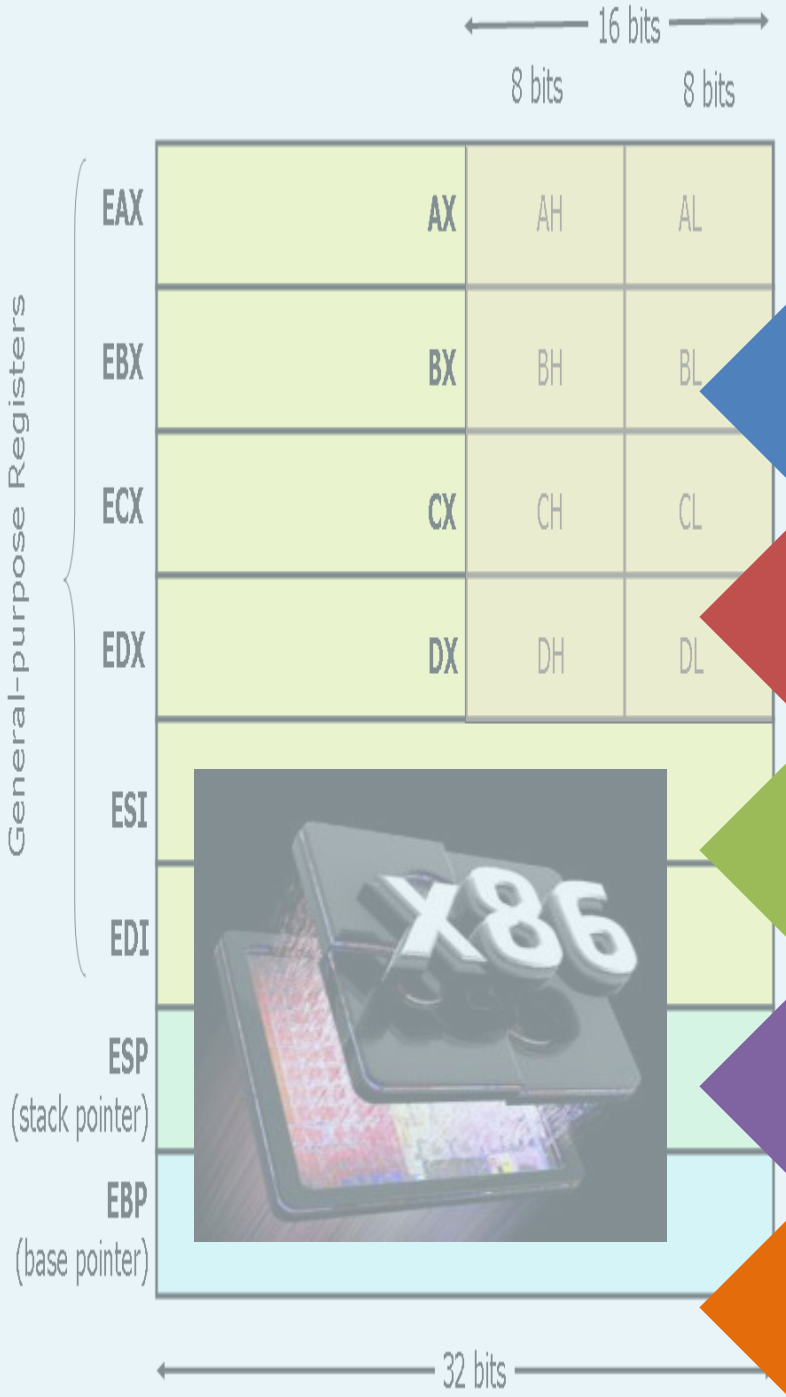
Computer Architecture and IOT

Chapter2

Microprocessor X86 និងភាសាAssembly



You will Learn

1. 8086 Microprocessor
2. Register
3. Instruction of 8086
4. X86Asm Edition
5. Assembly Programming

Introduction

- X86 តំណាងអោយស៊េរីនៃឈ្មោះ CPU ដែលត្រូវបានបង្កើតឡើងតាំងពីឆ្នាំ 1978រហូតមកដល់បច្ចុប្បន្ន ដោយបានចាប់ផ្តើមពី intel 8086 ជាមូលដ្ឋានដើម្បីបង្កើត 80286, 80386, 80486, 80586, Pentium Pro, I , II , III , IV , Core I3, I5, I7, I9 និងឈ្មោះផ្សេងៗទៀតដែលផលិតដោយក្រុមហ៊ុន intel និងក្រុមហ៊ុនដទៃទៀត។ ទាំងនេះជាស្ថាបត្យកម្មដែលត្រូវបានប្រើនៅក្នុងកុំព្យូទ័រលើតុ (Desktop) និងកុំព្យូទ័រយួរដៃ(Laptop)ភាគច្រើន។ ចំណែកឯ workstation និង server ពេលបច្ចុប្បន្នក៏ប្រើ CPU នៃគ្រួសារ X86 និង X64 ដែរ។

