

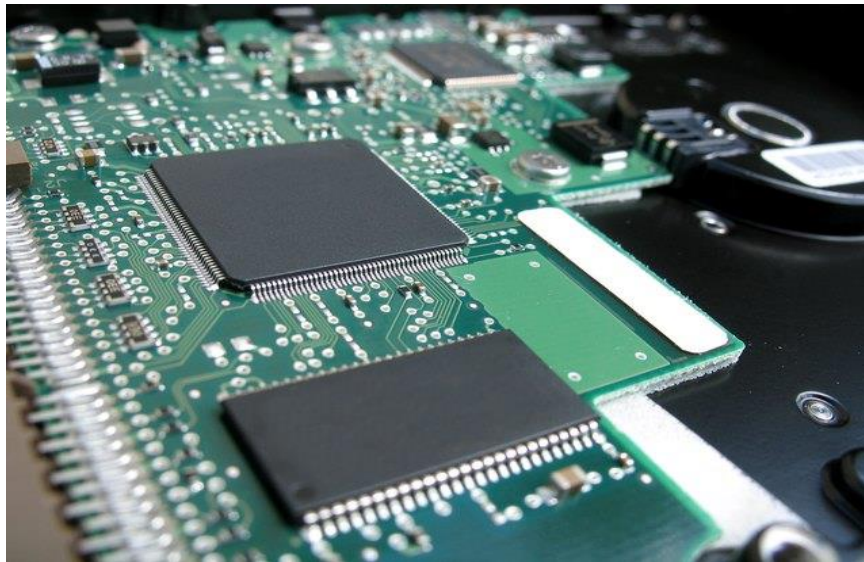


جامعة القادسية
كلية التربية



Lecture 16

Microprocessors



Prepared by:

Firas Abdulrahman Yosif

Jump Instructions

These instructions transfer the program control from one address to other address.
(Not in a sequence).

The purpose of a jump instruction is to alter the execution path of instructions in the program. The code segment register and instruction pointer keep track of the next instruction to be executed. Thus a jump instruction involves altering the contents of these registers. In this way, execution continues at an address other than that of the next sequential instruction. That is, a jump occurs to another part of the program.

الغرض من تعليمات القفز هو تغيير مسار تنفيذ التعليمات في البرنامج. يقوم سجل مقطع الكود ومؤشر التعليمات بتتبع التعليمات التالية التي سيتم تنفيذها. وبالتالي تتضمن تعليمات الانتقال تغيير محتويات هذه السجلات. بهذه الطريقة ، يستمر التنفيذ على عنوان آخر غير عنوان التعليمات التسلسلية التالية. أي ، تحدث قفزة إلى جزء آخر من البرنامج.

القاعدة العامة ان المعالج 8086 يقوم بتنفيذ البرنامج حسب ترتيب الاوامر من اول البرنامج الى نهايته بصورة متسلسلة. ولكن هناك بعض التطبيقات التي تتطلب الخروج من هذه القاعدة , مثلا تنفيذ عملية معينة او تنفيذ مجموعة اوامر عدة مرات في فترة زمنية محددة ففي هذه الحالة نستخدم اوامر القفز Jump Instructions داخل البرنامج للانتقال من مكان الى اخر الى اعلى البرنامج او الى اسفل البرنامج . وتعتبر Jump Instructions من الابعازات المهمة في التحكم بسير تنفيذ البرنامج وتنقسم الـ Jump Instructions الى نوعين اساسيين:

There are two types of jump instructions:

1- Unconditional jump

2- Conditional jump

في هذا الدرس سنتحدث عن القفز الغير مشروط :

1- Unconditional jump:

In an unconditional jump, no status requirements are imposed for the jump to occur. That is, as the instruction is executed, the jump always takes place to change the execution sequence.


في القفزة غير المشروطة ، لا توجد متطلبات مفروضة على حدوث القفزة. أي ، أثناء تنفيذ التعليمات ، تحدث القفزة دائماً لتغيير تسلسل التنفيذ.

عند تنفيذ اوامر القفز الغير مشروط Unconditional jump ينقل المعالج عملية التنفيذ الى المكان الجديد بدون قيد او شرط . والمكان الذي سيتم القفز اليه سيكون محدد بعلامة معينة (label) توضع امام الايعاز المراد القفز اليه. وقد يكون الـ jump داخل البرنامج اما الى الاعلى او الى الاسفل.

