

Lecture 1: Introduction to 8086 Mp

- A microprocessor (Mp) is a programmable controller on a chip.

The first Mp in the world, Intel 4004 was invented in 1971. 4004 Mp has:

- A memory of 4096 locations, each has 4-bits width.
- A speed of 50 KIPS (Kilo Instruction Per Second).
- An instruction set that contains 45 instructions.

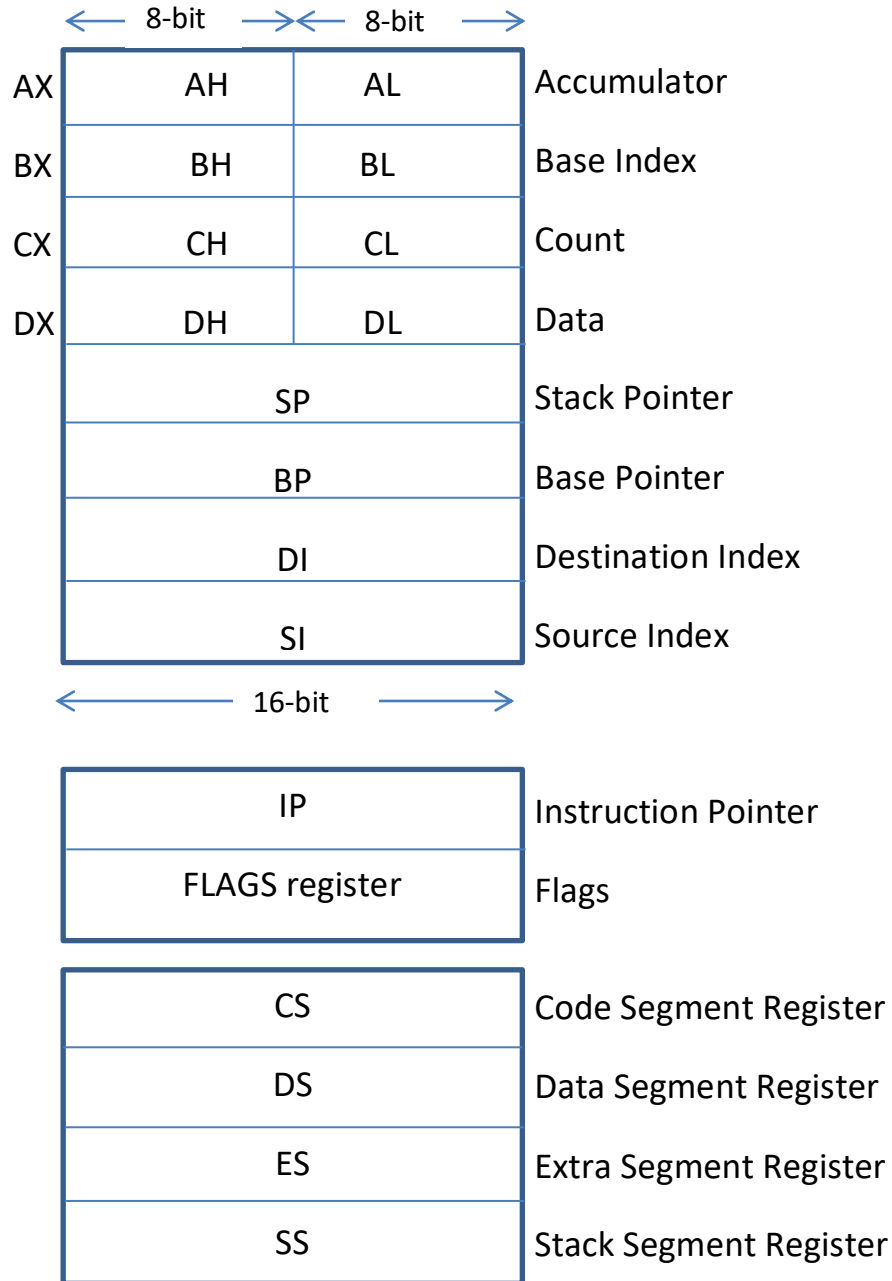
After 4004, Microprocessors gradually developed in the memory size, speed and no. of instructions.

- 8086 Mp was released by Intel in 1978. 8086 Mp has:
 - A memory of 1 Mega locations, each has 8-bits width.
 - A speed of 2.5 million KIPS (Kilo Instruction Per Second).
 - An instruction set that contains over 20,000 instructions.
 - A cash memory to speed up instructions execution.
 - A 16-line data bus.
 - A 20-line address bus.

المعالج هو رقاقة قابله للبرمجة تستخدم لعمليات متعددة, منها قراءة ادخال, معالجة معلومات, القيام بحسابات متنوعة, التخاطب والسيطرة على اجهزة معينة, ... الخ.

سبب دراسة معالج 8086 (مع انه لايقارن مع الامكانيات الموجودة في المعالجات الحديثة) هو انه نموذج مصغر للمعالجات الموجودة في الوقت الحاضر, حيث ان التطور الذي حصل هو في حجم الذاكرة وسرعة المعالج وعدد الايعازات , أما الاساس فهو ذاته.

The 8086 Mp Programming Model (or Software Model)



الرسم أعلاه يمثل نظرتنا الى المعالج , حيث ينظر المبرمج الى المعالج على أنه مجموعة من السجلات المتنوعة والتي يتم التحكم بالمعالج أو مخاطبة المعالج من خلالها باستخدام الايعازات, وستكون الايعازات الخاصة بمعالج الـ 8086 موضوع مادة المعالجات خلال الفصل الاول ان شاء الله.

- 8-bit registers are:
AH, AL, BH, BL, CH, CL, DH and DL.
- 16-bit registers are:
AX, BX, CX, DX, SP, BP, DI, SI, IP, FLAGS, CS, DS, ES and SS.
- Multipurpose registers are:
AX, BX, CX, DX, BP, DI and SI.
- Special purpose registers are:
IP, SP, FLAGS, CS, DS, ES and SS.

IP: addresses the next instruction in the code segment.

SP: addresses an area of memory called the stack.

CS: defines the starting address of the section of memory holding code.

DS: points to the starting address of the section of memory holding the data used by a program.

ES: additional data segment.

SS: points to the starting address of the stack segment.

Segment-Offset Address Combinations

segment	offset	Special purpose
CS	IP	Instruction address
SS	SP or BP	Stack address
DS	BX, DI, SI, 8-bit or 16-bit number	Data address
ES	---	---

الجدول اعلاه ذو اهمية بالغة حيث سيتم الرجوع اليه بشكل دائم لان المبرمج يجب ان يعلم السجل الذي يتم استخدامه كمؤشر (offset) عند التعامل مع أي من الـ segments الاربعة.

Important notes:

1 Byte = 8-bits

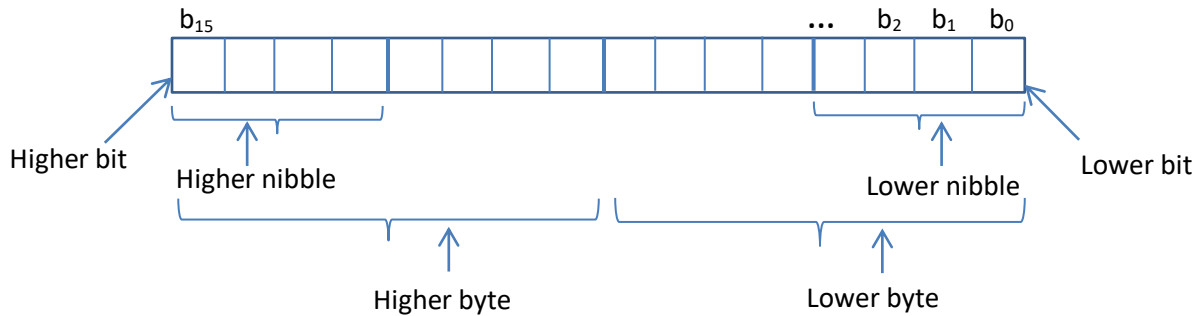
Word = 2 Bytes = 16-bits

Double word = 2 words = 4 Bytes = 32-bits

Nibble = 4-bits (1 Byte= 2 nibbles)

1 K Byte = 2^{10} Byte = 1024 Byte

1 M Byte = 2^{10} K Byte = $2^{10} * 2^{10}$ Byte



Requirements:

It is required from students at this level to understand the following:

- Binary numbering
- Hexadecimal numbering
- Converting between Binary, Hexadecimal and Decimal numbering systems.
- The difference between signed and unsigned numbers.

مطلوب من الطلبة مراجعة النقاط أعلاه قبل البدء في مادة المعالجات الدقيقة, والتي سبق وتم دراستها في مادة التقنيات الرقمية في الصف الأول.

HW:

1) Count in Hexadecimal from **0** to **100H** (Ascending)

2) Count in Hexadecimal from **400H** down to **360H** (Descending)

Reference text books:

- 1) "The Intel Microprocessors", by: Barry B. Brey
- 2) "The 8088 and 8086 Mp's programming, interfacing S/W, H/W and Applications", by: W. A. Triebel and A. Singh.

Lecture 2: Logical and Physical Addressing

Logical Addressing

- 8086Mp has a 1 MByte memory locations; each size of each location is 8-bits. In order to reach any location in this memory, we need to know the logical address of that location.
- 8086Mp memory is *logically* divided into segments. The size of each segment is 64 Kbyte.
- A programmer can reach the 8086Mp memory only by using logical addressing. Logical Address is written as follows: **(Segment : Offset)**
- The segment part of the address is fixed for all the memory locations that are in this segment (which is 1000 H in the below example), while the offset value changes from 0000 H to FFFF H for each segment.
- Offset is like a pointer that points to a specific location inside the segment.
- The segment value can only be stored in one of the segment registers while the offset is stored in one of the offset registers associated with that segment (go back to the segment-offset combinations table given in the previous lecture).

لا يمكن للمبرمج ان يصل الى اي موقع من الذاكرة ويتعامل معها الا من خلال العنونة المنطقية. يتكون العنوان المنطقي من قسمين, الاول يحمل عنوان الـ (Segment) والثاني يحمل قيمة المؤشر الذي يؤشر الى موقع محدد في داخل هذا الـ Segment. كل Segment يحتوي على 64 Kbyte من المواقع وذلك يتطلب ان يكون طول السجل الذي يحتوي قيمة الـ segment يساوي 16-bit. يتم تخزين قيمة الـ segment في احد السجلات الاربعة (CS, DS, ES, SS) كما سيأتي لاحقاً. أما قيمة المؤشر فهي تتراوح من (0000 H) لأول موقع من الـ segment الى (FFFF H) لآخر موقع منه. فاذا كانت قيمة الـ segment تساوي مثلاً 6A00 H فإن العنوان المنطقي لأول موقع من الـ segment هو (6A00 H : 0000 H) والعنوان المنطقي لآخر موقع من الـ segment هو (6A00 H : FFFF H).

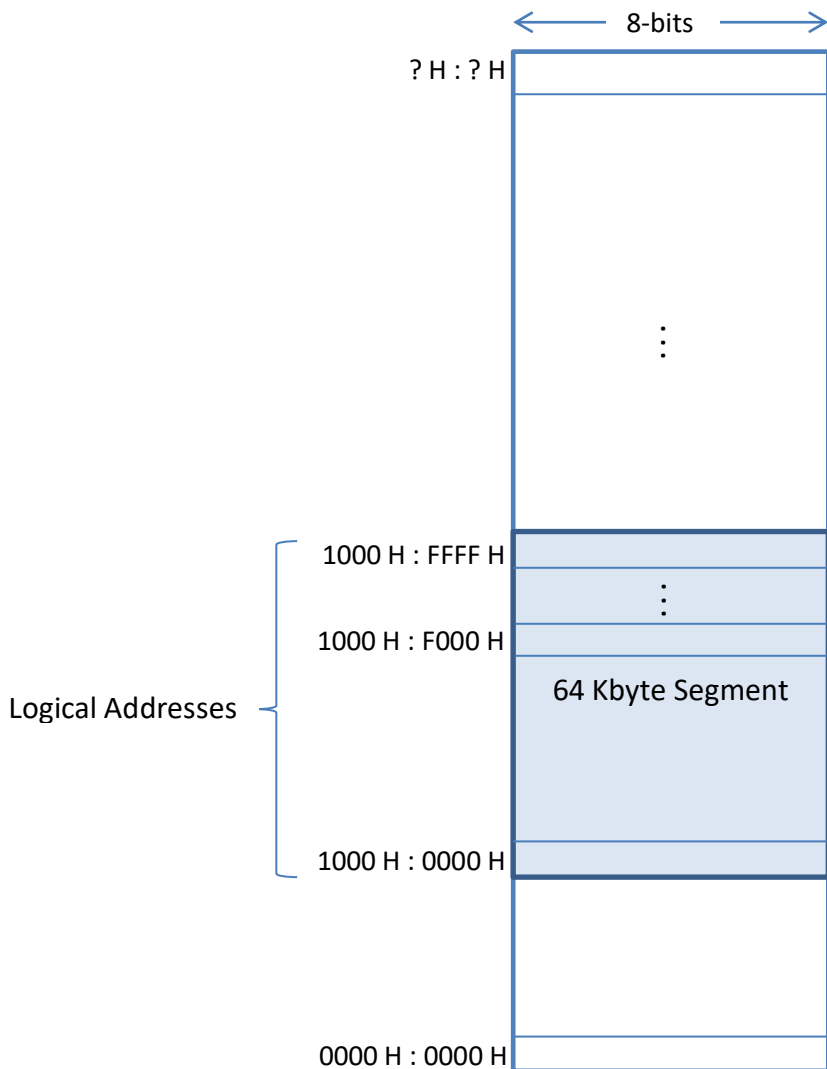
بما أن المؤشر (offset) طوله يساوي 16-bit فمعناه ان حجم الـ segment الواحد يكون 64 Kbyte (لان الحجم الكلي الذي يمكن عنونته باستخدام 16 بت هو 2 للأس 16 وهو يساوي 64Kbyte).

المثال المعطى في الرسم التالي يوضح مقطعاً من الذاكرة (المحدد باللون الازرق الفاتح), وقد تم تأشير العناوين المنطقية عليه. يتم كتابة العنوان سواء المنطقي او الفيزيائي الى جانب الذاكرة على اليمين أو اليسار أما ما يكتب في داخل موقع الذاكرة فيعتبر محتوى الموقع (أي الـ data) وليس عنواناً ويكون طوله 8-bits.

ملاحظة 1: عند رسم الذاكرة نقوم باعتبار اول موقع منها هو الموجود في الاسفل واخر موقع منها هو الموجود في الاعلى. وذلك للمشي مع الطريقة المستخدمة في الكتاب المنهجي رقم 1 المعطى في المحاضرة السابقة. علماً ان بعض المصادر تستخدم عكس ذلك.

ملاحظة 2: يتم استخدام النظام السادس عشر في كتابة العناوين المنطقية والفيزيائية, والذي يفترض ان الطالب يعرفه بشكل جيد.

ملاحظة 3: تم كتابة الموقع الاخير للذاكرة بشكل علامات استفهام لان الطالب سيفهم كيفية ايجاده بعد ان يتم التطرق الى العنونة الفيزيائية.



Physical Addressing (PA):

(Some references call it: effective address (EA))

In order to access any location in the 1 Mbyte memory, the Mp must generate a 20-bit memory address, which is called a Physical Address (PA). Physical address is calculated from the logical address as follows:

$$\text{Physical Address (PA)} = \text{Segment} * 16 + \text{offset}$$

where multiplying the segment by 16 is simply achieved by shifting it to the left and adding a zero to the lower digit.

العنوان الفيزيائي هو العنوان الحقيقي والذي يتكون من رقم طوله عشرين بت (لأن حجم الذاكرة الكلي هو 1 MByte). ويقوم المعالج بحساب العنوان الفيزيائي المطلوب التعامل معه من خلال العنوان المنطقي، وذلك بإضافة صفر إلى الـ segment (والذي يعادل ضرب في 16) ثم الجمع مع قيمة المؤشر offset. فالمفروض أن نحصل على رقم يحتوي

على 5 hexa digits (لان كل اربعة بت تمثل 1 hexa digit) وتتراوح قيمة العنوان الفيزيائي بين (00000 H) لاول عنوان من الذاكرة, الى (FFFFFF H) لآخر عنوان منها.

Example: Find the PA of the next instruction, if CS =1000 H, DS =A000 H, ES = A80 H, IP =2000 H.

Solution:

The logical address of the next location is stored in (CS: IP), so that DS and ES will not be required to calculate the PA of the next instruction.

يقوم المعالج عند تنفيذ اي برنامج بخزن عنوان الابعاز التالي الذي سيقوم بتنفيذه في العنوان المنطقي (CS: IP). نلاحظ أن قيم السجلات الاخرى الموجودة في السؤال غير داخلة في الحل وهي معطاة لغرض التأكد من فهم الطالب للموضوع.

$$\begin{array}{r} \text{PA} = 1\ 0\ 0\ 0\ 0 \\ \quad \quad \quad 2\ 0\ 0\ 0 \quad + \\ \hline \quad \quad \quad 1\ 2\ 0\ 0\ 0\ \text{H} \end{array}$$

Example: If a segment register value is (2100H), what are the starting and ending physical addresses of this segment?

Solution:

- For the starting address, the offset value is (0000 H), so PA is:

$$\begin{array}{r} \text{PA} = 2\ 1\ 0\ 0\ 0 \\ \quad \quad \quad 0\ 0\ 0\ 0 \quad + \\ \hline \quad \quad \quad 2\ 1\ 0\ 0\ 0\ \text{H} \end{array}$$

- For the ending address, the offset value is (FFFF H), so PA is:

$$\begin{array}{r} \text{PA} = 2\ 1\ 0\ 0\ 0 \\ \quad \quad \quad \text{F}\ \text{F}\ \text{F}\ \text{F} \quad + \\ \hline \quad \quad \quad 3\ 0\ \text{F}\ \text{F}\ \text{F}\ \text{H} \end{array}$$

Example: If SS =A022 H, DI =F019 H, BP =2E60 H, IP =1111 H, find the PA of the stack.

Solution:

The stack is addressed using the logical address (SS: BP) or (SS: SP). The value of SP is not given in the question, so that BP is used, so PA is:

الـ segment الخاص بالـ stack مخزون في الـ SS ومن خلال الرجوع الى الجدول المعطى في المحاضرة السابقة, نعلم بان المؤشرات المستخدمة معه هي SP و BP فقط. وبما ان قيمة SP غير معطاة في السؤال فنستخدم BP.

$$\begin{array}{r} \text{PA} = \text{A } 0 \text{ 2 2 } 0 \\ + \\ \text{2 E 6 0} \\ \hline \text{A 3 0 8 0 H} \end{array}$$

Example: Find a logical address that can take you to the location (255AF H) of the memory.

Solution: There are many logical addresses that lead to the given PA:

عملية ايجاد العنوان المنطقي من الفيزيائي هي مشابهة لاجاد رقمين حاصل جمعهما يساوي رقماً معيناً أي أن هنالك العديد من الحلول, ولكن يجب التأكد من صحة الرقمين في العنوان المنطقي من خلال وضع صفر الى عنوان الـ segment وجمعه مع الـ offset فنحصل عندها على العنوان الفيزيائي الذي يجب ان يطابق العنوان الفيزيائي المعطى.

255A H: 000F H

or

2000 H: 55AF H

or

2500 H: 05AF H

(and many other combinations)

More examples can be found in [Ref. 1, table 2-1]

Best Regards

Dr. Zainab Alomari

Lecture 3: Addressing Modes

8086Mp has a group of instructions that are used by the programmer to specify the task required to be done by the Mp. The language used for programming 8086Mp is called (Assembly Language).

(MOV) instruction is the most used 8086Mp instruction, and it is used as follows:

MOV AX, BX



In this example:

- AX is the **destination**
- BX is the **source**
- The value of BX will be moved (copied) to AX
- The old value of AX will be removed and now AX=BX
- BX will not be changed

In all instructions:

- The source and destination **MUST** have the same length.
- The destination is called so because the result of the operation performed by the instruction is always stored in the destination.
- The source is **NOT** changed by the instruction.

يتم التعامل مع المعالج من خلال اللغة الخاصة به (والتي سيتم تعلم أساسياتها خلال الفصل الأول), ويكون ذلك من خلال النظر الى المعالج على انه مجموعة من السجلات التي يمكن استخدامها للقيام بعمليات متنوعة.

الايجاز الابسط والاكثر استخداماً هو (MOV) وهو ايعاز نقل او نسخ قيمة الـ (source) الى الـ (destination). وسيتم استخدام هذا الايعاز لشرح الانواع المختلفة من العنونة (Addressing Modes).

Addressing Modes

1) Register Addressing Mode

في هذا النوع يكون كل من الـ (source) والـ (destination) عبارة عن سجل (register) ويجب ان يكونا بنفس الطول (إما كلاهما 8بت أو كلاهما 16بت).

MOV Register, Register

في الجدول التالي مجموعة من الامثلة على هذا النوع من الايعازات.

<i>Assembly Language</i>	<i>Size</i>	<i>Operation</i>
MOV AL,BL	8-bits	Copies BL into AL
MOV CH,CL	8-bits	Copies CL into CH
MOV AX,CX	16-bits	Copies CX into AX
MOV SP,BP	16-bits	Copies BP into SP
MOV DS,AX	16-bits	Copies AX into DS
MOV SI,DI	16-bits	Copies DI into SI
MOV BX,ES	16-bits	Copies ES into BX
MO / ES,DS	—	Not allowed (segment-to-segment)
MOV BL,DX	—	Not allowed (mixed sizes)
MOV CS,AX	—	Not allowed (the code segment register may not be the destination register)

هنالك حالات يمنع استخدامها وهي الحالات التالية:

- 1- أن يكون طول الـ (source) والـ (destination) غير متساوي (كما في المثال قبل الأخير من الجدول السابق).
- 2- أن يكون كلا السجلين (أي كل من الـ (source) والـ (destination)) هو من الـ (segment registers) الأربعة وهي (CS, DS, ES, SS) (كما في المثال 8 من الجدول السابق), والسبب أنها سجلات خاصة واستخدامها في الايعازات يكون بشكل محدود فقط. أما اذا كان أحد السجلين فقط هو من الـ (segment registers) فهذا ممكن ما عدا حالة واحدة وهي أن يكون الـ CS هو الـ (destination) فهذا لا يجوز (كما في المثال الأخير من الجدول السابق) والسبب أنه يحتوي قيمة مهمة تمثل عنوان الـ segment الذي يحتوي على الايعازات في الذاكرة وفي حالة تغيير قيمته يتم فقدان البرامج.

ملاحظة: جميع السجلات لا يمكن تجزئتها في الايعازات وانما يتم التعامل معها على انها 16بت دائماً, ما عدا (AX, BX, CX, DX) حيث يمكن تجزئة كل منهم الى جزئين من 8بت (AL, AH, BL, BH, CL, CH, DL, DH).

2) Immediate Addressing Mode

MOV Register , NUMBER

في هذا النوع يكون الـ (source) رقماً وليس سجلاً ويكون بنفس طول الـ (destination) إما 8بت أو 16بت. ويعطى الرقم إما بالنظام السادس عشري (Hexadecimal) وذلك بأن يكتب في آخره الحرف H (كما في المثال 2 من الجدول أدناه), أو بالنظام الثنائي (Binary) وذلك بان يكتب في آخره الحرف B (كما في المثال الأخير من الجدول أدناه), أو بالنظام العشري (decimal) وذلك بان لا يكتب في آخره اي شيء (كما في الأمثلة 1 و3 و4 من الجدول أدناه). أما اذا كان رمزاً وليس رقماً فيتم وضعه داخل علامتي (' ') وهنا يتم اخذ قيمة الـ (ASCII) الخاصة بهذا الرمز (وهي عبارة عن رقم طوله 8بت يقابل الرمز المعطى بين علامتي الاقتباس) ويتم اعطاؤه للـ destination والذي يجب ان يكون سجلاً بطول 8بت (كما في المثال 5 من الجدول أدناه). واذا تم وضع رمزين داخل علامتي الاقتباس معناه سيتم أخذ 16بت (8بت لكل رمز) ووضعها في الـ destination الذي يجب ان يكون سجلاً بطول 16بت (كما في المثال 6 من الجدول أدناه).

<i>Assembly Language</i>	<i>Size</i>	<i>Operation</i>
MOV BL,44	8-bits	Copies a 44 decimal (2CH) into BL
MOV AX,44H	16-bits	Copies a 0044H into AX
MOV SI,0	16-bits	Copies a 0000H into SI
MOV CH,100	8-bits	Copies a 100 decimal (64H) into CH
MOV AL,'A'	8-bits	Copies an ASCII A into AL
MOV AX,'AB'	16-bits	Copies an ASCII BA into AX
MOV CL,11001110B	8-bits	Copies a 11001110 binary into CL

Examples: MOV AL, 15H (AL=15H)

MOV BX, 6F0AH (BX=6F0AH) or (BL=0AH and BH=6FH)

MOV CX, 1 H (CX=1H) or (CL=01H and CH=00H)

MOV 1FH, AL
MOV 47H, DH
MOV 55, DX

NOT ALLOWED ☹️ (Destination CAN NOT be a number!)

نلاحظ أنه لا يمكن ان يكون الـ destination رقماً لأن عملية النقل تتم اليه, فهي عملية غير منطقية. وكذلك الحال اذا كان كل من الـ source والـ destination عبارة عن ارقام فهذا ايضاً لا يجوز.

MOV DS, 100H NOT ALLOWED ☹️

It is NOT allowed to use immediate addressing mode with segment registers (CS, DS, ES, SS). In order to put 100H in DS, we can do the following:

MOV AX, 100H

MOV DS, AX ALLOWED 😊

لا يمكن استخدام هذا النوع من الـ segment registers مع الـ segment registers الاخرى, وانما نقوم باستخدام احد السجلات الاخرى كمخزن وسطي لنقل القيمة باستخدام النوع الاول (register addressing mode) وهذه الطريقة يمكن استخدامها لتغيير قيمة اي من الـ segment registers ما عدا الـ CS كما ذكر سابقاً.

3) Direct Addressing Mode

MOV destination , [NUMBER]

MOV [NUMBER], source

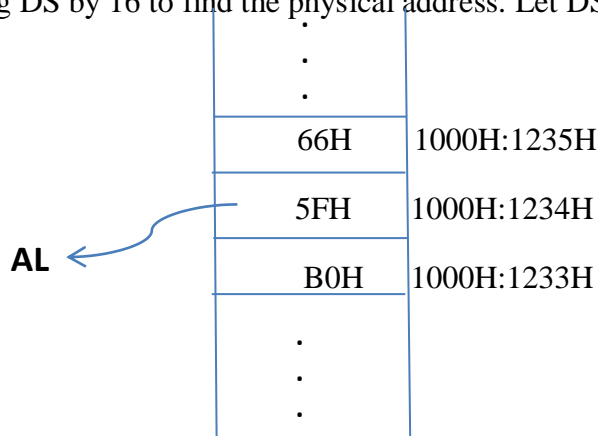
في هذا النوع يتم التعامل مع الذاكرة واستخدامها كـ source أو كـ destination, حيث يتم استخدام اقواس مربعة للوصول الى الذاكرة ويتم وضع رقم مابين هذه الاقواس ليساعدنا على الوصول الى العنوان المطلوب.

Example 1:

MOV AL , [1234H]

In this example, an 8-bit value is taken from a memory location and copied to AL.

How can the Mp calculate the physical address and go to the correct memory location? The physical address is calculated from (**segment : offset**). The number given in the instruction inside square brackets is the **offset**. When the offset is a number, then the segment is always **DS**. The Mp will take the offset from the instruction which is 1234H and adds it with DS value after multiplying DS by 16 to find the physical address. Let DS=1000H, then:



(After execution: AL=5FH)

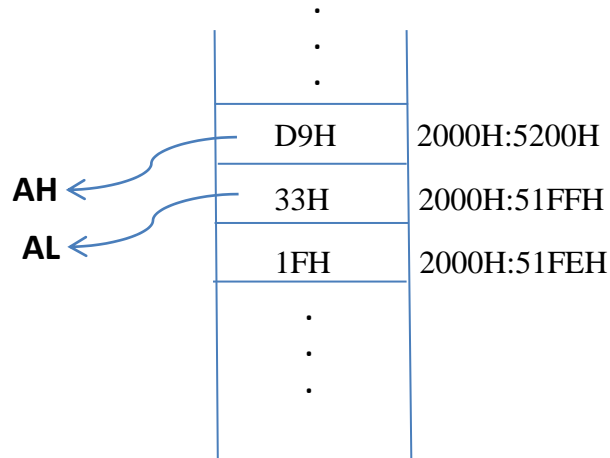
كيفية حساب العنوان الفيزيائي المطلوب الذهاب اليه من الذاكرة:

يتم استخدام القيمة الموجودة بين الاقواس المربعة حيث تمثل الـ offset ثم تجمع مع قيمة الـ DS بعد ضربه في 16, كما في المثال السابق.

Example 2:

MOV AX, [51FFH]

In this example, 16-bits will be moved from the memory to AX. The memory PA will be calculated in the same way as in example 1. This time two bytes are required to be copied from the memory to AX, so that: the first byte (lower byte) of AX (which is AL) will be taken from the specified memory location and the second byte (higher byte) of AX (which is AH) will be taken from the next memory location. Let DS=2000H, then:



After execution: AX=D933H.

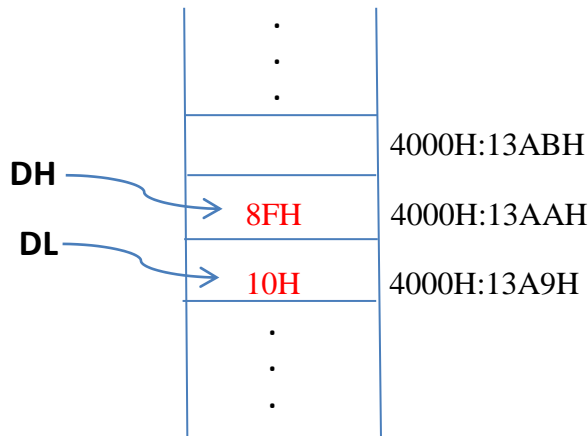
ملاحظة: القيم المخزونة في الذاكرة هنا هي مجرد مثال وتعطى في السؤال.

Note: in the two previous examples, the memory location was the source so that it is not changed; only the destination is changed.

Example 3:

MOV [13A9H], DX

In this example, two bytes will be copied from DX to the memory location at offset=13A9H in DS. Let DS=4000H and DX=8F10H, then:



(after execution DX is not changed, only two memory locations are changed)

MOV [4F00H], [651BH] **NOT ALLOWED** (memory to memory is not allowed) ☹

MOV [3A88H], 5H This is allowed **BUT**: the length of the moved data is not clear (16-bit or 8-bits), therefore we need to specify this length using either (byte ptr) or (word ptr), as follows:

MOV byte ptr[3A88H], 5H One memory location at DS:3A88H will be given the value 05H

MOV word ptr[3A88H],5H Two memory locations will be changed as follows:

DS:3A88H will be given the value (05H)

DS:3A89H will be given the value (00H)

MOV 5H, [3A88H] **NOT ALLOWED** (destination can NOT be a number) ☹

MOV DS, [3000H] **NOT ALLOWED** (segment registers can NOT be used with memory) ☹

4) Indirect Addressing Mode

في هذا النوع يتم اعطاء offset ليس بشكل رقم وانما بشكل سجل (register) قيمته تمثل offset لموقع الذاكرة المطلوب, حيث يوضع هذا السجل بين اقواس مربعة.

MOV Destination, [Register]

MOV [Register], Source

Examples: MOV AX, [BX]
 MOV [DI], AL
 MOV [BP], BX
 MOV DX, [SI]

هناك سجلات محددة يمكن ان تستخدم ك-offset وهي موجودة في جدول الـ (segment-offset combinations) المعطى في المحاضرة الاولى. وحسب هذا الجدول فان هذه السجلات هي (BP, BX, SI, DI), أما (SP) فيمكن استخدامه كسجل فقط وليس كمؤشر الى الذاكرة, وسجل الـ IP لا يمكن استخدامه بشكل مباشر مطلقاً في الابعازات لابلشكل سجل ولا بشكل مؤشر.

ملاحظة: عند استخدام السجل register للتأشير على الذاكرة يطلق عليه اسم الـ offset أو هناك تسمية ثانية وهي مؤشر (pointer).