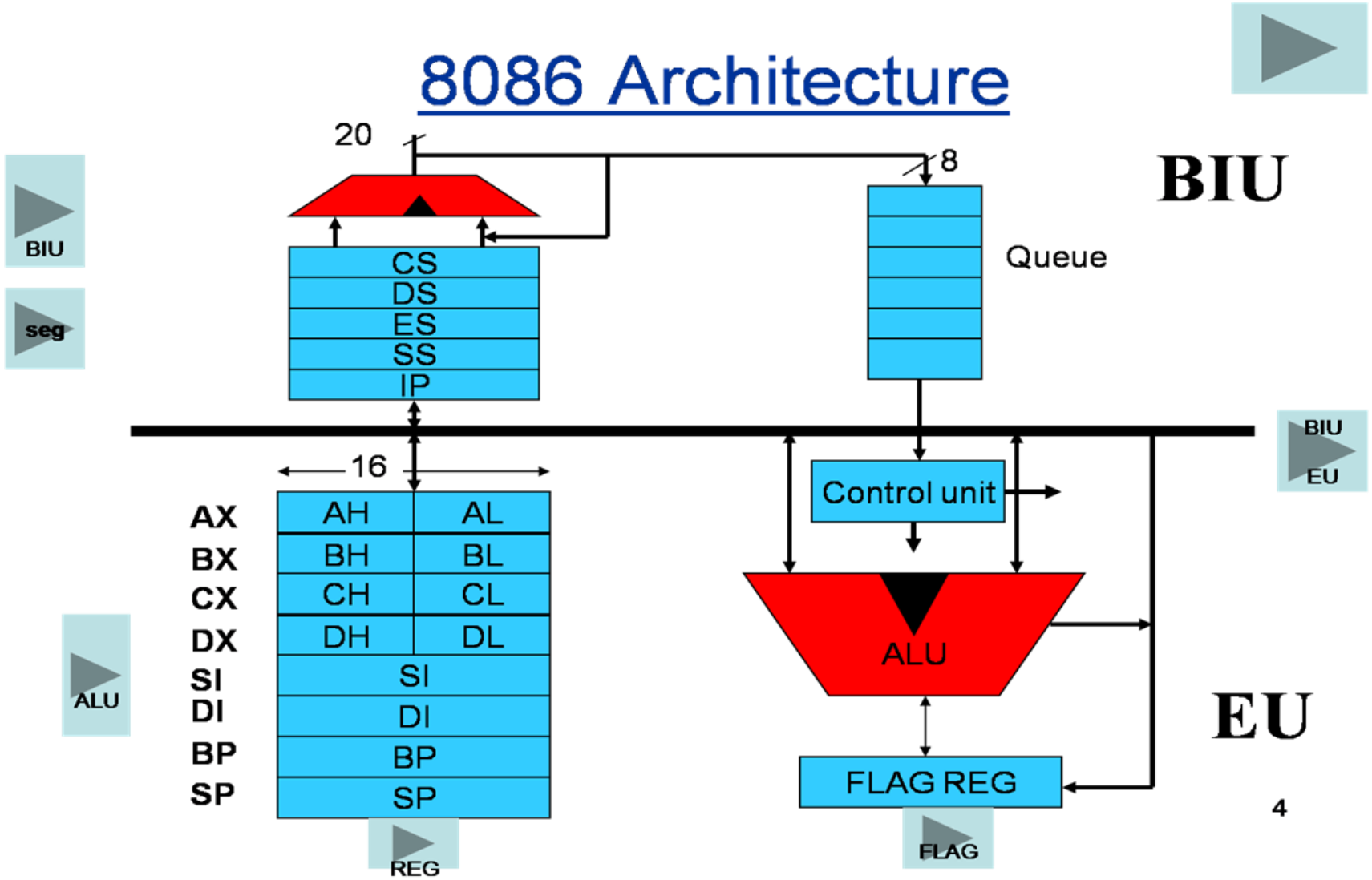
8086 Microprocessor

- 8086 / 8088 Microprocessor ចែកចេញជាផ្នែកសំខាន់ៗ គឺ Bus Interface Unit (BIU) , Execute Unit (EU) , និង Instruction Queue ដែលធ្វើអោយវាអាចដំណើរការ Code Instruction និង ដំណើរការប្រមាណវិធីផ្សេងៗ ដែលកើតចេញពីប្រូក្រាមមួយ ។
- តួនាទីសំខាន់របស់ Execute Unit ប្រតិបត្តិ Instruction , ដែលបានមកពី BIU ជាអ្នកផ្តល់ Instruction និង Data ទៅអោយវា ។ EU មាន ALU , CU , Index Register , Pointer Register និង General – Purpose Register ដែល Hardware ទាំងនេះអាចប្រតិបត្តិ Instruction , ធ្វើប្រមាណវិធី Arithmetic និង Logic និង ផ្ទុក Data រឺ លេខ Address ។

8086 Microprocessor

- BIU មាន Instruction Queue និង Segment Registers ។ មុខងាររបស់ Instruction Queue ផ្ទុក Code Instruction ដែលត្រូវប្រតិបត្តិ និង មានទំហំ 4 bytes(8086) , រឺ 6 bytes(8088) ។ ចំណែក Segment Register ផ្ទុកលេខ Address ចាប់ផ្តើមនៃផ្នែ Segment និមួយៗ ។ មុខងារដ៏សំខាន់របស់ BIU គឺ ទទួលយក Instruction ពី Memory និង ផ្លាស់វាទៅផ្ទុកក្នុង Instruction Queue ។ ក្នុងពេលកំពុងប្រតិបត្តិរបស់ 8086/8088 គឺវាប្រតិបត្តិ Instruction 1 , Instruction 2 ត្រូវបានផ្ទុកនៅក្នុង Instruction Queue និងធ្វើតាមវិធីនេះជាបន្តបន្ទាប់ រហូតដល់ចប់ Instructions ។