Instruction	Meaning	Format	Operation	Flags affected
JMP	Unconditional jump	JMP operand	Jump is to the address specified by operand	non

Example (1):

```
* MOV AX, 5
MOV BX, AX
JMP *
```



في المثال اعلاه يكون الـ JUMP بدون قيد او شرط نحو الاعلى بعد وضع علامة على الايعاز المراد القفز اليه.

Example (2):

MOV BL, 3

MOV CL, BL

JMP *

MOV DL,5

*MOV [SI], CL

HLT

في المثال اعلاه يتم الـ JUMP بدون شرط الى اسفل البرنامج.

Example (3): What's this program print?

MOV AX, 234H

MOV BX, 889H

JMP X1

ROR AX,3

SUB AX, BX

X1: ADD AX, BX

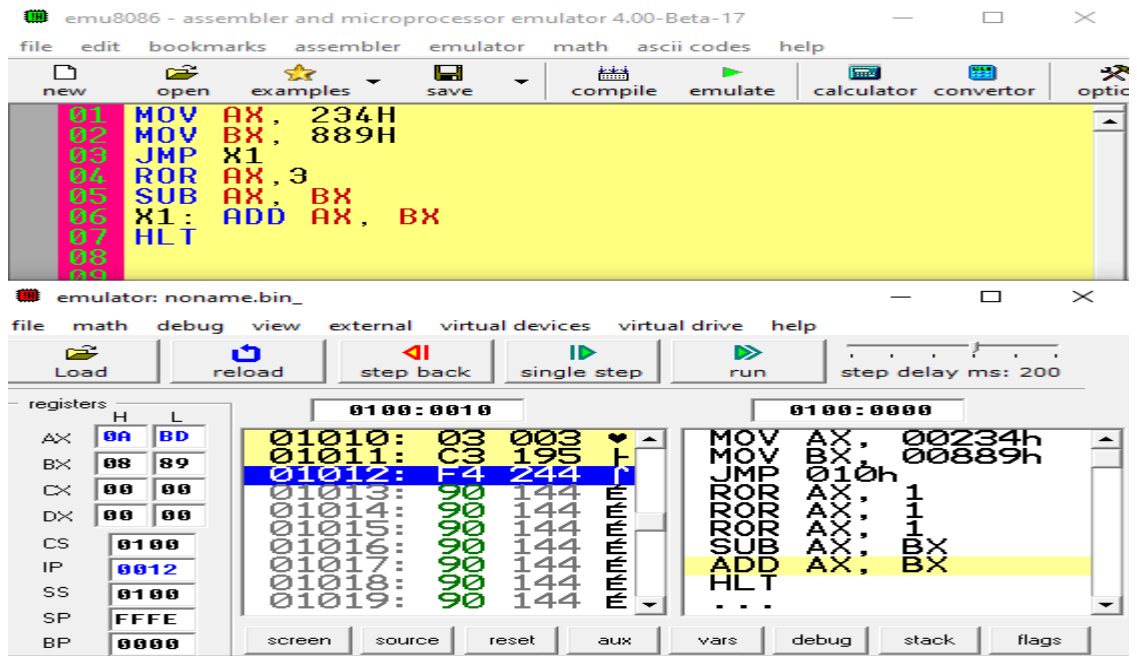
HLT

Answer:

AX= 0ABDH

Microprocessors

لاحظ التنفيذ باستخدام الـ Emulator:



Example (4): What's this program print?