كيفية حساب العنوان الفيزيائي المطلوب الذهاب اليه من الذاكرة:

يتم استخدام قيمة السجل الموجود بين الاقواس المربعة حيث تمثل الـ offset ثم تجمع مع قيمة الـ segment بعد ضربه في 16. يتم معرفة الـ segment من خلال الرجوع الى جدول الـ (segment-offset combinations) المعطى في المحاضرة الاولى, فإذا كان المؤشر المستخدم هو BX أو SI أو DI فسيكون الـ segment هو DS واذا كان المؤشر المستخدم هو BP فسيكون الـ segment هو SS.

Example: Let BX=1000H, DS=100H. Find the value of AX after executing the following instruction (note that the memory is given in the question): **MOV AX, [BX]**

⋮	
77H	02003H
A7H	02002H
34H	02001H
12H	02000H
⋮	

Solution: the Physical Address = DS * 16 + BX

$$PA = 01000$$

$$\begin{array}{r} 1000 + \\ \hline 02000H \end{array}$$

∴ After execution, AX= 3412H

ملاحظة: يمكن تأشير العنوان على الذاكرة بالشكل المنطقي أو الفيزيائي كلاهما صحيح. ولكن في حال تم اعطائه في السؤال بطريقة العنوان الفيزيائية فحتاج الى حساب العنوان الفيزيائي من العنوان المنطقي لنصل للموقع او المواقع المطلوبة من الذاكرة.

ملاحظة: الحالات التالية غير مسموح بها:

MOV [1000], [DI]

MOV [BX], [DI]

MOV [100H], [200H]

NOT ALLOWED (no memory to memory operation) ☹

MOV DS, [DI]

NOT ALLOWED (segment registers can NOT be used with memory) ☹

في الجدول التالي مجموعة من الامثلة المتنوعة على الـ Indirect Addressing Mode.

Assembly Language	Size	Operation
MOV CX,[BX]	16-bits	Copies the word contents of the data segment memory location address by BX into CX
MOV [BP],DL	8-bits	Copies DL into the stack segment memory location addressed by BP
MOV [DI],BH	8-bits	Copies BH into the data segment memory location addressed by DI
MOV [DI],[BX]	—	Memory-to-memory moves are not allowed except with string instructions

5) Register Relative Addressing Mode

في هذا النوع يتم استخدام سجل مع رقم لاعطاء offset داخل الاقواس المربعة.

MOV Destination, [Register + Displacement]

MOV [Register + Displacement], Source

where the displacement is any 16-bit number.

كيفية حساب العنوان الفيزيائي المطلوب الذهاب اليه من الذاكرة:

يتم اعتبار كل ما موجود بين القوسين المربعين هو offset وبالتالي يتم جمعه مع الـ segment بعد ضربه في 16. ويمكن معرفة الـ segment من نوع السجل المستخدم بين القوسين من خلال الرجوع الى جدول الـ (segment-offset combinations), فاذا كان السجل المستخدم كمؤشر (offset) هو BX أو DI أو SI فالـ segment هو DS واذا كان السجل المستخدم كمؤشر (offset) هو BP فالـ segment هو SS.

Example: Let BX=100H, DS=200H, find the value of AX after executing the following instruction (the memory is given in the question): **MOV AX, [BX+1000H]**

9AH	03102H
70H	03101H
11H	03100H
F5H	030FFH

Solution:

$$PA = DS * 16 + BX + 1000H$$

$$PA = 2000$$

$$0100$$

$$\underline{1000} +$$

$$03100H$$

∴ After execution, AX = 7011 H

ملاحظة: في بعض المصادر يتم وضع الـ displacement خارج الاقواس المربعة وهذا ايضاً ممكن, مثال:

MOV [BX]+1234H , AL

ملاحظة: في حالة استخدام هذا النوع من العنونة مع رقم فيجب تحديد طول العملية هل هي 8بت أم 16بت, كالتالي:

MOV byte ptr[BX+20H], 4AH One memory location at (DS:BX+20H) will be given the value 4AH.

MOV word ptr[BX+20H],5A12H Two memory locations will be changed as follows:

(DS: BX+20H) will be given the value (12H)

(DS:BX+21H) will be given the value (5AH)

MOV byte ptr[BX+20H],5A12H **NOT ALLOWED** (can you know why?) ☹

6) Base-Plus-Index Addressing Mode (or Base-Indexed Addressing Mode)

في هذا النوع يتم استخدام سجلين أحدهما (BP أو BX) والآخر (SI أو DI) لإعطاء الـ offset:

MOV destination , [Base Register + Index Register]

MOV [Base Register + Index Register] , source

where:

Base Registers are: **BX** and **BP**

Index Registers are: **SI** and **DI**

كيفية حساب العنوان الفيزيائي المطلوب الذهاب اليه من الذاكرة:

في كل الحالات يتم استخدام الـ DS كـ segment عند حساب الـ PA, ما عدا حالة واحدة وهي اذا تم استخدام الـ BP فعندها يعتبر الـ SS هو الـ segment عند حساب الـ PA. في الجدول التالي أمثلة على هذا النوع.

<i>Assembly Language</i>	<i>Size</i>	<i>Operation</i>
MOV CX,[BX+DI]	16-bits	Copies the word contents of the data segment memory location address by BX plus DI into CX
MOV CH,[BP+SI]	8-bits	Copies the byte contents of the stack segment memory location addressed by BP plus SI into CH
MOV [BX+SI],SP	16-bits	Copies SP into the data segment memory location addresses by BX plus SI
MOV [BP+DI],AH	8-bits	Copies AH into the stack segment memory location addressed by BP plus DI

Example: Let BX=AF00H
SI = 1FABH
DS=8000H
SP=F43EH
SS=15FFH

Give the memory locations that will be changed after executing the following instruction:

MOV [BX+SI], SP

Solution:

PA= 8CEABH (can you verify this?)

F4H	8CEACH
3EH	8CEABH

Note: if the instruction is modified to: MOV [BP + SI], SP

then the PA is calculated using SS instead of DS.

7) Base Relative-Plus-Index Addressing Mode

في هذا النوع من العنونة يتم استخدام سجلين أحدهما (BP أو BX) والآخر (SI أو DI) لإعطاء offset, بالإضافة الى رقم يضاف اليهما, أي أن هذا النوع هو حاصل دمج النوعين السابقين.

`MOV destination , [Base Register + Index Register + displacement]`

`MOV [Base Register + Index Register + displacement] , source`

where:

Base Registers are: **BX** and **BP**

Index Registers are: **SI** and **DI**

Displacement is any 16-bit number

كيفية حساب العنوان الفيزيائي المطلوب الذهاب اليه من الذاكرة:

في كل الحالات يتم استخدام الـ DS كـ segment عند حساب الـ PA, ما عدا حالة واحدة وهي اذا تم استخدام الـ BP فعندها يستخدم الـ SS كـ segment عند حساب الـ PA. أما قيمة الـ offset فهي قيمة ما موجود بين قوسي الذاكرة وهي قيمة الـ Index Register زائداً قيمة الـ Base Register زائداً قيمة الرقم (displacement).

Example:

Let DS= 1000H, SS= 50FFH, ES= 9A00H, BX= 20H, SI= 10H, find the value of AX after executing the following instruction (the memory is given in the question):

`MOV AX, [BX + SI + 100H]`

67H	10133H
7BH	10132H
ACH	10131H
DH	10130H
81H	1012FH

Solution:

$$PA = DS * 16 + (BX + SI + 100H) = 10130 H$$

$$\therefore AX = AC0DH$$

Note: if the instruction is modified to: `MOV AX, [BP + SI + 100H]`
then the PA is calculated using SS instead of DS as the segment.

Notice that all the tables and some examples are from the reference book (The Intel Microprocessors)- chapter 3.

Best Regards
Dr. Zainab Alomari

Lecture 4:

Exchange Instruction

يقوم هذا الايعاز بتبديل محتويات الـ source الى destination مع بعضهما.

Xchg Register , Register

Register , Memory

Memory , Register

لا يجوز استخدام نوع الـ Immediate addressing mode هنا لان كل من القيمتين الموجودتين بجوار الايعاز سوف تستبدل مع الاخرى.

Xchg Memory , Memory

NOT Allowed ☹

Example

Write a piece of code in Assembly language to exchange the contents of DH and DL.

Solution

Xchg DL, DH

Example

Write a piece of code in Assembly language to exchange the contents of DH and DL WITHOUT using Xchg instruction.

OR: What are the equivalent instructions of Xchg DH, DL?

Solution

MOV AL, DH

MOV DH, DL

MOV DL, AL

Example

Write a piece of code in Assembly language to exchange between the contents of memory location (FF100H) and AH.

Solution (1)

MOV DX, FF00H

MOV DS, DX

Xchg [100H], AH

Solution (2)

MOV DX, FF00H

MOV DS, DX

MOV BX, 100H

Xchg [BX], AH

Solution (3)

MOV DX, FF00H

MOV DS, DX

MOV AL, AH

MOV AH, [100H]

MOV [100H], AL

Notice that solution (1) is the most efficient solution because:

1. It requires less space in the memory compared to the other two solutions.
2. It uses less number of registers to perform the required operation.

نلاحظ ان الحل الاول اكثر كفاءة من الحلين الاخرين وذلك لانه يحتاج عدد اقل من الابعازات وبالتالي مساحة اقل للخرن من الذاكرة ولانه يستخدم عدد اقل من السجلات للقيام بالعملية المطلوبة.

Example

Let DS=1200H and BX= 11AAH, give the new values of all the registers and/or memory locations that are affected by executing the following instruction:

Xchg [1234H], BX

(the memory plot is given in the question)

Solution

We need to find the PA of the memory location:

00H	13235H
FFH	13234H
A7H	13233H

$$PA = DS * 16 + 1234H = 13234H$$

$$\therefore BX_{new} = 00FFH$$

11H	13235H
AAH	13234H
A7H	13233H

Translate Instruction

XLAT instruction has no destination or source.

قبل استخدام هذا الايعاز يتم تجهيز جدول يسمى (Look up table) في الذاكرة وهذا الجدول يحتوي على قيم لها علاقة مع تسلسلها في الجدول, مثلاً: يكون محتوى المواقع من الجدول يساوي تربيع تسلسلها أو قيمة نريد الوصول اليها حسب التسلسل الذي تكون مخزونة فيه. بتعبير اخر: نعطي تسلسل من الجدول ويعطينا الايعاز XLAT محتوى هذا التسلسل, ويتم وضع هذا التسلسل في AL قبل استدعاء هذا الايعاز.

عند تنفيذ هذا الايعاز يقوم المعالج بجمع قيمة كل من BX وAL باعتبارهما offset أما الـ segment فهو DS ثم يتم الذهاب الى موقع معين من الجدول ويتم وضع قيمته في AL, كالتالي:

$$[DS * 16 + BX + AL] \rightarrow AL$$

قبل استدعاء الايعاز XLAT يتم اعطاء BX وDS القيم التي تخص اول موقع في الـ (Look up table), وبالتالي فاننا اذا ثبتنا قيمة BX فان قيمة AL ستعتبر المؤشر بحيث اذا كانت $0=AL$ يتم الذهاب الى اول موقع من الـ (Look up table) واذا $AL=1$ يتم الذهاب الى ثاني موقع وهكذا, والقيمة التي يتم الذهاب اليها تنقل الى AL نفسها.

ملاحظة: صيغة هذا الايعاز هي فقط كلمة (XLAT).

Example

Find the physical address of the memory location that will be used by XLAT instruction if: DS=3000H, BX=100H, AL=3FH.

Solution

$$\begin{array}{r}
 PA= \quad 30000 \\
 \quad \quad 100 \\
 \quad \quad \quad 3F \quad + \\
 \hline
 \end{array}$$

3033FH (so this is the PA of the memory location that contain the value that will be given to AL after executing XLAT instruction).

Example

Write a piece of code in Assembly language to find: $y=x^2$ where $(0 \leq x \leq 15)$, using XLAT instruction. Assume that x is a value stored in DH, and the look up table is stored at 8000H:2100H in the Data Segment.

Solution

```
MOV AX, 8000H
MOV DS, AX
MOV BX, 2100H
MOV AL, DH
XLAT
HLT
```

	.
	.
	.
10H	82104H
09H	82103H
04H	82102H
01H	82101H
00H	82100H

ملاحظة 1: كان من الممكن حل السؤال باستخدام ايعاز الضرب Mul ولكن هذا المثال فقط للتوضيح, فعادة ما يتم استخدام هذا الايعاز عندما لا يكون هنالك علاقة بين العنوان والمحتوى مثلاً ان يكون التسلسل هو ترقيم للطلبة والمحتوى هو درجات لهؤلاء الطلبة.

ملاحظة 2: اقصى طول للـ (Look up table) هو 2^8 أي 256 موقع فقط وذلك لاننا نستخدم AL للتأشير اليها وهو يتكون من 8بت فقط.

Q) Write a piece of code in Assembly language to find $y=2^x$ where $(0 \leq x \leq 7)$, using XLAT instruction, assuming that x is stored at DS:SI while the look-up table is stored starting at DS:BX.

DS=4600H, BX=5000H, SI=FF0AH.

ملاحظة: يمكن ان يكون السؤال بطريقة اخرى بحيث تكون قيم السجلات DS و BX غير معطاة وانما يعطى رسم الذاكرة ومن القيم المؤشرة على العنوان لاول موقع من الـ (Look up table) يمكن معرفة قيمة DS:BX.

Questions:

A) Write a piece of code to do each of the following:

- 1- exchange between AX and the contents of memory locations: 90103H and 90104H.
- 2- exchange between 2 bytes of data stored at (11000H) in the Stack Segment with other two bytes of data stored at (50F06H) in the Data Segment.

B) Give the equivalent instructions of XLAT instruction.

Load Effective Address (LEA)

LEA Register(16-bit) , Memory

توجد صيغة واحدة فقط لهذا الابعاز وهي:

Example

LEA SI, [100H] ; $SI_{new} = 100H$

LEA DI, [BX+DI+ 5H] ; (Let BX=20H, DI=1000H, then: $DI_{new}=1000H+20H+5H=1025H$)

إذا لا يتم الذهاب الى الذاكرة بهذا الابعاز وانما فقط يؤخذ العنوان الموجود ما بين الاقواس.

LEA SI, 100H **NOT ALLOWED** ☹

Load DS and Load ES (LDS and LES)

LDS Register(16-bit) , Memory

LDS instruction loads 2bytes from memory to a 16-bit register, then loads the next 2bytes to DS register.

LES Register(16-bit) , Memory

LES instruction loads 2bytes from memory to a 16-bit register, then loads the next 2bytes to ES register.

Example

Let DS=1200H, find the values of all the registers that are affected by executing the following instruction: (the memory plot is given with the question)

LDS SI, [200H]

Solution

$$\begin{array}{r} 12000 \\ + 200 \\ \hline 12200H \end{array}$$

$\therefore SI_{new} = 0020H, DS_{new} = 1300H$

1BH	⋮
13H	12203H
00H	12202H
00H	12201H
20H	12200H

Notice that in this example, DS_{old} is required to reach the memory and bring the new values of SI and DS.

Notice that:

LDS AL, [100H] } **NOT ALLOWED** ☹️ (Destination is an 8-bit register)
LES AL, [100H] }

LDS AX, 100H } **NOT ALLOWED** ☹️ (source is not a memory)
LES AX, 100H }

Best Regards
Dr. Zainab Alomari

Lecture5: FLAGS Register

FLAGS register is a 16-bits register that contain 9-used bits while the rest bits are not used. These bits are as follows:

b ₁₅	b ₁₄	b ₁₃	b ₁₂	b ₁₁	b ₁₀	b ₉	b ₈	b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁	b ₀
				O	D	I	T	S	Z		A		P		C

Status Flags:

1) **Carry Flag (CF):**

CF=1 when there is a carry-out or a borrow-in after executing the instruction.

CF=0 otherwise.

2) **Parity Flag (PF):**

PF=1 if the number of ones in the result is even.

PF=0 if the number of ones in the result is odd.

3) **Auxiliary Carry Flag (AF):**

AF=1 if there is a carry-out into the high nibble from the low nibble, or a borrow-in from the high nibble into the low nibble of the lower byte of the result.

AF=0 otherwise.

4) **Zero Flag (ZF):**

ZF=1 when the result of executing an instruction is zero.

ZF=0 when the result of executing an instruction is not zeros.

5) **Sign Flag (SF):**

SF=1 if the MSB (Most Significant Bit) is 1 (negative number).

SF=0 if the MSB (Most Significant Bit) is 0 (positive number).

Or we can directly say (SF=MSB).

Notice that for unsigned numbers, SF value is neglected.

6) **Overflow Flag (OF):**

OF=1 if the signed result is out of range. This means that there is a carry added to the sign bit but no carries out of the sign bit. (when working with unsigned numbers, OF value is neglected).

Example:

$$\begin{array}{r}
 127 \quad = \quad 7F \quad = \quad 01111111 \\
 1 + \quad \quad \quad 1 + \quad \quad \quad 00000001 + \\
 \hline
 \textcircled{+128} \quad \quad \quad \textcircled{0}10000000 = \textcircled{-128} \quad \text{NOT CORRECT} \textcircled{\otimes}
 \end{array}$$

Control Flags:

1) **Trap Flag (TF):**

If TF is set to 1, it permits executing the program step by step (one instruction at a time).

2) **Interrupt Flag (IF):**

If IF is 0, all interrupt requests are ignored. If IF is set to 1, interrupts are recognized.

3) **Direction Flag (DF):**

(Used with *String Instructions*).

ملاحظة: ايعازي mov و xchg لا يؤثران على الـ flags.

Best Regards
Dr. Zainab Alomari

Lecture 6: Addition and Subtraction Instructions

1- Addition Instruction

يقوم هذا الايعاز بجمع محتويات الـ source والـ destination مع بعضهما وخزن النتيجة في الـ destination.

ADD Destination, Source

Example

ADD AX, BX ; (AX = AX + BX)

All status flags are affected by his instruction (CF, PF, ZF, SF, AF, OF)

يمكن ان تكون الارقام signed أو unsigned.

Example

Set CL to 0FH and CH to 1FH, Find the value of CL and CH with the values of the status flags after executing the following code:

MOV CL, 0FH

MOV CH, 1FH

ADD CL, CH

Solution

CL = 0 0 0 0 1 1 1 1

CH = 0 0 0 1 1 1 1 1 +

0 0 1 0 1 1 1 0

∴ $CL_{new} = 2EH, CH_{new} = CH_{old}$

Status flags:

CF=0

ZF=0

PF=1

SF=0

AF=1

OF=0

Examples

Assembly Language	Operation
ADD AL,BL	AL = AL + BL
ADD CX,DI	CX = CX + DI
ADD CL,44H	CL = CL + 44H
ADD BX,245FH	BX = BX + 245FH
ADD [BX],AL	AL adds to the contents of the data segment memory location address by BX with the sum stored in the same memory location
ADD CL,[BP]	The byte contents of the stack segment memory location addressed by BP add to CL
ADD BX,[SI + 2]	The word contents of the data segment memory location addressed by the sum of SI plus 2 add to BX with the sum stored in BX
ADD [BX + DI],DL	DL adds to the contents of the data segment memory location addressed by BX plus DI with the sum stored in the same memory location

ملاحظات مهمة عند التعامل مع signed numbers:

- (1) عند الطرح نأخذ الـ 2's complement للرقم الثاني ثم نجمع.
- (2) يتم أخذ الـ 2's complement للرقم من خلال قلب الـ 0 الى 1 والواحدات الى اصفار ثم الجمع مع 1 (او ممكن استخدام طرق اخرى).
- (3) اذا كنا نقوم بجمع ارقام signed numbers وبعد الجمع كان الـ OF=1 فمعناه أنه قد حصل خلل في الجمع لعدم كفاية عدد البتات حيث يتأثر الـ sign bit أي الـ MSB والنتيجة تكون خطأ. أما اذا كانت الارقام unsigned فنهمل قيمة الـ OF.

Example:

$$\begin{array}{r}
 +48 \quad 0011\ 0000 \\
 +80 \quad 0101\ 0000 \\
 \hline
 +128 \quad 0100\ 0000 = -128
 \end{array}$$

نلاحظ في هذا المثال ان الـ OF يساوي 1 لان هنالك carry من البت قبل الاخير الى بت الاشارة ثم لم يكن هنالك carry من بت الاشارة, وبالتالي فهناك حالة overflow.

- (4) في حالة جمع الارقام الـ signed وكان هنالك carry فيتم اهماله.

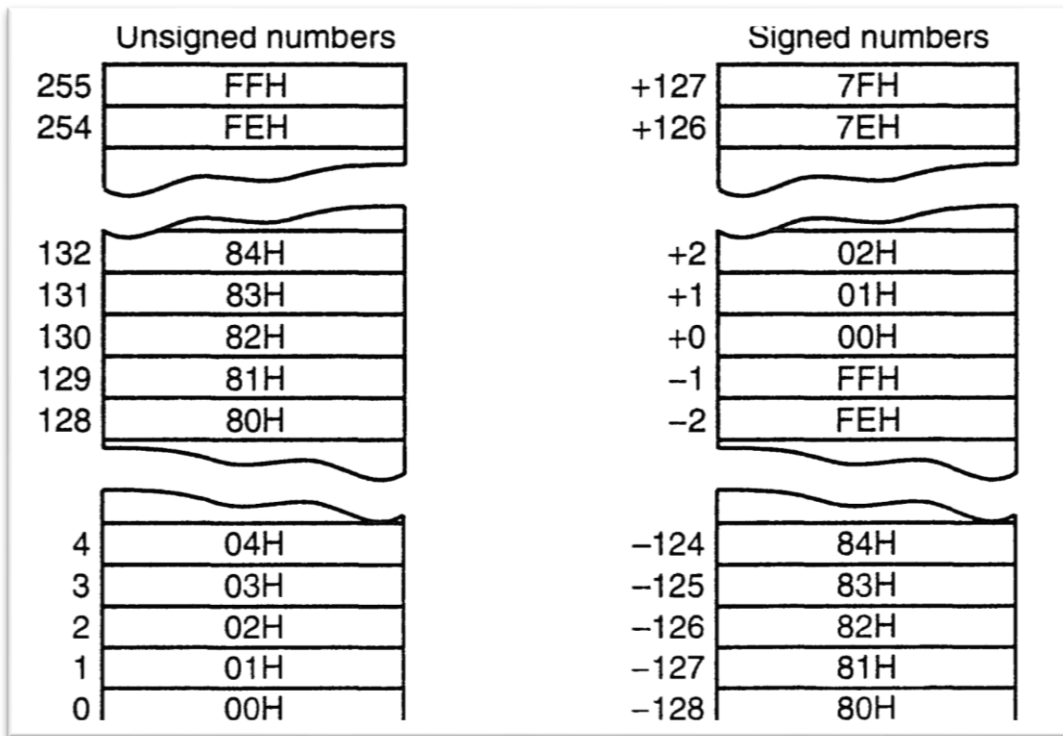
Example:

$$\begin{array}{r}
 +48 \quad 0011\ 0000 \\
 -48 \quad 1101\ 0000 \\
 \hline
 0 \quad 1000\ 0000 = 0
 \end{array}$$

Discard carry

- (5) لمعرفة قيمة الـ signed numbers:
- a. اذا كان البت الاخير او بت الاشارة يساوي 0 معناه ان الرقم موجب فنقوم مباشرة بحسابه كقيمة من التحويل من الثنائي الى العشري والاشارة موجبة.
- b. اذا كان البت الاخير من الرقم يساوي 1 معناه ان الرقم سالب فنقوم بأخذ الـ 2's complement للرقم (مع بت الاشارة) ثم نقوم بايجاد قيمة الناتج في النظام العشري والتي ستكون اشارتها سالبة.
- (6) يتم استخدام نظام الـ 2's complement في الانظمة الحديثة لتمثيل الارقام السالبة حيث أن نظام الـ 1's complement لديه مشكلة في تمثيل الصفر حيث له قيمتين (صفر موجب وصفر سالب).

A figure that shows the difference between 8-bit signed and unsigned numbers



2- Add with Carry Instruction

ADC destination, source

في هذا الايعاز يتم الجمع مع اخذ قيمة الـ CF بنظر الاعتبار.

Examples:

ADC AX, BX ; (AX = AX + BX + CF)

ADC BL, 5H ; (BL = BL + 5H + CF)

Notes:

- All status flags are affected by his instruction (CF, PF, ZF, SF, AF, OF)
- ADC instruction is used when the added numbers are wider than 16-bits.

Example

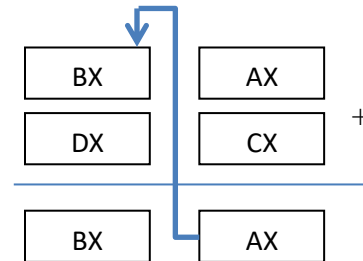
Add two 32-bit numbers using assembly language instructions, the first number is stored in (BX AX) and the second number is stored in (DX CX).

Solution

ADD AX, CX

ADC BX, DX

HLT



Example

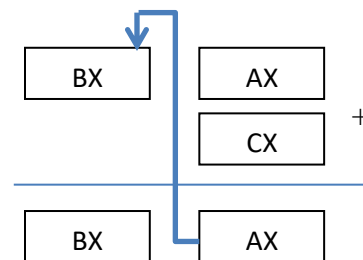
Add a 32-bit number with 16 bit number, where the first number is in (BX AX) and the second number is in CX.

Solution

ADD AX, CX

ADC BX, 0

HLT



Examples

<i>Assembly Language</i>	<i>Operation</i>
ADC AL,AH	AL = AL + AH + carry
ADC CX,BX	CX = CX + BX + carry
ADC DH,[BX]	The byte contents of the data segment memory location addressed by BX add to DH with carry with the sum stored in DH
ADC BX,[BP + 2]	The word contents of the stack segment memory location address by BP plus 2 add to BX with carry with the sum stored in BX

3- Increment Instruction

This instruction adds 1 to a register or memory contents.

Examples

<i>Assembly Language</i>	<i>Operation</i>
INC BL	BL = BL + 1
INC SP	SP = SP + 1
INC BYTE PTR [BX]	Adds 1 to the byte contents of the data segment memory location addressed by BX
INC WORD PTR [SI]	Adds 1 to the word contents of the data segment memory location addressed by SI

4- Subtraction Instruction

يقوم هذا الابعاز بطرح محتويات الـ source من الـ destination وخرن النتيجة في الـ destination.

SUB Destination, Source

Example

SUB CL, BL ; (CL = CL - BL)

All status flags are affected by his instruction (CF, PF, ZF, SF, AF, OF)

Instead of holding carry:

- CF is used to hold the borrow.
- AF is used to hold the half-borrow.

Example

Write the instructions required to subtract 44H from 22H and put the result in CH.

Solution

MOV CH, 22H

SUB CH, 44H

HLT

Examples

<i>Assembly Language</i>	<i>Operation</i>
SUB CL,BL	CL = CL - BL
SUB AX,SP	AX = AX - SP
SUB DH,6FH	DH = DH - 6FH

5- Subtract with Borrow Instruction

SBB destination, source (borrow باعتبارها) بنظر الاعتبار الـ CF اخذ قيمة الـ CF مع اذ قيمة الـ CF (باعتبارها)

Examples:

SBB CL, BL ; (CL = CL - BL - CF)

SBB BX, 5H ; (BX = BX - 5H - CF)

Examples

<i>Assembly Language</i>	<i>Operation</i>
SBB AH,AL	AH = AH – AL – carry
SBB AX,BX	AX = AX – BX – carry
SBB CL,2	CL = CL – 2 – carry
SBB BYTE PTR[DI],3	Both a 3 and carry subtract from the contents of the data segment memory location addressed by DI
SBB [DI],AL	Both AL and carry subtract from the data segment memory location addressed by DI
SBB DI,[BP + 2]	Both carry and the word contents of the stack segment memory location addressed by the sum of BP and 2 subtract from DI

Example

Write a piece of code in Assembly language to compute: $Z=X - Y$, where:

X is a 32-bit number stored at 92200H,

Y is a 32-bit number stored at 94200H,

Z is stored at 96200H.

Solution

Let: 92200H → 9000H:2200H

94200H → 9000H:4200H

96200H → 9000H:6200H

The three numbers are stored in the data segment with the value of DS equal to 9000H.

```
MOV AX, 9000H
```

```
MOV DS, AX
```

```
MOV AX, [2200H]
```

```
MOV BX, [2202H]
```

```
SUB AX, [4200H]
```

```
SBB BX, [4202H]
```

```
MOV [6200H], AX
```

```
MOV [6202H], BX
```

```
HLT
```

For the same example, if X is stored in the stack segment while Y and Z are in the data segment (using same addresses):

```
MOV AX, 9000H
```

```
MOV DS, AX
```

```
MOV SS, AX
```

```
MOV BP, 2200H
```

```
MOV AX, [BP]
```

```
MOV BX, [BP+2]
```

```
SUB AX, [4200H]
```

```
SBB BX, [4202H]
```

```
MOV [6200H], AX
```

MOV [6202H], BX

HLT

6- Decrement Instruction

This instruction subtracts 1 from a register or memory contents.

Examples

<i>Assembly Language</i>	<i>Operation</i>
DEC BH	$BH = BH - 1$
DEC CX	$CX = CX - 1$
DEC BYTE PTR [DI]	Subtracts 1 from the byte contents of the data segment memory location addressed by DI
DEC WORD PTR[BP]	Subtracts 1 from the word contents of the stack segment memory location addressed by BP

Notice that the example tables in this lecture are taken from Reference text book 1.

Best Regards
Dr. Zainab Alomari

Lecture 7: Multiplication and Division Instructions

1- Multiplication Instruction

يقوم هذا الابعاز بالضرب باستخدام الصيغ التالية:

MUL operand (for UNSIGNED numbers multiplication)

IMUL operand (for SIGNED numbers multiplication)

where: operand can be a **register** or a **memory**.

- If operand is 8-bits, multiplication is performed between AL and the operand and the result will be 16-bits stored in AX. This is called (8-bits multiplication)

Example:

MUL BL ; (AX = BL * AL) حاصل ضرب 8 بت * 8 بت يساوي 16 بت.

IMUL DH ; (AX = DH * AL)

- If operand is 16-bits, multiplication is performed between AX and the operand and the result will be 32-bits stored in (DX AX). This is called (16-bits multiplication)

Example:

MUL BX ; (DX AX = BX * AX) حاصل ضرب 16 بت * 16 بت يساوي 32 بت.

IMUL BP ; (DX AX = BP * AX)

NOTE:

MUL 12H **NOT Allowed** ☹ (NO immediate multiplication)

MUL DS **NOT Allowed** ☹ (NO segment registers multiplication)

Examples

<i>Assembly Language</i>	<i>Operation</i>
MUL CL	AL is multiplied by CL; the unsigned product is in AX
IMUL DH	AL is multiplied by DH; the signed product is in AX
IMUL BYTE PTR[BX]	AL is multiplied by the byte contents of the data segment memory location addressed by BX; the signed product is in AX

<i>Assembly Language</i>	<i>Operation</i>
MUL CX	AX is multiplied by CX; the unsigned product is in DX–AX
IMUL DI	AX is multiplied by DI; the signed product is in DX–AX
MUL WORD PTR[SI]	AX is multiplied by the word contents of the data segment memory location addressed by SI; the unsigned product is in DX–AX

Example

Find the result of x^2 using Assembly language, if x is an 8-bits signed number stored in the stack segment at offset (500H). Store the result in CX.

Solution

MOV BP, 500H

MOV AL, [BP]

IMUL AL

MOV CX, AX

HLT

2- Division Instruction

يقوم هذا الایعاز بالقسمة باستخدام الصیغ التالیة:

Div operand (for UNSIGNED numbers division)

IDiv operand (for SIGNED numbers division)

where: operand can be a **register** or a **memory**.

- If operand is 8-bits: AX is divided by the operand ($\frac{AX}{operand}$). Result will be stored in **AL** and remainder will be stored in **AH**.

Example:

Div BL ; (AL = $\frac{AX}{BL}$, and AH = remainder)

IDiv DH ; (AL = $\frac{AX}{DH}$, and AH = remainder)

- If operand is 16-bits: (DX AX) is divided by the operand ($\frac{DX AX}{operand}$). Result will be stored in **AX** and remainder will be stored in **DX**.

