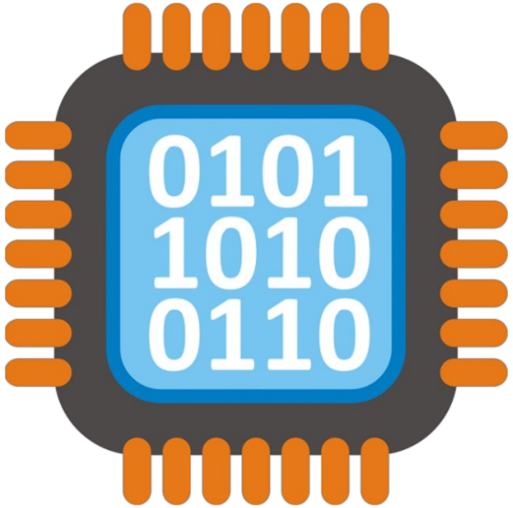
بسم الله الرحمن الرحيم

Secure Assembly Coding

Week # 4 Lectures

Dr. Qasem Abu Al-Haija

Department of Cybersecurity

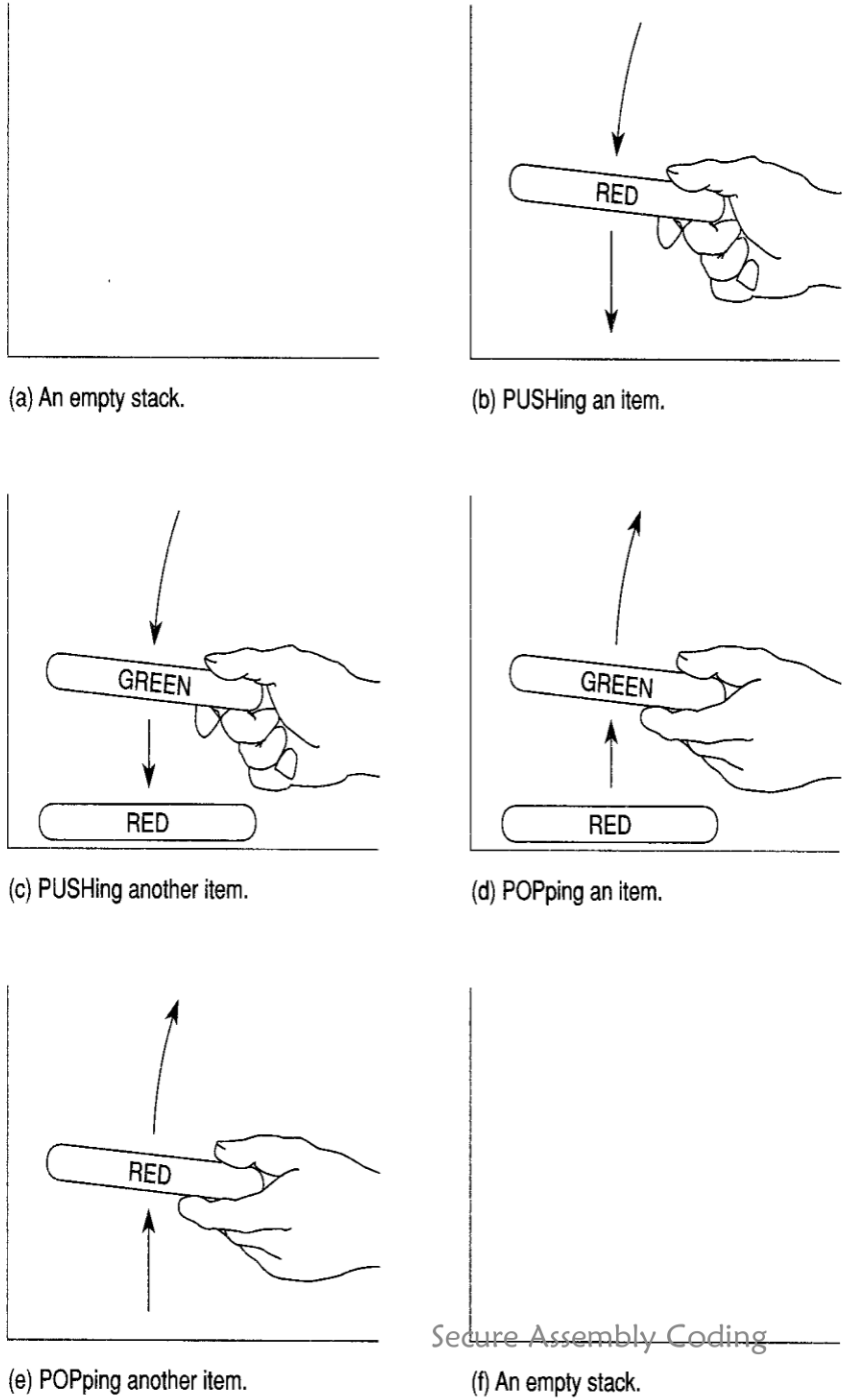


Concept of STACK

- Stack is a region of memory (RAM), which is defined by assembly program (Typically used by subroutines).

- Contains some RAM locations for RD/WR data words.
- 8086 stack is LIFO (Last In First Out) memory.

- 8086 stack is accessed by two instructions:
  - PUSH operation (write) decrements SP twice (-2) and put the word at SS:SP.
  - POP operation (read) retrieves the word at SS:SP and increments SP twice (+2).



(a) An empty stack.

(b) PUSHing an item.

(c) PUSHing another item.

(d) POPping an item.

(e) POPping another item.

(f) An empty stack.

Example of using STACK

- Example: A program stack is to use 100h bytes where:
- SS= 58A1h,
- A = 1234h
- B = 5678h.

- Figure 1 shows "empty" stack, where:
- The initial value of SP register = 0100H
- The stack starts at address 58A10H and ends at address 58B0FH (100 bytes).

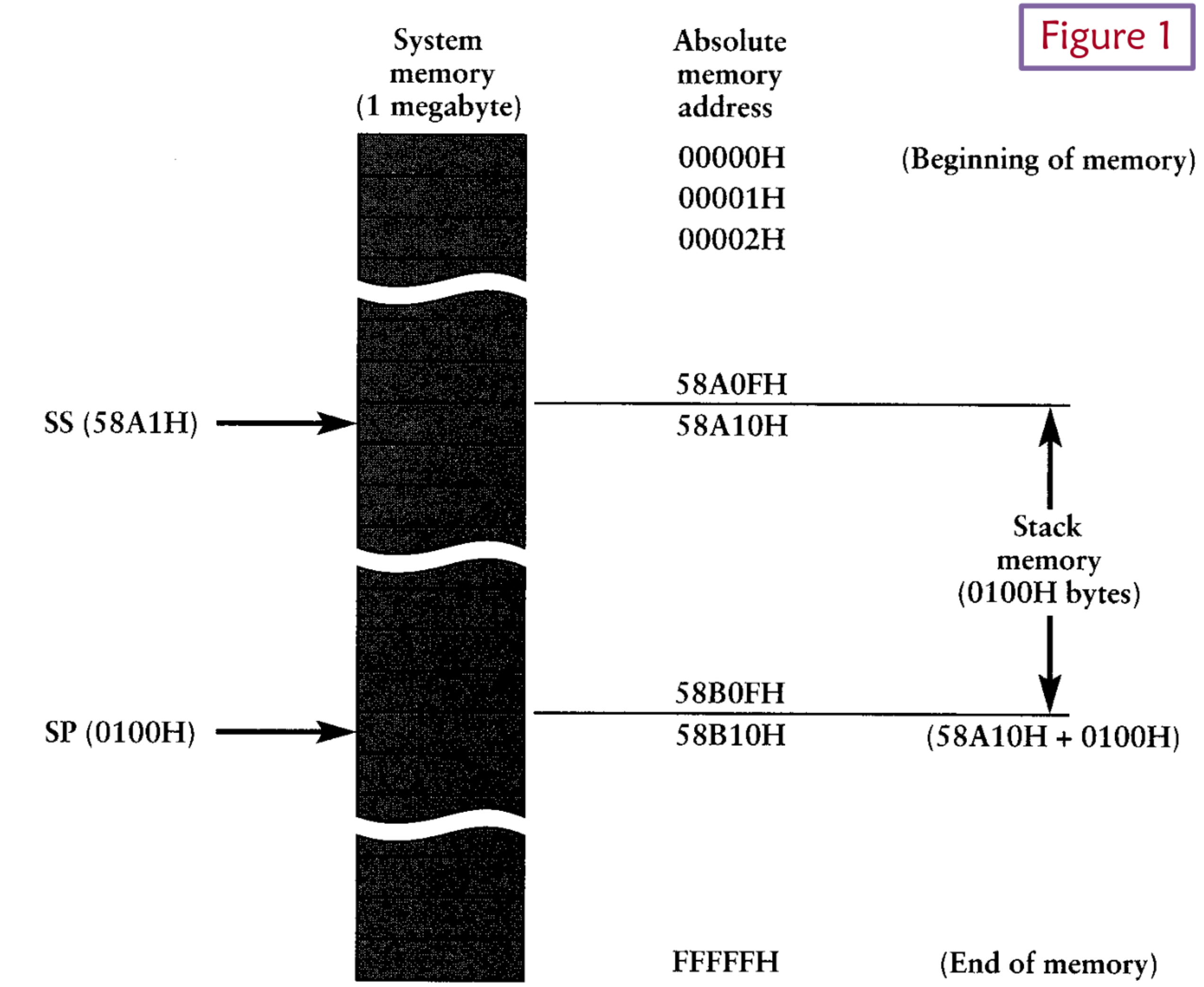


Figure 1

Example of using STACK

- Figure 2 shows the stack after two Push operations: Push A and Push B

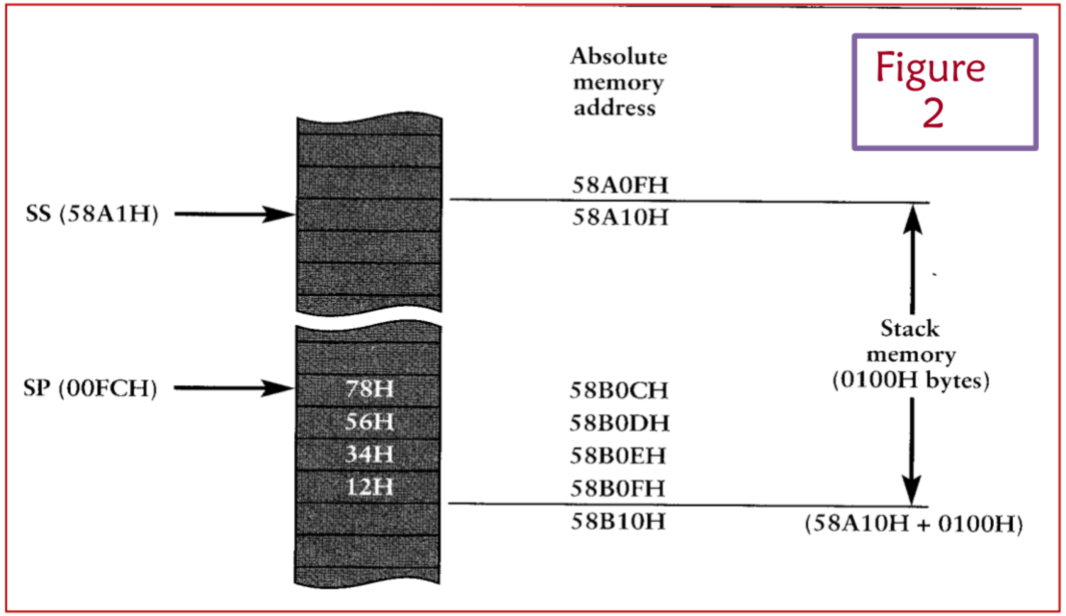


Figure 2

- Figure 3 shows the stack after two Pushes and one Pop operations (Pop B).