MOV CH, 23H

MOV DH, 88H

X1: ROL CH, 3

JMP X2

SHL CH, 2

SUB CH, DH

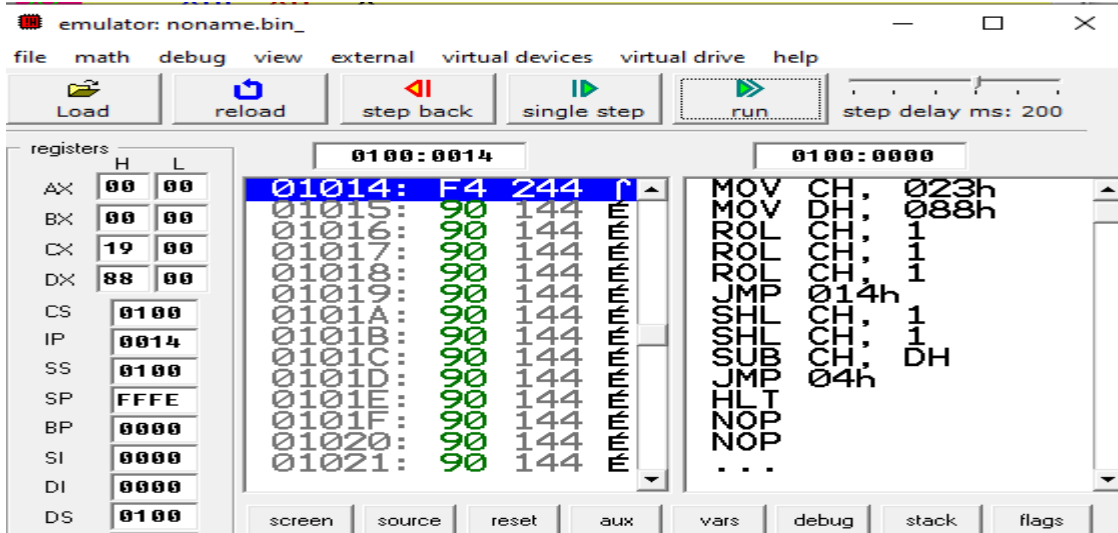
JMP X1

X2: HLT

Answer:

CH=19h

Microprocessors



- **Comparison instruction (CMP)**

The comparison instruction (CMP) is a subtraction between two register or between register and memory location which the destination operand never changes. A comparison is useful for checking the entire contents of a register or a memory location against another value, a CMP is normally followed by a conditional jump instruction.

ايغاز المقارنة (CMP) عبارة عن طرح محتويات سجلين او طرح محتويات سجل مع موقع ذاكرة حيث ان معامل الوجهة او الهدف لا يتغير أبداً. تعد المقارنة مفيدة لفحص المحتويات الكاملة لسجل أو موقع ذاكرة مقابل قيمة أخرى، عادةً ما يتبع CMP تعليمات قفزة مشروطة.

Format: CMP dest., source

الاية عمل الامر CMP هو يتم طرح المصدر من الهدف ولكن هنا لا يوضع النتيجة في الهدف حيث يبقى الهدف دون تغيير, وهذا هو الاختلاف بين الـ CMP والـ SUB. والـ destination ممكن ان يكون سجل او موقع ذاكرة والـ source ممكن ان يكون سجل او موقع ذاكرة او قيمة فورية.

Examples:

- i) `CMP CL, BL` ; CL-BL
- ii) `CMP AX, SP` ; (طرح يستخدم للمقارنة فقط بدون تغيير محتوى السجلين) AX-SP
- iii) `CMP [DI], CH` ; CH subtracts from the byte contents of the data segment memory location addressed by DI .

Example:

`MOV AL, 9`

R1: `MOV DH, 44H`

`CMP AL, 10H` ; compare AL against 10H

`JNZ R1` ; ZF=1 if AL لايساوي 10H jump to location R1

`MOV DH, 50H`

`HLT`

2. Conditional Jump

For a conditional jump instruction, status conditions that exist at the moment the jump instruction is executed decide whether or not the jump will occur. If this condition or conditions are met, the jump takes place, otherwise execution continues with the next sequential instruction of the program. The conditions that can be referenced by a conditional jump instruction are status flags such as carry (CF), parity (PF), and overflow (OF).

للحصول على تعليمات القفز الشرطي ، يتم عن طريق شروط الحالة التي تكون موجودة في اللحظة التي يتم فيها تنفيذ تعليمات الانتقال تقرر ما إذا كان هنالك قفزة أم لا. إذا تم استيفاء هذا الشرط أو الشروط ، تأخذ القفزة مكان معين داخل البرنامج ، وإلا يستمر التنفيذ مع التعليمات التسلسلية التالية من البرنامج. الشروط التي يمكن الرجوع إليها بواسطة شرط تعليمات القفز هي أعلام الحالة (status flag) مثل (CF) ، (PF) ، (OF).

ينقل التحكم في الـ conditional jump الى موقع جديد سواء الى اعلى البرنامج او الى اسفل البرنامج اذا تحقق شرط القفز. اما اذا لم يتحقق شرط القفز فان القفز لن يتم وسيستمر البرنامج في مساره الطبيعي. ان شرط القفز توضع دائما على الاعلام .