Example:

Div BX ; (AX = $\frac{DX AX}{BX}$, and DX = remainder)

IDiv SI ; (AX = $\frac{DX AX}{SI}$, and DX = remainder)

NOTE:

Div 12H **NOT Allowed** ☹ (NO immediate division)

IDiv DS **NOT Allowed** ☹ (NO segment registers division)

Examples

Assembly Language	Operation
DIV CL	AX is divided by CL; the unsigned quotient is in AL and the remainder is in AH
IDIV BL	AX is divided by BL; the signed quotient is in AL and the remainder is in AH
DIV BYTE PTR[BP]	AX is divided by the byte contents of the stack segment memor location addressed by BP; the unsigned quotient is in AL and the remainder is in AH

Assembly Language	Operation
DIV CX	DX-AX is divided by CX; the unsigned quotient is in AX and the remainder is in DX
IDIV SI	DX-AX is divided by SI; the signed quotient is in AX and the remainder is in DX

ملاحظة مهمة:

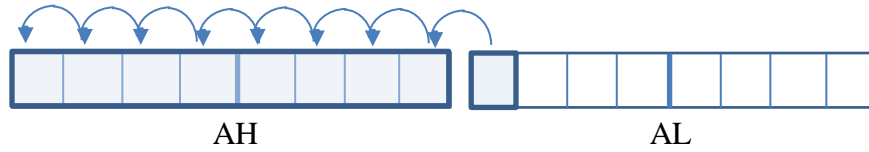
(1) في حالة أن المطلوب هو تقسيم 8بت على 8بت (مثلاً أن يكون المطلوب تقسيم AL على BL) فيجب تمديد قيمة AL الى AX ويتم ذلك بالطريقة التالية:

1- For unsigned numbers: put zeros in AH.

2- For signed numbers: use CBW.

(CBW=Convert Byte to Word): This instruction extends AL to AX by repeating the MSB of AL (sign bit) in AH.

ايغاز CBW يقوم بتمديد القيمة الموجودة في AL الى AX وذلك من خلال ملئ البتات الموجودة في AH بنفس قيمة البت الاخير من AL فبالنتالي اذا كان هذا البت واحد تمتلئ AH بالواحدات واذا كان صفر تمتلئ AH بالاصفار.

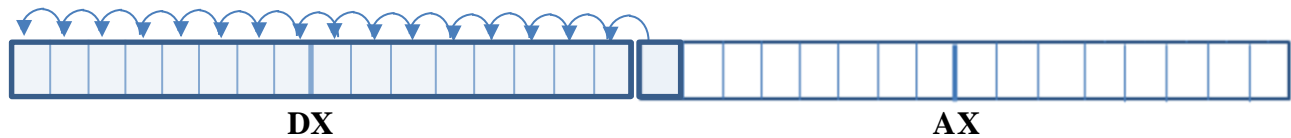


(2) وفي حالة أن المطلوب هو تقسيم 16 بت على 16 بت (مثلاً أن يكون المطلوب تقسيم AX على BX) فيجب تمديد قيمة AX الى AX DX ويتم ذلك بالطريقة التالية:

1- For unsigned numbers: put zeros in DX.

2- For signed numbers: use CWD.

(CWD=Convert Word to Double word): This instruction extends AX to DX AX by repeating the MSB of AX (sign bit) in DX.



ايغاز CWD يقوم بتمديد القيمة الموجودة في AX الى DX وذلك من خلال ملئ البتات الموجودة في DX بنفس قيمة البت الاخير من AX فبالنتالي اذا كان هذا البت واحد تمتلئ DX بالواحدات واذا كان صفر تمتلئ DX بالاصفار.

Example

Write the Assembly language instructions required to find the result of dividing the byte stored at DS:BX by the byte stored in the next location. Store the result and remainder in the locations next to the two numbers. Assume the two numbers are:

1) Unsigned

2) Signed

Solution

1) If numbers are unsigned:

```
MOV AL, [BX]
MOV AH, 0H
DIV Byte ptr[BX+1]
MOV [BX+2], AX
HLT
```

2) If numbers are signed:

```
MOV AL, [BX]
CBW
```



```
iDIV Byte ptr[BX+1]
MOV [BX+2], AX
HLT
```

Note: Some of the applications that may need the division operation are: calculating the average, checking if a number is even, checking if a number is prime, etc.

Example

What is the result of executing the following code:

```
MOV DX, 99H
MOV AL, 0A1H
CBW
CWD
HLT
```

Solution

$AX_{\text{new}} = \text{FFA1H}$, $DX_{\text{new}} = \text{FFFFH}$

*_*_*_*_*_*_*_*

Q1) Write the equivalent instructions of:

- 1) CBW
- 2) CWD

Q2) Write the Assembly language instructions required to find: $Z=X/Y$, where X and Y are 16-bit numbers stored in BX and CX, respectively. Store the result at 91F00H in the data segment, and store the remainder at 55A80H in the stack segment. Assume that X and Y are:

- 1) Unsigned numbers
- 2) Signed numbers

Notice that the example tables in this lecture are taken from Reference text book 1.

Best Regards
Dr. Zainab Alomari

Lecture 8: JUMP Instruction

1- Unconditional Jump

JMP operand(16-bits) ; operand can be any 16-bit value (immediate, register or memory)

This instruction is used to specify the location of the next instruction to be executed.

يقوم هذا الايعاز بتغيير قيمة المؤشر الذي يؤشر الى الايعاز التالي الذي سيتم تنفيذه فبالتالي يحصل قفز الى مكان اخر داخل او خارج البرنامج.

ملاحظة: لا يؤثر هذا الايعاز على اي من الـ flags.

Types of Jump operation:

1) Intrasegment Jump

In this type, the value of the new IP is directly given by the instruction. **CS is not changed.**

Examples

JMP 1234H ; ($IP_{new} = 1234H$)

JMP BX ; ($IP_{new} = BX$)

JMP BL **NOT Allowed ☹ (operand must be 16-bits)**

JMP DS **NOT Allowed ☹ (operand can't be a segment register)**

Examples

Let BX= 1000H, IP=100H, find the value of IP after executing the following instruction:

1342H:0100H JMP [BX] ; notice that 1342H:0100H is the location (CS:IP) where the instruction is stored.

Solution

$IP_{new} = 200H$

∴ The next location is stored at (1342H:0200H).

02H	DS: 1001H
00H	DS: 1000H
A7H	DS: 0FFFH

NOTE: any type of memory addressing modes can be used here, example:

JMP [SI+1]

JMP [DI+BX+3]

ملاحظة: يمكن استخدام الذاكرة لجلب القيمة الجديدة للـ IP وباستخدام اي نوع من انواع addressing modes التي اخذناها سابقاً, وبدون الحاجة الى تحديد حجم القيمة المأخوذة من الذاكرة بـ (word ptr) لانه طالما الاليعاز هو jmp فسيتم اخذ بايتين من الذاكرة ووضعها في IP.

2) Intersegment Jump

في هذا النوع يتم تغيير كل من الـ CS و IP وذلك باستخدام احدى الطريقتين التاليتين:

(1) يتم اعطاء قيمة CS و IP بشكل مباشر في الاليعاز وذلك باحدى الصيغتين التاليتين:

JMP FAR CS_{new}: IP_{new}

JMP FAR PTR CS_{new}: IP_{new}

Example:

JMP FAR 4000H:200H ; CS_{new}=4000H, IP_{new}=200H

(2) يتم اخذ قيمة CS و IP من الذاكرة باستخدام الصيغة التالية:

JMP Dword ptr [] ; any addressing mode can be used here.

Examples:

JMP Dword ptr [71FFH]

JMP Dword ptr [BX]

JMP Dword ptr [SI+500H]

ملاحظة: في هذه الحالة سيتم اخذ اربع بايتات من الذاكرة, أول بايتين منها تعطى للـ IP وثاني بايتين تعطى للـ CS.

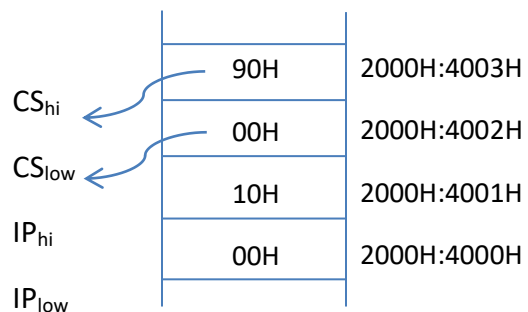
Example

Let BX= 4000H, DS= 2000H, find the new values of CS, IP and BX after executing the following instruction (the memory is given in the question):

8000H:0100H JMP Dword ptr[BX]

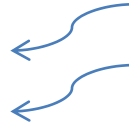
Solution

IP_{new} = 1000H



$CS_{new} = 9000H$

$BX_{new} = BX_{old}$



1- Conditional Jump

هي انواع القفز التي تعتمد على قيمة عامل او بت معين وحسب القيمة سيتم او لا يتم القفز الى الموقع المطلوب.

Conditional Jump Instructions

<i>Assembly Language</i>	<i>Tested Condition</i>	<i>Operation</i>
JA	$Z = 0$ and $C = 0$	Jump if above
JAE	$C = 0$	Jump if above or equal
JB	$C = 1$	Jump if below
JBE	$Z = 1$ or $C = 1$	Jump if below or equal
JC	$C = 1$	Jump if carry
JE or JZ	$Z = 1$	Jump if equal or jump if zero
JG	$Z = 0$ and $S = 0$	Jump if greater than
JGE	$S = 0$	Jump if greater than or equal
JL	$S \neq 0$	Jump if less than
JLE	$Z = 1$ or $S \neq 0$	Jump if less than or equal
JNC	$C = 0$	Jump if no carry
JNE or JNZ	$Z = 0$	Jump if not equal or jump if not zero
JNO	$O = 0$	Jump if no overflow
JNS	$S = 0$	Jump if no sign (positive)
JNP or JPO	$P = 0$	Jump if no parity or jump if parity odd
JO	$O = 1$	Jump if overflow
JP or JPE	$P = 1$	Jump if parity or jump if parity even
JS	$S = 1$	Jump if sign (negative)
JCXZ	$CX = 0$	Jump if CX is zero
JECXZ	$ECX = 0$	Jump if ECX equals zero
JRCXZ	$RCX = 0$	Jump if RCX equals zero (64-bit mode)

Note:

(Above and Below) are used with **Unsigned** numbers.

(Greater and Less) are used with **Signed** numbers.

Example

Write the assembly language instructions required to copy the contents of 10 memory locations (from 9A000H to 9A009H) to the memory locations that start at 96050H.

Solution

```
MOV AX, 9000H
MOV DS, AX
MOV BX, A000H
MOV SI, 6050H
MOV CX, 0AH
N1 : MOV AL, [BX]
      MOV [SI], AL
      INC BX
      INC SI
      DEC CX
      JNZ N1
      HLT
```

ANOTHER SOLUTION:

```
MOV AX, 9000H
MOV DS, AX
MOV BX, 0H
MOV CX, 0AH
N1 : MOV AL, [BX+A000H]
      MOV [BX+6050H], AL
      INC BX
      DEC CX
      JNZ N1
      HLT
```

Loop Instructions:

1) LOOP instruction:

```
LOOP N1
```

This instruction decrements CX and jumps if (CX ≠ 0) to N1.

(ملاحظة: النوع الاول من ايعازات loop هو فقط المطلوب من الطلبة اما النوعين التاليين فهما للاطلاع فقط)

2) LOOPE and LOOPZ instructions:

LOOPE N1

LOOPZ N1

These instructions decrement CX and check both of CX and ZF as follows:

if (CX \neq 0 and ZF = 1) jump to N1.

3) LOOPNE and LOOPNZ instructions:

LOOPNE N1

LOOPNZ N1

These instructions decrement CX and check both of CX and ZF as follows:

if (CX \neq 0 and ZF = 0) jump to N1.

*_*_*_*_*_*_*_*_*_*

Questions:

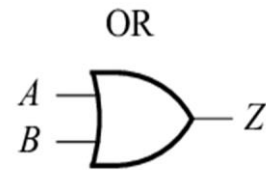
- 1) Re-solve the last example, with replacing (JNZ) instruction with (LOOP) instruction.
- 2) Write the equivalent instructions of (LOOP NXT).

Best Regards
Dr. Zainab Alomari

Lecture 9: Logic Instructions

1- OR Instruction

OR destination, source



Inputs		Output
A	B	Z
0	0	0
0	1	1
1	0	1
1	1	1

Annotations: A green arrow points from the output '0' to '=A'. A blue arrow points from the output '1' to '=1'. There are also green and blue lines connecting the input values to their respective output values.

يقوم هذا الايعاز بعمل Logic OR بين كل بت من destination مع البت الذي يقابله من source ويتم تخزين النتيجة في destination. من النظر الى Truth Table الخاصة ببوابة OR نلاحظ بأنه:

- عند عمل OR لـ A مع صفر فالنتيجة هي A.
- عند عمل OR لـ A مع واحد فالنتيجة هي 1.

بمعنى أنه يمكننا الاستفادة من ايعاز OR اذا اردنا جعل بت معين يساوي واحد (set) مع المحافظة على باقي البتات بدون تغيير, كما في المثال التالي:

Example

Set the MSB and LSB of BX using Assembly language.

Solution

OR BX, 8001H

HLT

XXXX XXXX XXXX XXXX

1000 0000 0000 0001 OR

1xxx xxxx xxxx xxx1

ملاحظة: الحل هو نفسه مهما كانت قيمة BX لا تؤثر, ولهذا كتبناها بشكل x والتي تعني (don't care condition).

Examples

Assembly Language

Operation

OR AH,BL

AH = AH OR BL

OR SI,DX

SI = SI OR DX

OR DH,0A3H

DH = DH OR A3H

OR SP,990DH

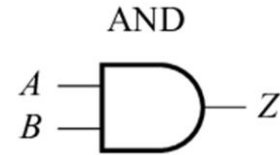
SP = SP OR 990DH

OR DX,[BX]

DX is ORed with the word contents of the data segment memory location addressed by BX

2- AND Instruction

AND destination, source



Inputs		Output
A	B	Z
0	0	0
0	1	0
1	0	0
1	1	1

Annotations: A green arrow points from the '0' output of the first row to '=0'. A blue arrow points from the '1' output of the last row to '=A'.

يقوم هذا الابعاز بعمل Logic AND بين كل بت من destination مع البت الذي يقابله من source ويتم خزن النتيجة في destination. من النظر الى Truth Table الخاصة ببوابة AND نلاحظ بأنه:

- عند عمل AND لـ A مع صفر فالنتيجة هي 0.
- عند عمل AND لـ A مع واحد فالنتيجة هي A.

بمعنى أنه يمكننا الاستفادة من ابعاز AND اذا اردنا جعل بت معين يساوي صفر (reset) مع المحافظة على باقي البتات بدون تغيير, كما في المثال التالي:

Example

Reset the MSB and LSB of AX using Assembly language.

Solution

AND AX, 7FFEh

HLT

xxxx xxxx xxxx xxxx

0111 1111 1111 1110 AND

0xxx xxxx xxxx xxx0

ملاحظة: الحل هو نفسه مهما كانت قيمة AX لا تؤثر, ولهذا كتبناها بشكل x والتي تعني (don't care condition).

Examples

Assembly Language

Operation

AND AL,BL

AL = AL AND BL

AND CX,DX

CX = CX AND DX

AND CL,33H

CL = CL AND 33H

AND DI,4FFFFH

DI = DI AND 4FFFFH

AND AX,[DI]

AX is ANDed with the word contents of the data segment memory location addressed by DI

3- XOR Instruction

XOR destination, source

Exclusive OR



Inputs		Output
A	B	Z
0	0	0
0	1	1
1	0	1
1	1	0

Annotations: Green arrows point from the output '0' in the first row to '=A'. Blue arrows point from the output '1' in the second and third rows to '=A-bar'.

يقوم هذا الابعاز بعمل Logic XOR بين كل بت من destination مع البت الذي يقابله من source ويتم خزن النتيجة في destination. من النظر الى الـ Truth Table الخاصة ببوابة XOR نلاحظ بأنه:

- عند عمل XOR لـ A مع صفر فالنتيجة هي A.
- عند عمل XOR لـ A مع واحد فالنتيجة هي \bar{A} .

بمعنى أنه يمكننا الاستفادة من ايعاز XOR اذا اردنا قلب بت معين مع المحافظة على باقي البتات بدون تغيير. كما في المثال التالي:

Example

Complement the MSB and LSB of DX using Assembly language.

Solution

XOR DX, 8001H

HLT

XXXX XXXX XXXX XXXX

1000 0000 0000 0001 XOR

$\bar{X}XXX XXXX XXXX XXX\bar{X}$

ملاحظة: الحل هو نفسه مهما كانت قيمة DX لا تؤثر. ولهذا كتبناها بشكل x والتي تعني (don't care condition).

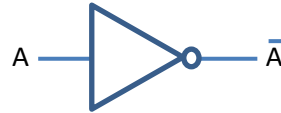
Examples

Assembly Language	Operation
XOR CH,DL	CH = CH XOR DL
XOR SI,BX	SI = SI XOR BX
XOR AH,0EEH	AH = AH XOR EEH
XOR DI,0DDH	DI = DI XOR 0DDH
XOR DX,[SI]	DX is Exclusive-ORed with the word contents of the data segment memory location addressed by SI

4- NOT Instruction

NOT operand ; (this instruction takes the 1's complement of the operand)

where: operand can be a register or a memory.



Example

What is the value of AL after executing the following code:

```
MOV AL, 55H
AND AL, 1FH
OR AL, 0C0H
XOR AL, 0FH
NOT AL
HLT
```

Solution

$AL_{new} = 25H$

5- Negative Instruction

NEG operand ; (this instruction takes the 2's complement of the operand)

where: operand can be a register or a memory.

الايعازات المكافئة لايعاز NEG: يوجد ثلاث طرق للقيام بالعمل المكافئ لعمل ايعاز Neg وهي:

- 1 NOT ثم الجمع مع 1
- 2 XOR مع FFFFH ثم الجمع مع 1
- 3 عملية طرح (sub) الرقم من الصفر فبالتالي ستكون النتيجة سالبة هذا الرقم

TABLE 5-19 NOT and NEG instructions

Assembly Language	Operation
NOT CH	CH is one's complemented
NEG CH	CH is two's complemented
NEG AX	AX is two's complemented
NOT BYTE PTR[BX]	The byte contents of the data segment memory location addressed by BX is one's complemented

6- Test Instruction

TEST destination, source

This instruction performs AND between destination and source, WITHOUT affecting destination (only flags are affected by the result):

Examples

<i>Assembly Language</i>	<i>Operation</i>
TEST DL,DH	DL is ANDed with DH
TEST CX,BX	CX is ANDed with BX
TEST AH,4	AH is ANDed with 4

NOTE: Test instruction is usually used to test a bit or more, depending on the Zero Flag (ZF):

ZF=0 if the bit under test is not zero.

ZF=1 if the bit under test is zero.

The required bit is usually tested against immediate number, example:

1 to test bit 0 (TEST AL, 01H)

2 to test bit 1 (TEST AL, 02H)

4 to test bit 2 (TEST AL, 04H)

8 to test bit 3 (TEST AL, 08H)

and so on.

Example

Find the value of the ZF after executing the following instruction:

TEST byte ptr[1000H], 80H

Solution

ZF= 0

5AH	DS:1001H
90H	DS:1000H

This result means that the value of the bit under test is 1 (which is the MSB or b_7).

Example

Find the number of **even** values in a block of 50 16-bit signed numbers stored in the memory starting at DS:DI. Store the result in BL.

Compare Instruction

CMP destination, source

This instruction performs SUB (destination – source) without affecting the destination (only flags are affected). CMP is usually followed with a conditional jump.

يقوم هذا الايعاز بعمل طرح أي نفس عمل ايعاز SUB ولكن لا يغير الـ destination وانما فقط تؤثر النتيجة على الـ flags.

Examples

TABLE 5–7 Comparison instructions

<i>Assembly Language</i>	<i>Operation</i>
CMP CL,BL	CL – BL
CMP AX,SP	AX – SP
CMP AX,2000H	AX – 2000H
CMP [DI],CH	CH subtracts from the contents of the data segment memory location addressed by DI
CMP CL,[BP]	The byte contents of the stack segment memory location addressed by BP subtract from CL

ملاحظة: عادةً ما يتبع ايعاز CMP بايعاز من ايعازات القفز المشروط، كما في الامثلة التالية:

Examples

CMP AX, BX

JE N1

CMP CX, 10H

JA N1

Example

Write the assembly language instructions required to copy the contents of 10 memory locations (from 9A000H to 9A009H) to the memory locations that start at 96050H.

(هذا المثال موجود في المحاضرة السابقة ولكن هنا سنعيد الحل باستخدام ايعاز CMP).

Solution

```
MOV AX,9000H
MOV DS, AX
MOV BX, 0
N1: MOV AL, [BX+A000H]
    MOV [BX+6050H], AL
    INC BX
    CMP BX, 0AH
    JNZ N1
    HLT
```

Example

Write the assembly language instructions required to find the square of 20 8-bits signed numbers stored in the memory starting at address 8E000H,. Store the results in memory locations starting at 81000H.

Solution

```
MOV AX, 8000H
MOV DS, AX
MOV BX, 0H
MOV SI, 0H
NEXT:MOV AL, [BX+E000H]
    IMUL AL          ;( AX=AL*AL)
    MOV [SI+1000H], AX
    ADD SI, 2
    INC BX
    CMP BX, 14H    ; or  CMP BX, 20
    JNZ NEXT
    HLT
```

Notice that the example tables in this lecture are taken from Reference text book 1.

Best Regards
Dr. Zainab Alomari

Lecture 10: Stack Instructions

The stack instructions are important instructions that store and retrieve data from the LIFO (last-in, first-out) stack memory. Stack instructions are: PUSH and POP instructions.

1- PUSH Instruction

PUSH operand

يقوم هذا الابعاز بخزن 16بت في الستاك

This instruction stores the (operand) value in the stack, where (operand) is any 16-bit value.

Operand can be: 16-bit register

Segment register (CS, DS, ES and SS)

Immediate value

Memory

Examples:

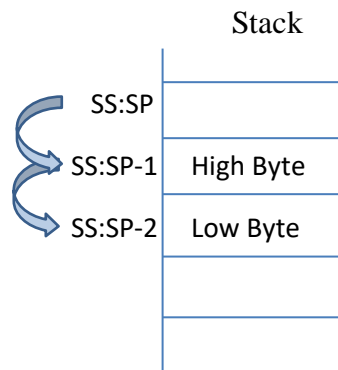
Push BX

Push [DI]

Push 15H

Push DS

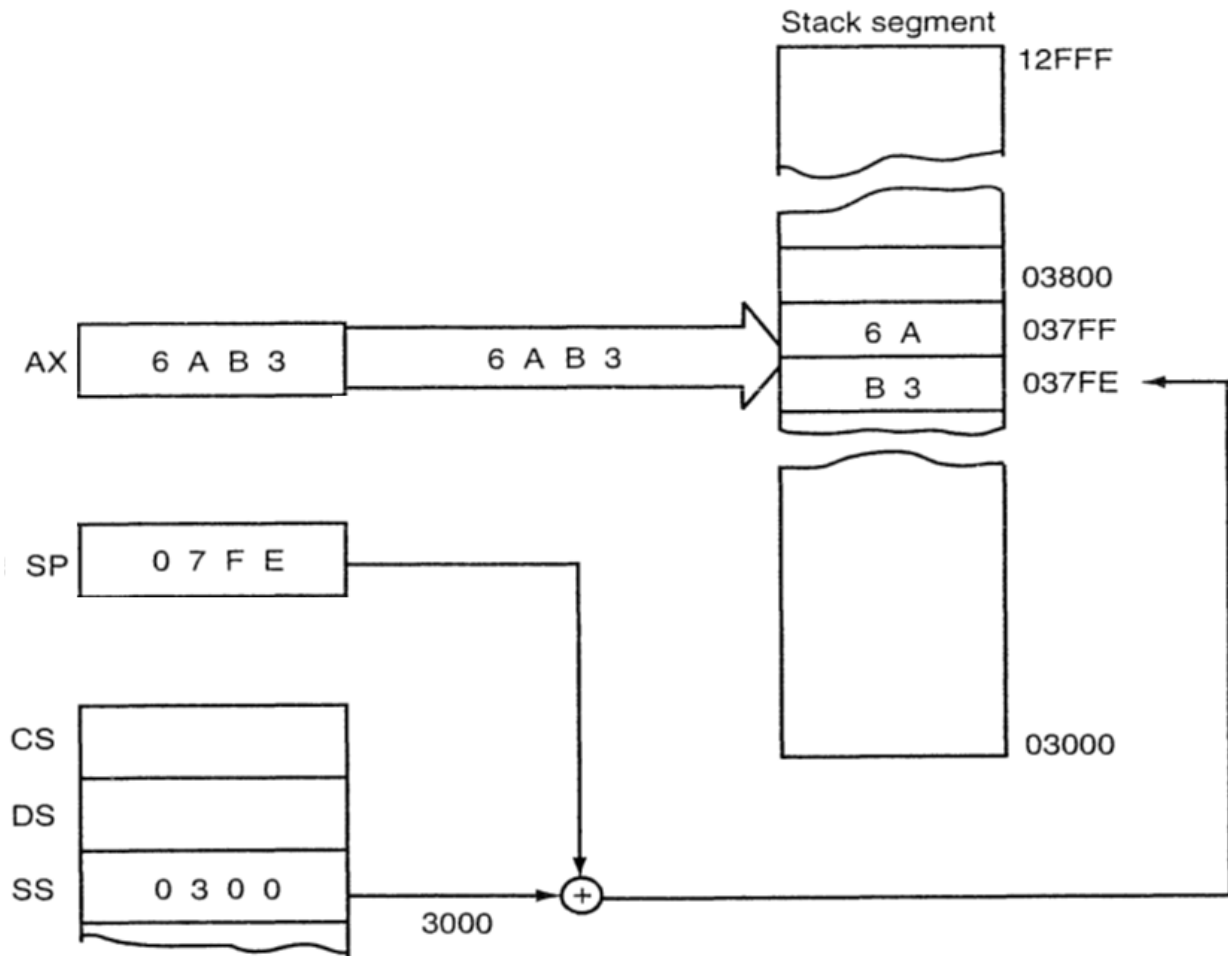
ملاحظة: يتم خزن البايتات في الذاكرة كالتالي:



After storing the 16-bits into the stack, SP is decremented by 2 ($SP_{new} = SP - 2$).

باستخدام الذاكرة المؤشر عليها باستخدام SS:SP يتم خزن الـ 16بت بالطريقة التالية: الموقع الحالي لا يتم الخزن فيه وانما يتم تنقيص المؤشر SP بمقدار 1 ثم يتم خزن البايت الـ high ثم تنقيص SP بمقدار 1 مرة ثانية وخزن البايت الـ low, وبعد الانتهاء من تنفيذ ايعاز Push ستكون قيمة SP الجديدة هي (SP-2).

Example



The effect of the `PUSH AX` instruction on `SP` and stack memory location `37FFH` and `37FEH`. This instruction is shown at the point **after execution.**

Example:

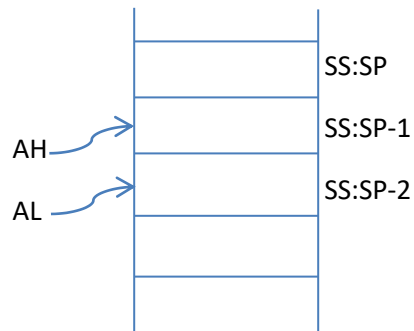
Push AX

Solution

`AH` \rightarrow `SS:[SP-1]`

`AL` \rightarrow `SS:[SP-2]`

`SPnew` = `SP - 2`



Example:

Let BX=100H, SP=500H, SS=1FF0H what is the result of executing the following instruction (the memory is given with the question):

Push [BX]

33H	DS:101H
A0H	DS:100H
1FH	DS:FFH
75H	DS:FEH

(Data Segment)

Solution

2 bytes will be taken from (DS:BX) to the stack.

These two bytes are (33A0H)

$$SP_{\text{new}} = 500H - 2 = 4FEH$$

	1FF0H:500H
33H	1FF0H:4FFH
A0H	1FF0H:4FEH

(Stack Segment)

2- POP Instruction

POP operand

يقوم هذا الابعاز باخراج 16بت من الستاك وخرنها بداخل ال operand

This instruction takes 16-bits from the stack, and puts it in the operand.

Operand can be: 16-bit register 8-bit registers are NOT allowed
Segment register (DS, ES and SS) CS is NOT allowed
Memory

Examples:

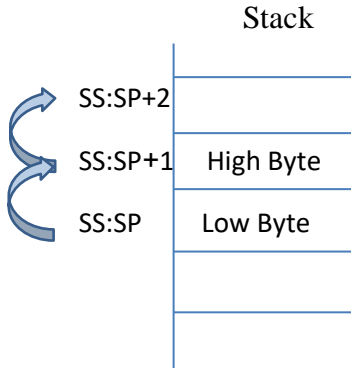
POP BX

POP [DI]

POP DS

POP 15H NOT Allowed ☹

ملاحظة: يتم أخذ البايتات من الستاك كالتالي:



After taking the 16-bits from the stack, SP is increased by 2 ($SP_{new} = SP+2$).

باستخدام الذاكرة المؤشر عليها باستخدام SS:SP يتم أخذ 16 بت بالطريقة التالية: يتم أخذ البايٲ الـ low من الموقع الحالي ثم يتم زيادة المؤشر SP بمقدار 1 ثم يتم أخذ البايٲ الـ high , ثم زيادة المؤشر SP بمقدار 1 مرة ثانية, وبعد الانتهاء من تنفيذ ايعاز POP ستكون قيمة SP الجديدة هي (SP+2).

Example:

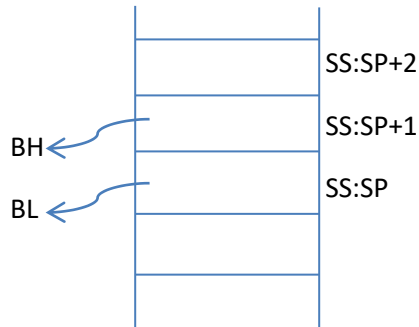
POP BX

Solution

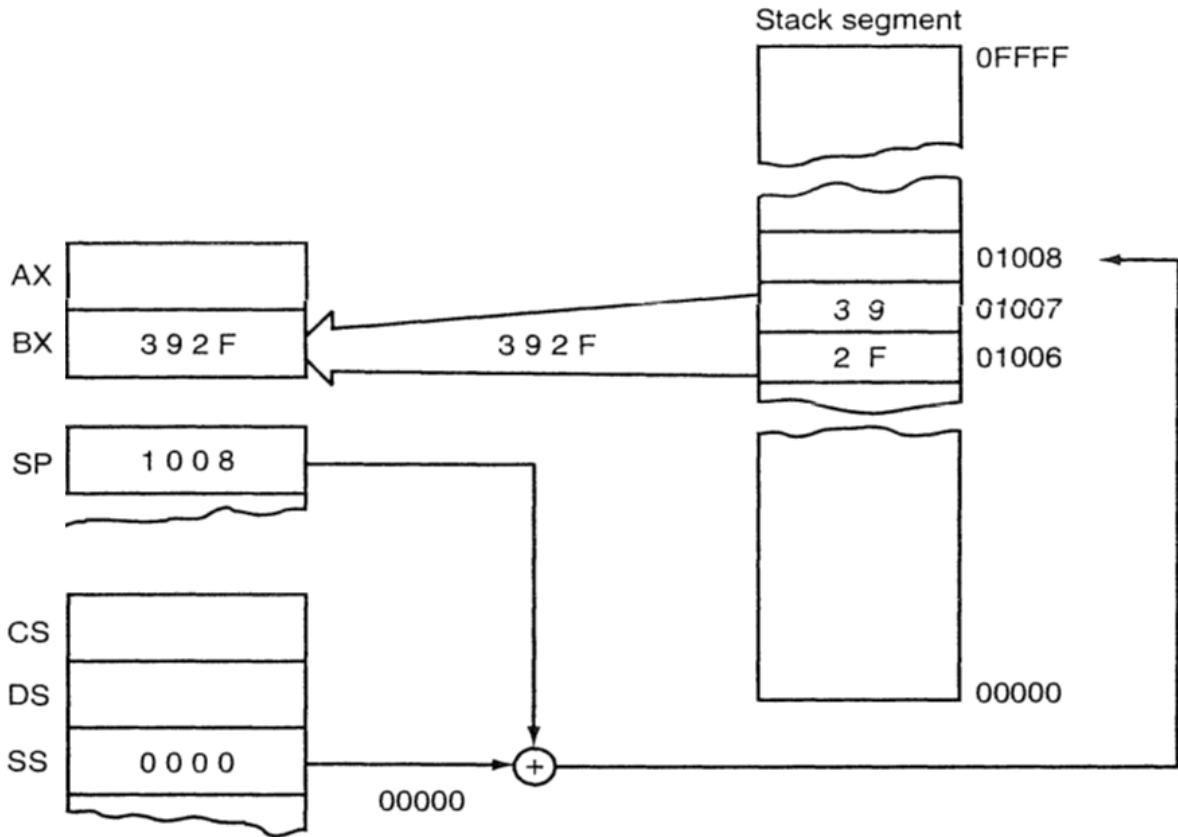
BL \leftarrow SS:[SP]

BH \leftarrow SS:[SP+1]

$SP_{new} = SP + 2$



Example:



The POP BX instruction, showing how data are removed from the stack. This instruction is shown after execution.