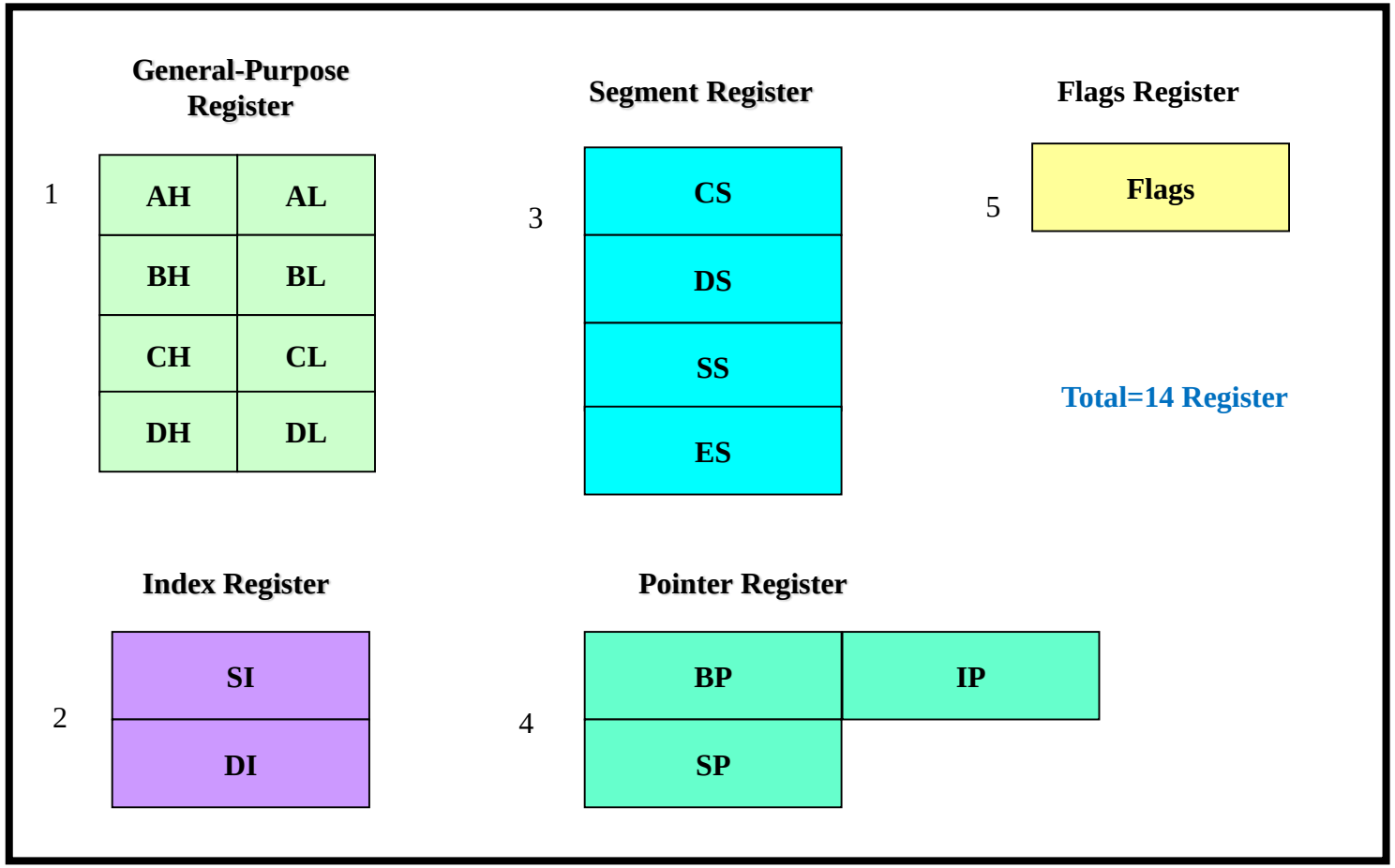
8086/8088 Block Diagram



Registers

- 8086 / 8088 មាន Internal Register គឺមានចាប់ 8 bit ទៅ 16 bit , ដូចជា AL, BL, CL, DL, AH, BH, CH, DH 8bit Register ដោយ "L" តាងអោយ Low Register និង "H" តាងអោយ Hight Register ។ រីឯ AX, BX, CX, DX គឺជា 16 bit Register , ដោយអក្សរ "X" នៅខាងក្រោយតាងអោយ 16 bit ។ Register ដែលបានរៀបរាប់ខាងលើមានឈ្មោះថា General Purpose Register និងមាន Register ដទៃទៀតដូចជា Index Registers, Segment Registers, Pointer Registers និង Flag Register.

Registers of 8086

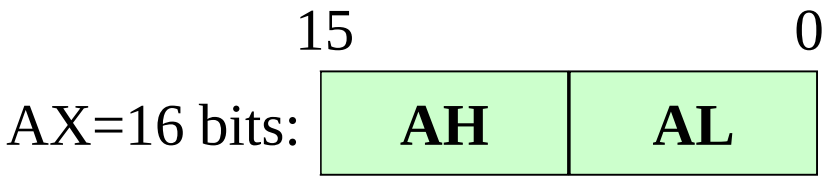


General Purpose Register

1. Accumulator Register (AX)

AX Register បានវិវត្តមកពី A Register របស់ 8085 ប្រើសំរាប់ប្រមាណវិធីដែលជាប់ទាក់ទងទៅនឹង Input និង Output រួមទាំងប្រមាណវិធី Arithmetic ជាច្រើនទៀត ។ ឧទាហរណ៍ ការចែកលេខ , ការគុណលេខ និង បកប្រែ Instruction ត្រូវប្រើ AX ។ Instruction ខ្លះដែលបង្កើតជាកូដមានប្រសិទ្ធភាពក៏ជាប់ទាក់ទងប្រើ AX ។

AX ផ្សំពី AH (Accumulator High) = 1byte និង AL (Accumulator Low)= 1Byte និង វាមានទំហំស្មើនឹង 2 bytes ។

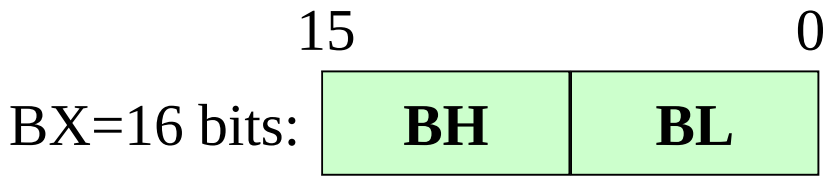


General Purpose Register

2. Base Register (BX)

Base Register ជា General – Purpose Register មួយផ្សំពី BH និង BL និង មានទំហំ 2byte ។

- មុខងារសំខាន់ របស់ BX គឺ :
  - –ផ្ទុកតំលៃ Data រឺ លទ្ធផលនៃប្រមាណវិធី Arithmetic និង Logic មួយ
  - –ឆ្ពោះទៅកាន់លេខ Offset Address នៃ Data ដែលមាននៅក្នុងផ្នែ Data Segment ។

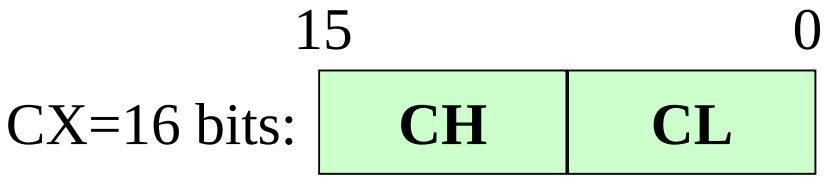


General Purpose Register

3. Count Register (CX)

Count Register ជា General – Purpose Register ផ្សំពី CH និង CL និង មានទំហំ 2 Byte ។