Microprocessors

الجدول ادناه يوضح الاعلام التي لها علاقة مع القفز المشروط:

Opcode	Operand	Description
JC	address	Used to jump if carry flag CY = 1
JNC	address	Used to jump if no carry flag (CY = 0)
JE/JZ	address	Used to jump if equal/zero flag ZF= 0
JNE/JNZ	address	Used to jump if not equal/zero flag ZF= 1
JO	address	Used to jump if overflow flag OF = 1
JNO	address	Used to jump if no overflow flag OF = 0
JP/ JPE	address	Used to jump if parity/parity even PF = 1
JNP/ JPO	address	Used to jump if not parity/parity odd PF = 0
JS	address	Used to jump if sign flag SF = 1
JNS	address	Used to jump if not sign SF = 0

Example (1) What's this program print?

```
MOV AL, 0FFH
```

```
MOV BX,00
```

```
ADD AL, 1
```

```
JC X1 ; IF CF=1 start jump to address X1 else continue without jump
```

```
MOV BX, 1000H
```

```
NEG BX
```

```
X1: NOT BX
```

```
HLT
```

Answer) BX=FFFF

Example (2) What's contain of AL after executing this program?

MOV AL, 34H

SUB AL, 3

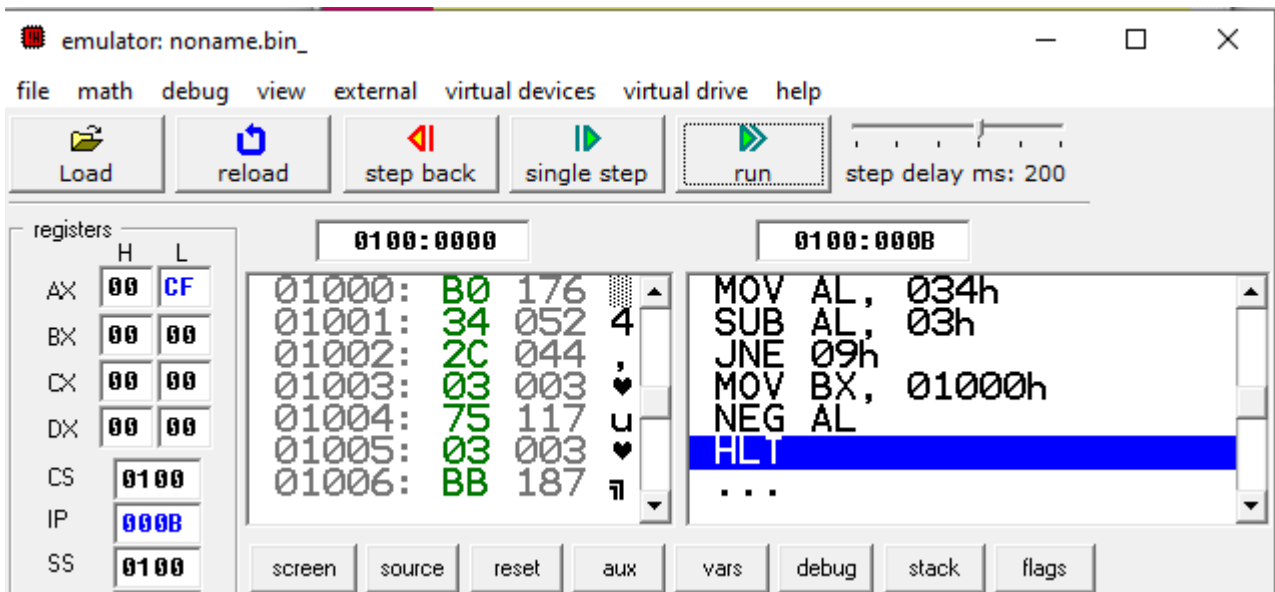
JNZ X1 ; IF ZF=1 start jump to address X1 else continue without jump

MOV BX, 1000H

X1: NEG AL

HLT

Answer) AL=CF



Example(3): Write a program in assembly language to summation of memory locations start with (400H – 409H) in DS= 3000H, then put result in DL.

Sol.) CLC
 MOV AX, 3000H
 MOV DS, AX
 MOV CL, A
 MOV SI, 400H
 MOV AL, [SI]
Loop: ADC AL, [SI+1]
 INC SI
 DEC CL
 JNZ loop
 MOV DL, AL
 HLT

Microprocessors

Example (4): Write a program in assembly language to store value ff in even locations start with (700H – 709H).

SOL.)

MOV SI, 700H

MOV AL, 0FFH

MOV CL, 5; CL is counter for memory locations .

