG

จริง ๆ แล้วทำงานคู่กับรีจิสเตอร์ DS



8086 Real-Address Mode
จำเป็นหรือไม่
ช่วยให้เขียนโปรแกรมง่ายขึ้น
หรือทำให้โปรแกรมเขียนยาก
?

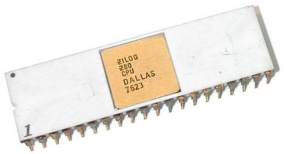
Intel 8086 เปิดตัวปี 1978 เพื่อเป็น Microprocessor ทางเลือกที่ เร็ว , แรง , รองรับหน่วยความจำได้มากถึง 1MB



Max. CPU clock rate 5 MHz to 10 MHz
Data width 16 bits
Address width 20 bits

แต่ในปีนั้นเครื่องไมโครคอมพิวเตอร์ เริ่มแพร่หลายแล้ว

A MOS Technology 6502
เปิดตัว 1975
Max. CPU clock rate 1 MHz to 3 MHz
Data width 8
Address width 16



Zilog Z80
1976
Max. CPU clock rate 2.5, 4, 6, 8 MHz to 10 MHz
Data width 8 bits
Address width 16 bits

Apple II
4KiB, 8KiB, 12KiB, 16KiB, 20KiB, 24KiB, 32KiB, 36KiB, 48KiB, or 64KiB

Atari 800
64KB

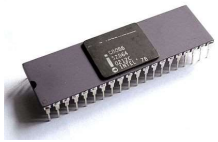
TRS-80 Model I
4kB - 48kB

Commodore 64
64kB

Nintendo Entertainment System
8kB

ZX Spectrum
16 KB / 48 KB / 128 KB

Intel 8086 เปิดตัวปี 1978 ในปีถัดมาได้ได้มีการลด spec ของ data bus ลงแล้วเปลี่ยนชื่อเป็น 8088



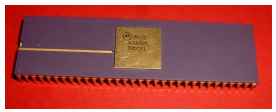
Max. CPU clock rate 5 MHz to 10 MHz
Data width 16 bits
Address width 20 bits

Max. CPU clock rate 5 MHz to 16 MHz
Data width 8 bits
Address width 20 bits

Segmented Memory Addressing

IBM Personal Computer
CPU Intel 8088 @ 4.77 MHz
Memory 16 kB – 640 kB

Motorola 68000 เปิดตัวปี 1979



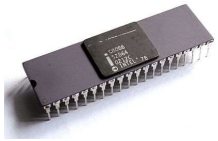
Bits 16/32-bit
Data width 16 bits
Address width 24 bits

Flat Memory Addressing

Apple Lisa
512KB, 1MB, 2MB

Macintosh
128K

Intel 8086 เปิดตัวปี 1978 ในปีถัดมาได้ได้มีการลด spec ของ data bus ลงแล้วเปลี่ยนชื่อเป็น 8088