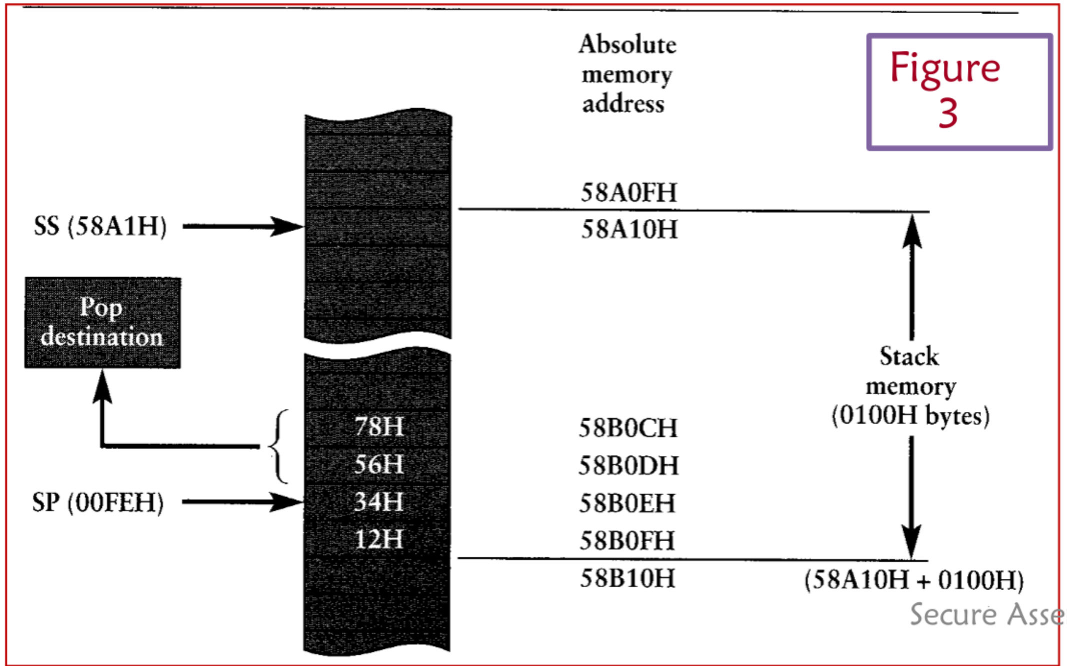
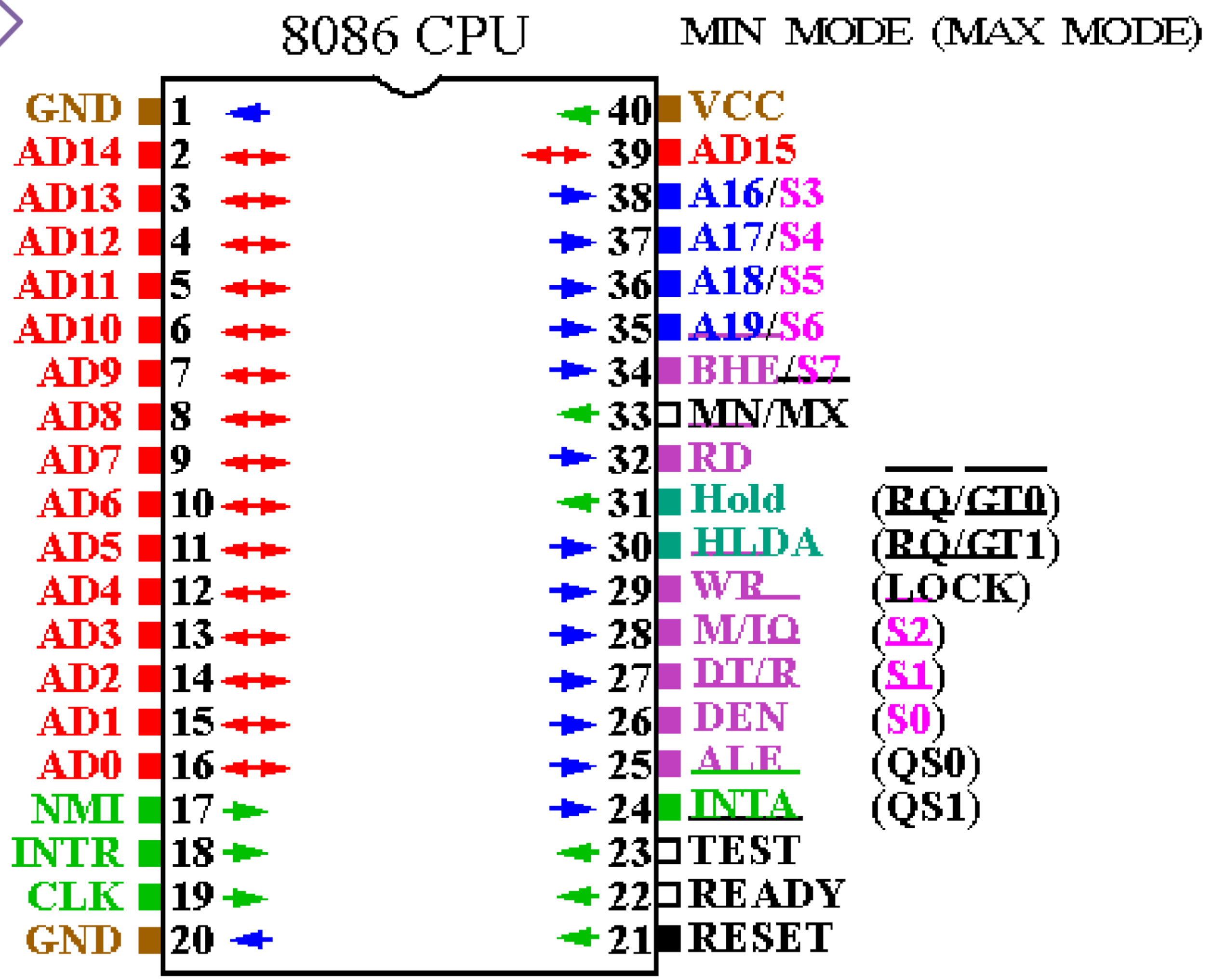


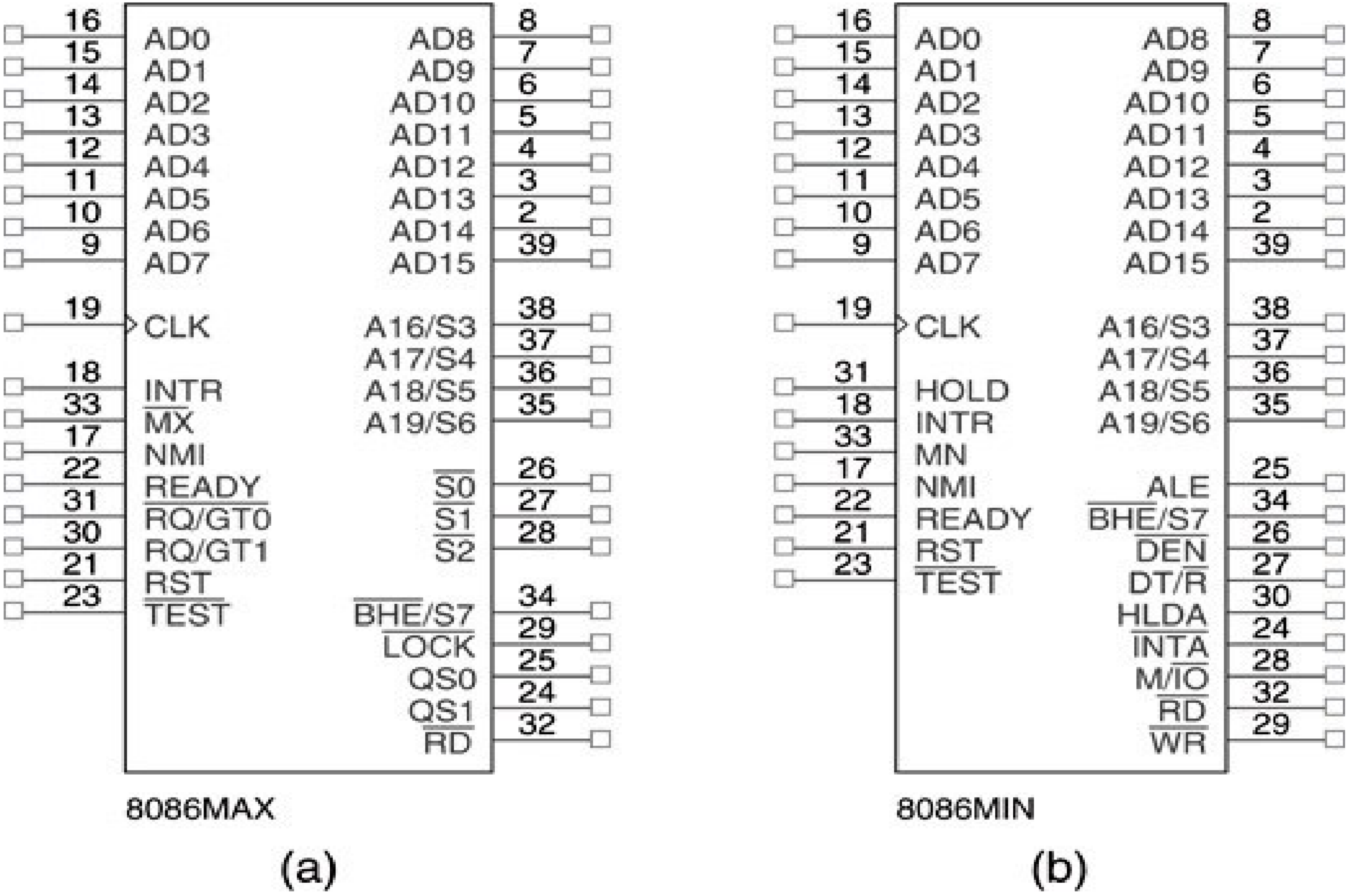
Figure 3

8086 PIN DIAGRAM AND FUNCTIONS

The direction of the arrows is important



8086 PIN DIAGRAM AND FUNCTIONS



8086 PIN DIAGRAM AND FUNCTIONS

- 8086 can be operated in MINIMUM mode and MAXIMUM mode.
- The mode is controlled by $MN/\overline{MX}$ pin.
- Min mode: all control signals for memory & I/O are generated by 8086.
- Max mode: some control signals must be externally generated.
- This requires the addition of an external bus controller(8288) with 8086.
- Some pins in 8086 have same function in both modes and some pins have different function in the two modes.

FUNCTION OF COMMON PINS

- **Vcc Pin:** Provides a +5V signal to the 8086 with tolerance of ±10%.
- **GND Pin:** The return of power supply (Vcc), 8086 has 2 GND pins.
- **ALE Pin:** is Address Latch Enable. Its used with the Multiplexed Bus as follows:
  - If ALE is high→ AD15-AD0 & A19|S6-A16|S3 will carry address bits.
  - If ALE is low → AD15-AD0 carry data & A19|S6-A16|S3 will carry status.
- **AD15-AD0:** 8086 Multiplexed address pins (A0-A15) and data bus (D0-D15).
- **A19|S6 - A16|S3:** Multiplexed address (A16-A19)and status bus bits (S6-S3).
- **$\overline{\text{TEST}}$ Pin:** used with WAIT instruction are used to poll for an external event.

FUNCTION OF COMMON PINS

- **RESET Pin:** If held high for a minimum of 4 clock cycles, it causes 8086 to reset.
  - Reset signal initializes CS & IP to FFFFH & 0000H and other registers to 0000H.
- **CLK Input:** with duty cycle of 33% to provide proper internal timing for 8086
- **NMI input:** Used to request a Non-Maskable hardware interrupt.
- **INTR input:** Used to request Maskable hardware interrupt and controlled by IF.
  - When IF=1 ➔ **INTR** is enabled / When If IF=0 ➔ **INTR** is disabled.
- **$\overline{\text{RD}}$ signal:** If low➔ 8086 **Reads** data from memory or I/O device via data bus.
- **READY input:** it inserts **WAIT** states when 8086 interface with slow peripheral.

Function of pins used in minimum mode

- **$M/\overline{IO}$ Pin:** If its high → Memory operation and if its low → an I/O operation.
- **$\overline{WR}$ Pin:** If low →8086 Writes data to memory or Output device via data bus.
- **$DT/\overline{R}$ Pin:** Controls data bus to transmit (if high) or receive (if low) data.
- **$\overline{DEN}$ Pin:** Data bus enable signal to transfer data if the pin is low.
- **$\overline{INTA}$ Pin:** Used to place interrupt vector into data bus in response to INTR.
- **HOLD pin:** Generated by DMA controller to request DMA operation from MP.
- **HLDA pin:** Indicates that 8086 entered the hold state and is connected to

  HLDA input of DMA controller.

Function of pins used in Maximum mode

- S2,S1,S0 (The states bits) signals.
  - Normally decoded by 8288 to indicate the function of current bus cycle.
- LOCK output Pin.
  - Lock peripherals off the system and activated using LOCK prefix.
- RQ/GT0 & RQ/GT1 (The request/grant pins).
  - These lines are bi-directional used to request & grant for a DMA operation.
- QS1 and QS0 (The queue status bits).
  - Show the status of the internal instruction queue in 8086.
  - These pins are provided for access by the numeric coprocessor (8087).