- មុខងារសំខាន់របស់ CX គឺ :
  - ផ្ទុកតំលៃ Data បានដូច AX
  - ផ្ទុកចំនួន Loop ដែលត្រូវធ្វើប្រមាណវិធីដែលមាន n ចំនួន
  - ផ្ទុកចំនួនដងនៃការប្រមាណវិធី SHL , SHR , រឺ Shift Bits នៃប្រមាណវិធីផ្សេងៗទៀត



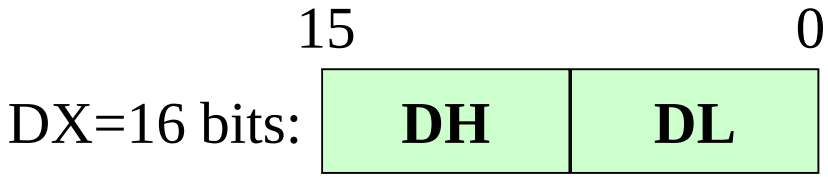
General Purpose Register

4. Data Register(DX)

DX Register គឺជា General – Purpose Register ដែលផ្សំពី DL និង DH មានទំហំ 2 Bytes ។

- មុខងារសំខាន់របស់ DX គឺ

–អាចផ្ទុក Data និង លទ្ធផលបានដូច AX ហើយចំពោះប្រមាណវិធីគុណ ប្រសិនបើលទ្ធផលមានទំហំធំ នោះត្រូវប្រើ DX register ។ DL អាចផ្ទុកចំនួន Loop ដូច CX ។



Accessing part of Registers

- 16 bit (AX,BX,CX,DX) registers បានបន្ថែមទំហំទៅជា 32 bit ត្រូវបានហៅថា EAX, EBX,ECX,EDX.

32-bit	16-bit	8-bit (high)	8-bit (low)
EAX	AX	AH	AL
EBX	BX	BH	BL
ECX	CX	CH	CL
EDX	DX	DH	DL

Flag Registers

FR គឺជា Register ដែលមានទំហំ 2 bytes មានមុខងារផ្ទុក Flags ទាំងអស់ដែលបានមកពីប្រមាណវិធីមួយ រឺ កើតចេញពីការប្រើ Program មួយ ។ 8086 Microprocessor មាន Flags ទាំងអស់ចំនួនប្រាំបួនដែលមានដូចបង្ហាញក្នុងរូបខាងក្រោម :

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	R	R	R	OF	DF	IF	TF	SF	ZF	U	AF	U	PF	U	CF

Flag Registers

ការប្រើ Flags ជាការសំខាន់ណាស់ក្នុងប្រមាណវិធី Condition និង ប្រមាណវិធី Loop ។ គ្រប់ Jump instruction ត្រូវបានប្រើជាមួយ Flags ដើម្បីឆ្ពោះទៅកាន់ Target Address នៃ Program Assembly មួយ ។ ជាពិសេសដើម្បីប្រើ Flags ក្នុង Program ត្រូវចេះកំនត់ Flags អោយបានច្បាស់លាស់ ។

U= Undefine	R= Reserve	DF= Direction Flag	IF= Interrupt Flag
TF=Trace Flag	SF= Sign Flag	AF= Auxiliary Flag	CF= Carry Flag

Flag Registers

a. Carry Flag

- CF= 1 នៅពេលធ្វើប្រមាណវិធីមួយកើតមានត្រាទុកត្រង់ខ្ទង់ Bits ទី 7 ចំពោះប្រមាណវិធី 8 bits រឺ ខ្ទង់ Bits ទី 15 ចំពោះប្រមាណវិធី16 bits , ខ្ទង់ Bits ទី 31 ចំពោះប្រមាណវិធី 32bits ។

- CF = 0 នៅពេលប្រមាណវិធីមួយគ្មានត្រាទុកត្រង់ខ្ទង់Bits ទី 7 ចំពោះប្រមាណវិធី 8 bits រឺ ខ្ទង់ Bits ទី 15 ចំពោះប្រមាណវិធី 16 bits , ខ្ទង់ Bits ទី 31 ចំពោះប្រមាណវិធី 32 bits ។

Flag Registers

b. Auxiliary Carry Flag (AF)

- AF = 1 នៅពេលប្រមាណវិធីមួយកើតមានត្រាទុកត្រង់ខ្ទង់ Bits ទី 3 ចំពោះប្រមាណវិធី 8 bits , 16 bits និង 32 bits.

- AF = 0 នៅពេលធ្វើប្រមាណវិធីមួយគ្មានត្រាទុកត្រង់ខ្ទង់ Bits ទី 3 ចំពោះប្រមាណវិធី 8 bits.

Flag Registers

c. Parity Flag (PF)

- បន្ទាប់ពីធ្វើប្រមាណវិធីមួយ , PF ពិនិត្យទៅលើលេខលទ្ធផលនៃប្រមាណវិធីត្រង់ Lower byte ជាចំនួនគូរដែលបានមកពីការរាប់លេខ 1 នោះ PF=1 ។

- បន្ទាប់ពីធ្វើប្រមាណវិធីមួយ , PF ពិនិត្យទៅលើលេខលទ្ធផលត្រង់ Lower byte។ បើ Lower byte ជាចំនួនសេសដែលបានមកពីការរាប់លេខ 1 នោះ PF មិនកំនត់ ។

Flag Registers

d. Zero Flag (ZF)

- ZF = 1 នៅពេលលេខលទ្ធផលនៃប្រមាណវិធី Arithmetic រឺ Logical មានតំលៃស្មើ 0

- ZF = 0 នៅពេលលេខលទ្ធផលនៃប្រមាណវិធី Arithmetic មានតំលៃខុសពី 0 ។

e. Sign Flag (SF)

- SF = 1 នៅពេលលទ្ធផលនៃប្រមាណវិធី Arithmetic រឺ Logical មានលេខត្រង់ MSB ស្មើនឹង 1 ។

  នៅពេល SF = 1 , បញ្ជាក់លទ្ធផលនោះជាចំនួនអវិជ្ជមាន ។

- SF = 0 នៅពេលលទ្ធផលនៃប្រមាណវិធី Arithmetic រឺ Logical មានលេខត្រង់ MSB ស្មើនឹង 0 ។