Example:

Push AX
POP DX } = MOV DX, AX


NOTE: the Push and POP instructions can be used to save the values of registers before using them in some tasks.

Example:

If we are required to use AX and DX in Div or Mul instructions, and they contain important information, we can use stack instructions as follows:

Push AX

Push DX



(perform required task here using AX and DX)

POP DX

POP AX

Example:

Exchange between AX and BX using stack instructions.

Solution

Push AX

Push BX

POP AX

POP BX

HLT

3- PUSHF and POPF Instructions

PUSHF saves the two bytes of the FLAGS register into the stack.

POPF takes two bytes from the stack and puts them in the FLAGS register.

Example:

Save the contents of FLAGS register in DX.

Solution

PUSHF

POP DX

HLT

Example:

Clear all the bits of the FLAGS register.

Solution

```
MOV DX, 0
```

```
PUSH DX
```

```
POPF
```

```
HLT
```

Example:

Clear the high byte of the FLAGS register.

Solution

```
PUSHF
```

```
POP DX
```

```
MOV DH, 00
```

```
PUSH DX
```

```
POPF
```

```
HLT
```

Example:

Write the required instructions to set the interrupt flag:

Solution:

```
PUSHF
```

```
POP AX
```

```
OR AX, 0200H
```

```
PUSH AX
```

```
POPF
```

```
HLT
```

Flags Control Instructions

LAHF (AH ← FLAGS register lower byte)

SAHF (AH → FLAGS register lower byte)

Example:

Save the current contents of the low byte of the FLAGS register into ES:BX, then load this byte with a new value from DS:SI.

Solution

LAHF

MOV ES:[BX], AH

MOV AH, [SI]

SAHF

HLT

Other instructions that allow changing a single bit in the FLAGS register without affecting other bits:

CLC (clear carry flag CF)

STC (set carry flag CF)

CMC (complement carry flag CF)

CLI (clear Interrupt flag IF)

STI (set interrupt flag IF)

Notice that some examples in this lecture are taken from Reference text book 1.

*Best Regards
Dr. Zainab Alomari*

Lecture 11: Shift and Rotate Instructions

1- Shift Instructions

1) Logical Shift

Shifting to the left:

SHL Operand, shift ; shifting the operand to the left

يقوم هذا الابعاز بتزحيف البتات الموجودة في الـ(operand) الى اليسار وعدد مراتب التزحيف يحدده الـ(shift).

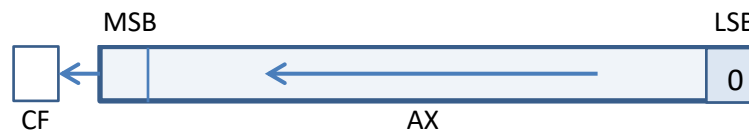
- يمكن ان يكون الـOperand سجلاً أو ذاكرة، ويمكن ان يكون بطول 8بت أو 16بت.
- مقدار الـshift يكون إما (immediate value) أو مخزوناً في سجل CL حصراً.
- يتم اضافة اصفار في البتات التي ستفرغ من جهة اليمين.
- يتم وضع اخر بت تم تزحيفه في الـCF.

Example:

Let AX=877FH , what is the new value of AX after executing the following instruction:

SHL AX, 1

Solution



$AX_{old} = 1000\ 0111\ 0111\ 1111\ b$

$AX_{new} = 0000\ 1110\ 1111\ 1110\ b$

= 0EFE H

ZF = 0 , CF = 1 (CF = last shifted bit)

Example:

Let AX=877FH , what is the new value of AX after executing the following instructions:

MOV CL, 10H

SHL AX, CL

Solution

$AX_{old} = 1000\ 0111\ 0111\ 1111\ b$

$AX_{new} = 0000\ 0000\ 0000\ 0000\ b$

= 0H

ZF = 1, CF = 1 (CF = last shifted bit = LSB)

Note: shifting a number to the **left** by n bits equals to multiplying the number by 2^n (used with unsigned numbers)

عملية تشيفت الى اليسار تعادل ضرب الرقم بمقدار يعتمد على عدد بتات التزحيف, كما في الامثلة الاتية:

SHL AX, 1 ; to multiply AX by 2

SHL AX, 2 ; to multiply AX by 4

SHL AX, 3 ; to multiply AX by 8

Shifting to the right:

SHR Operand, shift ; shifting the operand to the right

يقوم هذا الابعاز بتزحيف البتات الموجودة في الـ(operand) الى اليمين و عدد مراتب التزحيف يحدده الـ(shift).

- يمكن ان يكون الـOperand سجلاً أو ذاكرة, ويمكن ان يكون بطول 8بت أو 16بت.
- مقدار الـshift يكون إما (immediate value) أو مخزوناً في سجل CL حصراً.
- يتم اضافة اصفار في البتات التي ستفرغ من جهة اليسار.
- يتم وضع اخر بت تم تزحيفه في الـCF.

Example:

Let $AX=8F01H$, what is the new value of AX after executing the following instruction:

SHR AX, 3H

Solution

$AX_{old} = 1000\ 1111\ 0000\ 0001\ b$

$AX_{new} = 0001\ 0001\ 1110\ 0000\ b$

= 11E0H

ZF = 0, CF = 0 (CF = last shifted bit)

Example:

Let $AX=8F01H$, what is the new value of AX after executing the following instruction:

`SHR AX, 11H`

Solution

(11H = 17d)

$AX_{old} = 1000\ 1111\ 0000\ 0001\ b$

$AX_{new} = 0000\ 0000\ 0000\ 0000\ b$

= 0H

ZF = 1, CF = 0 (CF = last shifted bit)

Note: shifting a number to the **right** by n bits equals to dividing the number by 2^n (used with unsigned numbers)

عملية تشيفت الى اليمين تعادل قسمة الرقم بمقدار يعتمد على عدد بتات التزحيف, كما في الامثلة الاتية:

`SHR AX, 1` ; to divide AX by 2

`SHR AX, 2` ; to divide AX by 4

`SHR AX, 3` ; to divide AX by 8

2) Arithmetic Shift

تستخدم عند التعامل مع ارقام signed

Arithmetic Shift to the Left:

SAL = SHL (exactly the same)

Arithmetic Shift to the Right:

SAR operand, shift

أيضاً يتم التزحيف الى اليمين بهذا الابعاز ولكن يستخدم مع الارقام signed.

- يمكن ان يكون الOperand سجلاً أو ذاكرة, ويمكن ان يكون بطول 8بت أو 16بت.
- مقدار الshift يكون إما (immediate value) أو مخزوناً في سجل CL حصراً.
- بدل وضع اصفار الى جهة اليسار يتم تكرار قيمة البت الاخير (MSB) في البتات التي ستفرغ من جهة اليسار, وبالتالي اذا كان الرقم signed فان هذه العملية ستحافظ على بت الاشارة.
- يتم وضع اخر بت تم تزحيفه في الCF.

Example:

Let $AX=8000H$, what is the new value of AX after executing the following instruction:

$SAR AX, 10H$

Solution

($10H = 16d$)

$AX_{old} = 1000\ 0000\ 0000\ 0000\ b$

$AX_{new} = 1111\ 1111\ 1111\ 1111\ b$

= $FFFF\ H$

$ZF = 0, CF = 1$ (CF = last shifted bit)

نلاحظ بان جميع ايعازات التشفيت الاربعة تقوم بوضع صفر في الCF اذا كان مقتدار التشفيت اكبر من عدد البتات, ما عدا ايعاز SAR يقوم بوضع بت الاشارة في الCF.

Example:

Let $AX=FC44H$, what is the new value of AX after executing the following instruction:

$MOV CL, 12H$

$SAR AX, CL$

Solution

($12H = 18d$)

$AX_{old} = 1111\ 1100\ 0100\ 0100\ b$

$AX_{new} = 1111\ 1111\ 1111\ 1111\ b$

= $FFFF\ H$

$ZF = 0, CF = 1$ (CF = last shifted bit)

(In this example, Let $AX_{old} = 7FFFH \rightarrow AX_{new} = 0000H, CF=0, ZF=1$)

Examples

<i>Assembly Language</i>	<i>Operation</i>
$SHL AX, 1$	AX is logically shifted left 1 place
$SHR BX, 12$	BX is logically shifted right 12 places
$SAR SI, 2$	SI is arithmetically shifted right 2 places

2- Rotate Instructions

1) Rotate Left:

ROL operand, rotation ; Rotating the operand to the left

يقوم هذا الايعاز بتدوير البتات الموجودة في الـ(operand) الى اليسار وعدد مراتب التزحيف يحدده الـ(rotation).

- يمكن ان يكون الـOperand سجلاً أو ذاكرة, ويمكن ان يكون بطول 8بت أو 16بت.
- مقدار الـrotation يكون إما (immediate value) أو مخزوناً في سجل CL حصراً.
- يتم التدوير بحيث توضع البتات المزحفة من اليسار في مكان البتات التي ستفرغ من جهة اليمين.
- يتم وضع اخر بت تم تدويره في الـCF.

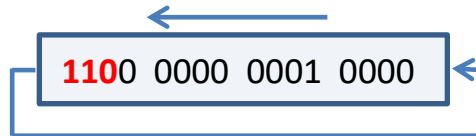


Example

Let AX=C010H , what is the new value of AX after executing the following instruction:

ROL AX, 3

Solution



$AX_{\text{new}} = 0000\ 0000\ 1000\ 0110\ \text{B}$

=0086H

CF = 0, ZF = 0

Example

What is the new value of AL after executing the following instruction:

ROL AL, 8

Solution

$AL_{\text{new}} = AL_{\text{old}}$

CF = last rotated bit = LSB

Example

What is the new value of AX after executing the following instruction:

ROL AX, 10H

Solution

$$AX_{\text{new}} = AX_{\text{old}}$$

CF = last rotated bit = LSB

2) Rotate Right:

ROR operand, rotation ; Rotating the operand to the right

يقوم هذا الايعاز بتدوير البتات الموجودة في الـ(operand) الى اليمين وعدد مراتب التزحيف يحدده الـ(rotation).

- يمكن ان يكون الـOperand سجلاً أو ذاكرة, ويمكن ان يكون بطول 8بت أو 16بت.
- مقدار الـrotation يكون إما (immediate value) أو مخزوناً في سجل CL حصراً.
- يتم التدوير بحيث توضع البتات المزحفة من اليمين في مكان البتات التي ستفرغ من جهة اليسار.
- يتم وضع اخر بت تم تدويره في الـCF.



Example

Let AL=F0H , what is the new value of AL after executing the following instruction:

ROR AL, 4

Solution

$$AL_{\text{old}} = 1111\ 0000\ \text{b}$$

$$AL_{\text{new}} = 0000\ 1111\ \text{b} = 0FH$$

$$CF = 0$$

نلاحظ أنه يمكن الاستفادة من هذا الايعاز في تبديل AL مع AH بدون الحاجة لاستخدام ايعاز Xchg باستخدام الايعاز التالي: (ROR Ax, 8) أو (ROL Ax, 8).

3) Rotate with Carry Left:

RCL Operand, rotation

فرق هذا الايعاز عن ROL هو أنه يقوم بالتدوير معتبراً الـ CF من ضمن البتات التي سيتم تدويرها الى اليسار, بالتالي سيتم استخدام قيمتها القديمة على انها اول بت يتم تدويره وفي النهاية ستحتوي على قيمة اخر بت تم تدويره.



4) Rotate with Carry Right:

RCR Operand, rotation

فرق هذا الايعاز عن ROR هو أنه يقوم بالتدوير معتبراً الـ CF من ضمن البتات التي سيتم تدويرها الى اليمين, بالتالي سيتم استخدام قيمتها القديمة على انها اول بت يتم تدويره وفي النهاية ستحتوي على قيمة اخر بت تم تدويره.



Example

Let AX=FF05H, CF=0, CL=3, what is the difference between the results of the following instructions:

RCR AX, CL

ROR AX, CL

Solution

[1] RCR AX, CL

$$AX_{old} = 1111\ 1111\ 0000\ 0101 \quad \boxed{0}$$

$$AX_{new} = 0101\ 1111\ 1110\ 0000 \quad \boxed{1}$$

∴ AX_{new} = 5FE0 H, CF = 1

[2] ROR AX, CL

$$AX_{old} = 1111\ 1111\ 0000\ 0101$$

$$AX_{new} = 1011\ 1111\ 1110\ 0000$$

∴ AX_{new} = BFE0 H, CF = 1

Examples

<i>Assembly Language</i>	<i>Operation</i>
ROL SI,14	SI rotates left 14 places
RCL BL,6	BL rotates left through carry 6 places
RCR AH,CL	AH rotates right through carry the number of places specified by CL
ROR WORD PTR[BP],2	The word contents of the stack segment memory location addressed by BP rotate right 2 places

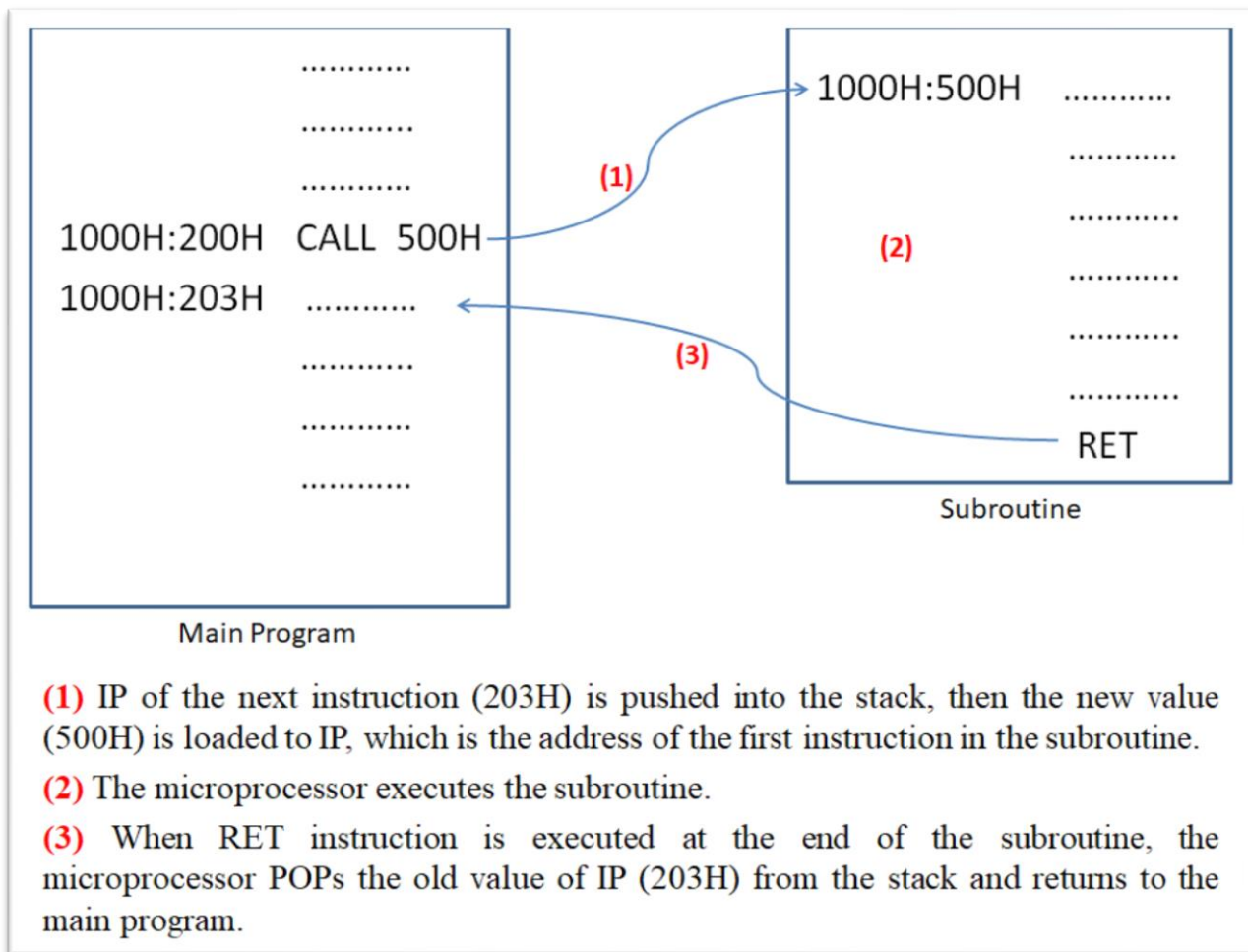
Notice that example tables in this lecture are taken from Reference text book 1.

Best Regards
Dr. Zainab Alomari

Lecture 12: Subroutine Instructions

1- Call Instruction

ايغاز Call يقوم بالقفز الى برنامج فرعي يتم تنفيذه ثم العودة لاكمال البرنامج من بعد المكان الذي تم القفز منه. عملية القفز تعني تغيير المؤشر الخاص بالبرنامج وهو إما IP فقط او تغيير كل من CS و IP.



- (1) IP of the next instruction (203H) is pushed into the stack, then the new value (500H) is loaded to IP, which is the address of the first instruction in the subroutine.
- (2) The microprocessor executes the subroutine.
- (3) When RET instruction is executed at the end of the subroutine, the microprocessor POPS the old value of IP (203H) from the stack and returns to the main program.

Call Instructions are divided into two types: Intersegment and Intrasegment Calls.

1) Intrasegment Call

In this type of Call, only IP is changed within the same CS.

CALL operand

Operand can be: immediate value
16-bit register
memory

Examples:

CALL 1234H ; IP_{new}= 1234H

CALL BX ; IP_{new}= BX

CALL [BX] ; IP_{new}= 8A60H (the memory is given in this example)



In this type of Call, the following happens:

(IP) of the instruction that follows call is pushed into the stack, with decrementing SP by 2.

- يقوم هذا النوع من الـ Call بخزن قيمة IP للايعاز الذي يقع بعد ايعاز Call في الستاك بنفس طريقة عمل ايعاز push وذلك قبل ان يغير الـ IP الى القيمة الجديدة (والتي تعود لاول ايعاز في البرنامج الفرعي المطلوب تنفيذه), ثم يقوم بتقليص قيمة SP بمقدار 2.
- سبب الاحتفاظ بقيمة الـ IP هذه في الستاك هو لاجل العودة اليها بعد الانتهاء من تنفيذ البرنامج الفرعي (subroutine).

Example

What is the result of executing the following instruction?

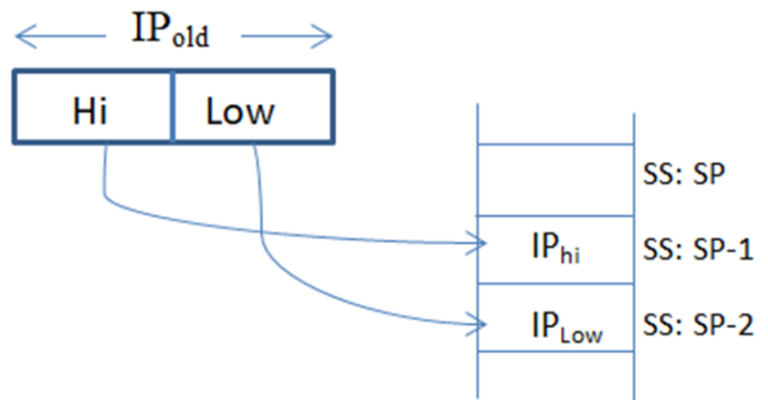
CALL [1234H]

Solution

IP_{Hi} → SS:[SP-1]

IP_{Low} → SS:[SP-2]

SP= SP-2



(IP_{old} is the IP of the instruction that follows CALL instruction in the main program)

2) Intersegment Call

في هذا النوع يتم القفز خارج الـ segment بحيث يتم تغيير قيمة كل من CS و IP في ابعاز CALL وليس الـ IP فقط. ويتم اعطاء قيم CS و IP إما مباشرة او تؤخذ قيمهما من الذاكرة, كما في الامثلة التالية:

Examples

CALL 1000H:2000H

CALL DWORD PTR [DI]

CALL FAR PTR [SI+5]

In this type of Call, the following happens:

Both of CS and IP of the instruction that follows call are pushed into the stack, with decrementing SP by 4, and then CS and IP are loaded with the new values.

ملاحظة: في هذا النوع من الـ CALL يتم وضع CS_{Hi} ثم CS_{Low} ثم IP_{Hi} ثم IP_{Low} في الـ stack.

$CS_{Hi} \rightarrow SS: [SP-1]$

$CS_{Low} \rightarrow SS: [SP-2]$

$IP_{Hi} \rightarrow SS: [SP-3]$

$IP_{Low} \rightarrow SS: [SP-4]$

$SP = SP - 4$

Example

Give the new values of all the given registers with drawing all the memory locations (addresses and values) that are affected or used by the following instruction (memory is given with the question):

CALL FAR PTR [BX]

Let BX= 88AH

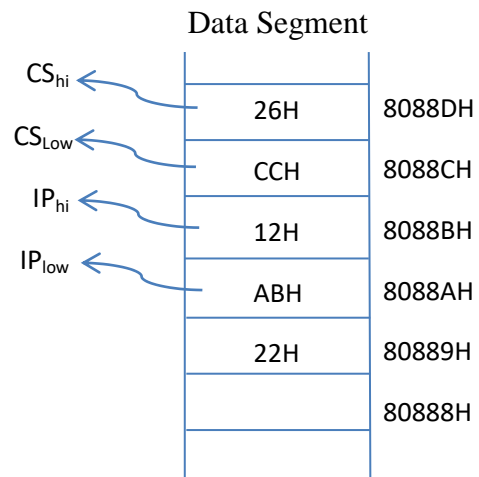
SP= 100H

DS= 8000H

CS= 2000H

IP= 30H

SS= B00H



Solution

```
8 0 0 0 0
+ 8 8 A
-----
8 0 8 8 A H
```

∴ From the given memory:

$$CS_{\text{new}} = 26\text{CCH}$$

$$IP_{\text{new}} = 12\text{ABH}$$

$$SP_{\text{new}} = SP_{\text{old}} - 4 = 00\text{FCH}$$

$$BX_{\text{new}} = BX_{\text{old}}$$

$$DS_{\text{new}} = DS_{\text{old}}$$

$$SS_{\text{new}} = SS_{\text{old}}$$

Example

Give the new values of all the given registers with drawing all the memory locations (addresses and values) that are affected or used by the following instruction:

CALL FAR PTR [SI]

Let BX= 1000H
SI= 200H
DI= F100H
SP= 7E90H
DS= 5A00H
CS= 6400H
IP= 5A0H
SS= 19F0H

You are given that ($IP_{\text{new}} = 6\text{F7BH}$, $CS_{\text{new}} = \text{A590H}$).

Stack Segment

	B00H:100H
20H	B00H: FFH
00H	B00H: FEH
00H	B00H: FDH
30H	B00H: FCH

Solution

Data Segment		Stack Segment	
A5H	5A00H:203H		19F0H:7E90H
90H	5A00H:202H	64H	19F0H:7E8FH
6FH	5A00H:201H	00H	19F0H:7E8EH
7BH	5A00H:200H	05H	19F0H:7E8DH
		A0H	19F0H:7E8CH

$$SP_{\text{new}} = 7E90H - 4 = 7E8CH$$

$$SI_{\text{new}} = SI_{\text{old}}$$

$$DI_{\text{new}} = DI_{\text{old}}$$

$$DS_{\text{new}} = DS_{\text{old}}$$

$$BX_{\text{new}} = BX_{\text{old}}$$

$$SS_{\text{new}} = SS_{\text{old}}$$

2- Return Instruction

Every subroutine must end by executing (RET) instruction to return control to the main program.

يقوم هذا الابعاز بارجاع المعالج الى البرنامج الرئيسي الذي تم القفز منه, وبالضبط الى الابعاز الذي يلي ابعاز CALL وذلك من خلال استرجاع قيمة IP أو كل من CS و IP من الستاك بنفس مبدأ عمل ابعاز POP.

After executing RET instruction the following happen:

A new value for IP is fetched from the stack (if the CALL was intrasegment) with increasing SP by 2, or a new value for CS and IP are fetched from the stack (if the CALL was intersegment) with increasing SP by 4.

1) RET in Intrasegment CALL

$$IP_{\text{Low}} \leftarrow SS: [SP]$$

$$IP_{\text{Hi}} \leftarrow SS: [SP+1]$$

$$SP_{\text{new}} = SP + 2$$

Stack Segment	
	SS: SP+2
IP _{Hi}	SS: SP+1
IP _{Low}	SS: SP

2) RET in Intersegment CALL

$IP_{Low} \leftarrow SS:[SP]$

$IP_{Hi} \leftarrow SS:[SP+1]$

$CS_{Low} \leftarrow SS:[SP+2]$

$CS_{Hi} \leftarrow SS:[SP+3]$

Stack Segment

	SS: SP+4
CS _{Hi}	SS: SP+3
CS _{Low}	SS: SP+2
IP _{Hi}	SS: SP+1
IP _{Low}	SS: SP

$SP_{new} = SP + 4$

Best Regards
Dr. Zainab Alomari

Lecture 13: Delay Loops

Each instruction in 8086Mp takes a specific number of clock cycles for execution.

The time required to execute any instruction = the no. of required clock cycles * clock time (T)

where T is (1/f), where f is the Microprocessor frequency. For 8086Mp, f=5MHz or 10MHz.

الوقت الذي يستغرقه اي ايعاز ليتم تنفيذه من قبل المعالج هو عدد الـ clock cycles التي يحتاجها الايعاز مضروباً في زمن الـ clock الواحدة وهذا الزمن (T) هو مقلوب التردد. وفي معالج 8086 يكون التردد إما 5MHz أو 10MHz.

فاذا علمنا عدد الـ clock cycles التي يحتاجها كل ايعاز (وهو يعطى للطالب) وكذلك تردد المعالج فيمكن ان نعلم الزمن الكلي الذي يستغرقه برنامج معين ليتم تنفيذه.

Examples

XOR → 3T

Push → 11T

POP → 8T

MOV → 4T

JNZ → 16/4T

CALL → 19T

RET → 16T

Loop → 17/5T

(where T is the clock time)

Example

Calculate the delay time taken by the following instructions:

```
MOV CX, 3
N1: LOOP N1
```

(if: MOV → 4T, Loop → 17/5T)

Solution

MOV instruction: 1 time

LOOP instruction: 3 times (2 times: jump is done, 1 time: jump is not done)

$$\begin{aligned} \therefore \text{Delay} &= 4T + (3-1)*17T + 5T \\ &= (4 + 34 + 5)T = 43T \end{aligned}$$

Let frequency = 5MHz

$$\therefore \text{Delay} = 43*1/(5*10^6) = 8.6 \mu\text{sec}$$

NOW if the code is changed to:


```
N1:  MOV CX, 3
      LOOP N1
```

Delay time = ∞

Example

Write a subroutine that generates a delay of 200msec if 8086Mp frequency is 5MHz.

Solution

```
CS:IP CALL 300H  CS:300H  PUSH CX
                                     MOV  CX,N
                                     RPT: LOOP RPT
                                     POP  CX
                                     RET
```

XOR	→	3T
Push	→	11T
POP	→	8T
MOV	→	4T
JNZ	→	16/4T
CALL	→	19T
RET	→	16T
Loop	→	17/5T

$$\begin{aligned} \text{Delay time} &= 19T \\ &+ 11T \\ &+ 4T \\ &+ (N-1)*17T + 5T \\ &+ 8T \\ &+ 16T \end{aligned}$$

$$200\text{msec} = 58T + 5T - 17T + 17NT$$

$$200\text{msec}/T = 46 + 17N$$

$$N=58821 \text{ d} \rightarrow N = \text{E5C5 H}$$

Example

For the code in the previous example, what is the value of N that maximizes the delay? And what is this maximum delay?

Solution

$$N_{\max} = \text{FFFF H}$$

$$\text{delay} / T = 46 + 17 N_{\max}$$


$$\therefore \text{delay} = 223\text{msec}$$

Notice that this subroutine is not enough when the required delay time is greater than 223msec.

Example

Write a subroutine that generates a delay of 500msec:

Solution

CS:IP CALL 500H  CS:500H PUSH CX
MOV CX, N
RPT: NOP
PUSH AX
POP AX
LOOP RPT
POP CX
RET

XOR	→	3T
Push	→	11T
POP	→	8T
MOV	→	4T
JNZ	→	16/4T
CALL	→	19T
RET	→	16T
Loop	→	17/5T
NOP	→	3T

$$\begin{aligned} \text{Delay time} &= 19T \\ &+ 11T \\ &+ 4T \\ &+ N (3T + 11T + 8T) \\ &+ (N-1)*17T + 5T \\ &+ 8T \\ &+ 16T \end{aligned}$$

$$500\text{msec} / T = 63 - 17 + 17N + 22N$$

$$N = 64101 \text{ d} \rightarrow N = \text{FA65 H}$$

Example

Write a subroutine that generates a delay of 10sec if the 8086Mp frequency is 5MHz:

Solution

```

CS:IP CALL A00H → CS:A00H  PUSH CX
                                PUSH AX
                                MOV  AX,N
LP2:  MOV  CX,0FFFFH
LP1:  NOP
                                PUSH AX
                                PUSH AX
                                POP  AX
                                POP  AX
                                LOOP LP1
                                DEC  AX
                                JNZ  LP2
                                POP  AX
                                POP  CX
                                RET

```

XOR	→	3T
Push	→	11T
POP	→	8T
MOV	→	4T
JNZ	→	16/4T
CALL	→	19T
RET	→	16T
Loop	→	17/5T
NOP	→	3T
DEC	→	2T

$$\begin{aligned}
 \text{Delay time} = & 19T \\
 & + 11T \\
 & + 11T \\
 & + 4T \\
 & + N [4T + 65535 * (3T + 11T + 11T + 8T + 8T) + 65534 * (17T) + 5T + 2T] \\
 & + (N-1)*16T + 4T \\
 & + 8T \\
 & + 8T \\
 & + 16T
 \end{aligned}$$

$$10 \text{ sec} = 65T + 3801040 \text{ NT}$$

At $f = 5\text{MHz}$:

$$N = 13.15 \approx 13 \text{ d} \rightarrow N = 0C \text{ H}$$

In order to perform a delay of 40seconds using the same code, only the value of N is changed.

At delay = 40sec

$$40 \text{ sec} = 65T + 3801040 \text{ NT}$$

$$N = 52.6 \approx 53 \text{ d} \rightarrow N = 35 \text{ H}$$

Example

Find the delay time of the following subroutine:

```
CS:IP CALL 600H → CS:600H PUSH CX
                                     PUSH AX
                                     MOV CX, 800H
NXT: MOV AX, 01H
                                     PUSH AX
                                     POP CX
                                     LOOP NXT
                                     POP AX
                                     POP CX
                                     RET
```

Solution

All the instructions of this subroutine will be executed only one time.

$$\text{Delay time} = 19T + 11T + 11T + 4T + 4T + 11T + 8T + 5T + 8T + 8T + 16T$$

$$= 105T = 105 / (5 \times 10^6) = 21 \mu\text{sec}$$

Example

What happens to the delay time of the previous subroutine if instruction 4 is changed to:

MOV AX, 2

Solution

Delay time = ∞

Example

What happens to the delay time of the previous subroutine if instruction 5 is changed to:

PUSH CX

Solution

The instruction (Loop NXT) and the 3 instructions before it will be executed 800H times.