Downloading, installing and Preliminaries of EMU8086

EMU8086

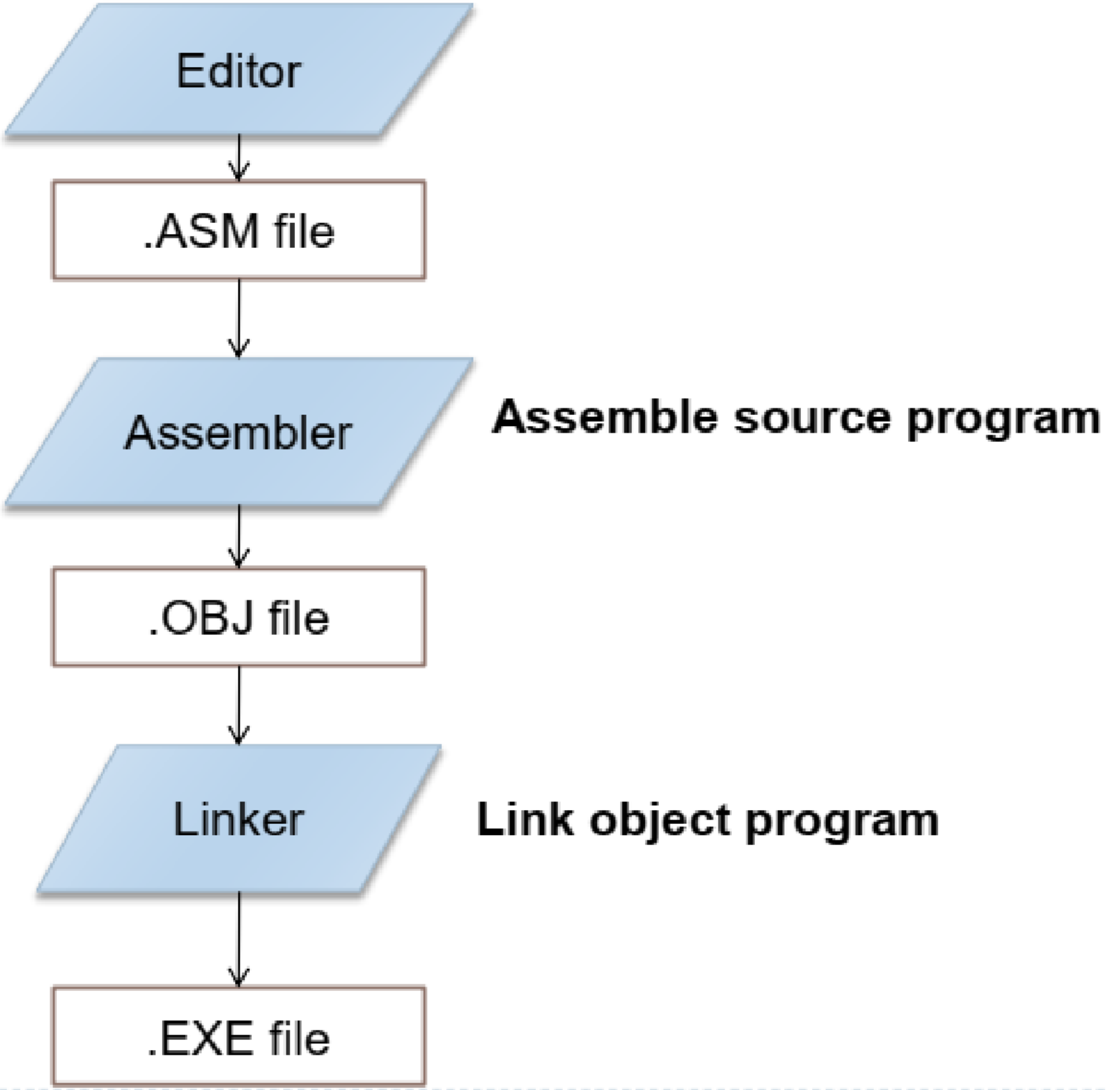
8086 EMULATION.

- Using Virtual-8086 mode to execute real-mode procedures in a protected-mode environment.

EMU8086

- This is a microprocessor emulator with an integrated 8086 Assembler.
- The emulator can run programs on a Virtual Machine,
- Emulates real hardware including screen, memory, and input and output devices.
- It helps you program in assembly language.
- The source code is compiled by assembler and then executed on Emulator step-by step,
- Allows you to watch registers, flags and memory while your program runs.

Programming Steps



Create source program

Assemble source program

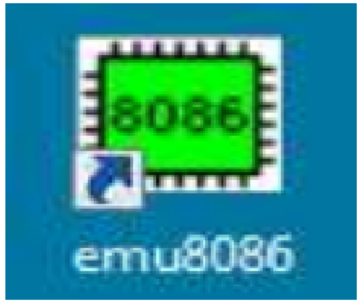
Link object program

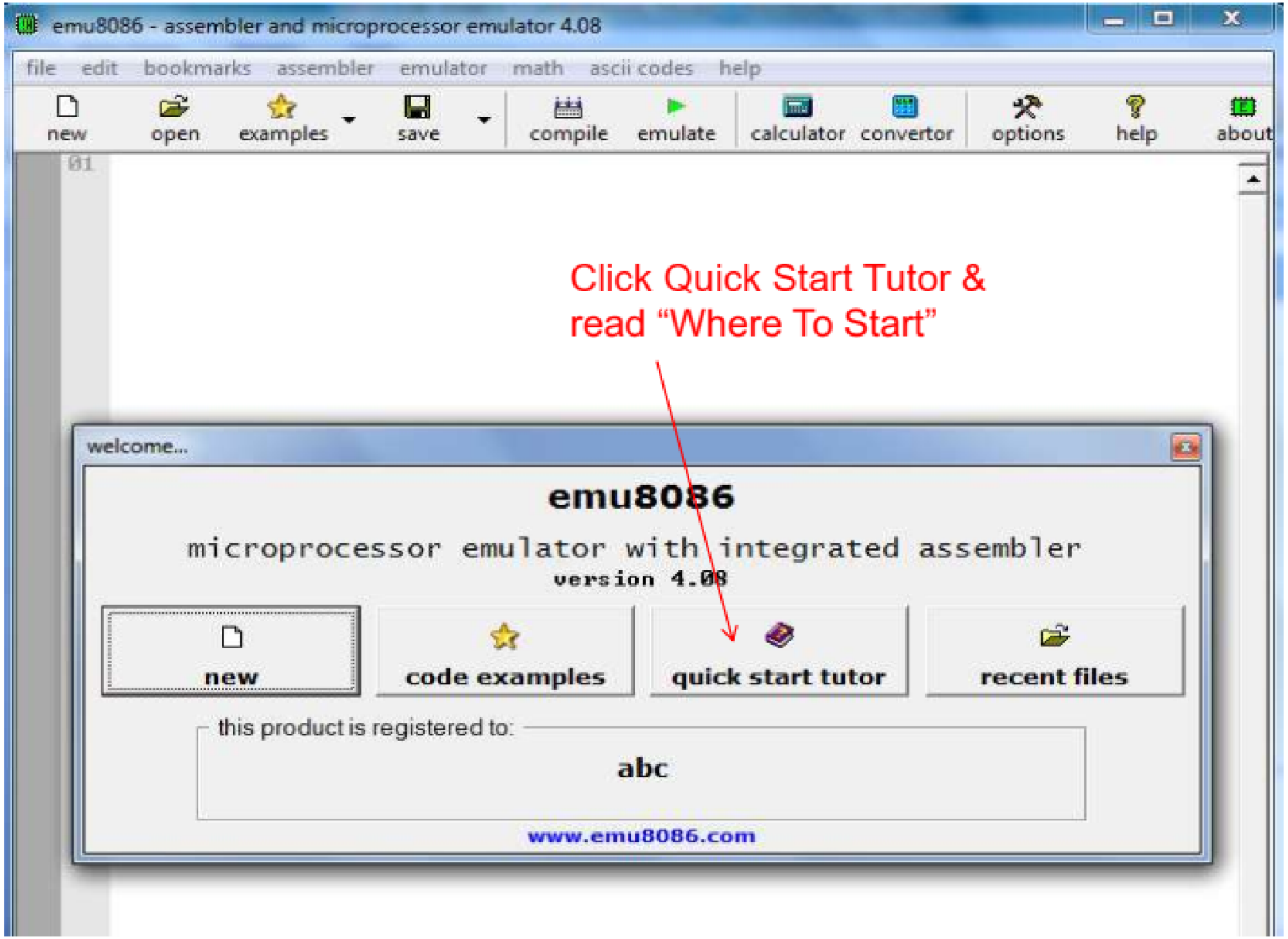
How to Open?

- Go To Start Menu → Find emu8086 in Programs

OR

- Double click following icon on your desktop:




emu8086 - assembler and microprocessor emulator 4.08
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help about
01
welcome...
new
code examples
quick start tutor
recent files

Click Quick Start Tutor & read “Where To Start”

emu8086

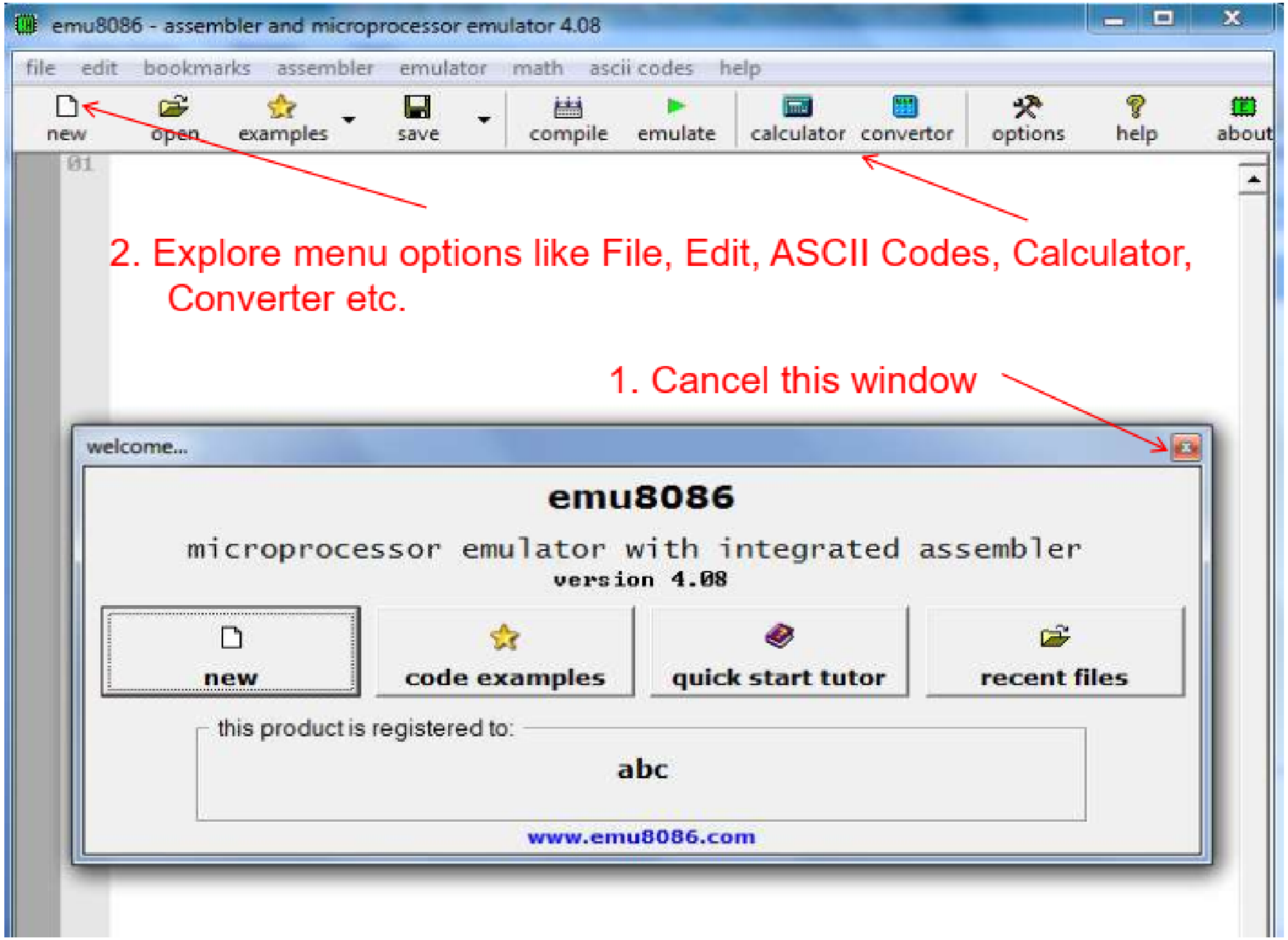
microprocessor emulator with integrated assembler

version 4.08

this product is registered to:

abc

www.emu8086.com



emu8086 - assembler and microprocessor emulator 4.08
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help about
01
welcome...
new
code examples
quick start tutor
recent files