  នៅពេល SF = 0 , បញ្ជាក់លទ្ធផលនោះជាចំនួនវិជ្ជមាន ។

Flag Registers

- **ឧទាហរណ៍ រក Flags តាមប្រមាណវិធី (39H + 2FH)**

		ចំលើយ	
	39H	00111001	57
ADD	2FH	00101111	47
	68H	01101000	104

CF = 0 ដោយគ្មាន Carry ត្រង់ខ្ទង់ B7 (ខ្ទង់ Bit គឺរាប់ពី 0 ទៅ)

PF=0, ដោយការរាប់លេខ 1′s ក្នុង Result ជាចំនួនសេស

AF = 1 ដោយមាន Carry ត្រង់ខ្ទង់ B3

ZF = 0 ដោយលទ្ធផលនៃប្រមាណវិធីមិនស្មើ0 (បើលទ្ធផលនៃប្រមាណវិធីស្មើ0 នោះ ZF = 1)

SF = 0 ដោយលេខត្រង់ MSB = 0 រឺ ខ្ទង់ Bits ទី 7 ស្មើ 0

Flag Registers

f. Trace Flag (TF)

- TF = 1 ពេល Programmer ប្រើ T Command រឺ Trace Command រឺ Trace Command ក្នុង Debug Program រឺ C-Program ។

- TF = 0 ពេល Programmer មិនប្រើ T Command រឺ Trance Command ។

ចំណាំ : T រឺ Trace Command ជាបញ្ជាអោយ Debug រឺ C Program ប្រតិបត្តិម្តងមួយ Statement ។

Flag Registers

g. Interrupt Flag (IF)

- IF គឺដំណើរការដូច IF របស់ 8085 ដែរ ។

h. Direction Flag(DF)

- DF = 1 ពេល Programmer ប្រើ STD (Set Direction Ins) នៅក្នុង Program ។

- DF = 0 ពេល Programmer ប្រើ CLD (Clear Direction Ins) នៅក្នុង Program ។

Flag Registers

ចំណាំ :

- STD ជា Instruction កំនត់អោយ DF = 1 ដើម្បីអោយ CPU អាចយកតួអក្សរពីស្តាំទៅខាងឆ្វេង រឺ ពីខាងក្រោមទៅខាងលើ ។

- CLD ជា Instruction កំនត់អោយ DF = 0 ដើម្បីអោយ CPU អាចយកតួអក្សរពីខាងឆ្វេងទៅខាងស្តាំ រឺ ពីខាងលើទៅខាងក្រោមលើ ។

Flag Registers

I. Overflow Flag (OF)

- OF =1 ពេលប្រមាណវិធីមួយមាន Carry ត្រង់ខ្ទង់ bit 6 , ប៉ុន្តែគ្មាន Carry ត្រង់ខ្ទង់ bit 7 រឺគ្មានខ្ទង់ទី 6 , ប៉ុន្តែមាន Carry ត្រង់ bit ទី 7 ចំពោះប្រមាណវិធី 8bit ។ ចំពោះប្រមាណវិធី 16 bit គឺមើល Carry ត្រង់ bit ទី 14 និង bit ទី 15 ។

- OF = 0 ពេលប្រមាណវិធីមួយមាន Carry ខ្ទង់ bit 6 និង ខ្ទង់bit 7 រឺគ្មាន Carry ត្រង់ខ្ទង់ bit 6 និងខ្ទង់ bit 7 ។ ចំពោះប្រមាណវិធី 16 bit គឺមើល Carry ត្រង់ bit ទី 14 និង bit ទី 15 ។

Index Registers

1.Destination Index Register (DI)

Destination Index Register ជាRegister មួយដែលមានទំហំ 2 bytes និងមិនអាចចែក Location ដូច AX, BX,DX រឺ CX បានទេ ។ មុខងាររបស់ DI គឺ :

- ផ្ទុកតំលៃ Data និង លទ្ធផលនៃការធ្វើប្រមាណវិធី Logic និង Arithmetic មួយឆ្ពោះទៅកាន់លេខ offset Address នៃ Data ដែលត្រូវដំណើរការ ។ ប្រើដូច BX ។

- ប្រើជា Pointer Register សំរាប់ឆ្ពោះទៅកាន់លេខ offset Address របស់ Data ជា String Character នៅក្នុងផ្នែ Extra Data Segment ។

Index Registers

2. Source Index Register (SI)

Source Index Register ជា Register មួយដែលមានទំហំ 2 bytes និងមិនអាចចែក Location ដូច AX, BX, DX រឺ CX បានទេ ។ មុខងាររបស់ SI គឺ :

- ផ្ទុកតំលៃ Data និង លទ្ធផលនៃការធ្វើប្រមាណវិធីមួយ
- ឆ្ពោះទៅកាន់ offset Address ប្រើដូច BX
- ឆ្ពោះទៅកាន់ offset Address Data ជា String Character នៅក្នុងផ្នែ Data Segment ។

Instruction of 8086

1. Move Instruction (MOV)

MOV គឺជា Instruction មួយប្រើសំរាប់ចំលង Data ពី Memory ទៅផ្ទុកនៅក្នុង Register រឺពី Register ទៅផ្ទុកក្នុង Memory វិញ ។ Instruction នេះក៏អាចចំលង Data ពី Register មួយទៅ Registers មួយទៀត ។

Syntax : MOV destination , source ; ពេលប្រើ MOV Instruction ត្រូវកំនត់ destination = source

ឧទាហរណ៍ : ប្រើ MOV ins ជាមួយ Register និងប្រាប់ពីមុខងាររបស់ Register ។

```
MOV AX , 1234H          ; ចំលងតំលៃ 1234H ចូលក្នុង AX register ; AX = 1234H
MOV CL , AL             ; ចំលងតំលៃពីក្នុង AL ចូលកាន់ CL register
MOV DI , SI             ; ចំលងតំលៃពីក្នុង SI register ទៅដាក់ក្នុង DI register
MOV CX , DX             ; ចំលងតំលៃពីក្នុង DX register ទៅដាក់ក្នុង CX register
```

Instruction of 8086

2. Addition Instruction (ADD)

ADD គឺជា Instruction មួយប្រើសំរាប់ប្រាប់ទៅ ALU ធ្វើប្រមាណវិធីបូកពីចំនួន ដោយ Data ហៅចេញដោយស្ម័យប្រវត្តិពី Register និង Data 2 ហៅចេញពី Register/Memory លទ្ធផលក្រោយពី ALU គណនារួចផ្ទុកនៅក្នុង Register ជា Destination ។