Delay time = 16.4 μ sec

Example

Find the delay time of the following subroutine:

```
CS:IP CALL 1ABCH          CS:1ABCH PUSH CX
                                MOV  CX, 400H
                                NXT: PUSH AX
                                POP  AX
                                LOOP NXT
                                POP  CX
                                RET
```

Solution

Delay time = 7.382 msec

* * * * *

Best Regards
Dr. Zainab Alomari

Lecture 1: In/Out Instructions

To deal with input/output devices, 8086Mp programmer needs to know three things:

- 1- **Type of device:** it is important to know if the device is an input or output device, in order to use the suitable instruction. Some devices are (input/output) devices.
- 2- **Port number:** This is a number given to the I/O device during manufacturing. Each I/O device has its own unique port number. Port numbers can be 8-bits or 16-bits number.
- 3- **Data Length:** An I/O device will send/receive a piece of information to/from 8086Mp. This information is of 8-bits or 16-bits length (depending on the type of the device).

Note1: These three parameters are fixed for each device and can't be modified.

Note2: the length of the port number and data of any device are not related together, there can be a device with a data length of 16-bits and a port number of 8-bits length, or a device with 8-bits data length and 16-bits port number.

Note3: some devices can be input and output in the same time, i.e. you can read data from such a device and also you can send data to it using its port number.

عند التعامل مع اجهزة الادخال والاخراج يجب معرفة ثلاث امور: هل الجهاز هو جهاز ادخال ام اخراج أم كلاهما؟ وما هو رقم البورت (Port number) الخاص به؟ وهل data length لهذا الجهاز هو 8بت أم 16بت؟ هذه المعلومات يجب ان تعطى في السؤال.

In instruction:

This instruction transfers data from external I/O device to **AL** or **AX**.

In AL, port no. ; used when the data length is 8-bits

In AX, port no. ; used when the data length is 16-bits

Out Instruction:

This instruction transfers data from **AL** or **AX** to external I/O device.

Out port no. , AL ; used when the data length is 8-bits

Out port no. , AX ; used when the data length is 16-bits

Note: only AL or AX is used by the 8086Mp to send/receive data to/from I/O devices:

AL if the data length of the device is 8-bits.

AX if the data length of the device is 16-bits.

Fixed and Variable Port Addressing

If the device has an **8-bits** port number, then it is used directly in the In/Out Instruction, as in the above examples. This is called (**Fixed Port Addressing**).

إذا كان رقم البورت الخاص بالجهاز يتكون من 8 بت فقط فيتم استخدام هذا الرقم مباشرةً في ايعاز in و out كما في الامثلة التالية:

Examples

In AL, 96H ; an 8-bit data is copied from an input device that has a port no.= 96H to AL.

In AX, 7AH ; a 16-bit data is copied from an input device that has a port no.= 7AH to AX.

Out 3FH, AL ; an 8-bit data is sent from AL to an output device that has a port no.= 3FH.

Out 19H, AX ; a 16-bit data is sent from AX to an output device that has a port no.= 19H.

While if the port number of a device is of **16-bits** length, it must be given to DX, and then DX is used in the In/Out Instruction. This is called (**Variable Port Addressing**).

أما إذا كان رقم البورت الخاص بالجهاز يتكون من 16 بت فلا يجوز ان يستخدم مباشرةً في ايعاز in و out بل يجب وضعه في DX باستخدام ايعاز MOV وبعدها تستخدم DX على انها رقم البورت, كما في الامثلة التالية:

Examples

```
Mov DX, 3FA0H  
In AL, DX
```

```
Mov DX, 99A3H  
In AX, DX
```

```
Mov DX, 51BCH  
Out DX,AL
```

```
Mov DX, 972H  
Out DX,AX
```

Example

(THE INTEL MICROPROCESSORS) \Example 4-12 (Page 140)

Write the required instructions to set the right most two bits of the speaker (port no.= 61H), then clear them after a specific delay time. (The speaker has an 8-bit register).

Solution

```
IN AL, 61H
OR AL, 3
OUT 61H, AL

MOV CX, 1000H
L1: NOP
LOOP L1

IN AL, 61H
AND AL, 0FCH
OUT 61H, AL
HLT
```

NOTE: You can also find some help on this topic from the tutorials available in the Emulator program.

*Best Regards
Dr. Zainab Alomari*

Lecture 2: Interrupts (Part 1)

Interrupts are special type of (CALL) instruction. If any interrupt occurs while the Mp is executing a program, it breaks the execution and starts executing a subroutine called Interrupt Service Routine (ISR). After executing the ISR, the Mp returns to the main program and continues from the point it stopped at.

يمتلك المعالج 8086 القدرة على التعامل مع الأجهزة الخارجية (أجهزة الادخال واجهزة الاخراج) مثل الطابعة, مصابيح الإضاءة, LED, الشاشة, مقياس الضغط, مقياس الحرارة, المروحة ... الخ.

وينتطلب التعامل مع هذه الاجهزة وجود تقنية المقاطعة, وهي قدرة اي جهاز أن يقوم بمخاطبة المعالج لطلب خدمة معينة, ويقوم المعالج بالاستجابة لهذه المقاطعة حتى في حال كونه يقوم بتنفيذ برنامج معين حيث يتوقف عن التنفيذ ويستجيب للجهاز صاحب الطلب, ويقوم بتنفيذ برنامج الخدمة الخاص بهذا الجهاز والذي يسمى (ISR) وبعد ذلك يعود الى البرنامج ويكمل من النقطة التي توقف عندها. وهذه الطريقة (خدمة الجهاز ثم العودة الى البرنامج الرئيسي) شبيهة بمبدأ عمل ايعاز CALL.

ماهو الـ (ISR)? هو عبارة عن برنامج فرعي مكتوب مسبقاً ومخزون في ذاكرة المعالج, يحتوي على ايعازات معينة يتم تنفيذها من قبل المعالج عند الاستجابة لطلب مقاطعة معين. لكل مقاطعة من المقاطعات ISR خاص بها.

There are 256 interrupts in the 8086Mp:

From Interrupt 0 → to Interrupt 255 (or: from Interrupt 0H → to Interrupt FFH).

These interrupts are divided into two types:

- 1- Interrupts that are reserved for the present and future products and system errors.
They are: (from Interrupt 0H → Interrupt 1FH), i.e. the first 32 interrupts.
- 2- Interrupts available for user. These are the interrupts from Interrupt 20H → Interrupt FFH.

Q1) How many interrupts are available for user?

المقاطعات المتوفرة في معالج 8086 على نوعين:

1- مقاطعات محجوزة (وهي أول 32 مقاطعة) ويقوم المعالج بتنفيذها عند الحاجة اليها, مثلاً المقاطعة الاولى (Interrupt 0) يتم تنفيذ الـ ISR الخاص بها من قبل المعالج في حالة حصول قسمة على 0, حيث تقوم الايعازات الموجودة في هذا الـ ISR بايقاف تنفيذ البرنامج الحالي والخروج منه وطباعة error على الشاشة. بعض من هذه المقاطعات محجوز ولكنه غير مستخدم حيث كان من المتوقع اضافة تطويرات على المعالج فتم حجزها لخدمة هذه التطويرات مستقبلاً.

2- مقاطعات متوفرة للمستخدم: يمكن للمستخدم أن يُعرف مقاطعة معينة ويعطيها رقم من 20H الى FFH ويقوم بخزن برنامج خدمة هذه المقاطعة (ISR) في الذاكرة, ويقوم باستدعاء هذه المقاطعة متى شاء خلال برنامجه باستخدام ايعاز (INT) ويكتب أمامه الرقم الخاص بالمقاطعة, وحين يصل المعالج خلال تنفيذ البرنامج الى ايعاز المقاطعة هذا يقوم بالقفز الى الـ ISR المطلوب وتنفيذه ثم العودة لإكمال البرنامج الرئيسي.
(سيأتينا لاحقاً كيف يختار المستخدم رقم المقاطعة وكيف يعلم المعالج موقع خزن الـ ISR)

Interrupt Vectors:

Each interrupt has an interrupt vector, which is a 4-byte vector containing the address (CS & IP) of the ISR. The first 2-bytes contain the IP and the second 2-bytes contain the CS.

- The interrupt vector of **Interrupt 0** is stored in the **first** 4-bytes of the memory (addresses: 00000H → 00003H).
- The interrupt vector of **Interrupt 1** is stored in the **second** 4-bytes of the memory (addresses: 00004H → 00007H).
- The interrupt vector of **Interrupt 2** is stored in the **third** 4-bytes of the memory (addresses: 00008H → 0000BH).

and so on.

فيما ان عناوين الـISR مخزونة بشكل متسلسل في الذاكرة ابتداءً من الموقع 00000H, فبالتالي ممكن تحديد عنوان الـISR لأي مقاطعة من خلال ضرب رقم المقاطعة في 4, مثال:

- The interrupt vector of **Interrupt 7** is stored in the **seventh** 4-bytes of the memory.
 $7 * 4 = 28d = 1CH$
∴ The interrupt vector is stored at addresses: (0001CH → 0001FH)

The Mp multiplies the interrupt number of the interrupt by 4 to find the address where the interrupt vector (CS and IP) is stored.

Thus for 256 interrupts, there are 256 interrupt vectors.

Interrupt Vector Table (IVT):

IVT is the part of the memory where the interrupt vectors of all the interrupts are stored. IVT is located at the beginning of the 8086Mp memory.

Q2) What is the size of IVT?

Q3) What are the first and last addresses of the IVT?

نلاحظ بأن العنوان الذي يخزن فيه الـCS&IP (أي الـInterrupt vector) لكل المقاطعات يقبل القسمة على 4 وذلك لان عنوان الـISR لكل مقاطعة يحتاج الى اربع مواقع, فبالتالي:

1- اذا كان المطلوب ايجاد عنوان مواقع الذاكرة المخزون فيها موقع الـISR الخاص بالمقاطعة من رقمها فنقوم بضرب رقم المقاطعة في 4 وذلك باضافة صفرين أمام الرقم بعد تحويله الى النظام الثنائي (مما يعادل ضرب في 4) كما في المثال السابق.

2- اذا كان المطلوب ايجاد رقم المقاطعة من معرفة عنوان الـIVT الخاص بها فنقوم بتقسيم العنوان على 4 وذلك بحذف بتين من يمين الرقم بعد تحويله الى النظام الثنائي (مما يعادل القسمة على 4).

Example

If an interrupt vector is stored in the IVT starting at address: (001A8H), find the number of the interrupt. Also give the addresses of the IVT where CS and IP of this interrupt are stored.

Sol:

001A8H
↓
0000 0000 0001 1010 1000

نحذف صفرين من جهة اليمين للقسمة على 4:

∴ The interrupt number is 6AH.

The four addresses of the IVT where CS and IP are stored are:
001A8H, 001A9H, 001AAH, 001ABH.

IP_{low} is stored at: 001A8H and IP_{Hi} is stored at: 001A9H

CS_{low} is stored at: 001AAH and CS_{Hi} is stored at: 001ABH

CS _{hi}	001ABH
CS _{low}	001AAH
IP _{hi}	001A9H
IP _{low}	001A8H

Interrupt Priorities:

Interrupts are served on a priority basis. Interrupt 0 has the highest priority, while interrupt 255 has the lowest priority.

For example: Interrupt 60H has lower priority than interrupt 5AH.

يتم العمل مع المقاطعات بنظام الأسبقية عندما يكون هنالك اكثر من طلب مقاطعة في الوقت الواحد, فيقوم المعالج بخدمة المقاطعة ذات الاسبقية الأعلى وهي التي رقم مقاطعتها أقل, وبعد الانتهاء من خدمتها ينتقل لخدمة المقاطعة ذات الاسبقية الاقل.

Q4) Sort the following interrupts according to their priorities, from the highest priority to the lowest:

Interrupt ABH, Interrupt A0H, Interrupt BAH and Interrupt BCH.

Answers:

Q1_Ans: There are 224 interrupts available for user.

Q2_Ans: $256 * 4 = 1024$ Byte = 1Kbyte.

Q3_Ans: The first address is 00000H and the last address is 003FFH.

Q4_Ans: 1- Interrupt A0H
 2- Interrupt ABH
 3- Interrupt BAH
 4- Interrupt BCH

Best Regards
Dr. Zainab Alomari

Lecture 3: Interrupts (Part 2)

There are three types of Interrupts:

1- Software Interrupts:

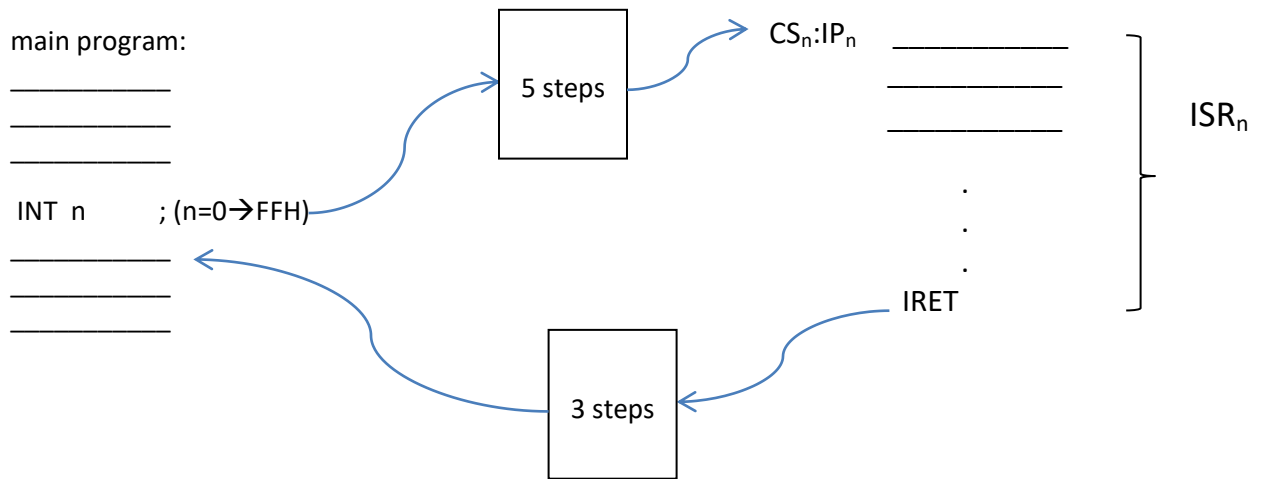
This interrupt uses the instruction (INT n), where n is the interrupt number and it can be any number between (0H→FFH).

هذا النوع من المقاطعات بإمكان المستخدم استدعائه باستخدام الأيعاز (INT n) وتكون قيمة n أي رقم من 0H→FFH.

When (INT) instruction is executed, 8086Mp performs the following 5 steps:

- 1) The Flags register (2bytes) is pushed onto the stack.
- 2) TF and IF are cleared (TF=0 and IF=0).
- 3) CS is pushed onto the stack.
- 4) IP is pushed onto the stack.
($SP_{new} = SP_{old} - 6$)
- 5) New values for CS and IP are fetched from the IVT.

عندما يقوم المعالج بتنفيذ برنامج معين ويصل إلى إيعاز (INT n) حيث تمثل n رقم المقاطعة (كما في المثال المرسوم)، يقوم المعالج بالقفز إلى الـ ISR الذي يحتوي إيعازات الخدمة التي تقدمها هذه المقاطعة، وعملية القفز هذه تتطلب الوصول إلى المكان المخزون فيه هذا الـ ISR والتي يتم الحصول عليها من جدول الـ IVT حسب رقم المقاطعة كما شرح في المحاضرة السابقة. ولغرض العودة إلى البرنامج الرئيسي بعد الانتهاء من تنفيذ الـ ISR، فإن المعالج بحاجة إلى الاحتفاظ بالقيم القديمة لكل من CS & IP (كما يحصل في إيعاز Call)، كما يقوم أيضاً بالاحتفاظ بحالة الأعلام Flags. يتم عمل Push لكل من الـ CS, IP & Flags register إلى الستاك، مما يؤدي إلى إنقاص قيمة المؤشر الخاص بالستاك (SP) بمقدار 6.



Each ISR ends with (IRET) instruction. When IRET instruction is executed, the Mp performs the following 3 steps:

- 1) IP takes its value back from the stack.
- 2) CS takes its value back from the stack.
- 3) Flags register takes its value back from the stack.
($SP_{new} = SP_{old} + 6$)

- ينتهي كل ISR بايعاز (IRET) والذي يقوم باستعادة قيم كل من CS, IP & Flags register من الستاك والتي كان قد تم تخزينها عند تنفيذ ايعاز (INT n), وذلك لغرض العودة لاكمال تنفيذ البرنامج الرئيسي من المكان الذي توقفنا عنده.
- نلاحظ هنا التشابه الكبير بين ايعازي (INT) و (Call), حيث أن الفرق هو اننا في ايعاز المقاطعة نقوم بتحديد رقم المقاطعة ويتم أخذ الـ CS&IP من الـ IVT أما في ايعاز Call فيتم اعطاء قيمة CS&IP مباشرة في الـ ايعاز.

Q) What are the benefits of INT instruction compared to Call instruction?

Answer:

- 1- No need to remember the address of the system call.
- 2- Each time INT instruction is used instead of Call instruction, 3 bytes are saved (because INT is 2 bytes long, while Call is 5 bytes long).

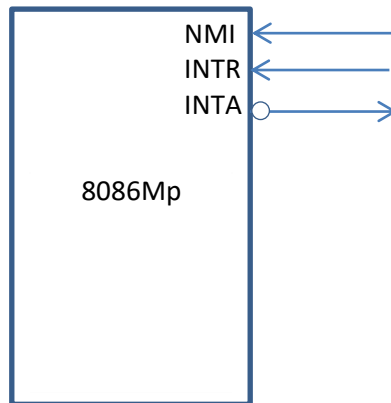
ما الفائدة من استخدام INT بدلاً من ايعاز Call ؟

الاجابة:

اولاً: إن طول الـ ايعاز INT هو بايتين فقط في حين أن طول ايعاز Call هو 5 بايت مما يوفر مواقع من الذاكرة.
وثانياً: لسنا بحاجة الى تذكر قيمة العنوان الخاص بالبرنامج الفرعي. بل نحتاج الى تذكر رقم المقاطعة فقط.

2- Hardware Interrupts:

There are three pins used for hardware interrupts in 8086Mp: (NMI, INTR and INTA).



Steps for hardware interrupt:

- 1) The I/O device requests a service by activating INTR pin (INTR=1).
- 2) The Mp accepts the request by pulsing \overline{INTA} pin to 0 logic. If the Mp was busy, he finishes his task and then pulses \overline{INTA} to 0 logic.
- 3) The I/O device puts the number of the interrupt on the first 8-bits of the data bus (D0→D7).
- 4) The Mp pushes the Flags register on to the stack.
- 5) TF and IF are cleared.
- 6) CS and then IP are pushed onto the stack.
- 7) New CS and IP values are fetched from the IVT.

Interrupt Flag (IF):

Hardware interrupts are disabled when interrupt flag (IF) is 0. When interrupt flag is set to 1, it enables hardware interrupts to be received. By default hardware interrupts are enabled (IF=1). Hardware interrupts are disabled (IF=0) automatically when software or hardware interrupt is in the middle of the execution.

There are two instructions that can be used to clear or set IF:

CLI ; Clear Interrupt Flag (IF=0). This disables INTR pin (no hardware interrupt is received).

STI ; Set Interrupt Flag (IF=1). This enables INTR pin (enables receiving hardware interrupts).

Non-Maskable Interrupt (NMI):

NMI is a special type of hardware interrupt that the system can't ignore. NMI is used to report important issues to the Mp such as errors or power failures. When this interrupt occurs, the Mp stores all the internal registers in a battery-backed up memory or an EEPROM. This interrupt is a system error and it is given a high priority (INT 2).

عندما يصبح هذا الإدخال (1 logic) فهذا يعني وجود مشكلة تتطلب قيام مقاطعة المعالج عن العمليات التي يقوم بها والقيام بمجموعة من الأمور أهمها تخزين قيم جميع السجلات في ذاكرة خاصة، وهذا يتم من خلال تنفيذه للمقاطعة رقم 2 الخاصة بال-NMI.

3- Internal Interrupts:

These interrupts are executed without the need to use INT instruction, due to a specific system event or error, like: divide by zero (which resumes INT 0). These interrupts are: INT 0, INT 1, INT 3 and INT 4.

Table 1 describes all the interrupts available in 8086Mp.

Table 1: Description of the function of each of the 8086Mp interrupts

Interrupt	Description
INT 0	Division by zero
INT 1	Single step
INT 2	NMI
INT 3	Break point
INT 4	Overflow
INT 5 → INT 31 (or: INT 5 → INT 1FH)	Not used
INT 32 → INT 255 (or: INT 20H → INT FFH)	User defined

Best Regards
Dr. Zainab Alomari

Lecture 4: Directives

Directives are included in the source code that contains assembly language code. Directives are used to give directions to the compiler (compilers convert the com file in the Emulator to machine code). Unlike assembly language instructions, directives have no machine code.

Examples

- (Org 100h) is a directive used to tell the compiler to load the program at offset 100H in the code segment.
- (END) is a directive to stop the compiler.

Note: in the com file, directives are given in violet color while instructions are written in blue.

هي عبارة عن توجيهات تعطى للـ compiler مع البرنامج المكتوب بلغة الـ Assembly, وليست موجهة الى الـ Mp كالايعازات, فبالنتالي لا يتم تحويلها الى الـ machine code. مثالها توجيه (Org 100h) الذي يحدد الموقع الذي سيتم خزن البرنامج فيه من الـ code segment, وتوجيه (END) الذي يقوم بايقاف الـ compiler. نلاحظ أن التوجيهات تعطى اللون البنفسجي في حين تعطى ايعازات لغة الـ Assembly اللون الازرق.

ملاحظة: الـ compiler هو البرنامج الذي يقوم بترجمة وتحويل ما مكتوب في الـ com file من ايعازات وتوجيهات.

Variables and Arrays

A) Variables:

A variable needs to have name, length (byte or word) and value:

Name	DB	Value
	DW	

Note1: Variables are defined at the end of the source code (in com files) after RET instruction.

Note2: DB means define byte, DW means define word.

Examples

- 1) To define a variable word named k1, which has the value (5F0H), we write:

```
k1 DW 5F0H
```

- 2) To define a variable byte named NUM, which has the value (70d), we write:

```
NUM DB 70
```

- 3) To define a variable word named Var1, which has the value (6AH), we write:

```
Var1 DW 6AH
```

Note: It is also possible to give the value of the variable in binary as:

Var1 DW 11010b (نلاحظ في هذه الحالة ان باقي البتات من جهة اليسار تساوي صفر)

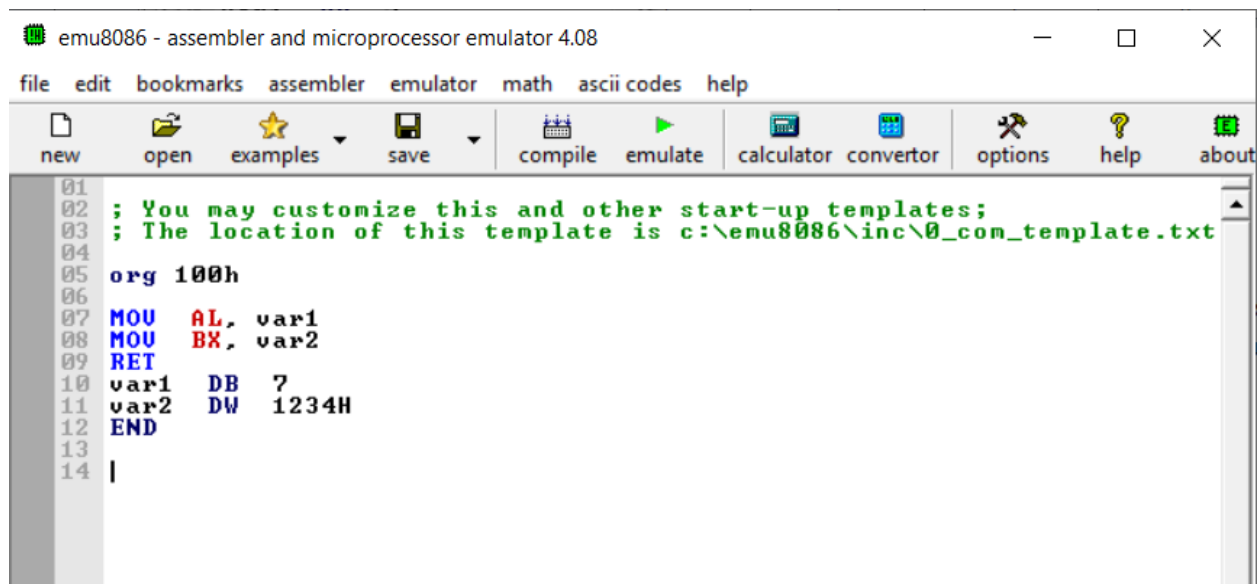
4) To define a variable byte named K2, which has no initial value, we write:

K2 DB ?

NOTE: Defining a variable or array is a directive (not an instruction).

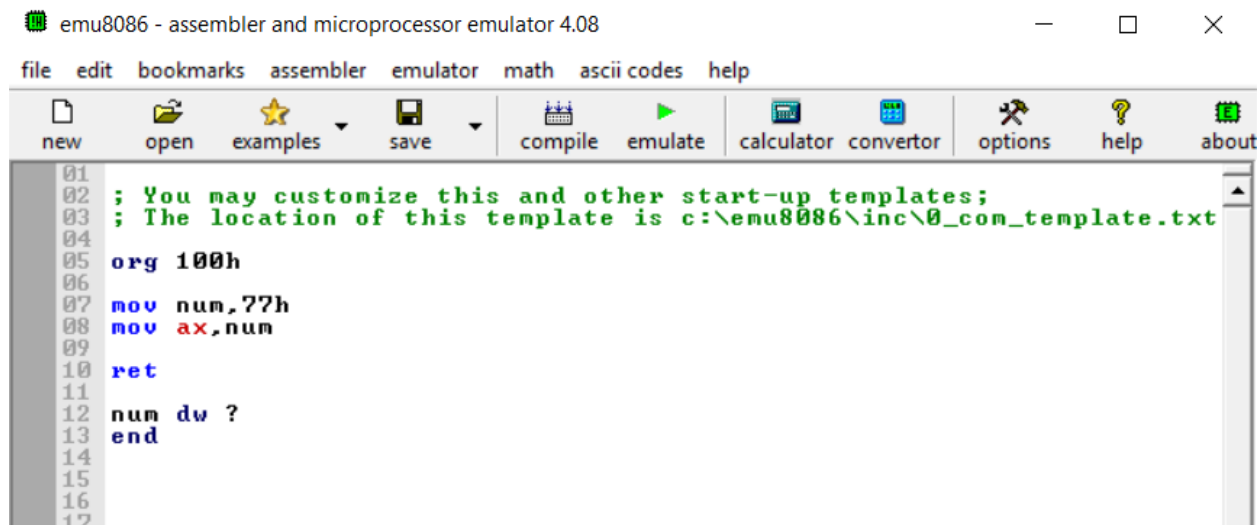
When a variable is defined, it is possible to use it in the Assembly language instructions, where a byte (or word) is allocated in the memory for this variable.

Example



```
01  
02 ; You may customize this and other start-up templates;  
03 ; The location of this template is c:\emu8086\inc\0_com_template.txt  
04  
05 org 100h  
06  
07 MOV AL, var1  
08 MOV BX, var2  
09 RET  
10 var1 DB 7  
11 var2 DW 1234H  
12 END  
13  
14 |
```

Example



```
01  
02 ; You may customize this and other start-up templates;  
03 ; The location of this template is c:\emu8086\inc\0_com_template.txt  
04  
05 org 100h  
06  
07 mov num, 77h  
08 mov ax, num  
09  
10 ret  
11  
12 num dw ?  
13 end  
14  
15  
16  
17
```

B) Arrays:

It is possible to define an array by giving the array name, length of each element (byte or word) and elements values.

Example

```
A DB 48h, 65h, 6ch, 6ch, 6fh, 0h      (A[0]=48h, A[1]=65h, ...)  
B DB 'Hello', 0                      (B[0]=48h, B[1]=65h, ...)
```

(Note that array B is an exact copy of A, where each character in the string (Hello) is stored as a byte with the equivalent ASCII code).

نلاحظ امكانية اعطاء القيم باكثر من طريقة: الثنائي, العشري, السادس عشر ورموز Characters (ويتم في النوع الاخير خزن المكافئ الـ ASCII لهذا الـ Character).

Now we can access any element in array A as:

```
MOV AL, A[4] ; AL= 6fH
```

Or by using one of the pointers: BX, SI or DI as:

```
MOV SI, 4  
MOV AL, A[SI]
```

خلال البرنامج لدينا امكانية للوصول الى القيم التي تم تعريفها في نهاية البرنامج, وبدون الحاجة لمعرفة الموقع الذي تم خزنها فيه من الذاكرة, وذلك باحدى الطريقتين السابقتين.

Getting the address of variables

Two ways can be used to access a variable address:

1) Using **LEA** instruction:

```
LEA BX, var1 ; BX = the offset address of var1
```

2) Using **OFFSET**:

```
MOV BX, OFFSET var1 ; BX = the offset address of var1
```

NOTE1: Any 16-bit register can be used instead of BX in these two examples.

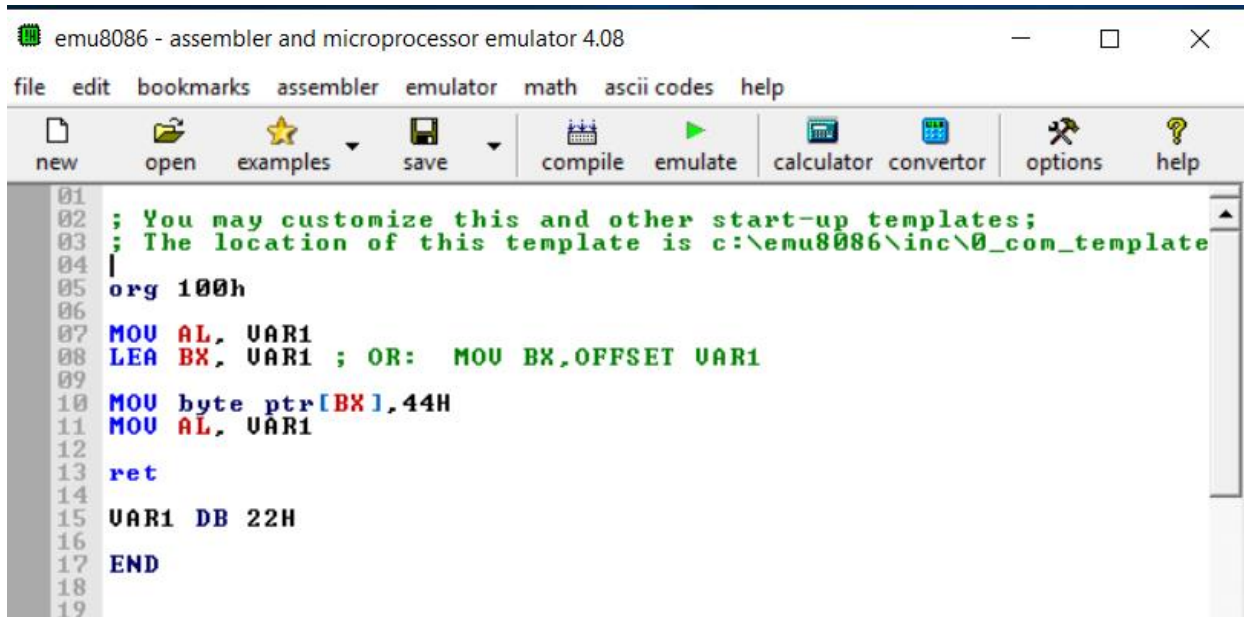
NOTE2: when com file is loaded, the value of DS is set to the same value of CS.

في حالة الحاجة للوصول الى العنوان الذي تم خزن متغير معين فيه من الذاكرة فيتم ذلك باحدى الطريقتين السابقتين, مع الانتباه الى أن العنوان الذي سنحصل عليه سيكون عبارة عن 16-bit offset وبالتالي يجب خزنه في سجل طوله 16-bit.

س/ في أي Segment سوف يتم خزن هذا المتغير؟

الجواب: عندما يتم تحميل الـ com file فانه يتم توحيد قيمة كل من الـ DS&CS بحيث يشير كلاهما الى نفس المقطع من الذاكرة, وبالتالي يمكننا اعتبار المتغير مخزوناً في أي منهما.