Max. CPU clock rate 5 MHz to 10 MHz

Data width 16 bits

Address width 20 bits

Max. CPU clock rate 5 MHz to 16 MHz

Data width 8 bits

Address width 20 bits

Segmented Memory Addressing

IBM Personal Computer

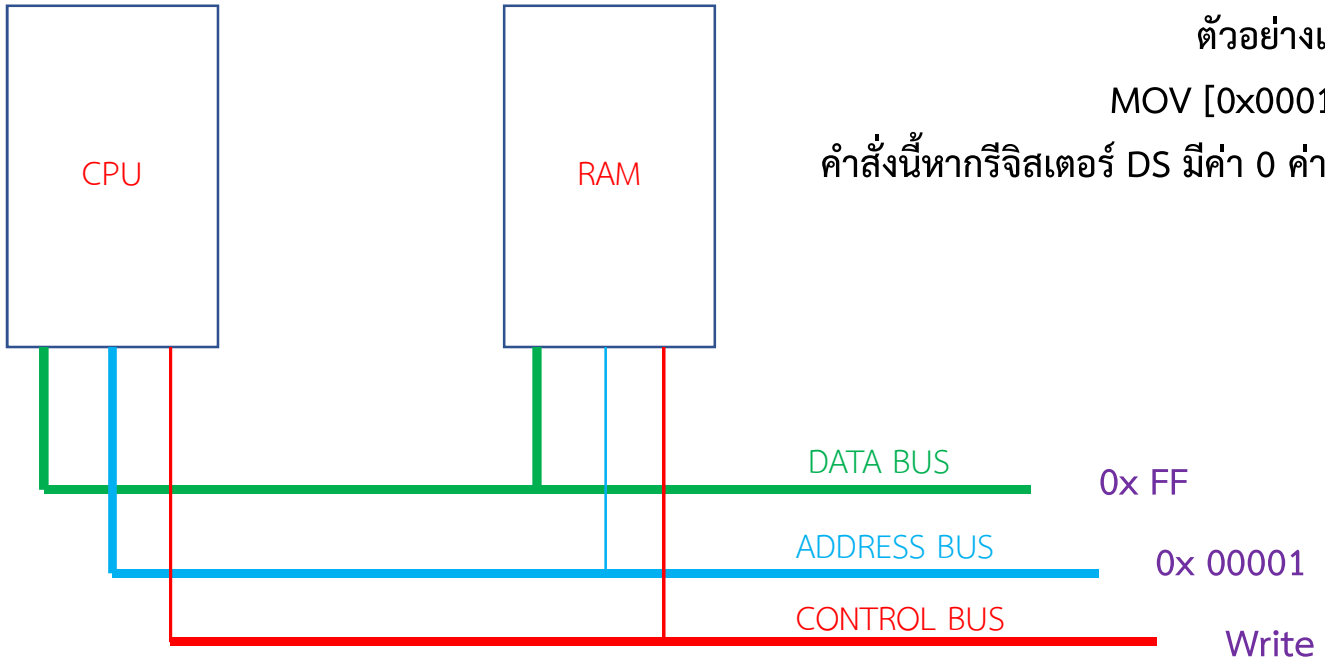
CPU Intel 8088 @ 4.77 MHz

Memory 16 kB – 640 kB

Release date August 12, 1981

Segmented Memory ทำให้เขียนโปรแกรมยาก แต่ก็มีข้อดี