2. Explore menu options like File, Edit, ASCII Codes, Calculator, Converter etc.

1. Cancel this window

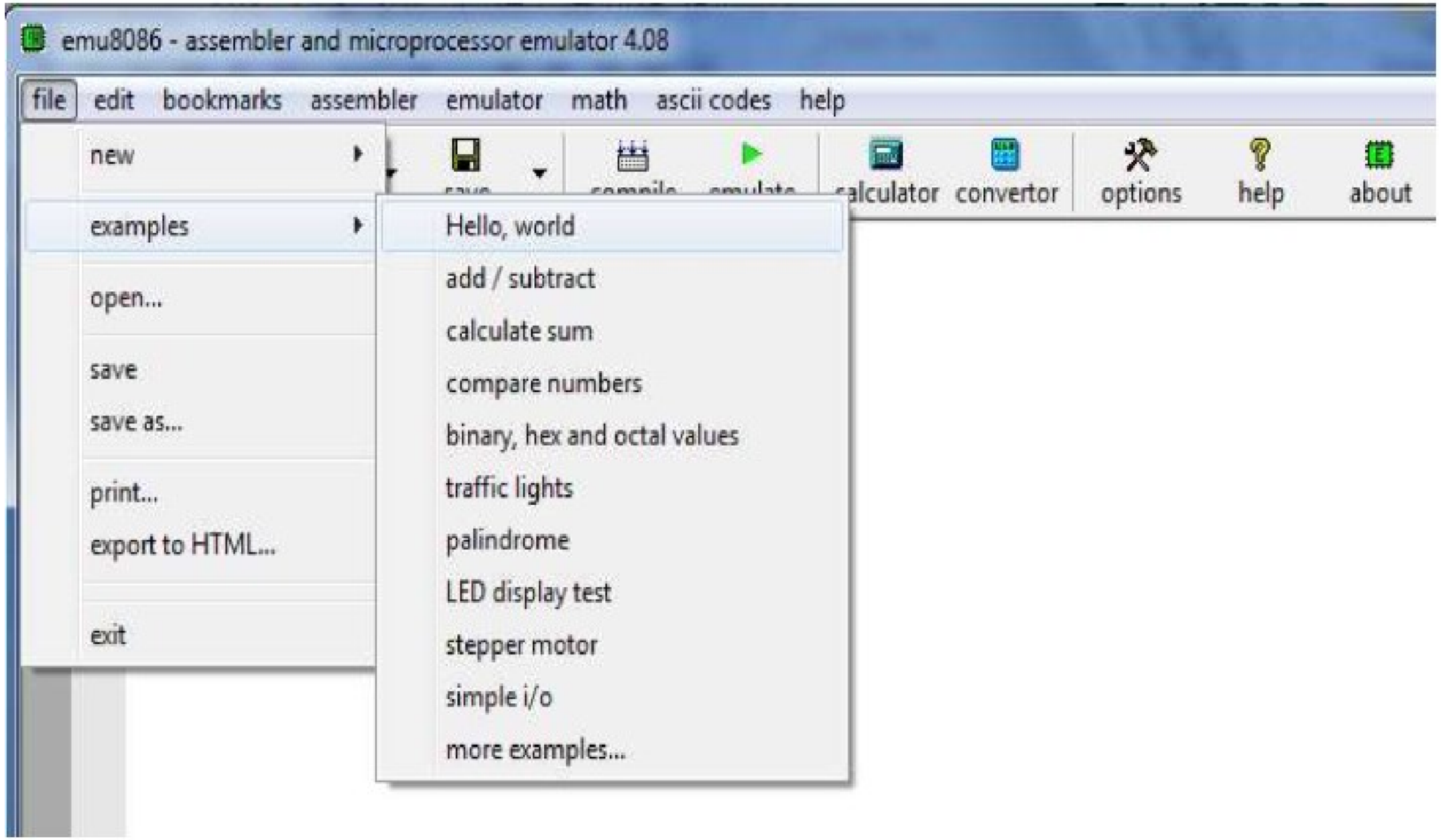
emu8086

microprocessor emulator with integrated assembler
version 4.08

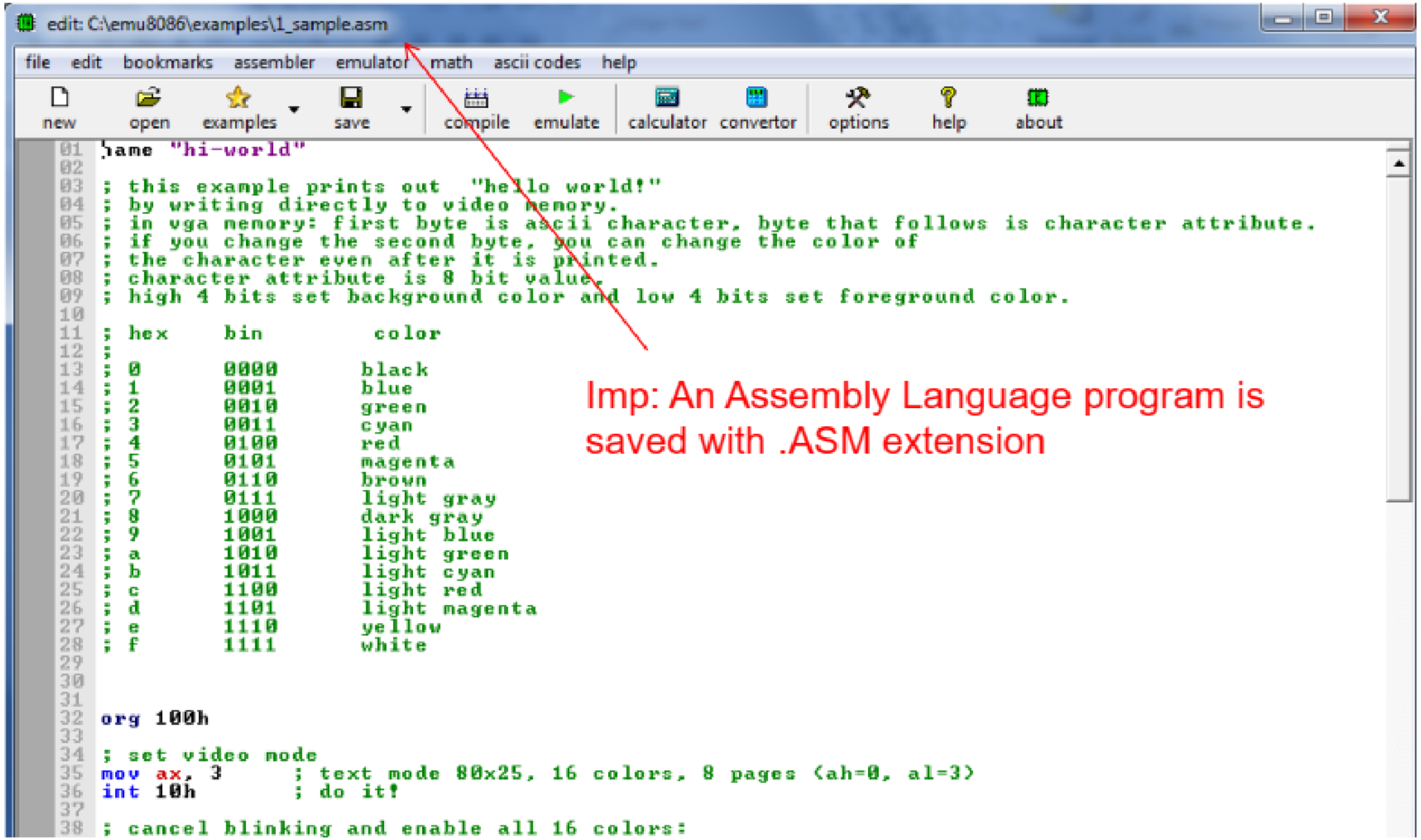
this product is registered to:
abc

www.emu8086.com

Open a Program from Examples



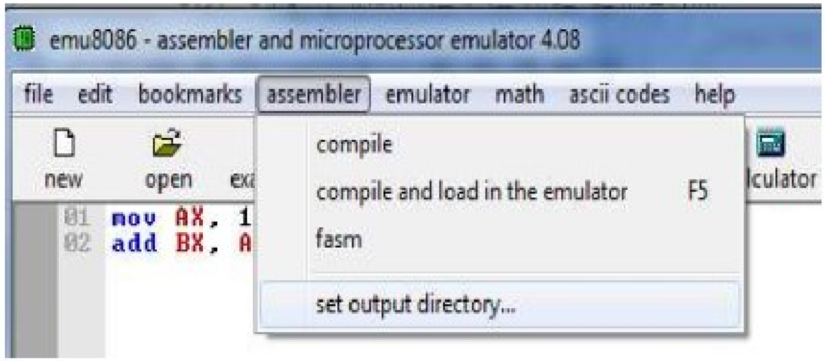
Program loaded in Editor



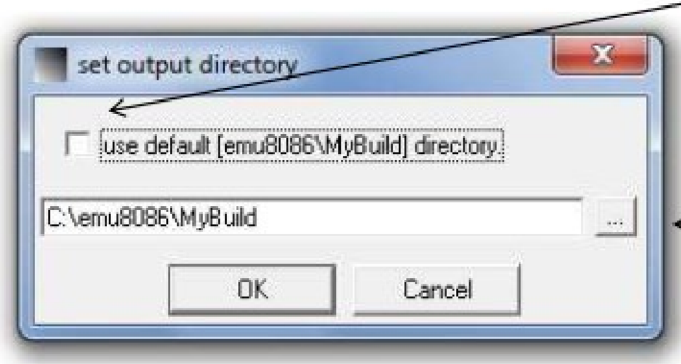
```
name "hi-world"

; this example prints out  "hello world!"
; by writing directly to video memory.
; in vga memory: first byte is ascii character, byte that follows is character attribute.
; if you change the second byte, you can change the color of
; the character even after it is printed.
; character attribute is 8 bit value,
; high 4 bits set background color and low 4 bits set foreground color.

; hex    bin         color
;
; 0      0000      black
; 1      0001      blue
; 2      0010      green
; 3      0011      cyan
; 4      0100      red
; 5      0101      magenta
; 6      0110      brown
; 7      0111      light gray
; 8      1000      dark gray
; 9      1001      light blue
; a      1010      light green
; b      1011      light cyan
; c      1100      light red
; d      1101      light magenta
; e      1110      yellow
; f      1111      white



org 100h

; set video mode
mov ax, 3     ; text mode 80x25, 16 colors, 8 pages (ah=0, al=3)
int 10h       ; do it!

; cancel blinking and enable all 16 colors:
```

Imp: An Assembly Language program is saved with .ASM extension

How to change Output Directory



```
mov AX, 1
add BX, A
```



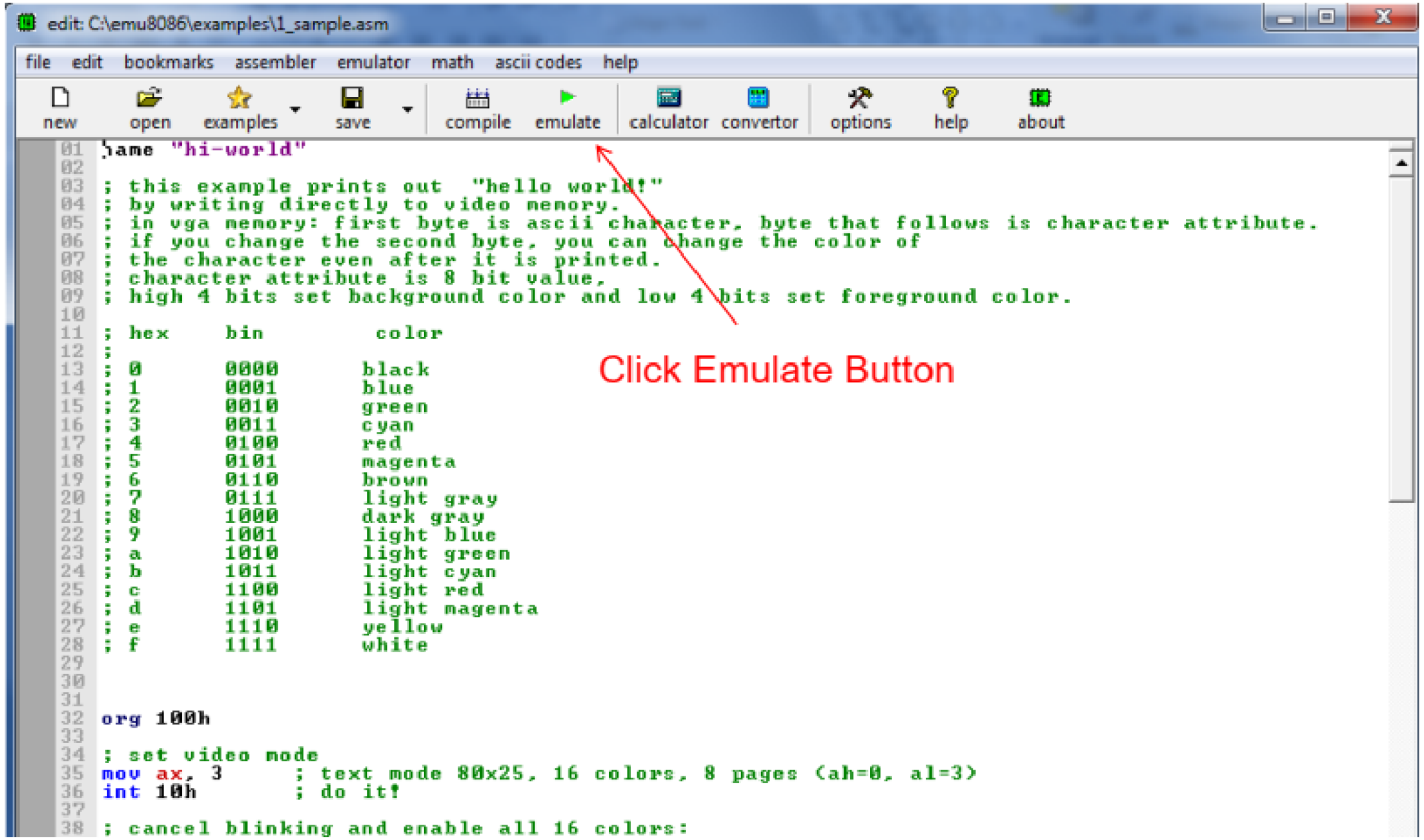
☐ use default [emu8086\MyBuild] directory