Syntax : ADD destination , source ; ពេលប្រើ ADD Instruction ត្រូវកំនត់ destination = destination + source

ឧទាហរណ៍ : ប្រើ ADD Instruction ជាមួយ Register ប្រាប់ពីមុខងាររបស់ Register ។

```
ADD AL , CL      ;គឺជាការបូកលេខពីរចំនួនដោយយកតំលៃក្នុង AL បូកនឹងតំលៃក្នុងCL ហើយលទ្ធលផ្ទុកនៅក្នុង AL
ADD AX ,1245H    ;AX=AX+1234H
ADD DX , x       ;DX=DX+x
ADD sum, DI      ;sum=sum+DI
```

Instruction of 8086

3. Subtraction Instruction (SUB)

SUB គឺជា Instruction មួយប្រើសំរាប់ប្រាប់ទៅ ALU ធ្វើប្រមាណវិធីដកពីចំនួន ដោយ Data ហៅចេញដោយស្វ័យប្រវត្តិពី Register និង Data 2 ហៅចេញពី Register/Memory លទ្ធផលក្រោយពី ALU គណនារួចផ្ទុកនៅក្នុង Register ជា Destination ។

Syntax : SUB destination , source ; ពេលប្រើ SUB Instruction ត្រូវកំនត់ destination = destination - source

ឧទាហរណ៍ : ប្រើ SUB Instruction ជាមួយ Register ប្រាប់ពីមុខងាររបស់ Register ។

```
SUB AL , CL        ;AL=AL-CL

SUB AX ,1234H      ;AX=AX-1234H

SUB r , DI         ;r=r-DI

SUB total, 15H     ;total=total-15H
```

4. Decrement Instruction (DEC)

DEC ជា Instruction សំរាប់ធ្វើការបន្ថយតម្លៃ1នៅក្នុង Register ។

Syntax : DEC Reg

ឧទាហរណ៍ : ប្រើ SUB Instruction ជាមួយ Register

```
DEC CX        ; CX = CX – 1

DEC CX        ; CX = CX – 1
```

ចំពោះប្រមាណវិធីខាងលើយើងអាចជំនួសដោយ SUB CX,2

Instruction of 8086

5. Increment Instruction (INC)

INC ជា Instruction សំរាប់ធ្វើការបន្ថែមតម្លៃ1នៅក្នុង Register ។

Syntax : INC Reg

ឧទាហរណ៍:

```
MOV BX      , 1230H ; BX = 1230H

INC BX      ; BX = BX + 1 = 1231H

INC BX      ; BX = BX + 1          ; ADD BX , 1
```

5. Jump Not Zero Instruction (JNZ)

JNZ គឺជា Instruction សំរាប់ឆ្ពោះទៅកាន់ Target Address បើសិនជា ZF = 0 និងមិនឆ្ពោះទៅកាន់ Target Address បើសិនជា ZF= 1 ។

តំលៃ ZF=0 នៅពេល register CX មានតំលៃខុសពី0 ផ្ទុយទៅវិញគឺ ZF = 1 នៅពេលតំលៃនៅក្នុង CX ស្មើសូន្យ ។

Syntax : JNZ Target Address (is called Label in c programming)

6. Multiplication Instruction (MUL)

MUL ជាបញ្ជាប្រើសម្រាប់ធ្វើប្រមាណវិធីគណនាផលគុណ។

Multiplication	Operand1	Operand2	Result
Byte * Byte	AL	Register or Memory	AX
Word * Word	AX	Register or Memory	DX:AX
Byte * Word	AL AH	Register or Memory	DX:AX
DWord * Dword	EAX	Register or Memory	EDX:EAX

Instruction of 8086

ឧទាហរណ៍១: គណនាផលគុន 10 * 20 (byte * byte) លទ្ធផលផ្ទុកក្នុង AX register

```
.Data
        x       db      10              ;       x=10    (byte)
        y       db      20              ;       y=20    (byte)
.Code
start:
        mov al,x                        ;       al=x
        mul y                           ;       ax=ax*y
        PrintDec ax
Ret
end start
```

Instruction of 8086

ឧទាហរណ៍២: គណនាផលគុន 10 * 20 (word * word) លទ្ធផលផ្ទុកក្នុង DX:AX register

```
.Data
        x       dw      100             ;       x=100   (word)
        y       dw      20              ;       y=200   (word)
.Code
start:
        mov ax,x        ;       ax=x
        mul y           ;       dx:ax=ax*y
        PrintDec ax     ;       display value of ax
        PrintDec dx     ;       display value of dx
Ret
end start
```

Instruction of 8086

ឧទាហរណ៍៣: គណនាផលគុន 10 * 20 (dword * dword) លទ្ធផលផ្ទុកក្នុង EDX:EAX register

```
.Data
        x       dd      1000            ;       x=1000  (double word)
        y       dd      2000            ;       y=2000  (double word)
.Code
start:
        mov eax,x       ;       eax=x
        mul y           ;       edx:eax=eax*y
        PrintDec eax    ;       display value of eax
        PrintDec edx    ;       display value of edx
Ret
end start
```

7. Division Instruction (VID)

DIV ជាបញ្ជាប្រើសម្រាប់ធ្វើប្រមាណវិធីចែក។

Division	Dividend	Divisor	Quotient	Remainder
Byte / Byte	AL=B , AH=0	Memory or Register	AL	AH
Word / Word	AX=W , DX=0	Memory or Register	AX	DX
Word / Byte	AX=Word	Memory or Register	AL	AH
Dword / Word	DX:AX=DWord	Memory or Register	AX	DX
Dword / Dword	EAX	Memory or Register	EAX	EDX

Instruction of 8086

ឧទាហរណ៍១: គណនាផលគុន 92 / 2 (byte * byte) លទ្ធផលផ្ទុកក្នុង AX register

```
.Data
        num     dd      92              ; x=92      (double word)
        two     dd      2               ; y=2       (double word)
.Code
start:
        mov edx,0               ; edx=0
        mov eax,num             ; eax=num
        div two                 ; edx=eax/two
        PrintDec    eax         ; display the value of eax
        PrintDec    edx         ; display the value of edx
Ret
end start
```

8. AND Instruction

AND ជាពាក្យបញ្ជាប្រើសម្រាប់អនុវត្តប្រមាណវិធីកត្តវិទ្យាឈ្នាប់ប្រសព្វរវាងទិន្នន័យពីរ និងលទ្ធផលផ្ទុកក្នុង destination។ destination អាចប្រើជា memory, register រីឯ source operand អាចប្រើជា immediate, memory , register.