Example



```
01 ; You may customize this and other start-up templates;
02 ; The location of this template is c:\emu8086\inc\0_com_template
03 |
04 |
05 org 100h
06
07 MOV AL, VAR1
08 LEA BX, VAR1 ; OR: MOV BX, OFFSET VAR1
09
10 MOV byte ptr[BX], 44H
11 MOV AL, VAR1
12
13 ret
14
15 VAR1 DB 22H
16
17 END
18
19
```

في المثال أعلاه, تم تغيير قيمة المتغير VAR1 من 22h الى 44h وتم التأكد من حصول التغيير من خلال وضع قيمة المتغير في AL, ولكن يمكن أيضاً تجييك قيمة هذا المتغير من خلال الذهاب الى الموقع الذي تم خزنه فيه في الذاكرة والذي يتمثل بـ (DS:BX) وملاحظة تغير القيمة الى 22h بعد تنفيذ البرنامج.

Note:

MOV AX, offset VAR1

In this instruction, if the variable VAR1 is an array, then AX will be the offset of the first element of this array, i.e. :

MOV AX, offset VAR1 = MOV AX, offset VAR1[0]
= LEA AX, VAR1 = LEA AX, VAR1[0]

Best Regards
Dr. Zainab Alomari

Lecture 5: DOS Interrupts

The interrupt types 20h-3Fh are serviced by DOS routines that provide high-level service to hardware as well as system resources such as files and directories. The most useful is INT 21H, which provides many functions for doing keyboard, video, and file operations.

INT 21H (INT 33)

This interrupt performs various operations depending on the value of AH.

1) Reading a Character

After executing (INT 21h), if AH is equal to (1) then a character is read from keyboard with its ASCII stored in AL.

Example

```
MOV AH, 01  
INT 21H
```

If the number 4 is inserted by keyboard after execution, then AL will be equal to (34H).

2) Printing a Character

After executing (INT 21h), if AH is equal to (2) then a character is printed on screen, the ASCII code of this character is taken from DL.

NOTE: after printing the character on screen, the value of DL is also copied to AL.

Example

```
MOV DL, 52H  
MOV AH, 02  
INT 21H
```

After execution, the letter (R) is printed on screen and AL = DL = 52H.

3) Printing a String

After executing (INT 21h), if AH is equal to (9) then a string of characters is printed on screen. The printed string must be stored starting at (DS:DX), i.e. the offset of the first character in the string is stored at offset = DX. The printing is stopped when it reaches \$.

Example

```
org 100h
MOV DX, OFFSET msg
MOV AH, 9
INT 21H
RET
msg DB "Hello World$"
```

4) Reading a String

After executing (INT 21h), if AH is equal to (10) then a string of characters is read from keyboard and stored starting at (DS:DX+2), where the bytes at DX and DX+1 are reserved for the buffer size and the number of characters stored in the buffer, respectively.

- قبل استخدام هذا النوع من مقاطعة 21h يجب تهيئة ذاكرة (Buffer) مسبقاً والتي يجب ان يُوشر DS:DX على اول موقع منها. البايت الاول من هذه البفر يمثل حجم البفر والبايت الثاني يمثل عدد القيم الموجودة في البفر (والذي يجب ان لا يتجاوز الطول المعطى في البايت الاول), وبالتالي الرموز التي يتم ادخالها تخزن ابتداءً من DS:DX+2.
- يتم اعتبار ادخال (enter) من قبل المستخدم علامة الانتهاء من الادخال ولذلك يتم تخصيص حقل لخرن قيمة الـ(enter) والذي يساوي (0DH) ولكنه لا يحسب ضمن الطول الموجود في الحقل الثاني. وبالتالي تكون اقصى قيمة للبايت الثاني هي قيمة البايت الاول - 1.

NOTE: This function does not add \$ at the end of the string. So to print this string using INT 21h at AH=9, you must store \$ at the string end first, then start printing from DS:DX+2.

Example

Write a program in Assembly language to read a string that has a maximum length of 10 characters, then print this string on the screen.

Solution

```
org 100h
MOV DX, OFFSET BUFFER
MOV AH, 0AH
INT 21H
MOV BH, 00
MOV BL, BUFFER[1]
MOV BUFFER[BX+2], '$'
MOV DX, OFFSET BUFFER+2
MOV AH, 9
INT 21H
ret
BUFFER DB 10, ?
END
```

Example

Write a program in Assembly language to print the value of an 8-bit number on the screen **IN BINARY**.

Solution

(To solve this question, it is important to notice that printing on screen starts from the left, therefore the number should be printed from the higher bit to the lower. Also remember that we need to know the ASCII code of any character in order to print it on screen.)

<u>ASCII</u>	<u>Hexadecimal</u>
0	30h
1	31h

```
org 100h
MOV CL,8      ; counter for printing times
MOV BL,NUM
RPT:MOV DL,30H ; the value of DL will be printed on screen
SHL BL,1     ; checking if the bit is 1 or 0
JNC PRNT    ; if the bit is zero, go directly to print
INC DL       ; if the bit is one, add one to DL and then print
PRNT:MOV AH,2
INT 21H
DEC CL
JNZ RPT
RET
NUM DB 45H   ; example number to be printed on screen in binary
```

Example

Write a program to print the value of any 16-bit number on screen **IN HEXADECIMAL**.

Solution

(To write this program, we need to check the digits of the number starting from the higher digit to the lower, if the digit is between 30H and 39H (after adding 30h to the digit) then we print the value directly on screen. Otherwise if the digit is more than 39H (then it is a value between A and F), then we need to add a shift of 7 to the value before printing. Why seven? This can be explained by looking at the ASCII codes list, we can see that there are 7 characters between numbers and letters.)

ASCII Hexadecimal

0	30h
1	31h
2	32h
3	33h
4	34h
5	35h
6	36h
7	37h
8	38h
9	39h

(other seven characters whose ASCII codes are: 3Ah,3Bh,3Ch,3Dh,3Eh,3Fh,40h)

A	41h
B	42h
C	43h
D	44h
E	45h
F	46h

org 100h

```
    MOV CL,16
RPT:SUB CL,4
    MOV BX,NUM
    SHR BX,CL      ; shifting the number by 12, 8, 4 and 0 (to take one digit each time)
    AND BX,000FH ; to make sure that only one digit is there after shifting
    ADD BX,30H
    CMP BX,39H    ; checking the number
        JBE PRNT ; if the number is between 30h and 39h, go directly to print
        ADD BX,7  ; if the number is between A and F, add 7 before printing
PRNT:MOV DL,BL   ; the number is only 8-bits long
    MOV AH,2
    INT 21H
    CMP CL,0
    JNZ RPT
RET
NUM DW 8E1FH     ; example number to be printed on screen in hexadecimal
```

Example

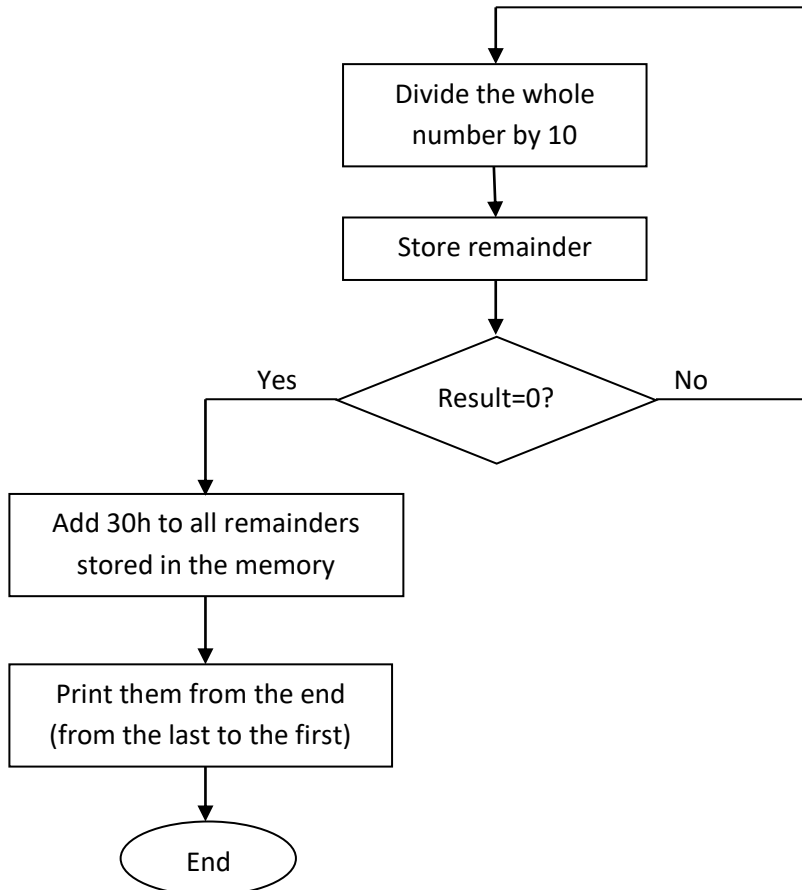
Write a program to print the value of any 16-bit number on screen **IN DECIMAL**.

Solution

(When converting a number from Hexadecimal to decimal, we divide the number by 10 and the remainder represents the decimal number. This method will be used in the code so that the remainder is printed on screen after adding 30h. The remainder is stored in an array in the memory so that it is printed from the last number to the first).

Simple example:

10 D h	<u>remainder</u>
10 1	3
0	1



org 100h

```
MOV SI, 00 ; counter for the number of remainders
MOV BX, OFFSET REMS
MOV CX, 0AH ; to divide by 10
MOV AX, NUM ; the number to be converted from Hexa to decimal
RPT: MOV DX, 00 ; DX is cleared for the division
DIV CX ; dividing the number by 10
MOV [BX], DL ; storing remainder (which is always between 0 and 9)
INC SI
INC BX
CMP AX, 00 ; division is repeated until the result of division is 0
JNZ RPT
PRNT: DEC BX ; if division result is zero, start printing
DEC SI
MOV DL, [BX]
ADD DL, 30H ; each number must be added with 30h before printing
MOV AH, 2
INT 21H
CMP SI, 0
JNZ PRNT
RET

NUM DW 0FFFFH ; example number to be printed on screen in decimal
REMS DB ? ; an empty array of bytes for storing division remainders
```

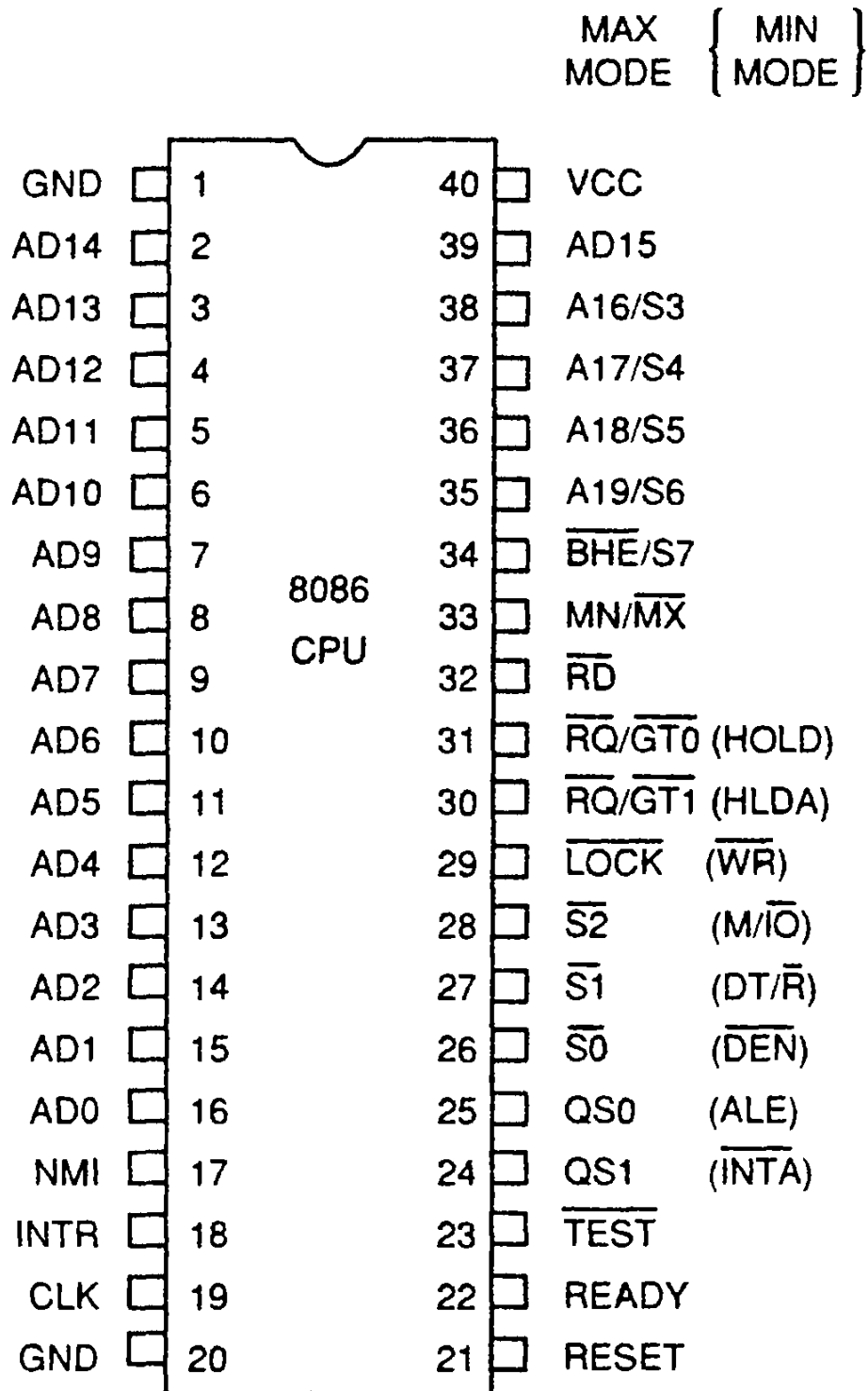
Example

Write a program to convert a character named (*char*):

- 1) from small to capital letter
- 2) from capital to small letter
- 3) if capital → small, if small → capital

Best Regards
Dr. Zainab Alomari

Lecture 6: 8086Mp Pin-out Diagram



There are 3 buses in 8086Mp:

- **Data bus:** 16 data lines (D0 – D15)
- **Address bus:** 20 Address lines (A0 – A19)
- **Control bus:** 3 lines ($\overline{M}/\overline{IO}$, \overline{RD} , \overline{WR})

Multiplexing in 8086Mp

- **Multiplexing** is the use of one line for two different signals.
- Address/Data lines (AD0 – AD15) are multiplexed.
- **Address/Data Multiplexing** means that the same pin carries an address bit at one time and a data bit at another time.
- Multiplexing is used to minimize the number of pins.

8086Mp Signal Descriptions

The following signal descriptions are common for both Minimum and Maximum Modes:

AD15-AD0: (Bidirectional signals)

These are the time multiplexed memory I/O address and data lines. Address remains on the lines during T1 state, while the data is available on the data bus during T2, T3, Tw and T4.

- هي خطوط مشتركة بين الـ (data and address) ويتم استخدامها عند التعامل مع الذاكرة أو أجهزة الـ I/O.
- عدد هذه الخطوط هو 16 خط وهي bidirectional أي ذات اتجاهين (ادخال واخراج).
- في T1 تستخدم هذه الخطوط لاعطاء العنوان (address), بينما في (T2, T3, Tw and T4) تستخدم للـ data.

A16/S3

A17/S4

A18/S5

A19/S6

(Status lines are output signals)

Status lines (S3-S6) are multiplexed with address lines (A16-A19). During addressing they contain the address, while during data transmission they contain status information.

S3 and S4: When data is transmitted through data lines, S3 and S4 lines give the segment used to transfer data to/from the microprocessor, as in the following table:

S4	S3	Segment	Description
0	0	ES	Data transfer is to/from ES
0	1	SS	Data transfer is to/from SS
1	0	CS/ none*	Data transfer is to/from CS
1	1	DS	Data transfer is to/from DS

* if this data is not for memory (Ex: Input/Output), then S4 S3 = 10 (none).

NOTE: status lines (S0-S2) are used during maximum mode.

S5: this status line gives the condition of the IF bit as:

When IF = 0 → S5=0

When IF = 1 → S5=1

S6: this status line is always '0' logic, indicating that 8086 is controlling the system bus.

$\overline{\text{BHE}}/\text{S7}$ (Bus High Enable/S7)

$\overline{\text{BHE}}$ line is used to enable or disable transferring data over the high data lines (D15-D8), according to the following table:

$\overline{\text{BHE}}$	A0	Description
0	0	word
0	1	One byte is transferred using (D8-D15)
1	0	One byte is transferred using (D0-D7)
1	1	None

* S7 is always at '1' logic.

CLK: is an input signal used for supplying the Microprocessor with the clock signal.

2 GND: input signals that are connected to ground.

VCC: input signal that receives the supply voltage, which must be +5 V.

$\overline{\text{MN}}/\text{MX}$: input signal to specify the mode of operation:

1 → Minimum mode: in this mode, the Microprocessor works alone.

0 → Maximum mode: in this mode, the Microprocessor works with one or more other microprocessors.

Control Signals ($\overline{M/\overline{IO}}$, \overline{RD} , \overline{WR}): (output signals)

\overline{RD} : (READ pin) when this pin is at logic '0', it indicates that the Mp is performing memory or I/O read operation.

\overline{WR} : (WRITE pin) when this pin is at logic '0', it indicates that the Mp is performing memory or I/O write operation.

$\overline{M/\overline{IO}}$: this pin is used to specify if the read or write operation performed by the Mp is with the memory or the I/O device.

$\overline{M/\overline{IO}}$	\overline{RD}	\overline{WR}	Description
0	0	1	I/O read
0	1	0	I/O write
1	0	1	memory read
1	1	0	memory write

Hardware Interrupt Lines (\overline{INTA} , INTR and NMI): explained previously.

ALE: (Address Latch Enable) is an output signal used to inform the memory or I/O device when a valid address is on the address bus.

\overline{DEN} : (Data ENable) is an output signal used to inform the memory or I/O device when they should read/write data on data bus.

HOLD and HLDA: Direct Memory Access Signals (DMA).

External devices can request to take control of the system bus by making HOLD signal =1. When 8086Mp accepts this request, it makes signal HLDA=1 and enters idle state (or z-state).

\overline{TEST} : is instruction (wait) is executed, this input is tested as follows:

If $\overline{TEST} = 1 \rightarrow$ execution will continue.

This output is used to decide the direction of data flow through the transreceivers

If $\overline{TEST} = 0 \rightarrow$ Mp remains in idle state until \overline{TEST} becomes =1.

READY: if a device that is performing a read or write operation with the Mp was not ready for next data transition, READY signal will be used by this device to tell the Mp to wait (READY=0) by inserting additional clock cycles between T3 and T4 in read/write bus cycles, until READY input becomes =1, which indicates that the device is ready to transfer data.

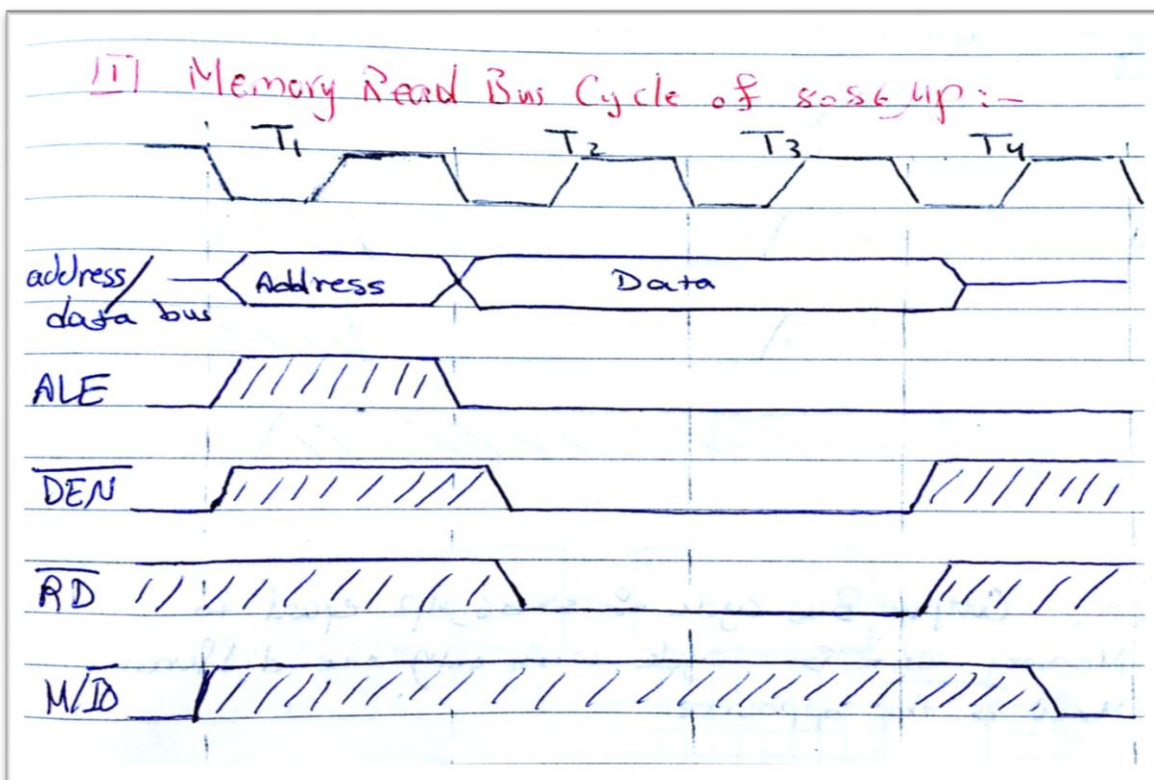
RESET: to reset the Mp. If RESET=1 → all registers will be initialized and reset service routine is executed.

DT/R: (Data Transmit/Receive) this output is used to decide the direction of data flow. '1' logic means data is transmitted from the microprocessor, '0' logic means data is received by the microprocessor.

8086Mp Bus Cycles

8086Mp uses memory and I/O devices in periods called (Bus cycles). Each bus cycle takes 4 system clocks (4T). 8086 Mp Bus Cycles are:

- 1- Memory Read Bus Cycle
- 2- Memory Write Bus Cycle
- 3- Input Bus Cycle
- 4- Output Bus Cycle



During T1: the address is put on the address/data bus, with ALE=1.

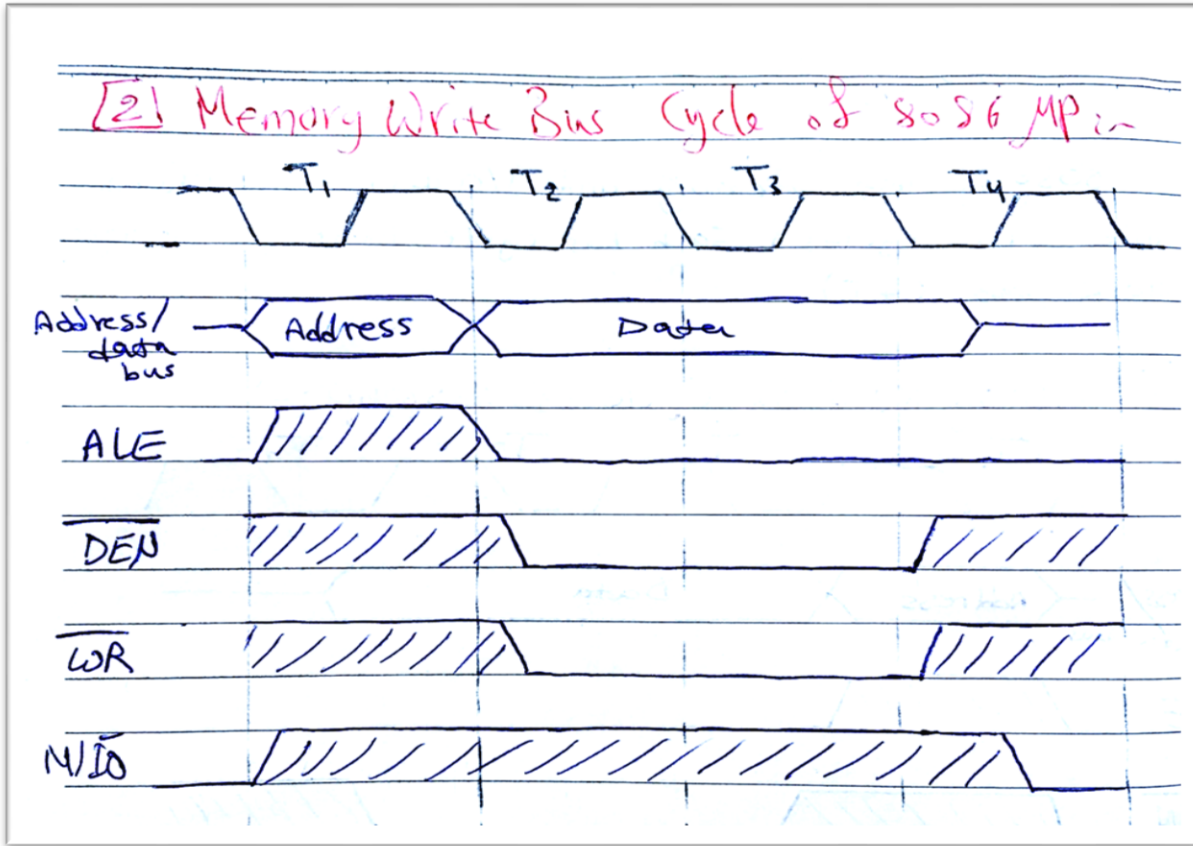
During T2: DEN becomes logic 0, to signal the memory or I/O device when to put data on to the bus, with RD=0.

During T3: 8086Mp reads data from address/data bus.

During T4: all bus signals are deactivated in preparation for the next clock cycle.

NOTE: Input bus cycle of 8086Mp is equal to Memory read bus cycle, with only one difference: M/I/O is equal to logic 0 during the four cycles (exactly the opposite).

ملاحظة مهمة: رسم الـ (Input bus cycle of 8086Mp) مساو للرسم اعلاه والخاص بالـ (Memory Read bus cycle of 8086Mp) عدا فرق واحد وهو أن الإشارة (M/I/O) تكون العكس مما هي عليه في الرسم أعلاه.

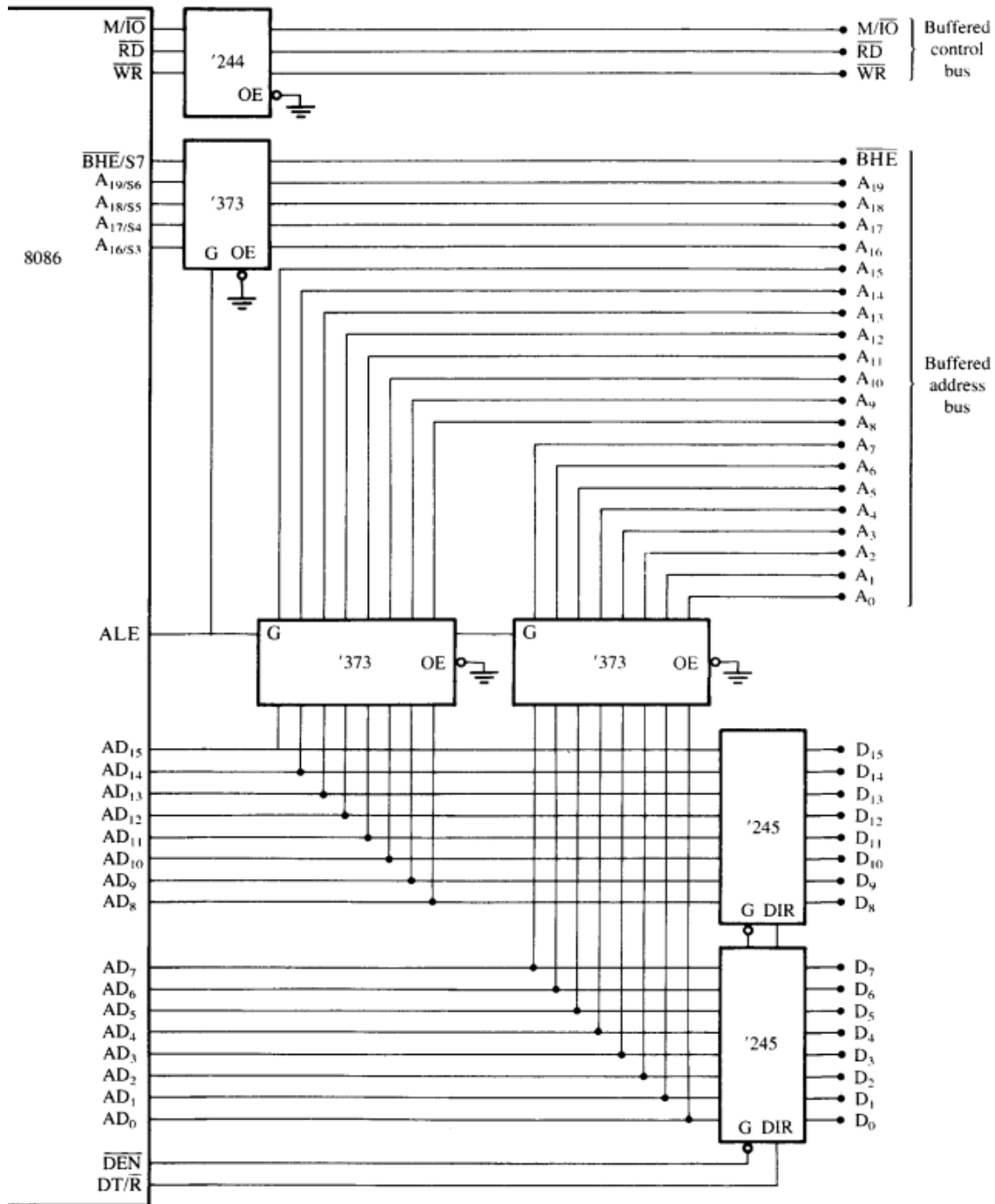


NOTE: Output bus cycle of 8086Mp is equal to Memory write bus cycle, with only one difference: M/I/O is equal to logic 0 during the four cycles (exactly the opposite).

ملاحظة مهمة: رسم الـ (Output bus cycle of 8086Mp) مساو للرسم اعلاه والخاص بالـ (Memory Write bus cycle of 8086Mp) عدا فرق واحد وهو أن الإشارة (M/I/O) تكون العكس مما هي عليه في الرسم أعلاه.

Best Regards
Dr. Zainab Alomari

Lecture 7: Demultiplexing, Buffering and Latching



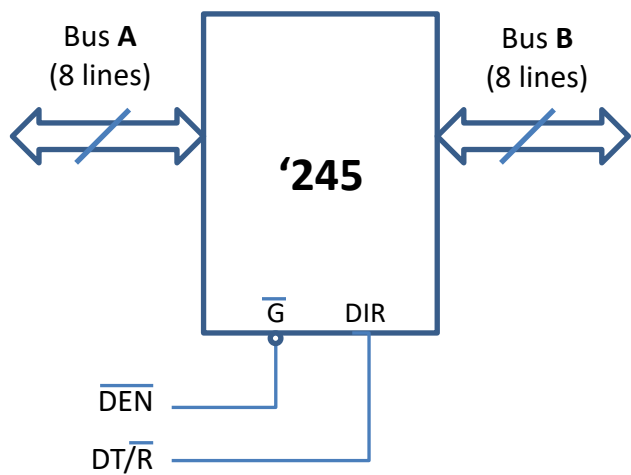
Fully Buffered 8086 Microprocessor

Demultiplexing is the operation of splitting multiplexed signals, like separating Address/Data lines into address lines and data lines.

- Data bus requires demultiplexing and buffering.
- Address bus requires demultiplexing, latching and buffering.
- Control bus requires only buffering.

74LS245

It is a bidirectional buffer used for demultiplexing and buffering data bus. It has 8-lines so that two buffer are required.