คำสั่ง MOV ที่ทำงานกับ Memory ไม่ได้สนใจว่า ข้อมูลภายนอกเป็น RAM หรือไม่



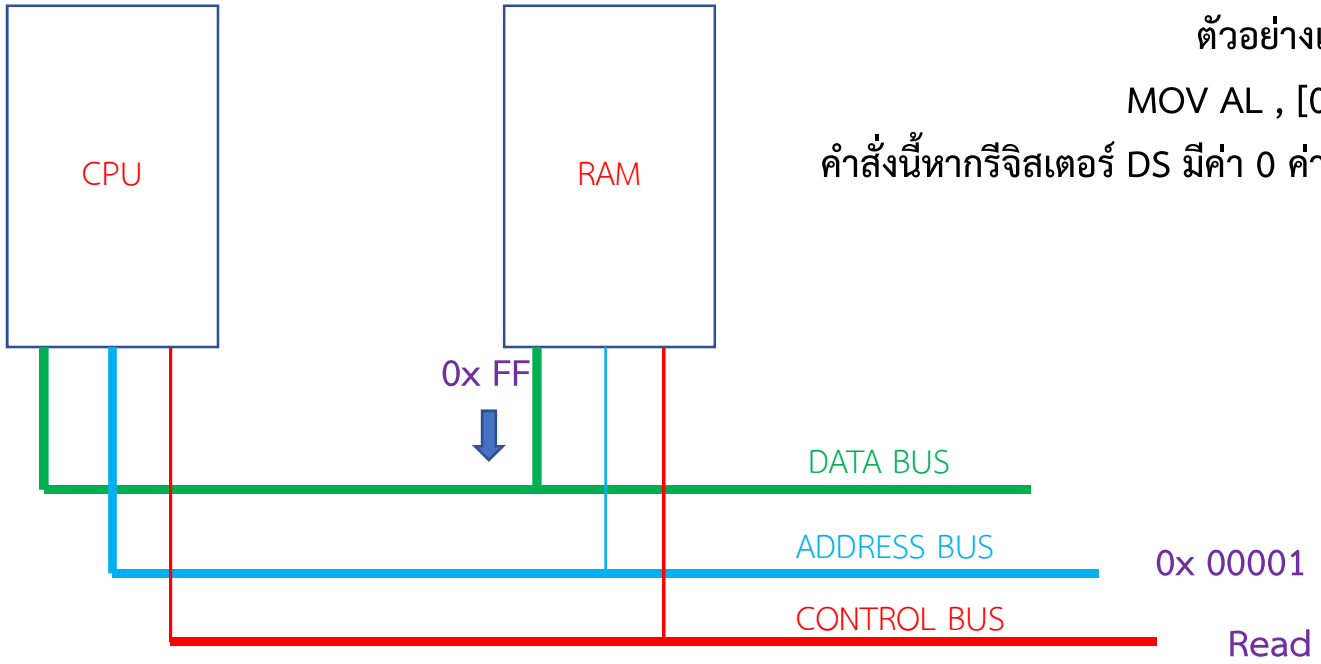
ตัวอย่างเช่น

MOV [0x0001] , 0xFF

คำสั่งนี้หากรีจิสเตอร์ DS มีค่า 0 ค่า ตำแหน่งจริงจะเป็น 0x0001

RAM จะทำการนำเอาข้อมูลจาก Data bus ไปเก็บในตำแหน่งที่ ระบุใน Address bus

คำสั่ง MOV ที่ทำงานกับ Memory ไม่ได้สนใจว่า ข้อมูลภายนอกเป็น RAM หรือไม่



ตัวอย่างเช่น

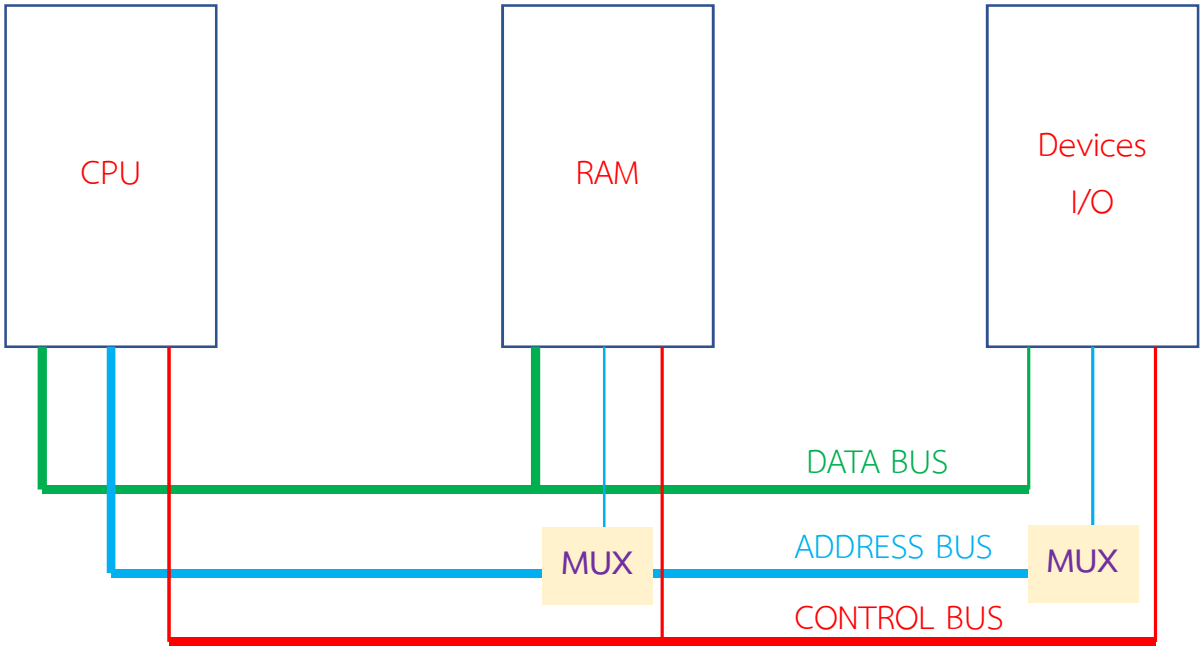
MOV AL , [0x0001]