C:\emu8086\MyBuild

Uncheck this check box

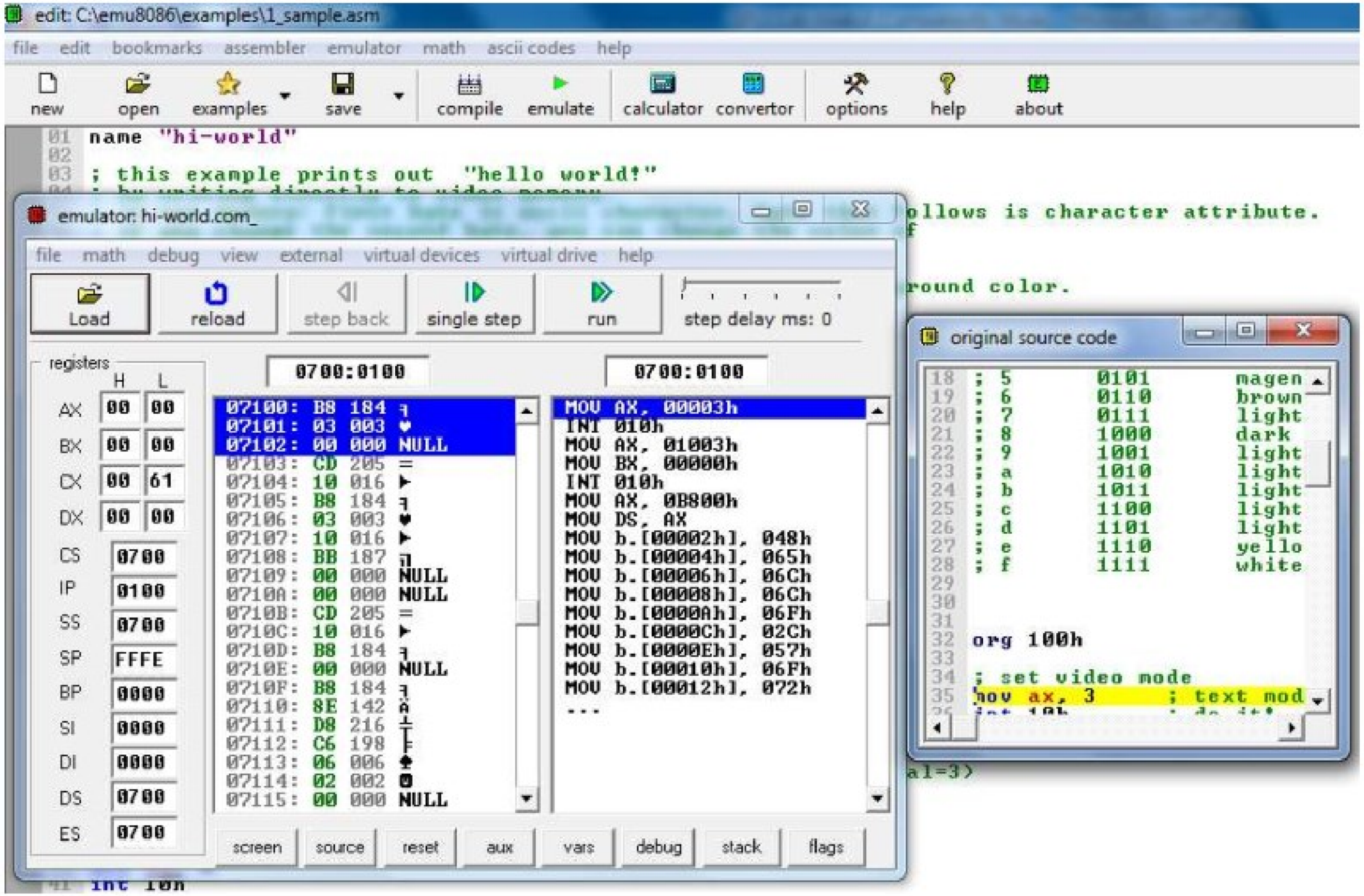
Browse path where executable file will be saved then Click Ok button

How To Run Program



```
name "hi-world"

; this example prints out  "hello world!"
; by writing directly to video memory.
; in vga memory: first byte is ascii character, byte that follows is character attribute.
; if you change the second byte, you can change the color of
; the character even after it is printed.
; character attribute is 8 bit value,
; high 4 bits set background color and low 4 bits set foreground color.

; hex    bin        color
;
; 0      0000      black
; 1      0001      blue
; 2      0010      green
; 3      0011      cyan
; 4      0100      red
; 5      0101      magenta
; 6      0110      brown
; 7      0111      light gray
; 8      1000      dark gray
; 9      1001      light blue
; a      1010      light green
; b      1011      light cyan
; c      1100      light red
; d      1101      light magenta
; e      1110      yellow
; f      1111      white



org 100h

; set video mode
mov ax, 3     ; text mode 80x25, 16 colors, 8 pages (ah=0, al=3)
int 10h       ; do it!

; cancel blinking and enable all 16 colors:
```

Click Emulate Button


edit: C:\emu8086\examples\1_sample.asm
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help about
emulator: hi-world.com_
file math debug view external virtual devices virtual drive help
Load reload step back single step run step delay ms: 0
original source code
screen source reset aux vars debug stack flags

```
name "hi-world"

; this example prints out  "hello world!"
```

```
ollows is character attribute.
f

round color.

al=3)
```

```
int 10h
```

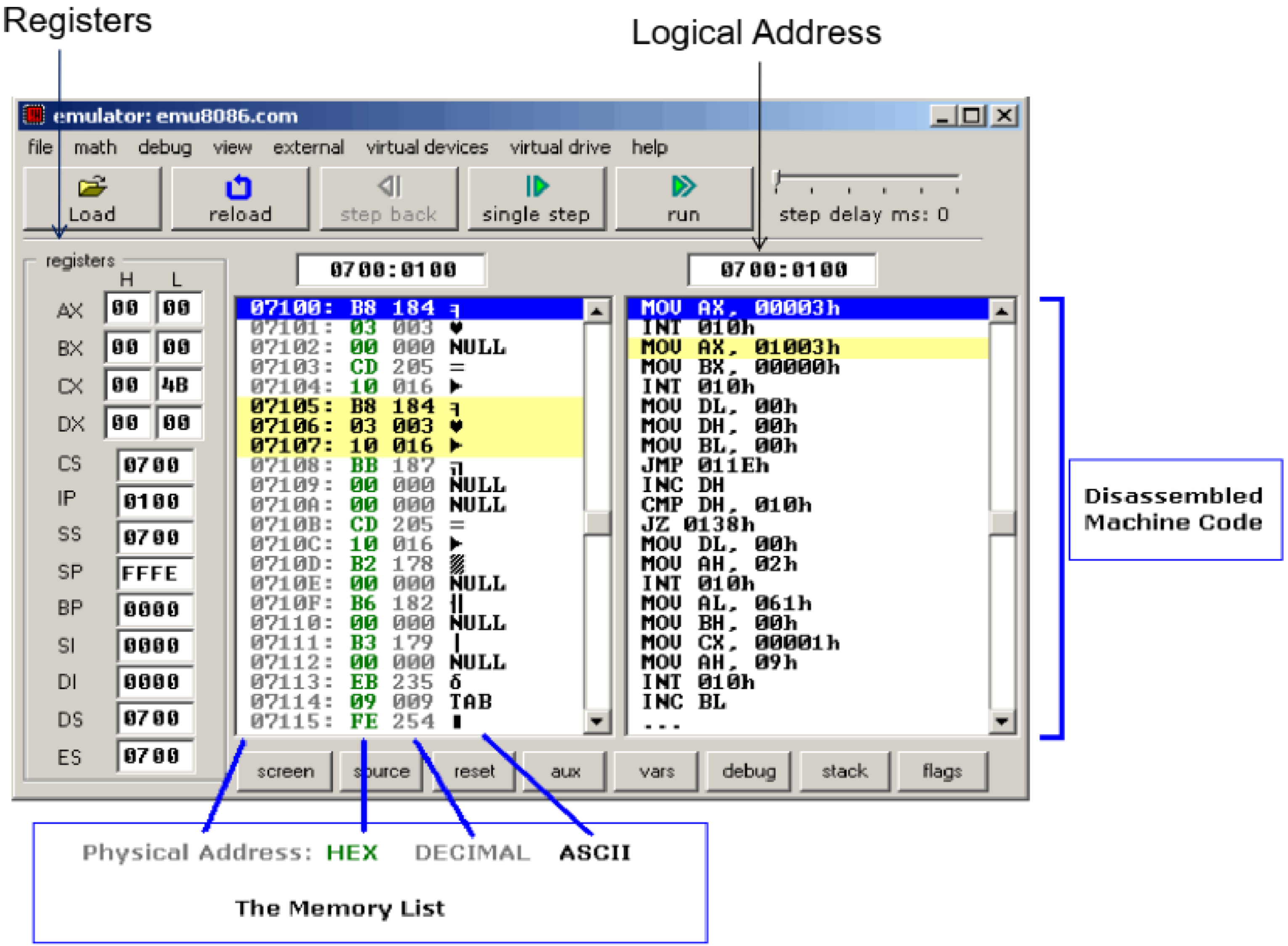
registers

	H	L
AX	00	00
BX	00	00
CX	00	61
DX	00	00
CS	0700	
IP	0100	
SS	0700	
SP	FFFE	
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	

```
0700:0100
07100: B8 184 ╕
07101: 03 003 ♥
07102: 00 000 NULL
07103: CD 205 ═
07104: 10 016 ►
07105: B8 184 ╕
07106: 03 003 ♥
07107: 10 016 ►
07108: BB 187 ╗
07109: 00 000 NULL
0710A: 00 000 NULL
0710B: CD 205 ═
0710C: 10 016 ►
0710D: B8 184 ╕
0710E: 00 000 NULL
0710F: B8 184 ╕
07110: 8E 142 Ä
07111: D8 216 ╪
07112: C6 198 ╞
07113: 06 006 ♠
07114: 02 002 ☻
07115: 00 000 NULL
```

```
0700:0100
MOV  AX, 00003h
INT  010h
MOV  AX, 01003h
MOV  BX, 00000h
INT  010h
MOV  AX, 0B800h
MOV  DS, AX
MOV  b.[00002h], 048h
MOV  b.[00004h], 065h
MOV  b.[00006h], 06Ch
MOV  b.[00008h], 06Ch
MOV  b.[0000Ah], 06Fh
MOV  b.[0000Ch], 02Ch
MOV  b.[0000Eh], 057h
MOV  b.[00010h], 06Fh
MOV  b.[00012h], 072h
...
```

```
; 5          0101        magen
; 6          0110        brown
; 7          0111        light
; 8          1000        dark
; 9          1001        light
; a          1010        light
; b          1011        light
; c          1100        light
; d          1101        light
; e          1110        yello
; f          1111        white



org 100h

; set video mode
mov ax, 3         ; text mod
```



Registers

Logical Address

0700:0100 0700:0100

	H	L
AX	00	00
BX	00	00
CX	00	4B
DX	00	00

CS	0700
IP	0100
SS	0700
SP	FFFE
BP	0000
SI	0000
DI	0000
DS	0700
ES	0700

```
07100: B8 184 ╕
07101: 03 003 ♥
07102: 00 000 NULL
07103: CD 205 ═
07104: 10 016 ►
07105: B8 184 ╕
07106: 03 003 ♥
07107: 10 016 ►
07108: BB 187 ╗
07109: 00 000 NULL
0710A: 00 000 NULL
0710B: CD 205 ═
0710C: 10 016 ►
0710D: B2 178 ▓
0710E: 00 000 NULL
0710F: B6 182 ╢
07110: 00 000 NULL
07111: B3 179 │
07112: 00 000 NULL
07113: EB 235 δ
07114: 09 009 TAB
07115: FE 254 ■
```

```
MOV AX, 00003h
INT 010h
MOV AX, 01003h
MOV BX, 00000h
INT 010h
MOV DL, 00h
MOV DH, 00h
MOV BL, 00h
JMP 011Eh
INC DH
CMP DH, 010h
JZ 0138h
MOV DL, 00h
MOV AH, 02h
INT 010h
MOV AL, 061h
MOV BH, 00h
MOV CX, 00001h
MOV AH, 09h
INT 010h
INC BL
...
```