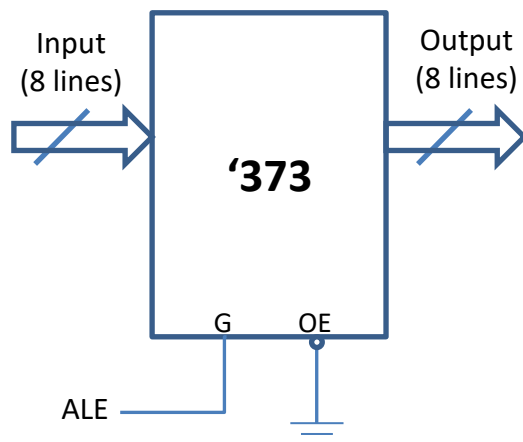


\overline{G}	DIR	Operation
0	0	Transfer data from bus B to bus A
0	1	Transfer data from bus A to bus B
1	X	Isolation

G = Gate (works as enable)
DIR = Direction of data transfer

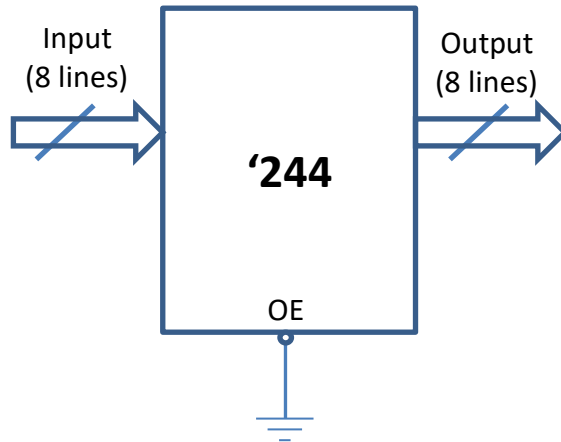
74LS373

It is a unidirectional buffer used for demultiplexing, buffering and latching address lines and \overline{BHE} . It has 8 inputs and 8 outputs, so that three buffers are required. (OE = Output Enable)



74LS244

It is a unidirectional buffer used for buffering the signals of the control bus, so that one buffer is required.



Best Regards
Dr. Zainab Alomari

Lecture 8: Memories (part 1)

Every Microprocessor-based system has a memory system. There are two main types of memories:

- Random Access Memory (RAM)
- Read Only Memory (ROM)

The size of the memory device is indicated as follows:

Memory device name (No. of Memory Locations X Bits per Location)

يتم تصنيع رقائق الذاكرة ليتم ربطها بالمعالجات واستخدامها ل تخزين وتحميل المعلومات والبرامج المختلفة. وتتوفر الكثير من رقائق الذاكرة في سوق العمل وبأحجام مختلفة. ويمكن معرفة حجم رقائق الذاكرة من خلال الأرقام التي تكون مكتوبة عليها بالطريقة أعلاه، حيث يتم إعطاء عدد المواقع مضروبة في حجم الموقع الواحد بين القوسين، أما الرقم الذي يسبق القوس فهو رقم مميز لكل نوع من الرقائق وبمطابقة أسم لها.

Example (1)

7216 (2K x 8) memory

This memory device contains 2K locations, 11 address lines and 8 data outputs.

- يتم معرفة عدد خطوط العنونة من خلال عدد المواقع (عدد خطوط الـ address يساوي N اذا كان عدد المواقع هو 2^N).
- يتم معرفة عدد خطوط الـ data من عدد البتات لكل موقع.

Example (2)

62256 (32K x 8) memory

This memory device contains 32K locations, 15 address lines and 8 data outputs.

Note: If only one number is mentioned for a memory device, this number represents the total number of memory bits and it is called (**bit capacity**).

الـ bit capacity يمثل حاصل ضرب الرقمين الموجودين في داخل الأقواس وهي عدد البتات الكلي. فبالتالي من معرفة عدد البتات الكلي (bit capacity) وعدد المواقع يمكن معرفة عدد البتات في الموقع الواحد، أو: من معرفة عدد البتات في الموقع الواحد والـ bit capacity يمكننا معرفة عدد المواقع الكلي.

Example (3)

(1K x 8) memory is listed as an 8K memory device, while (64K x 4) memory is listed as a 256K memory device.

Types of Memory Devices

There are two main memory devices:

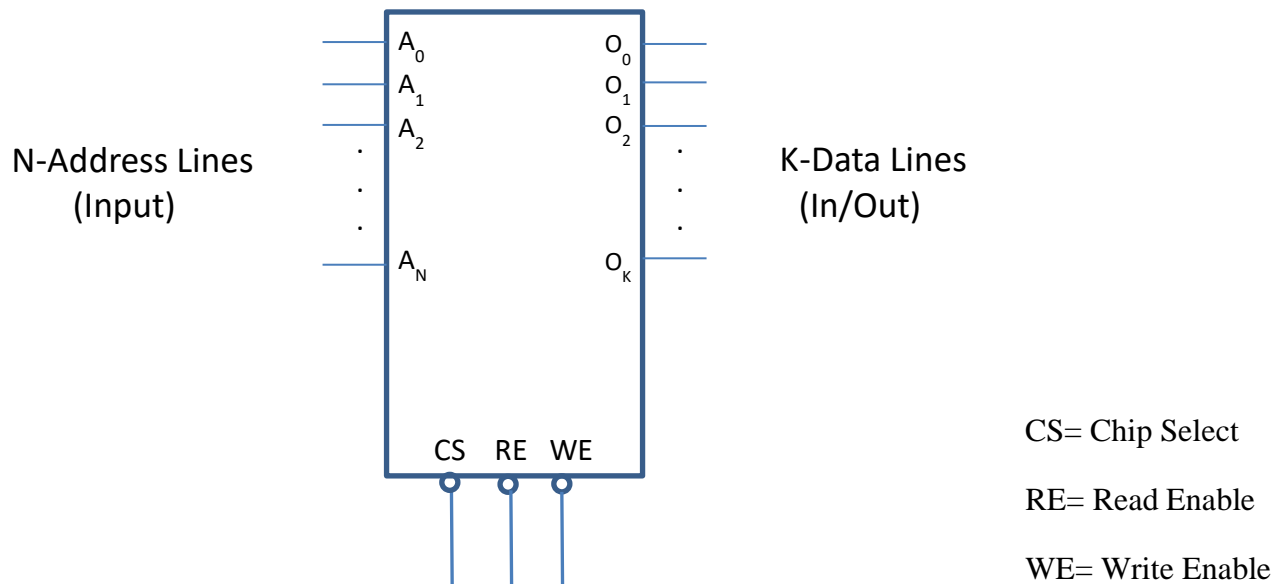
- Random Access Memory (RAM)
- Read Only Memory (ROM)

The main differences between these types are:

- 1- A RAM is written under normal operation. A ROM can be programmed, but normally it is only read.
- 2- RAMs are used to store temporary data, while ROMs are used to store permanent data.

General block diagram of RAM memory devices

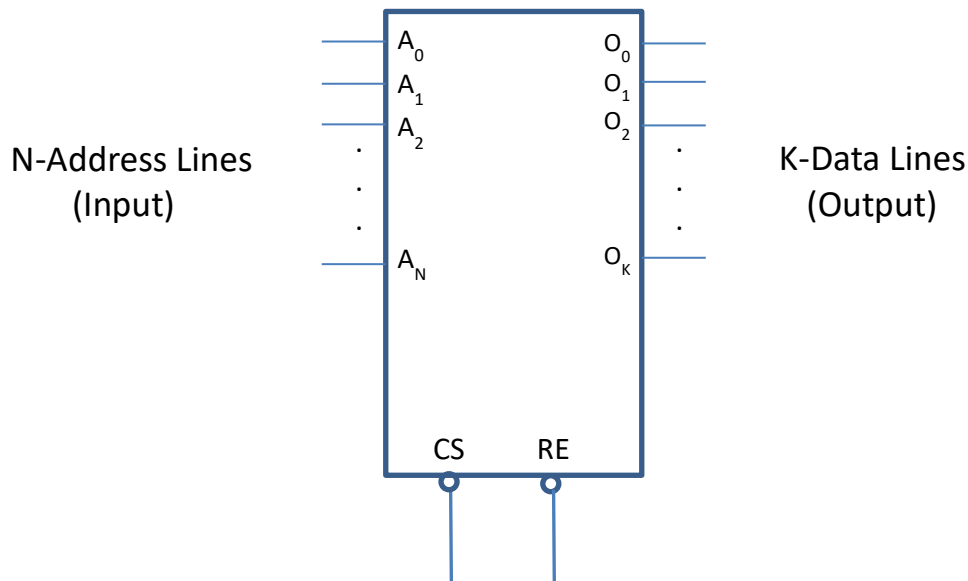
الشكل التالي يمثل block diagram الذي يمثل أي رقاقة RAM مهما كان حجمها, حيث تحتاج دائماً الى ادخال يمثل العنوان وخطوط الـ Input/Output التي تمثل خطوط الـ data. وتعمل الرقاقة عملية كتابة أو قراءة فقط في حالة أن ادخال الـ CS مفعّل (أي 0 logic). أما العملية هل هي قراءة أم كتابة للموقع المحدد عنوانه بخطوط العنوان فيتم معرفته من خلال الخطتين RE و WE أيهما مفعّل (أي أيهما 0 logic).



General block diagram of RAM memory devices

General block diagram of ROM memory devices

الفرق الرئيسي بين النوعين والملاحظ من الرسم أن خطوط الـ data في الـ ROM هي باتجاه واحد (output) كما أنه لا يوجد خط للكتابة (WE) وإنما للقراءة فقط (RE).



General block diagram of ROM memory devices

8086Mp Memory Organization and Interfacing

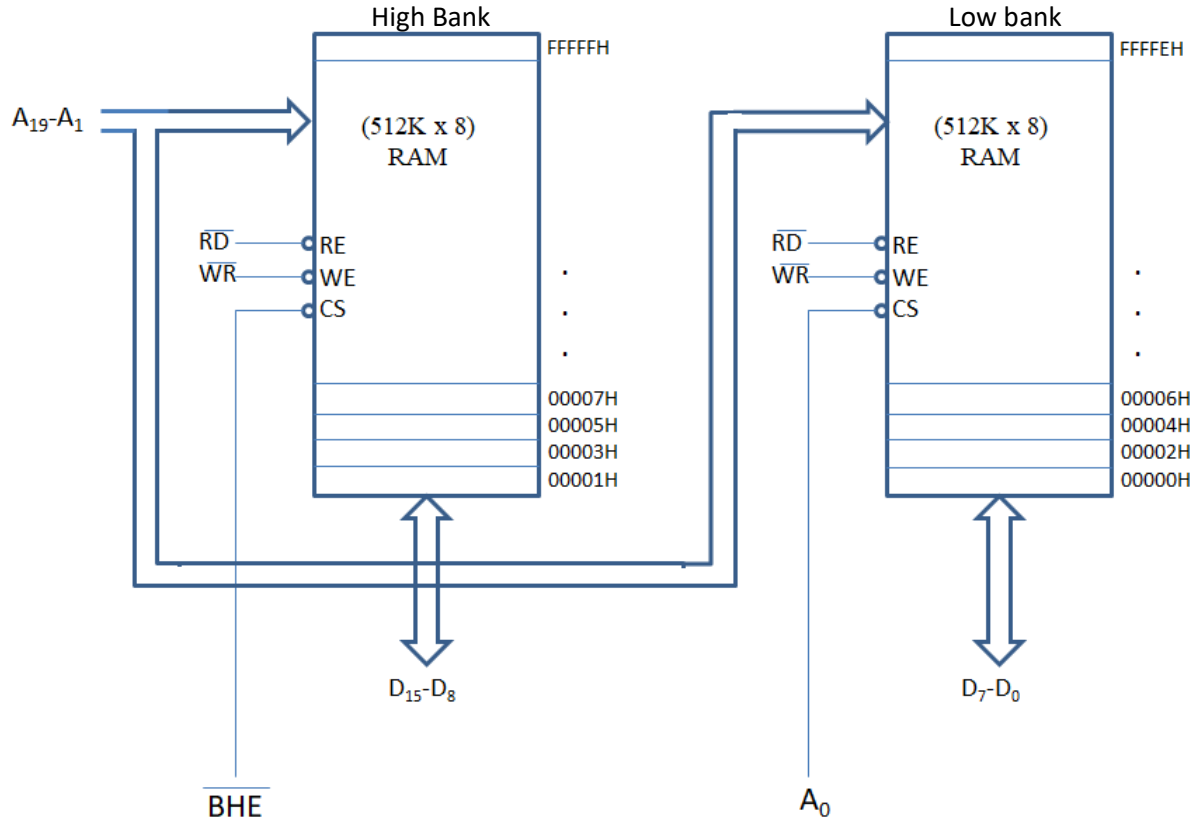
8086Mp 1MByte memory is implemented using 2 independent 512Kbyte banks:

- Low Bank (Even Bank)
- High Bank (Odd Bank)

نأتي الى تصميم الذاكرة الخاصة بالـ 8086Mp, ونلاحظ بأنها تم بناؤها باستخدام بلوكين حجم كل منهما 512Kbyte:

الأول: يحتوي المواقع ذات العناوين الزوجية ويسمى (Low bank) أو (Even bank) ويبدأ بالموقع 00000H وينتهي بالموقع FFFFEH .

والثاني يحتوي المواقع ذات العناوين الفردية ويسمى (High bank) أو (Odd bank) ويبدأ بالموقع 00001H وينتهي بالموقع FFFFFH .



High and Low Banks of 8086Mp Memory

ملاحظات مهمة جداً عن الرسم أعلاه:

بما أن حجم كل بلوك هو 512Kbyte فبالتالي عدد خطوط العنوان لكل من البلوكين هو 19 خط. هذه الخطوط تربط على الخطوط الخاصة بالaddress من الـ 8086Mp والتي هي من A_1 الى A_{19} أما A_0 فلأنه يحدد هل العنوان فردي أم زوجي فهو يربط على إشارة الـ CS للـ Low bank فاذا كان العنوان القادم من المعالج زوجي فإن A_0 سيكون يساوي 0 وبالتالي سيتم تفعيل إشارة CS لهذا البنك.

أما اذا كان العنوان المطلوب فردي فهناك إشارة يفعلها المعالج للإشارة الى كون العنوان فردي وهي \overline{BHE} . نلاحظ أنها تصبح تساوي 0 اذا كان العنوان فردي ولهذا يتم ربطها بالـ CS الخاص بالـ High bank.

كل هذا الكلام هو في حال كون الداتا المطلوب قرائتها أو كتابتها في الذاكرة هي بايت واحد فقط، كما في الأمثلة التالية:

- Mov AL, [200H]
- Mov [201H], AL

$\overline{\text{BHE}}$	A_0	Memory operation
0	0	Word (Both banks are enabled)
0	1	High byte (Only high bank is enabled)
1	0	Low byte (Only low bank is enabled)
1	1	No operation (Both banks are disabled)

Note:

- If address is even (ex: MOV AX,[300H]), then both of $\overline{\text{BHE}}$ and A_0 are logic 0, and the data transfer from low bank takes 1 bus cycle (4 clocks).
- If address is odd (ex: MOV AX, [301H]), then the first byte (lower byte) in high bank takes 1 bus cycle (4 clocks), where ($\overline{\text{BHE}}=0$, $A_0=1$).and then the second byte (high byte) in low bank takes 1 other bus cycle (4 clocks), where ($\overline{\text{BHE}}=1$, $A_0=0$).

أما في حالة ان الـ data المطلوب قرائتها أو كتابتها في الذاكرة هي word أي 16 بت, فبالتالي سنحتاج الى جلب البايتين من كلا البلوكين, لان كل موقع من الذاكرة كما هو معروف يحتوي على بايت واحد فقط. فكيف ستم العملية؟

أولاً: في حالة كون العنوان زوجي (مثلاً: MOV AX,[300H]) فسيتم تفعيل كل من A_0 و $\overline{\text{BHE}}$ (كلاهما سيصبح =0) وبالتالي سيتم أخذ أو اعطاء بايتين لموقعين من الذاكرة, الاول في البلوك الزوجي في العنوان 300H والثاني في البلوك الفردي في العنوان 301H.

ثانياً: في حالة كون العنوان فردي (مثلاً: MOV AX,[301H]) فسيتم تفعيل اشارة $\overline{\text{BHE}}$ وأخذ أو اعطاء قيمة الباييت الأول (lower byte) من الـ high bank وذلك يستغرق دورة كاملة للقراءة أو الكتابة أي (4 clock cycles) ثم بعدها يتم تفعيل اشارة A_0 بحيث يتم أخذ أو اعطاء الباييت التالي من الـ low bank وهذا سيستغرق دورة ثانية كاملة أي (4 other clock cycles).

إذاً: التعامل مع الذاكرة يستغرق bus cycle واحدة أي (4 clock cycles) سواءً في القراءة أو الكتابة ما عدا في حالة واحدة وهي أن المطلوب هو قراءة أو كتابة 16 بت والعنوان فردي, ففي هذه الحالة ستستغرق العملية (2bus cycles) أي (8 clock cycles).

Best Regards
Dr. Zainab Alomari

Lecture 9: Memory Devices (Part 2)

Address Decoding

1) Simple NAND gate Decoding

Example

Design the hardware required to implement (128K x 8) RAM starting at address 20000H, using (64K x 8) RAMs.

المطلوب في هذا السؤال تصميم وربط ذاكرة بحجم معين الى المعالج, باستخدام بلوكات RAM اقل منه حجماً, فنحتاج معرفة الحجم للذاكرة ومعرفة عنوان البداية لها والذي سيؤثر على طريقة الربط.

Solution

$$128K/64K = 2$$

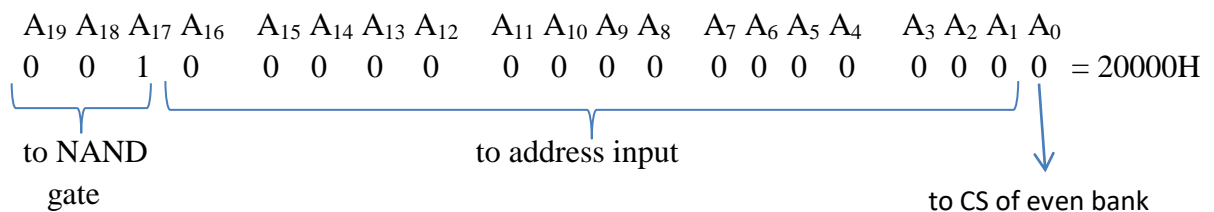
$$\therefore \text{No. of } (64K \times 8) \text{ RAMs} = 2$$

$$\text{No. of RAMs in each bank} = 1$$

$$\text{Total no. of address lines (for the 128Kbyte memory)} = 17$$

$$\text{No. of address lines for each } (64K \times 8) \text{ RAM} = 16$$

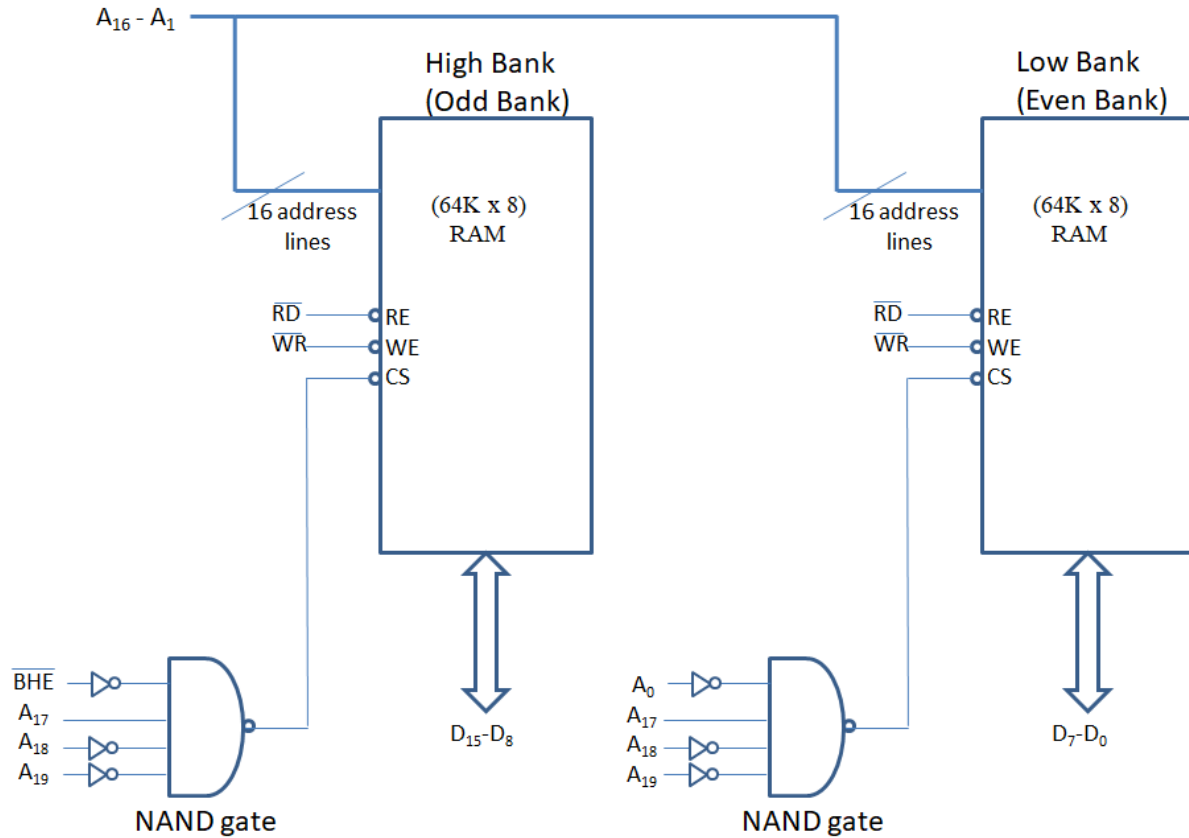
ملاحظة: دائماً يتم فصل البلوكات الى قسمين أحدهما even bank والثاني odd bank. بمعنى لو مثلاً احتجنا اربع بلوكات لبناء الذاكرة المطلوبة لكان كل بانك يحتوي على بلوكين , وهكذا. والان من حجم البلوك الواحد سنعرف عدد الخطوط التي نحتاج ربطها الى العنوان, وبما أن حجم كل بلوك هنا في هذا السؤال هو 64K فبالتالي نحتاج الى 16 خط لكل بلوك, فنقوم باستخدام خطوط العنوان بدءاً من A1 وبالتالي تنتهي بـA16, حيث أن A0 يتم ربطه على اشارة الـCS الخاصة بالـeven bank. تبقى الخطوط القادمة من المعالج 8086 وهي A17 و A18 و A19. هذه الخطوط المتبقية يتم ربطها على بوابة NAND.



(20000H is the starting address of the designed memory).

- 16 bits of this address (from A₁ to A₁₆) are used as the address input of the two RAMs.
- A₀ is connected to the CS of the low (even) bank with (A₁₉ A₁₈ A₁₇) using **NAND** gate.
- BHE is connected to the CS of the high (odd) bank with (A₁₉ A₁₈ A₁₇) using **NAND** gate.

نلاحظ أن خطوط العنوان (A_{19} A_{18} A_{17}) تربط حسب العنوان المطلوب اعطائه للذاكرة حيث أننا نضع بوابة القلب على كل من A_{18} و A_{19} فبالتالي اذا لم تكن هذه الاشارات الثلاثة 001 فمعناه أن العنوان الموجود على خطوط العنوان القادمة من المعالج لا تخص هذه الذاكرة اي ليست ضمن الرينج الخاص بها , فلا يتم تفعيل اشارات ال-CS.



NAND Output	\overline{CS} state
1	\overline{CS} is not active
0	\overline{CS} is active

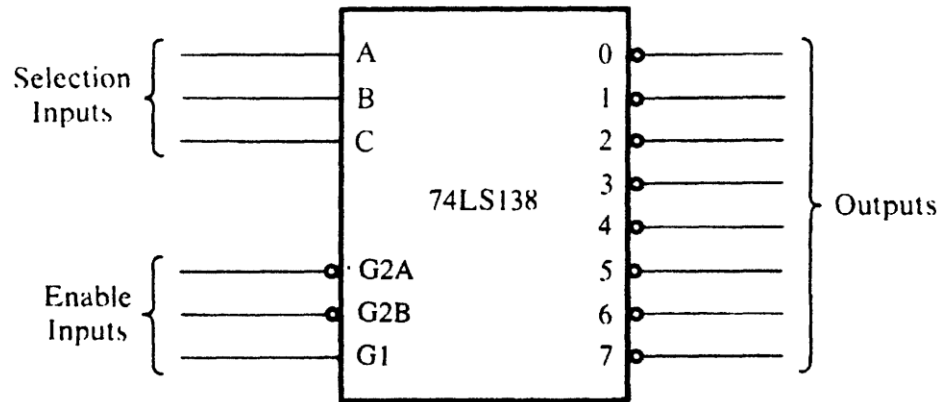
Q What is the ending address of the designed memory in the above example?

Answer: we fill the bits of the address from A_0 to A_{16} with ones and this is the ending address, (= 0011 1111 1111 1111 1111 = 3FFFFH)

2) 74LS138 Decoder

When more than one memory device is used in each bank, a decoder is needed to select the correct memory device from each bank.

74LS138 decoder has 3 inputs, 8 outputs and 3 enable inputs. When the enable inputs are all active, one output is activated according to the input. The work of this decoder is summarized in the following truth tables.



Inputs			Outputs												
Enable			Select												
$\overline{G2A}$	$\overline{G2B}$	G1	C	B	A	$\overline{0}$	$\overline{1}$	$\overline{2}$	$\overline{3}$	$\overline{4}$	$\overline{5}$	$\overline{6}$	$\overline{7}$		
1	X	X	X	X	X	1	1	1	1	1	1	1	1		
X	1	X	X	X	X	1	1	1	1	1	1	1	1		
X	X	0	X	X	X	1	1	1	1	1	1	1	1		
0	0	1	0	0	0	0	1	1	1	1	1	1	1		
0	0	1	0	0	1	1	0	1	1	1	1	1	1		
0	0	1	0	1	0	1	1	0	1	1	1	1	1		
0	0	1	0	1	1	1	1	1	0	1	1	1	1		
0	0	1	1	0	0	1	1	1	1	0	1	1	1		
0	0	1	1	0	1	1	1	1	1	1	0	1	1		
0	0	1	1	1	0	1	1	1	1	1	1	0	1		
0	0	1	1	1	1	1	1	1	1	1	1	1	0		

The Block Diagram and Truth Table of 74LS138 Decoder

Example

Design the hardware required to implement a (64K x 8) RAM start at address A0000H, using (16K x 8) RAMs.

Solution

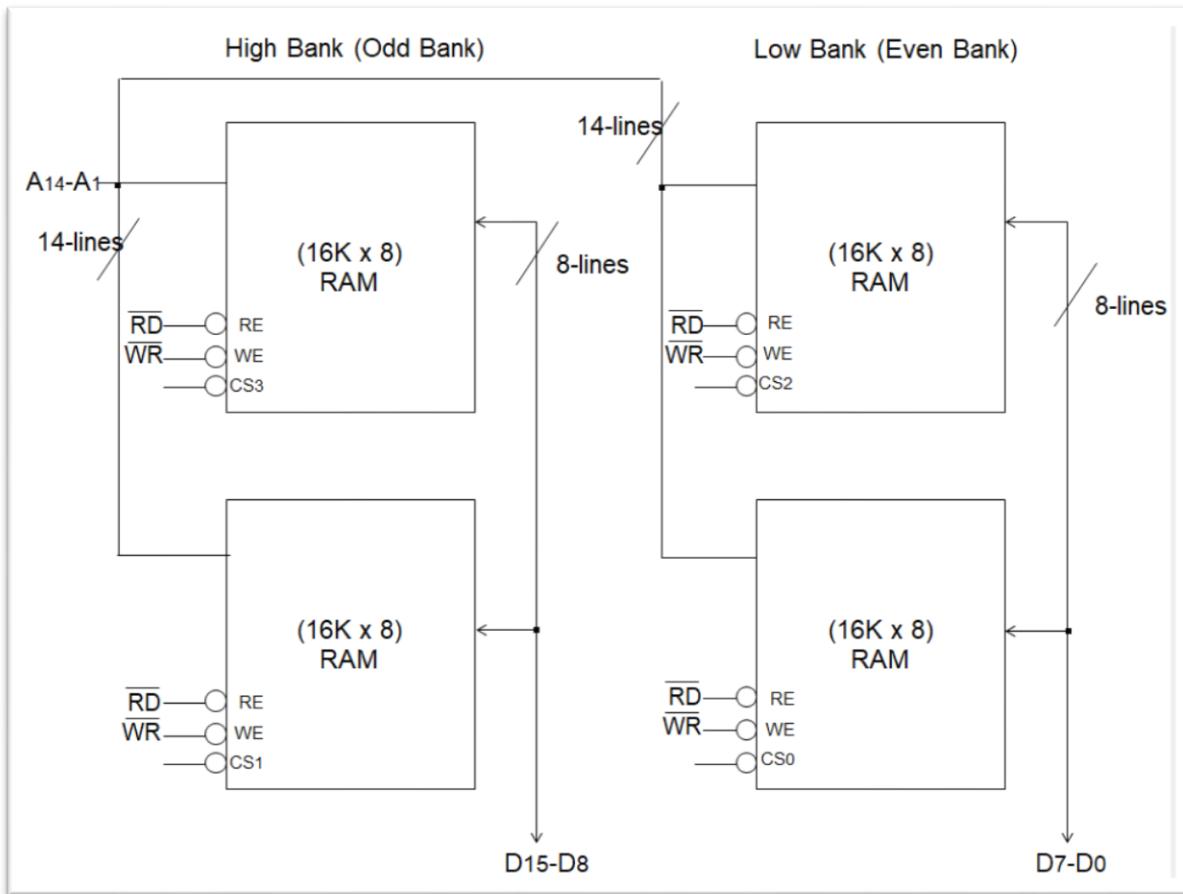
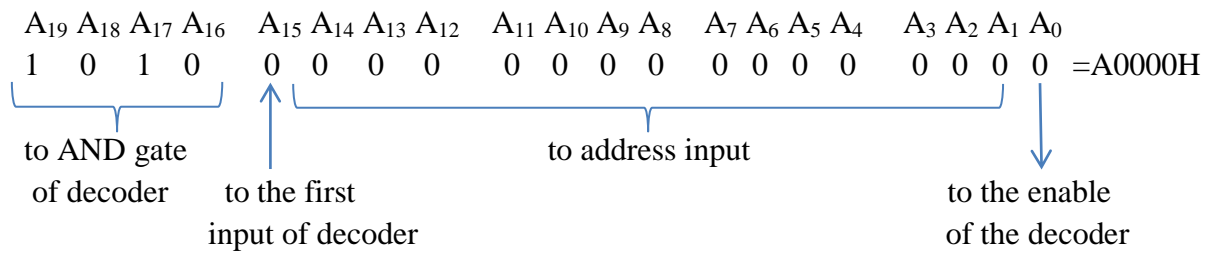
$$64K/16K = 4$$

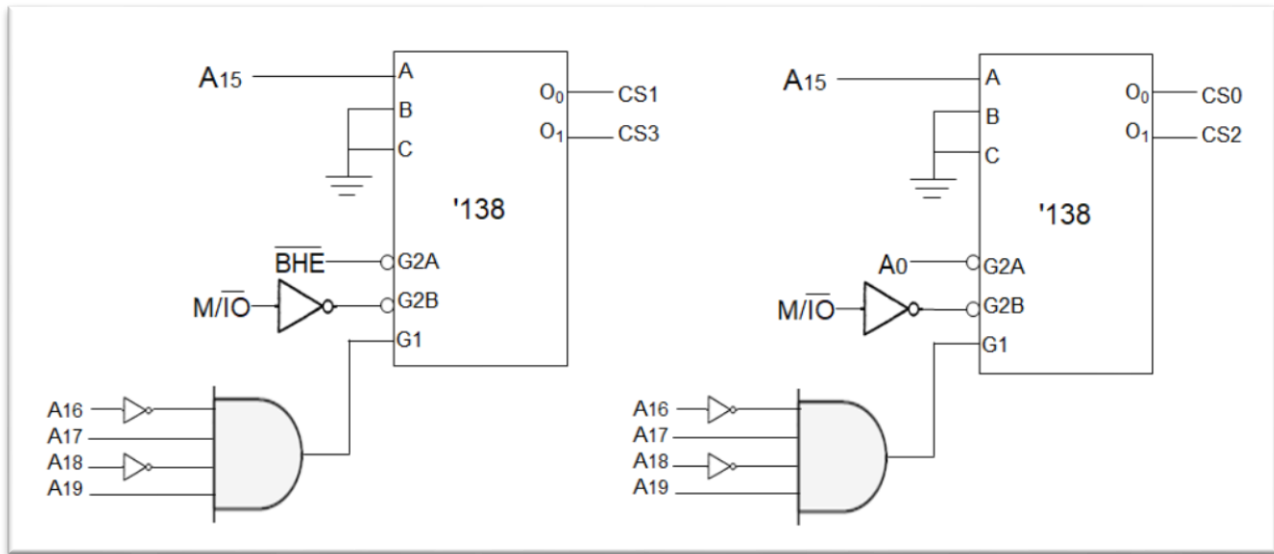
∴ No. of (64K x 8) RAMs = 4

No. of RAMs in each bank = 2

Total no. of address lines (for the 64K byte memory) = 16

No. of address lines for each (16K x 8) RAM = 14





Example

Design the hardware required to implement a (128K x 8) RAM start at address 00000H, using (32K x 8) RAMs.