คำสั่งนี้หากรีจิสเตอร์ DS มีค่า 0 ค่า ตำแหน่งจริงจะเป็น 0x0001

RAM จะทำการป้อนข้อมูลสู่ Data bus ตามตำแหน่งที่ ระบุใน Address bus

ในระบบคอมพิวเตอร์ ไม่ได้มีแค่แรม ยังมี I/O และอื่น ๆอีก

IBM ได้ออกแบบระบบ PC ให้อุปกรณ์ภายนอก Map ไปยัง Address bus

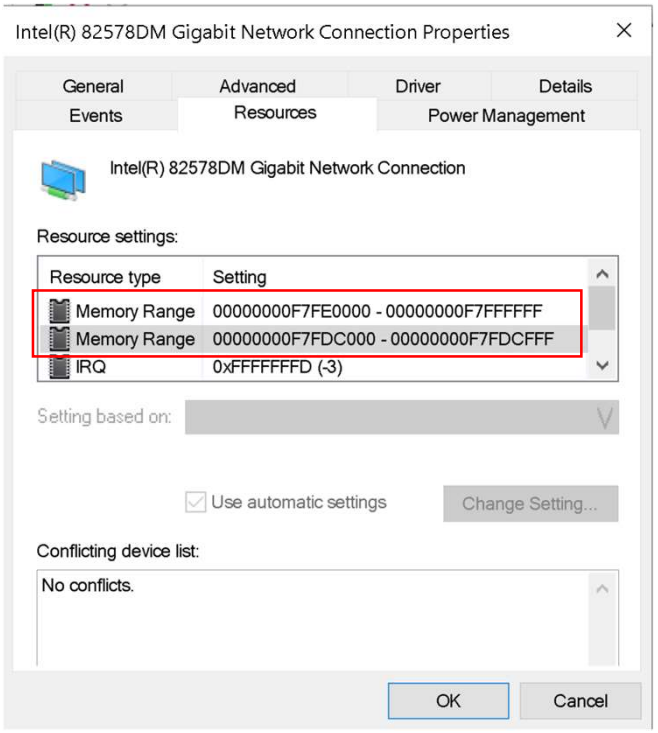


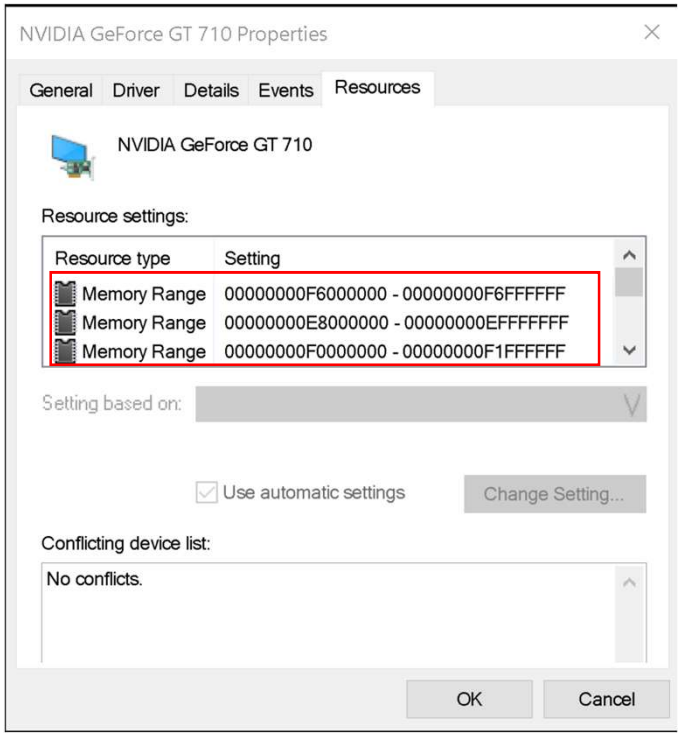
โดยใช้วงจร Multiplexer , Decoder เป็นตัวเลือกว่าจะให้อุปกรณ์ไหนทำงาน

ปัจจุบันนี้ I/O ส่วนใหญ่ถูกแยกการประมวลผลออกจากไมโครโพรเซสเซอร์ ไปอยู่ยังหน่วยประมวลผลร่วมที่เรียกว่า Chipset