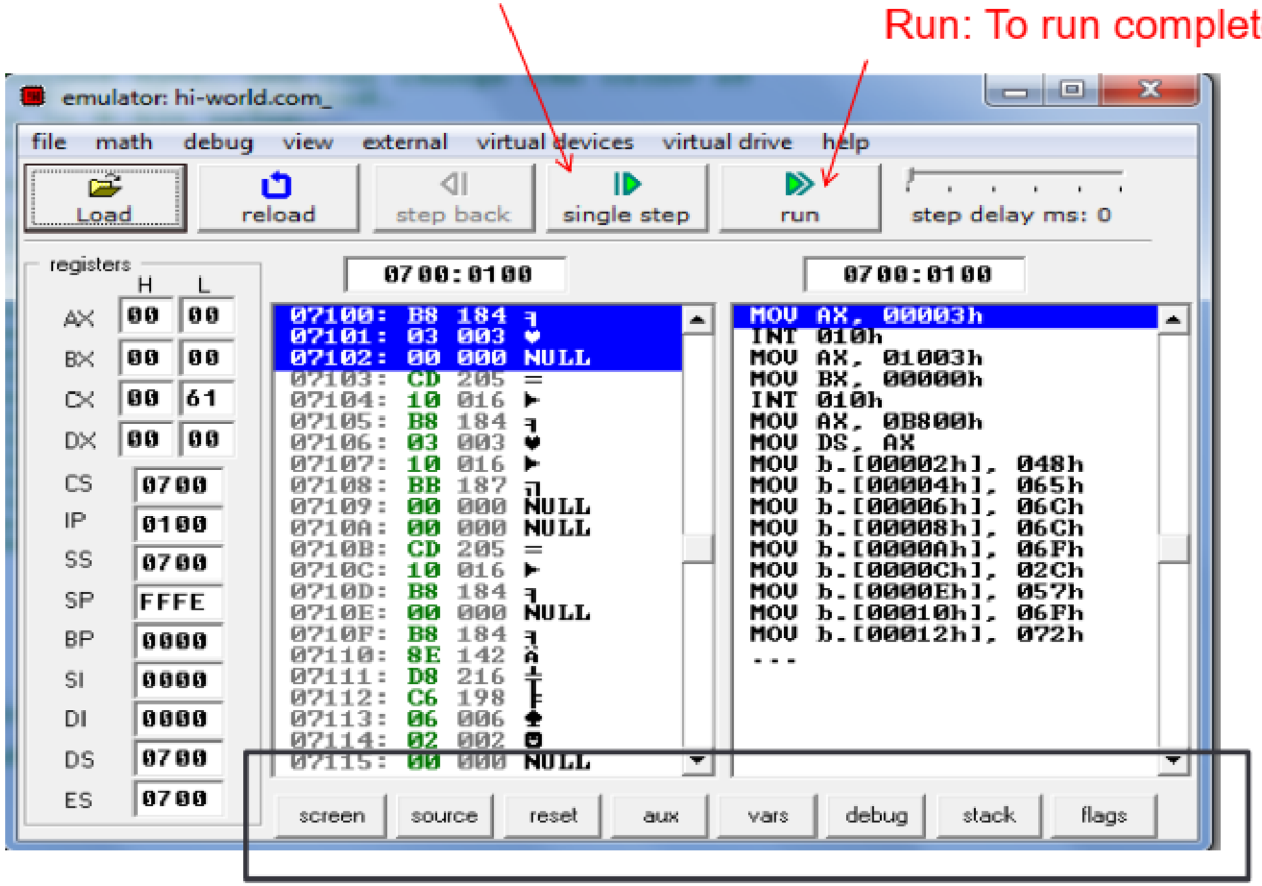
Disassembled Machine Code

Physical Address: HEX DECIMAL ASCII

The Memory List

Single Step: To execute instruction one by one i.e. stop after each instruction

Run: To run complete program



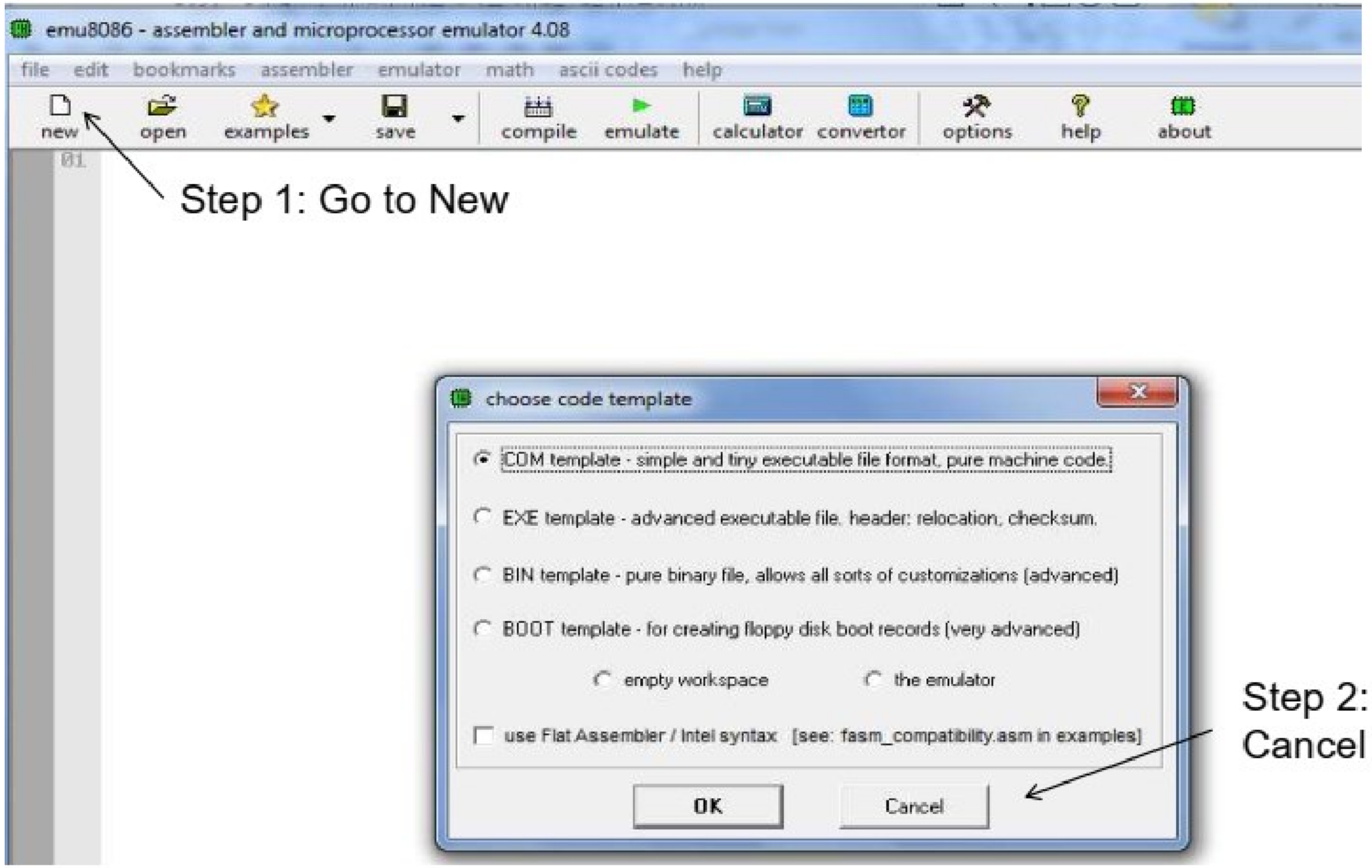
	H	L
AX	00	00
BX	00	00
CX	00	61
DX	00	00
CS	0700	
IP	0100	
SS	0700	
SP	FFFE	
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	

```
0700:0100
07100: B8 184 ╕
07101: 03 003 ♥
07102: 00 000 NULL
07103: CD 205 ═
07104: 10 016 ►
07105: B8 184 ╕
07106: 03 003 ♥
07107: 10 016 ►
07108: BB 187 ╗
07109: 00 000 NULL
0710A: 00 000 NULL
0710B: CD 205 ═
0710C: 10 016 ►
0710D: B8 184 ╕
0710E: 00 000 NULL
0710F: B8 184 ╕
07110: 8E 142 Ä
07111: D8 216 ╪
07112: C6 198 ╞
07113: 06 006 ♠
07114: 02 002 ☻
07115: 00 000 NULL
```

```
0700:0100
MOV  AX, 00003h
INT  010h
MOV  AX, 01003h
MOV  BX, 00000h
INT  010h
MOV  AX, 0B800h
MOV  DS, AX
MOV  b.[00002h], 048h
MOV  b.[00004h], 065h
MOV  b.[00006h], 06Ch
MOV  b.[00008h], 06Ch
MOV  b.[0000Ah], 06Fh
MOV  b.[0000Ch], 02Ch
MOV  b.[0000Eh], 057h
MOV  b.[00010h], 06Fh
MOV  b.[00012h], 072h
...
```

Explore what these buttons display!

How To Create a New File



● COM template - simple and tiny executable file format, pure machine code.

○ EXE template - advanced executable file. header: relocation, checksum.

○ BIN template - pure binary file, allows all sorts of customizations (advanced)

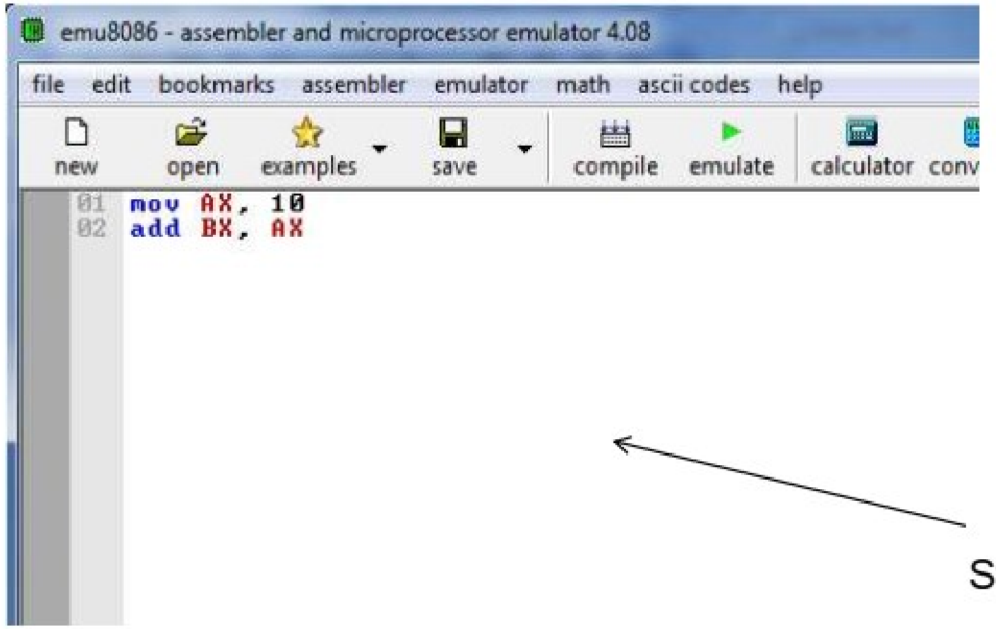
○ BOOT template - for creating floppy disk boot records (very advanced)

○ empty workspace ○ the emulator

□ use Flat Assembler / Intel syntax [see: fasm_compatibility.asm in examples]

Step 1: Go to New

Step 2: Click Cancel Button

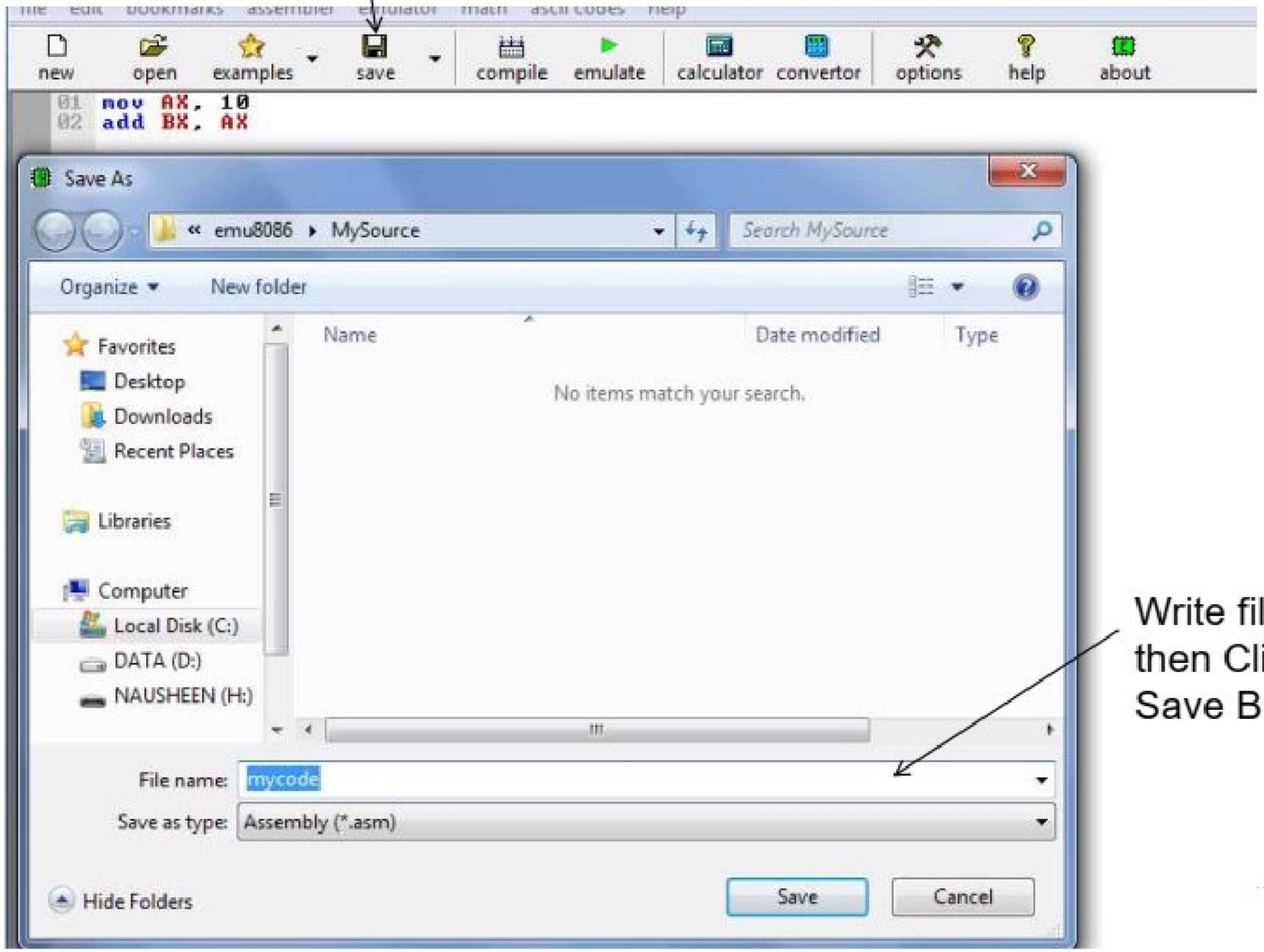


emu8086 - assembler and microprocessor emulator 4.08
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator conv

```
mov AX, 10
add BX, AX
```