Solution

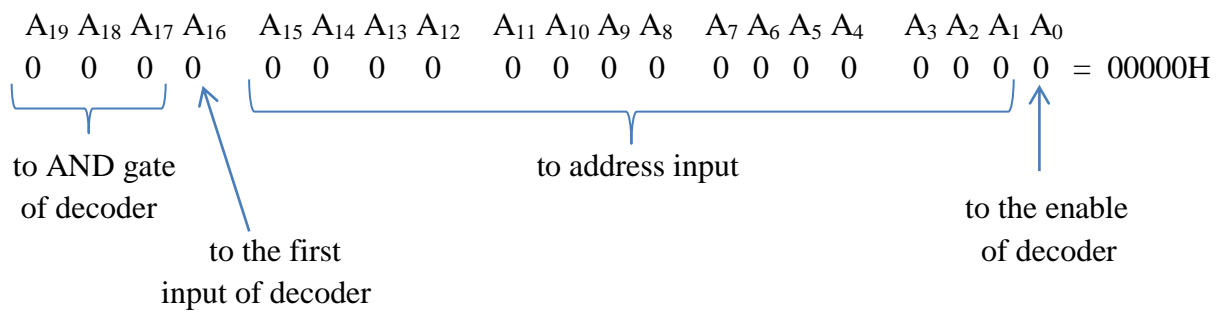
$$128K/32K = 4$$

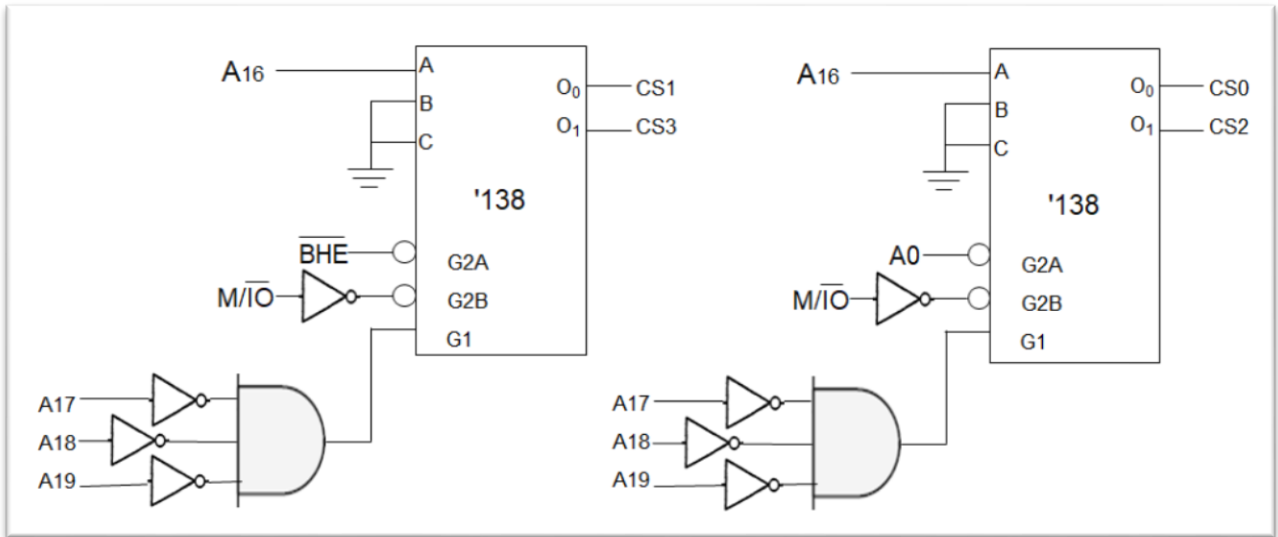
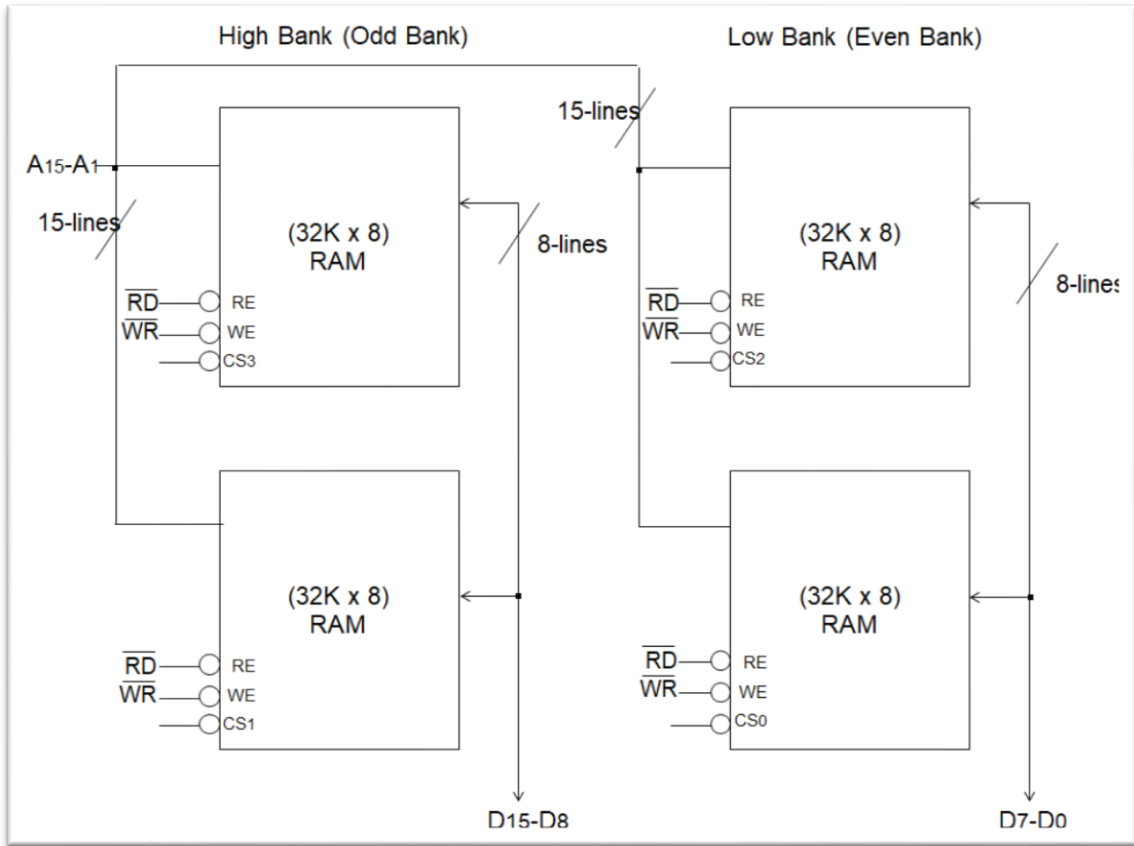
$$\therefore \text{No. of (32K x 8) RAMs} = 4$$

$$\text{No. of RAMs in each bank} = 2$$

$$\text{Total no. of address lines (for the 128K byte memory)} = 17$$

$$\text{No. of address lines for each (32K x 8) RAM} = 15$$





Best Regards
 Dr. Zainab Rami Alomari

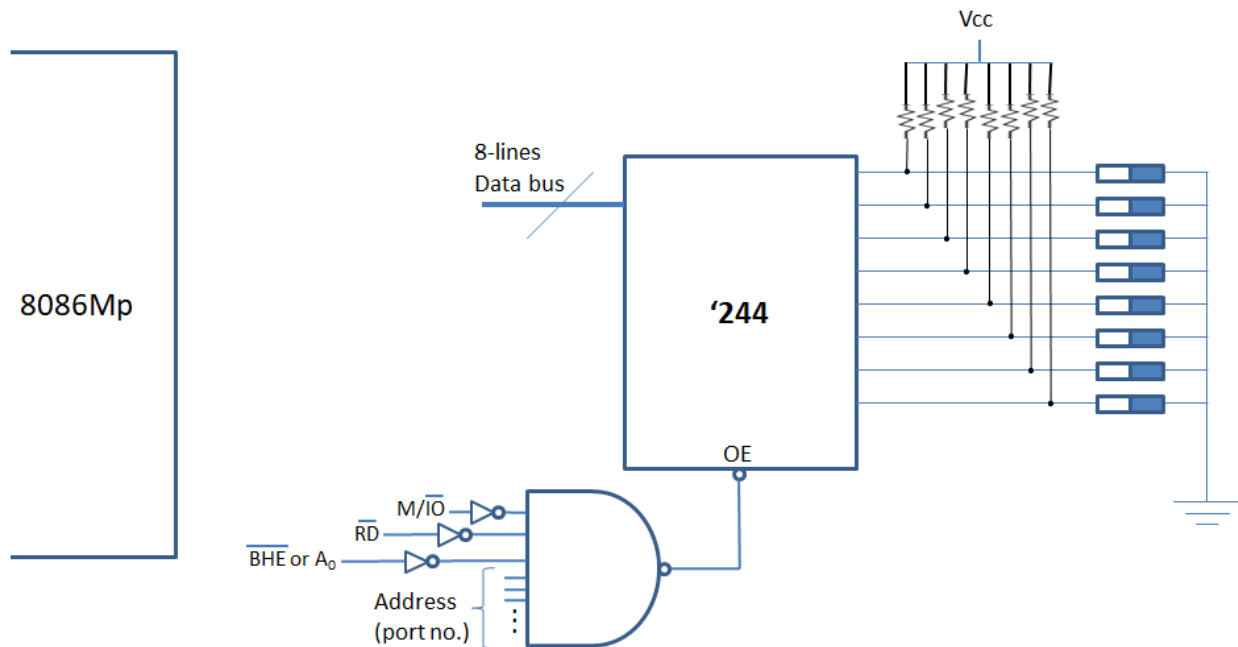
Lecture 10: Basic Input/Output Device Interfacing (Part 1)

1) Basic Input Devices Interfacing

74LS244 Buffer is used to interface input devices to the 8086Mp.

Example

The hardware required to connect 8 switches to 8086Mp is as follows:



NOTE1: I/O address (port no.) can be 8-bits or 16-bits. However, all 16 address lines (A0-A15) of the 8086Mp are used to enable I/O devices.

If address is 8-bits, the Mp puts zeros on (A8-A15). This is to differentiate between (for example): port no. 1F3AH (16 bits) and 3AH (8 bits).

Therefore, NAND gate has 19 inputs ($\overline{M/I0}$, \overline{RD} , \overline{BHE} or A_0 + 16 address lines).

NOTE2: I/O device data can be 8-bits or 16-bits.

1) **If data is 16 bits:** address is always even.

Low byte is transferred over (D0-D7) and high byte is transferred over (D8-D15).

2) **If data is 8 bits:** address can be:

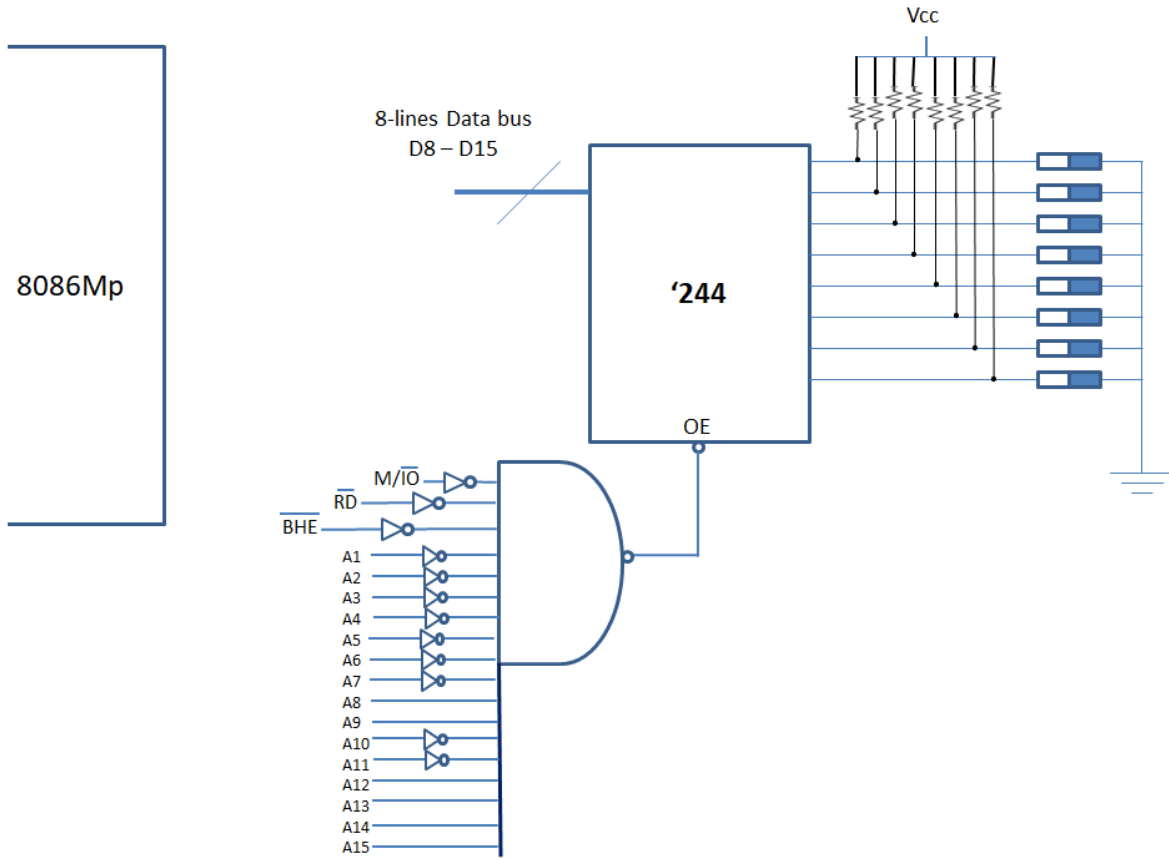
- Even: data is transferred over D0-D7.
- Odd: data is transferred over D8-D15.

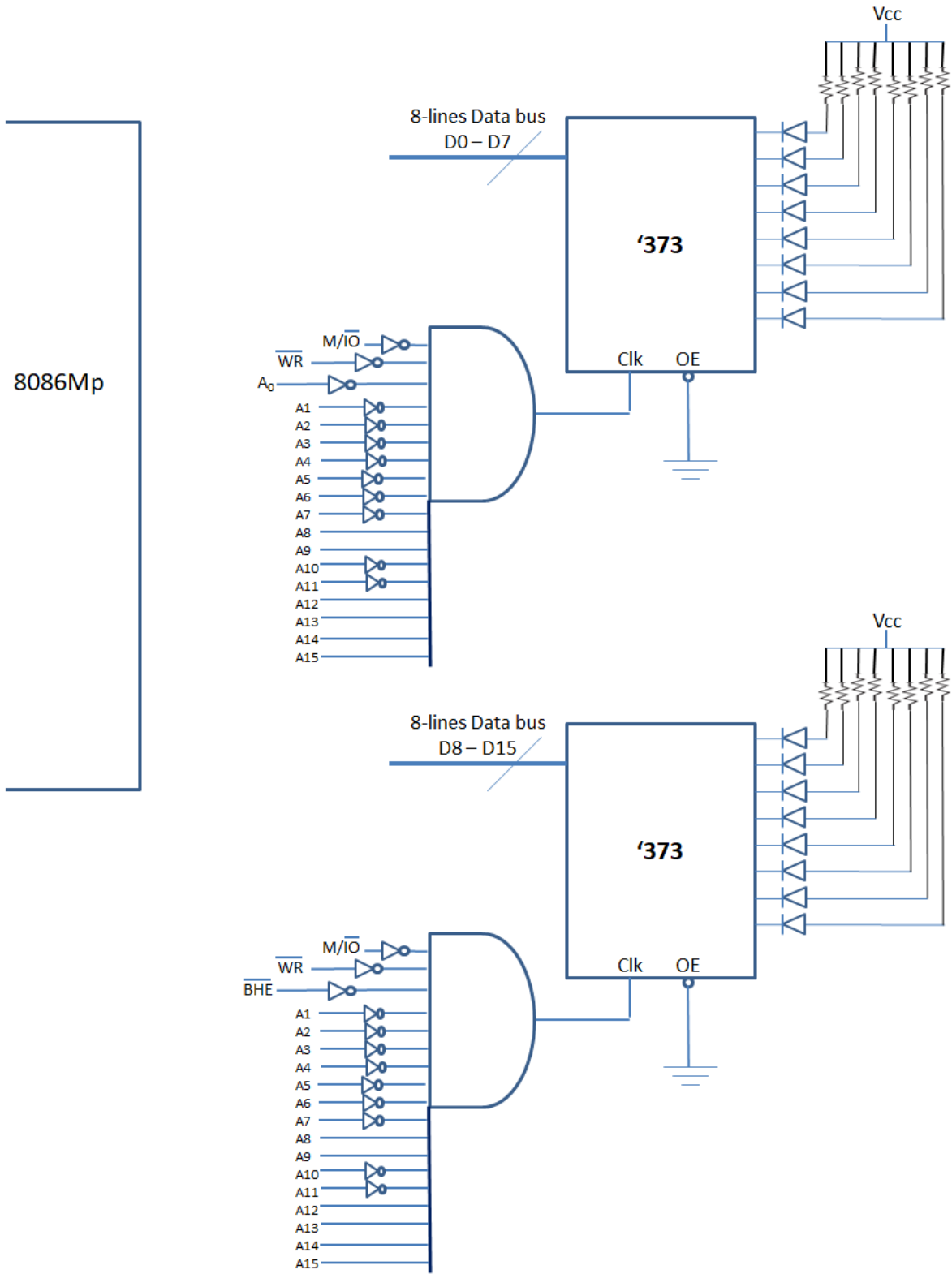
Example

Design the hardware required to interface 8 switches to the 8086Mp at port number F301H.

Solution

$$\begin{array}{cccccccccccccccc} A_{15} & A_{14} & A_{13} & A_{12} & A_{11} & A_{10} & A_9 & A_8 & A_7 & A_6 & A_5 & A_4 & A_3 & A_2 & A_1 & A_0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array} = \text{F301H}$$





Example

Write a code in Assembly language to control the LEDs in the previous example so that the first LED turns on for a specific delay, then the second LED is turned on for same delay and so on.

Solution

```
MOV DX, 0F300H
MOV AX, 0FFFEH
RPT: OUT DX, AX
CALL DELAY1
ROL AX, 1
JMP RPT
HLT
```

Best Regards
Dr. Zainab Alomari

Lecture 11: Basic Input/Output Device Interfacing (Part 2)

2) Basic Output Devices Interfacing

Single Pole Double Through Relay (SPDT)

SPDT is an electrically operated switch, used to control a circuit by a low power signal.

عند استخدام اي جهاز اخراج (Output Device) فيجب استخدام الـ SPDT عند الربط (ما عدا اذا كان الجهاز LEDs فلا حاجة لذلك), حيث تعمل كسويچ يقوم بتوصيل الـ Vcc الى الـ (Output Device) اذا كانت الاشارة القادمة من المعالج = 1 وتقطع التوصيل اذا كانت الاشارة القادمة من المعالج = 0.

والسبب: أن الاشارة القادمة من المعالج تكون بفولتية واطنة (Logic = 5Volt '1') في حين تحتاج الاجهزة الكهربائية الى فولتية تعادل (220Volt).

Example

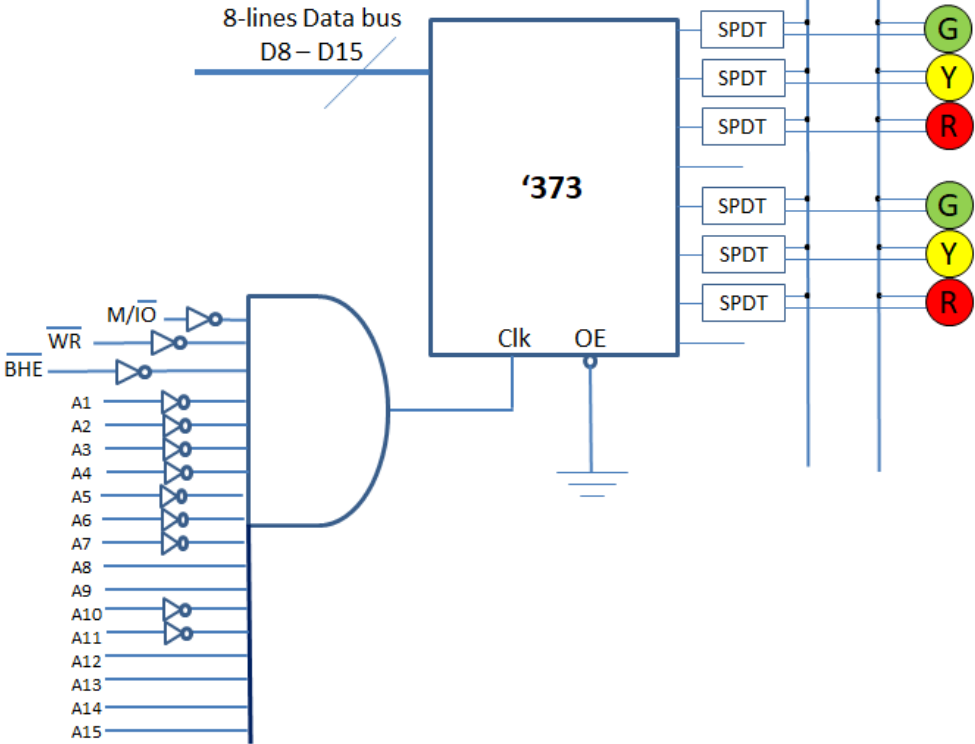
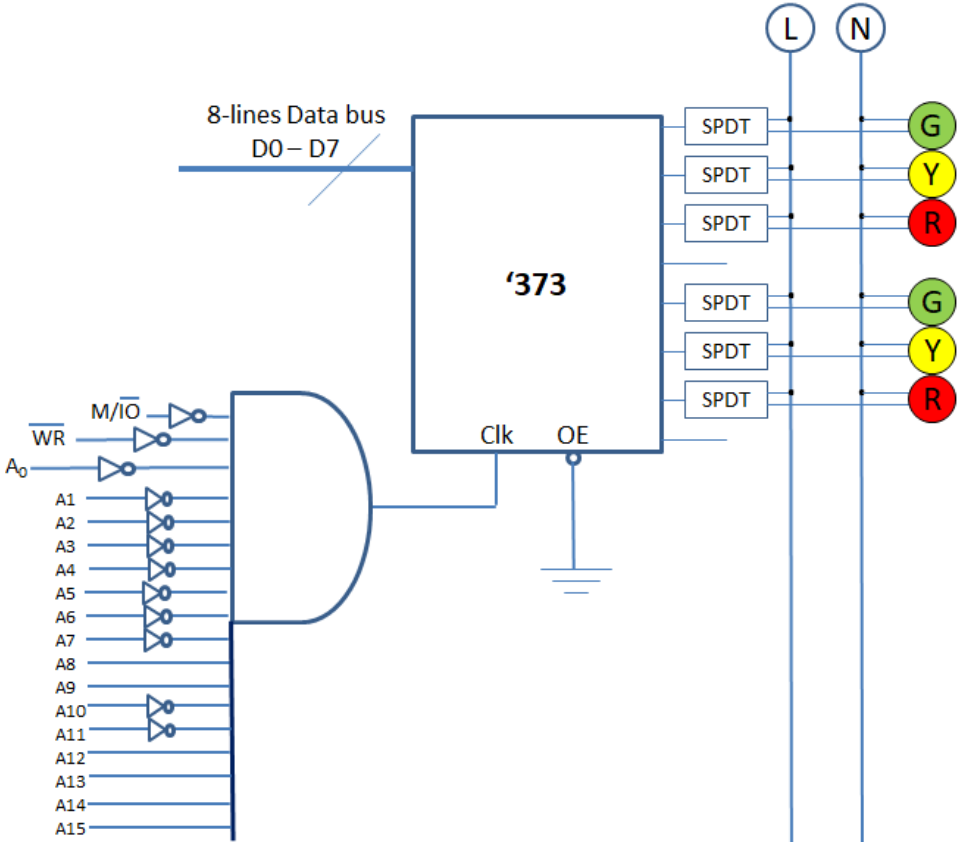
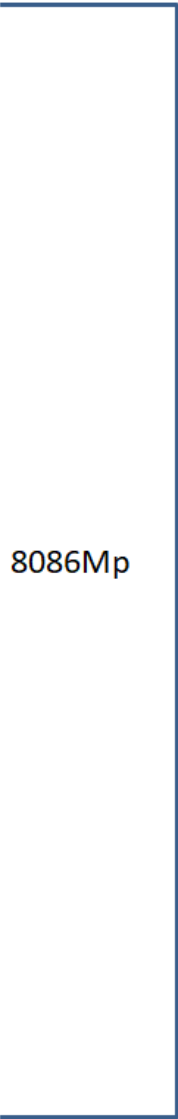
Design the hardware required to control a traffic light system (at port no. F300H) of four-road intersection. Write a code in Assembly language to control it as follows:

Green time = 2 minutes

Yellow time = 20 seconds

Solution

	X	R	Y	G	X	R	Y	G	X	R	Y	G	X	R	Y	G	
4441H=	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	1	2min
4442H=	0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	0	20sec
4414H=	0	1	0	0	0	1	0	0	0	0	0	1	0	1	0	0	2min
4424H=	0	1	0	0	0	1	0	0	0	0	1	0	0	1	0	0	20sec
4144H=	0	1	0	0	0	0	0	1	0	1	0	0	0	1	0	0	2min
4244H=	0	1	0	0	0	0	1	0	0	1	0	0	0	1	0	0	20sec
1444H=	0	0	0	1	0	1	0	0	0	1	0	0	0	1	0	0	2min
2444H=	0	0	1	0	0	1	0	0	0	1	0	0	0	1	0	0	20sec



Software:

```
MOV DX, 0F300H
RPT: MOV BX, 0
MOV CX, 4
NXT: MOV AX, Array[BX]
OUT DX, AX
CALL DELAY1 ; 20 min delay
MOV AX, Array[BX+1]
OUT DX, AX
CALL DELAY2 ; 20 sec delay
ADD BX, 2
LOOP NXT
JMP RPT
HLT
RET
Array DW 4441h, 4442h, 4414h, 4424h, 4144h, 4244h, 1444h, 2444h
END
```

Solution 2 (يمكن كتابة الكود بطريقة ثانية)

```
MOV DX, 0F300H
RPT: MOV CX, 4
LEA BX, Array
NXT: MOV AX, [BX]
OUT DX, AX
CALL DELAY1
MOV AX, [BX+2]
OUT DX, AX
CALL DELAY2
ADD BX, 4
LOOP NXT
JMP RPT
HLT
RET
Array DW 4441h, 4442h, 4414h, 4424h, 4144h, 4244h, 1444h, 2444h
END
```

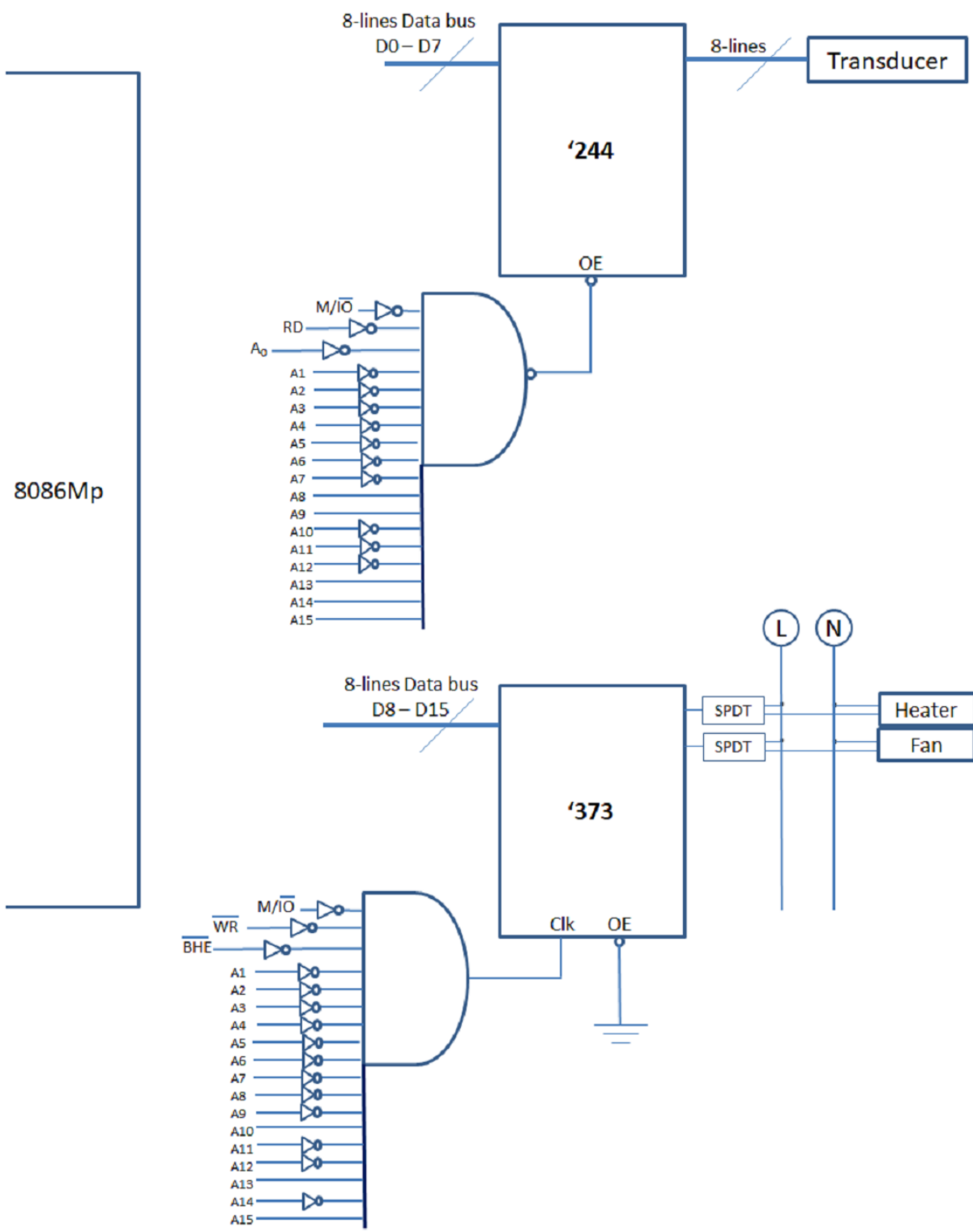
Example

Design the hardware and software required to interface a transducer, fan and heater to 8086Mp. The Mp reads the temperature from the transducer (8-bit signed data), and according to this temperature, it derives the fan and heater to keep temperature between 18°C and 25°C.

Connect the transducer to port no. E300H and connect the fan and heater to port no. A401H.

FAN	ON	
HEATER	OFF	
-----		25°C
FAN	OFF	
HEATER	OFF	
-----		18°C
FAN	OFF	
HEATER	ON	

Solution



Software:

```
START: MOV DX, E300H
      IN AL, DX
      CMP AL, 18
      JG N1
      MOV AL, 01      ; Heater is ON, Fan is OFF
      MOV DX, 0A401H
      OUT DX, AL
      JMP START
N1:   CMP AL, 25
      JG N2
      MOV AL, 00      ; Heater is OFF, Fan is OFF
      MOV DX, 0A401H
      OUT DX, AL
      JMP START
N2:   MOV AL, 02      ; Heater is OFF, Fan is ON
      MOV DX, A401H
      OUT DX, AL
      JMP START
      HLT
RET
```

Best Regards
Dr. Zainab Alomari