เพราะปัจจุบัน I/O มีความซับซ้อนมาก และมักจะทำงานช้ากว่าไมโครโพรเซสเซอร์มาก

หากไม่จำเป็นต้องใช้ความเร็วในการส่งข้อมูลสูงจริง ๆ จะไม่มีการเชื่อมต่อกับไมโครโพรเซสเซอร์โดยตรง





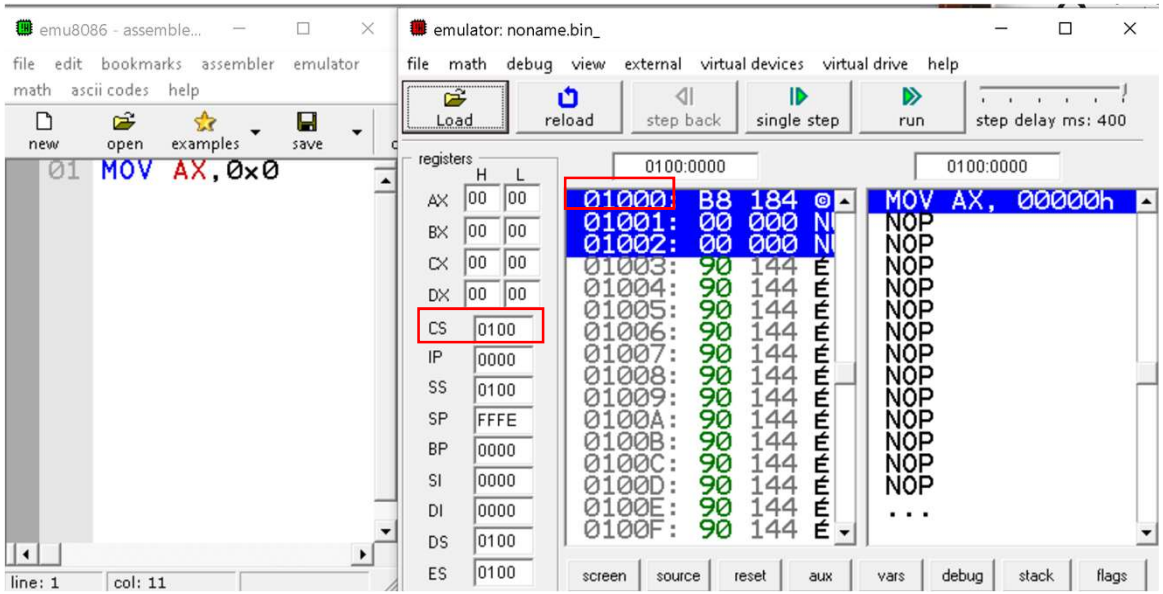
ตัวอย่างนี้เป็นไมโครโพรเซสเซอร์แบบ 64 บิต จึงอ้างตำแหน่งหน่วยความจำแบบ Flat ไม่มีการ Segment

F0000	BIOS
E0000	BIOS
D0000	VRAM การ์ดจอ
C0000	VRAM การ์ดจอ
A0000	VRAM การ์ดจอ
90000	
80000	
70000	
60000	
50000	
40000	
30000	
20000	
10000	
00000	00000-003FF Interrupt Vector Table

รองรับหน่วยความจำ 1MB ไม่ได้แปลว่า User จะใช้ได้ครบทั้ง 1MB
หากต้องการใช้ได้เต็มความจุ โปรแกรมเมอร์ต้องออกแบบคอมพิวเตอร์
และประกอบขึ้นใหม่เอง

แผนที่นี้ มีเฉพาะ BIOS เท่านั้น
ยังไม่รวมส่วนที่ใช้ไปโดยระบบปฏิบัติการ

F0000	BIOS
E0000	BIOS
D0000	VRAM การ์ดจอ
C0000	VRAM การ์ดจอ
A0000	VRAM การ์ดจอ
90000	
80000	
70000	
60000	
50000	
40000	
30000	
20000	
10000	
00000	00000-003FF Interrupt Vector Table



โปรแกรมที่เราเขียนจึงถูกโหลดเข้าสู่หน่วยความจำที่

Segment = 100H

เพื่อหลบ BIOS Interrupt Vector

โปรแกรมเริ่มที่ offset = 0 เมื่อแปลงจาก 0100:0000

จึงได้ตำแหน่งจริงเป็น 01000H