Loop: MOV [SI], AL

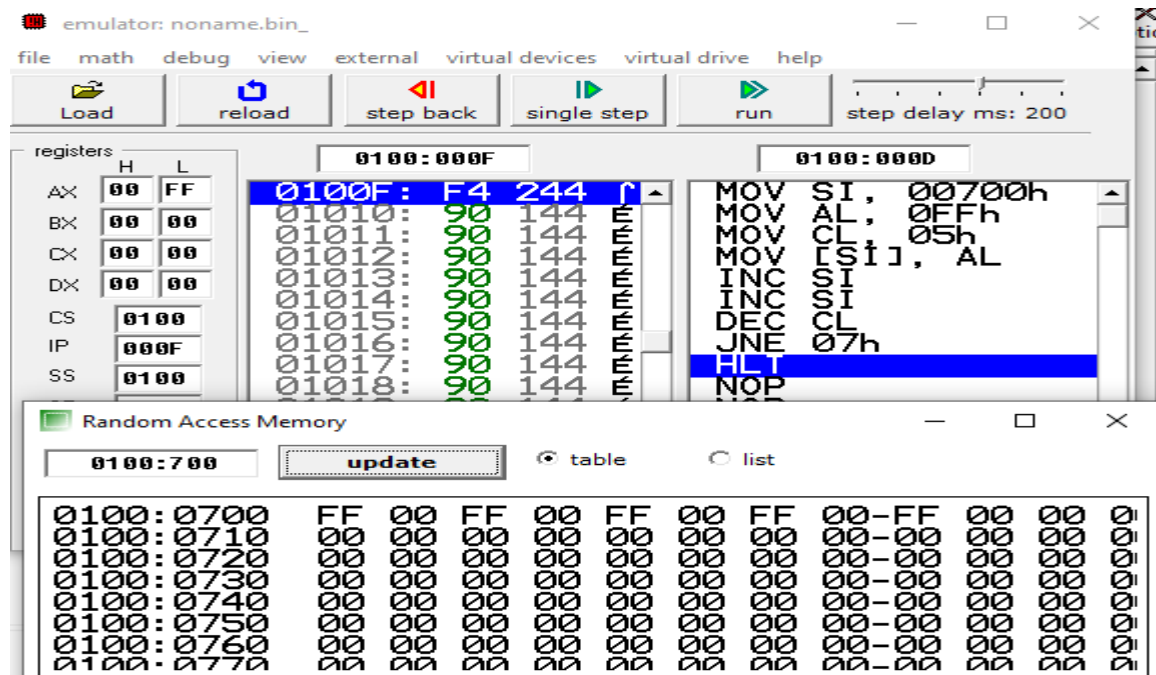
INC SI } increase memory locations twice to get even locations.

INC SI }

DEC CL

JNZ loop; if ZF=1 jump

HLT



Example (5): Write a program in assembly language to compute the one's number in BX register, if BX= 5555H.

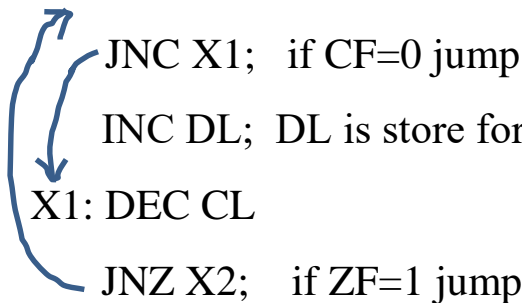
Solution)