Step 3: Write your code

Step 4: Save your code



```
mov AX, 10
add BX, AX
```

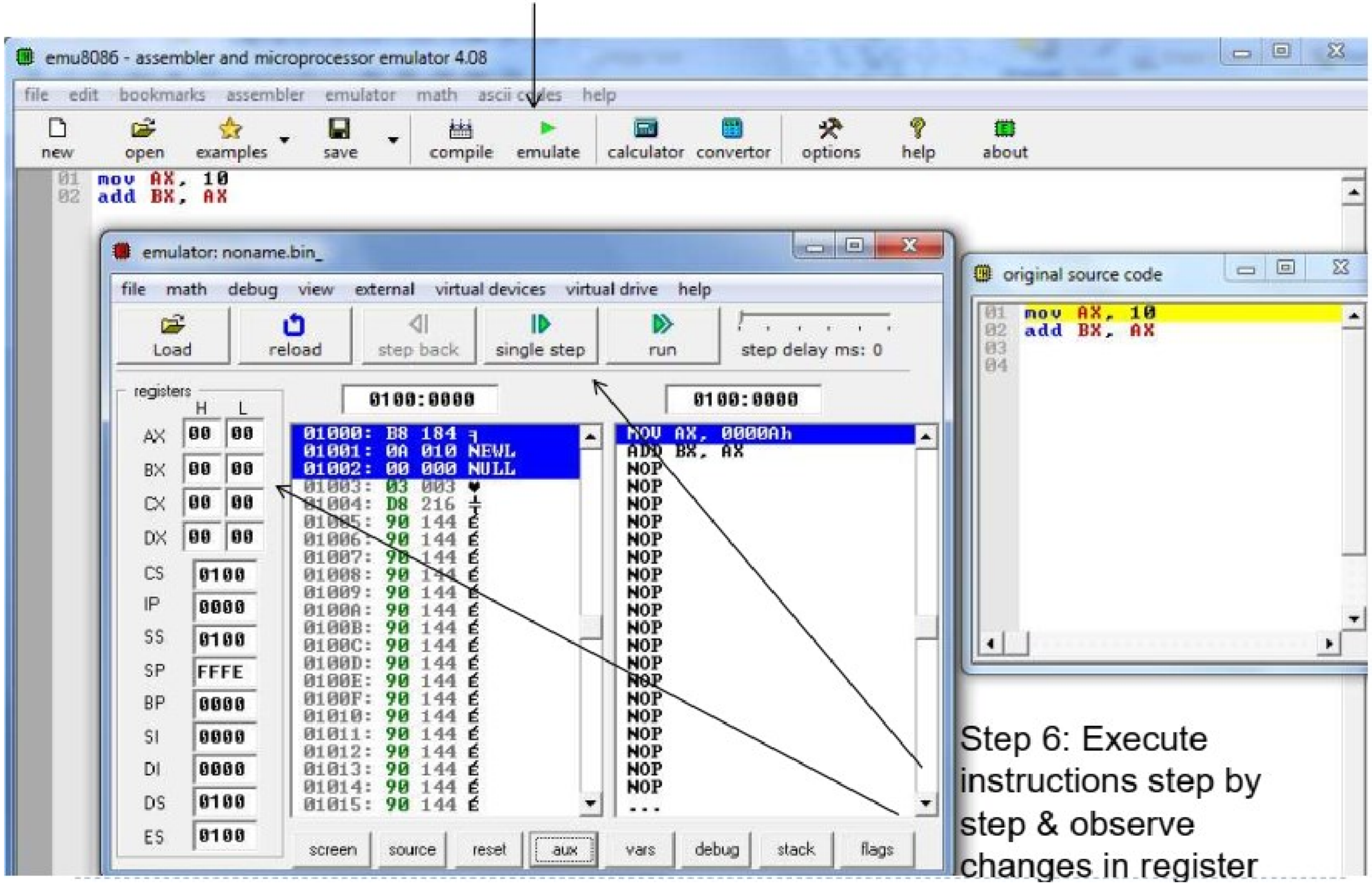
emu8086 › MySource

File name: mycode

Save as type: Assembly (*.asm)

Write filename then Click Save Button

Step 5: Emulate Program



```
mov AX, 10
add BX, AX
```

	H	L
AX	00	00
BX	00	00
CX	00	00
DX	00	00

CS	0100
IP	0000
SS	0100
SP	FFFE
BP	0000
SI	0000
DI	0000
DS	0100
ES	0100

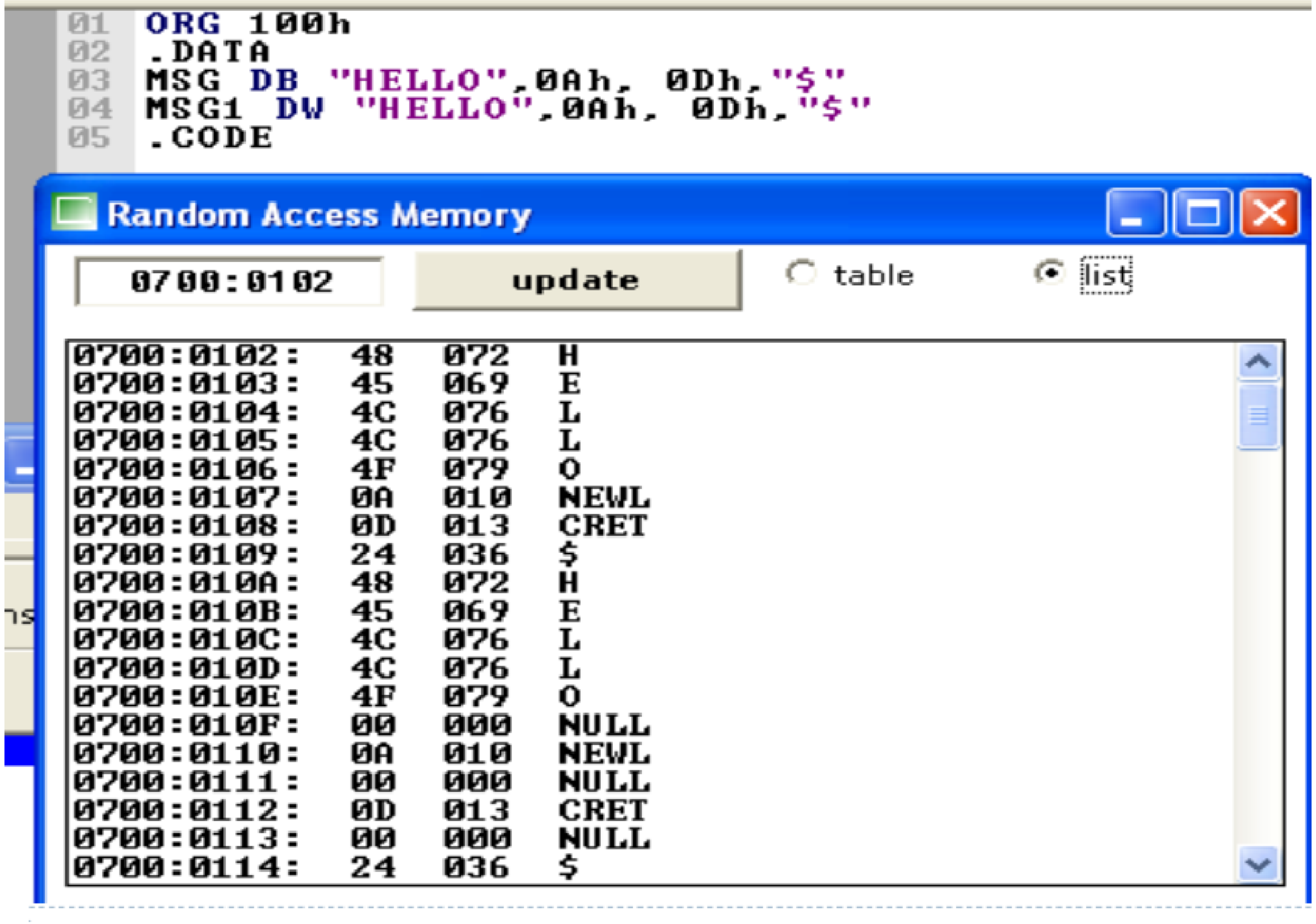
```
0100:0000

01000: B8 184 ╕
01001: 0A 010 NEWL
01002: 00 000 NULL
01003: 03 003 ♥
01004: D8 216 ╪
01005: 90 144 É
01006: 90 144 É
01007: 90 144 É
01008: 90 144 É
01009: 90 144 É
0100A: 90 144 É
0100B: 90 144 É
0100C: 90 144 É
0100D: 90 144 É
0100E: 90 144 É
0100F: 90 144 É
01010: 90 144 É
01011: 90 144 É
01012: 90 144 É
01013: 90 144 É
01014: 90 144 É
01015: 90 144 É
```

```
0100:0000

MOV AX, 0000Ah
ADD BX, AX
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
...
```

```
mov AX, 10
add BX, AX
```

Step 6: Execute instructions step by step & observe changes in register

Memory



```
ORG 100h
.DATA
MSG DB "HELLO",0Ah, 0Dh,"$"
MSG1 DW "HELLO",0Ah, 0Dh,"$"
.CODE
```

0700:0102 ○ table ● list

```
0700:0102:   48   072   H
0700:0103:   45   069   E
0700:0104:   4C   076   L
0700:0105:   4C   076   L
0700:0106:   4F   079   O
0700:0107:   0A   010   NEWL
0700:0108:   0D   013   CRET
0700:0109:   24   036   $
0700:010A:   48   072   H
0700:010B:   45   069   E
0700:010C:   4C   076   L
0700:010D:   4C   076   L
0700:010E:   4F   079   O
0700:010F:   00   000   NULL
0700:0110:   0A   010   NEWL
0700:0111:   00   000   NULL
0700:0112:   0D   013   CRET
0700:0113:   00   000   NULL
0700:0114:   24   036   $
```

Recall

Signed number representation: Ranges, Overflow

Number Representation

- An ***integer*** is a number which has no fractional part.
- Numbers can be represented as a combination of
  - Sign (plus or minus)
  - Value or magnitude

Unsigned Integer
(Natural Number)

- 8-bit storage location
  - 2^8 different values between 0 and 255
- 16-bit storage location
  - 2^{16} different values between 0 and 65535
- multiple storage locations
  - 4 consecutive 1-byte storage locations
  - provide 32 bits of range
  - 2^{32}, or 4,294,967,296 different values
  - difficult to calculate and manipulate

Signed-Integer Representation

- No obvious direct way to represent the sign in binary notation
- Options:
  - Sign-and-magnitude representation
  - 1’s complement (skip – confusing)
  - 2’s complement (most common)

Sign-and-Magnitude

- Use left-most bit for sign
  - 0 = plus; 1 = minus
- Total range of integers the same
  - Half of integers positive; half negative
  - Magnitude of largest integer half as large
- Example using 8 bits:
  - Unsigned: 1111 1111 = +255
  - Signed: 0111 1111 = +127
    1111 1111 = -127
  - Note: 2 values for 0:
    +0 (0000 0000) and -0 (1000 0000)

Calculation Algorithms

- Sign-and-magnitude algorithms complex and difficult to implement in hardware
  - Must test for 2 values of 0
  - Useful with BCD
  - Order of signed number and carry/borrow makes a difference
- Example: Decimal addition algorithm

Addition: 2 Positive Numbers	Addition: 1 Signed Number		
4 +2 6	4 - 2 2	2 - 4 -2	12 - 4 8

Ranges

No. of bits	Binary			
	Unsigned		Sign-magnitude	
	Min	Max	Min	Max
1	0	1		
2	0	3	-1	1
3	0	7	-3	3
4	0	15	-7	7
5	0	31	-15	15
6	0	63	-31	31
Etc.				