Syntax : AND destination , source ; destination=destination AND source

9. OR Instruction

OR ជាពាក្យបញ្ជាប្រើសម្រាប់អនុវត្តប្រមាណវិធីកត្តវិទ្យាឈ្នាប់ប្រជុំរវាងទិន្នន័យពីរ និង លទ្ធផលផ្ទុកក្នុង destination។ destination អាចប្រើជា memory, register រីឯ source operand អាចប្រើជា immediate, memory , register.

Syntax : OR destination , source ; destination=destination OR source

10. NOT Instruction

OR ជាពាក្យបញ្ជាប្រើសម្រាប់អនុវត្តប្រមាណវិធីកត្តវិទ្យាឈ្នាប់មិនរវាងទិន្នន័យពីរ និង លទ្ធផលផ្ទុកក្នុង destination។ destination អាចប្រើជា memory, register រីឯ source operand អាចប្រើជា immediate, memory , register.

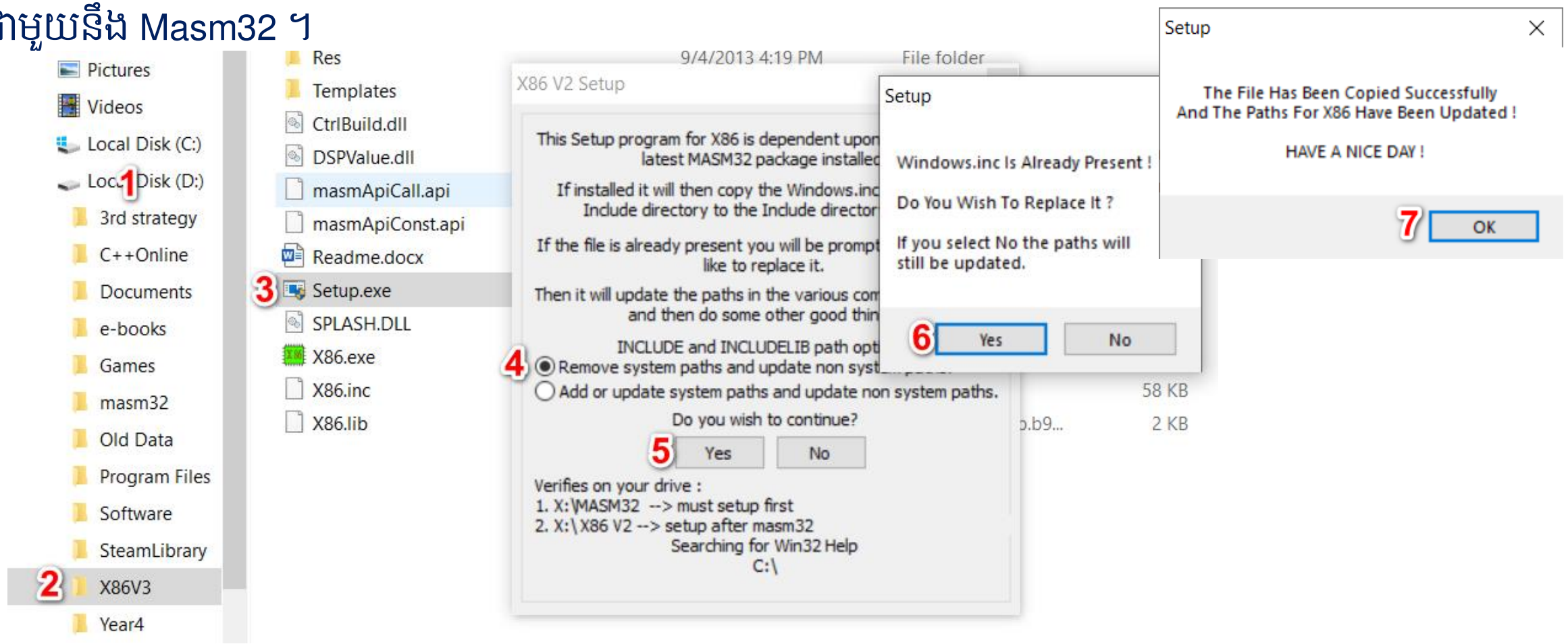
Syntax : NOT destination ; destination=NOT destination

X86Asm Editor

X86Asm ជាកម្មវិធី IDE មួយអាចសរសេរភាសា Assembly ដោយប្រើទំហំ 8bit , 16bit និង 32 bit Registers និងប្រើ Instructions របស់ CPU ចាប់ពីលេខ 8086 រហូតដល់ CPU បច្ចុប្បន្ននេះ។ វាអាចប្រើប្រាស់អនុគមន៍របស់ C/C++ និង API (Application Programming Interface) ដែលជួយសម្រួលក្នុងការសរសេរភាសា Assembly អោយកាន់តែមានភាពងាយស្រួល។ ជាងនេះទៅទៀតការសរសេរភាសា Assembly ប្រើសម្រាប់បង្កើតគំរោងថ្មីៗដោយប្រើជា Console ដោយផ្ទាល់ និងអាចប្រើ Resource Editor ដែលមានទំរង់ដូច Visual Basic ដែរក្នុងការបង្កើត Dialog ឬ Windows បានលឿនបំផុត ។

X86Asm Editor

ដើម្បីដំណើរការកម្មវិធី X86Asm Editor ទៅបានយើងត្រូវតំលើងកម្មវិធីនេះជាមុនសិន រួមជាមួយនឹង Masm32 ។



X86Asm Editor

ដើម្បីសរសេរកម្មវិធីមួយ យើងត្រូវបង្កើត project ជាមុនសិនដោយអនុវត្តតាមដំណាក់កាលខាងក្រោម:

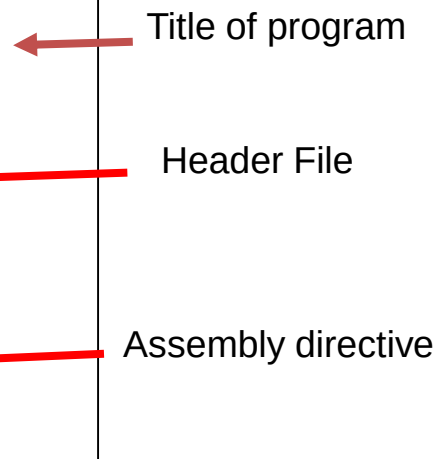
Assembly

- **Assembly or Assembler** គឺជាភាសាសរសេរកម្មវិធីមួយ ដែលបំលែងពី source code ទៅជាម៉ាស៊ីនកូដ សម្រាប់អោយកុំព្យូទ័រដំណើរការ។
- **Linker** (a companion program of assembler) គឺជាដំណើរការភ្ជាប់ file ផ្សេងៗរបស់ assembly រួមបញ្ចូលគ្នាជាfile តែមួយសម្រាប់ដំណើរការ។

add32bit.asm -> add32bit.obj -> add32bit.exe

Template of source code of assembly

```
1.  TITLE Program Template (template.asm)
2.  .586
3.  Include masm32rt.inc
4.  Include \X86V2\X86.inc
5.  Includelib \X86V2\X86.lib
6.  .data
7.  ; (insert variables here)
8.  .code
9.  start: ; The CODE entry point to the program.
10. ; (insert  instructions here)
11. ret
12. end start ; Tell MASM where the program
```



Assembly

- **.586 :** សម្រាប់បញ្ជាក់អោយ assembler ប្រើ 80386 instruction set.
- **.data :** គឺជាផ្នែកសំរាប់ប្រកាស់តម្លៃចាប់ផ្តើមក្នុងការសរសេរកម្មវិធី។
- **.code :** គឺជាផ្នែកសំរាប់សរសេរឃ្លាបញ្ជាផ្សេងៗសម្រាប់កម្មវិធី
- **start :** គឺជាផ្នែកសំរាប់ចាប់ផ្តើមដំណើរការ
- **end start :** គឺជាផ្នែកសំរាប់បញ្ចប់ដំណើរ
- **ret :** គឺជាផ្នែកសម្រាប់បញ្ជូនសញ្ញាត្រឡប់ទៅអោយប្រព័ន្ធប្រតិបត្តិការណ៍
- **Comment :** ប្រើសម្រាប់ពន្យល់ ឬសម្គាល់ឃ្លាបញ្ជា

Assembly

ដើម្បីប្រើប្រាស់ memory នៅក្នុងភាសា Assembly យើងត្រូវសិក្សាអំពីប្រភេទទិន្នន័យខាងក្រោម៖

[variable-name] define-directive initial-value

Directive	Purpose	Storage Space
DB	Define Byte	allocates 1 byte
DW	Define Word	allocates 2 bytes
DD	Define Doubleword	allocates 4 bytes
DQ	Define Quadword	allocates 8 bytes
DT	Define Ten Bytes	allocates 10 bytes

Invoke macro

invoke គឺជា macro ប្រើបម្រាប់ហៅឈ្មោះអនុគមន៍ រួមនិងធាតុរបស់វាមកប្រើប្រាស់ក្នុងការបញ្ជានៅក្នុង X86Asm IDE បានលឿននិងមានភាពងាយស្រួល។

INVOKE FUNCTION_NAME [, ARGUMENT_LIST] ; ARGUMENT_LIST

invoke crt_scanf, chr$ (“%d”), addr var

scanf(“%d”,&var);

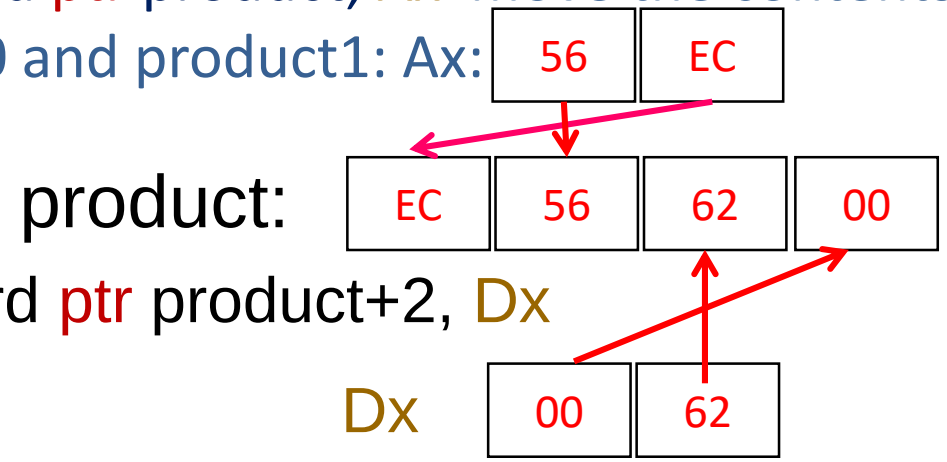
DbgDump instruction

```
;-----------------------------------------------
.Data
        word1       dw      1234h
        word2       dw      0567h
        product     dd      0

.Code
start:
    Mov ax, word1
    Mul word2
    Mov word ptr product, ax
    Mov word ptr product+2, dx
    DbgDump offset product,4
    PrintHex product


Ret
end start
;===============================================
```

```
00403004:  EC 56 62 00-                    -
product = 006256EC (RUPP.asm, 20)
```

DbgDump instruction

- DbgDump offset product
- DbgDump is a macro to see data and result in Memory .
- Offset is directive for defined variables are address .
  0 1 2 3
- product dd 0 ; ➡ product: | 00 | 00 | 00 | 00 |
- Mov word ptr product, Ax move the contents of ax to product0 and product1: Ax: | 56 | EC |



Decision Directive

If statement

Syntax :

```
.if condition
        StatementA
.endif
        StatementB
        StetementC
```

Decision Directive

If … else statement

Syntax :

```
.if condition
        StatementA
.else
        StatementB
.endif
```

Decision Directive

```
.While condition
        Statement(s)
        ;Dec or Inc Instruction
. Endw
```

Loop statement

```
.While condition
        Statement(s)
        ;Dec or Inc Instruction
. Endw
```