MOV CL, 10H

MOV DL, 0

MOV BX, 5555H

X2: SHR BX, 1

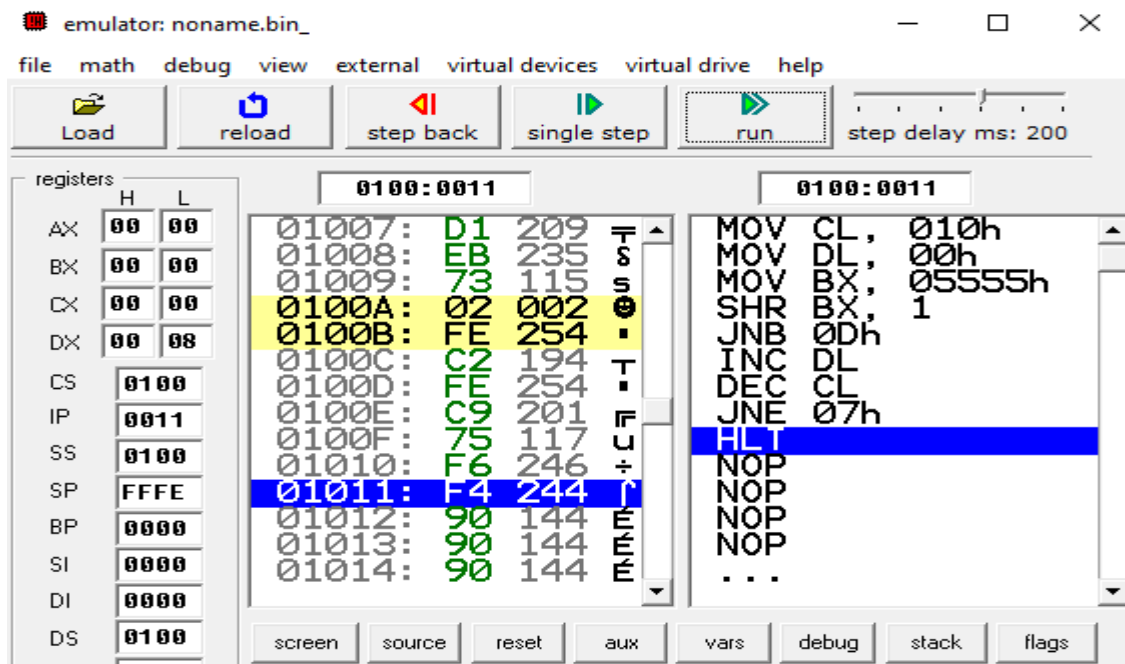


INC DL; DL is store for one's

X1: DEC CL

JNZ X2; if ZF=1 jump

HLT



Not: JNB (Jump if not below) when CF=0 and ZF = 1

Example (6): Write a program in assembly language to multiply memory locations (700H – 70FH) by 4, Using shift instructions.

Answer:

```
MOV CL, 10H
```

```
MOV SI, 700H
```

```
X1: SHL [SI], 2
```

```
INC SI
```

```
DEC CL
```

```
JNZ X1
```

```
HLT
```

Example (7): Write a program in assembly language to store even numbers (0 – 12H in memory locations start with (800H- 809H), if DS= 900H.

Answer) MOV AX, 900H

```
MOV DS, AX
```

```
MOV DI, 800H
```

```
MOV CL, 0AH
```

```
MOV DL, 0
```

```
again: MOV [DI], DL
```

```
INC DI
```

```
ADD DL, 2
```

```
DEC CL
```

```
JNZ again
```

```
HLT
```

Microprocessors

بعد تنفيذ البرنامج باستخدام الـ emulator لاحظ خزن الارقام الزوجية في عشرة مواقع داخل الذاكرة:

The top window displays the following assembly code:

```

0001 MOV AX, 900H
0010 MOV DS, AX
0021 MOV DI, 800H
0100 MOV CL, 0AH
0101 MOV DL, 0
0110 again: MOV [DI], DL
0111 INC DI
1000 ADD DL, 2
1001 DEC CL
1010 JNZ again
1011 HLT
    
```

The bottom window shows the emulator's state. The registers window displays:

registers	H	L
AX	00	00
BX	00	00
CX	00	00
DX	00	00
CS	0100	
IP	0000	
SS	0100	
SP	FFFE	
BP	0000	
SI	0000	

The disassembled instruction window shows the following instructions:

```

0100: B8 184 MOV AX, 00900h
0101: 00 000 MOV DS, AX
0102: 09 009 MOV DI, 00800h
0103: 8E 142 MOV CL, 0Ah
0104: D8 216 MOV DL, 00h
0105: BF 191 MOV [DI], DL
0106: 00 000 INC DI
0107: 08 008 ADD DL, 02h
0108: B1 177 DEC CL
0109: 0A 010 JNE 0Ch
010A: B2 178 HLT
010B: 00 000 ...
    
```

Random Access Memory

900:800 update table list

0900: 0800	00	02	04	06	08	0A	0C	0E	-10	12	00	00
0900: 0810	00	00	00	00	00	00	00	00	-00	00	00	00
0900: 0820	00	00	00	00	00	00	00	00	-00	00	00	00
0900: 0830	00	00	00	00	00	00	00	00	-00	00	00	00
0900: 0840	00	00	00	00	00	00	00	00	-00	00	00	00
0900: 0850	00	00	00	00	00	00	00	00	-00	00	00	00
0900: 0860	00	00	00	00	00	00	00	00	-00	00	00	00
0900: 0870	00	00	00	00	00	00	00	00	-00	00	00	00