Ranges: General Rule

No. of bits	Binary			
	Unsigned		Sign-magnitude	
	Min	Max	Min	Max
n	0	$2^n - 1$	$-(2^{n-1} - 1)$	$2^{n-1} - 1$

2’s Complement in Binary System

- 2’s complement representation:
  - Positive value represents itself
  - Negative value: invert and add “1”

Numbers	Negative		Positive	
Representation method	Complement		Number itself	
Range of decimal numbers	-128_{10}	-1_{10}	$+0_{10}$	127_{10}
Calculation	Inversion		None	
Representation example	10000000	11111111	00000000	01111111

Example: 2’s Complement

- Represent –5 in binary using 2’s complement notation
  1. Decide on the number of bits: `6 (for example)`
  2. Find the binary representation of the +ve value in 6 bits `000101`

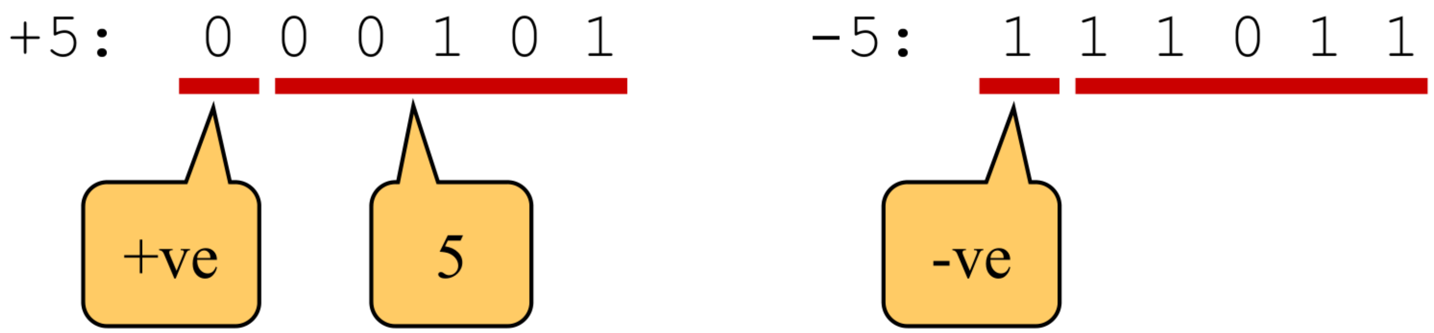
     +5
  3. Invert all the bits: `111010`
  4. Add 1

```
  111010
+      1
--------
  111011
```

-5

Sign Bit in 2's Complement

- In 2's complement representation, the MSB is the sign bit (as with sign-magnitude notation)
  - 0 = positive value
  - 1 = negative value



+5: 0 0 0 1 0 1

-5: 1 1 1 0 1 1

+ve

5

-ve

Estimating Integer Value of 2's Complement Representation

- Positive numbers begin with 0
- Small negative numbers (close to 0) begin with multiple 1's
  - 1111 1110 = -2 in 8-bit 2's complements
  - 1000 0000 = -128, largest negative 2's complements
  - Invert all 1's and 0's, add "1" and approximate the value

Exercise: 2's complement conversions

- What is -20 expressed as an 8-bit binary number in 2's complement representation?
  - Answer:___________

- 1100011 is a 7-bit binary number in 2's complement representation. What is the decimal sign-and-magnitude value?
  - Answer:___________

Exercise: 2's complement conversions

Answer

- What is -20 expressed as an 8-bit binary number in 2's complement notation?
  - Answer: _ 11101100 _

- 1100011 is a 7-bit binary number in 2's complement notation. What is the decimal value?
  - Answer:___ -29 ______

Detail for -20 -> 11101100

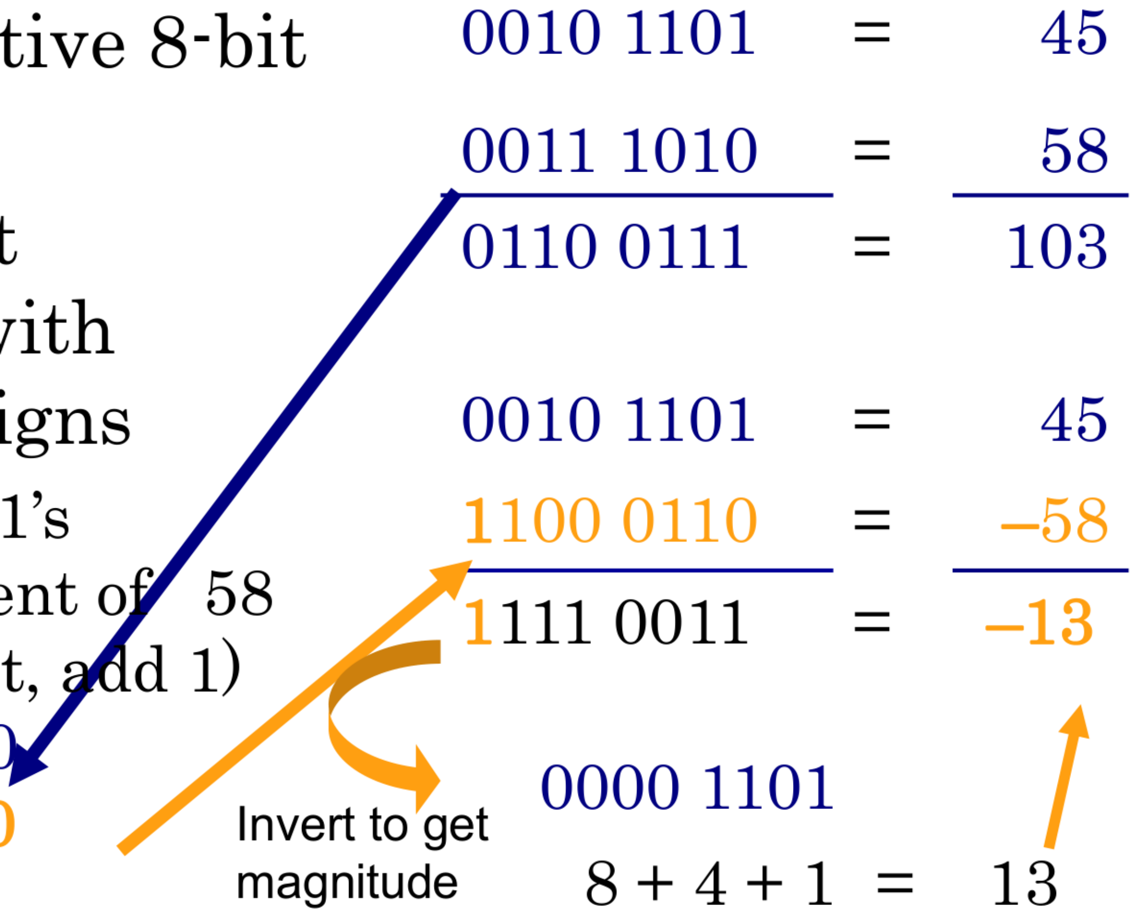
$\mathbf{-20_{10}}$: Positive Value =	**00010100**
Invert:	**11101011**
Add 1:	+ 1
	11101100

Detail for 1100011 -> - 29

2's Complement Rep:		1100011
Invert:		0011100
Add One:	+	1
		0011101
Converts to:	=	- 29

Arithmetic in 2's Complement

- Add 2 positive 8-bit numbers
- Add 2 8-bit numbers with different signs
  - Take the 1's complement of 58 (i.e. invert, add 1)
    0011 1010
    1100 0110



0010 1101 = 45
0011 1010 = 58
0110 0111 = 103

0010 1101 = 45
1100 0110 = –58
1111 0011 = –13

Invert to get magnitude

0000 1101

8 + 4 + 1 = 13

Addition with Carry in 2's Complement

- 8-bit number
  - Invert (add 1)
    0000 0010 (2_{10})
    1111 1110
  - Add
  - drop final carry out

$$
\begin{array}{rcr}
0110\ 1010 & = & 106 \\
1111\ 1110 & = & -2 \\
\hline
1\,0110\ 1000 & = & 104 \\
\text{(drop 1)} & & \\
\hline
0110\ 1000 & &
\end{array}
$$

Subtraction

- 8-bit number
  - Invert (add 1)
    0101 1010 (90_{10})
    1010 0110
  - Add
  - drop final carry out

0110 1010 = 106

-0101 1010 = 90

0110 1010 = 106

–1010 0110 = 90

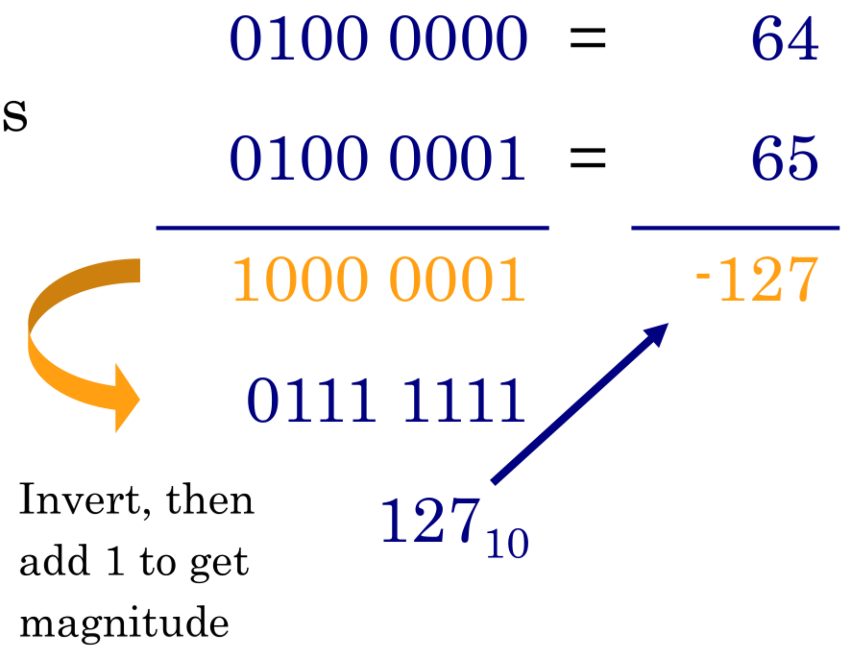
10001 0000

(drop 1)

0001 0000 = 16

Overflow

- 8-bit number
  - 256 different numbers
  - Positive numbers: 0 to 127
- Add
  - Test for *overflow*
  - 2 positive inputs produced negative result → *overflow!*
  - Wrong answer!
- Programmers beware: some high-level languages, e.g., some versions of BASIC, do not check for overflow adequately



0100 0000 = 64

0100 0001 = 65

1000 0001 -127

0111 1111

Invert, then add 1 to get magnitude

127_{10}

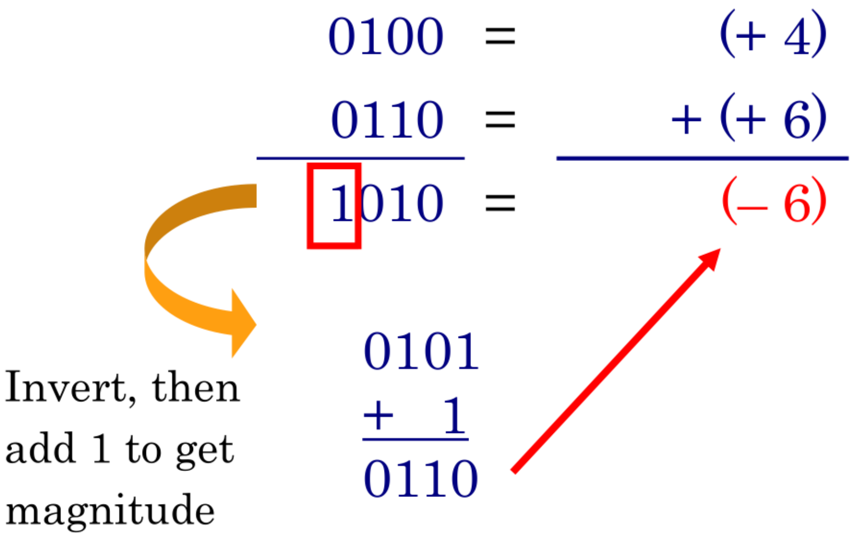
Overflow and Carry Conditions

- *Carry flag*: set when the result of an addition or subtraction exceeds fixed number of bits allocated
- *Overflow*: result of addition or subtraction overflows into the sign bit

Overflow/Carry Examples

- Example 1:
  - Correct result
  - No overflow, no carry
- Example 2:
  - Incorrect result
  - Overflow, no carry

$$\begin{array}{rcr} 0100 & = & (+\ 4) \\ \underline{0010} & = & \underline{+\ (+\ 2)} \\ 0110 & = & (+\ 6) \end{array}$$



$$\begin{array}{rcr} 0100 & = & (+\ 4) \\ \underline{0110} & = & \underline{+\ (+\ 6)} \\ \boxed{1}010 & = & (-\ 6) \end{array}$$

Invert, then add 1 to get magnitude

$$\begin{array}{r} 0101 \\ \underline{+\quad 1} \\ 0110 \end{array}$$

Overflow/Carry Examples

- Example 3:
  - Result correct ignoring the carry
  - Carry but no overflow

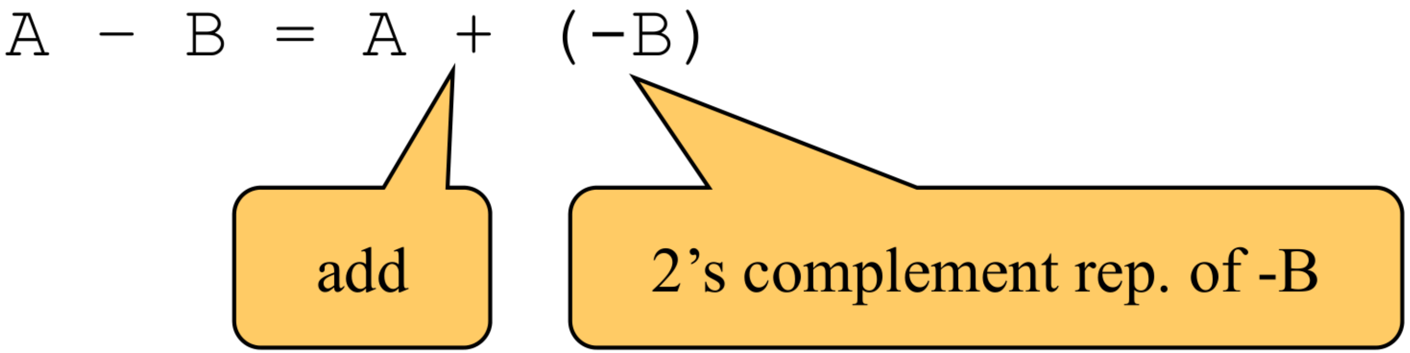
$$\begin{array}{rcr} 1100 & = & (-4) \\ 1110 & = & +(-2) \\ \hline 11010 & = & (-6) \end{array}$$

- Example 4:
  - Incorrect result
  - Overflow, carry ignored

$$\begin{array}{rcr} 1100 & = & (-4) \\ 1010 & = & +(-6) \\ \hline 10110 & = & (+3) \end{array}$$

2's Complement Subtraction

- Just add the opposite value!



```
A - B = A + (-B)
```

add

2's complement rep. of -B

Thank you