Yintaru 8-bit Microprocessor

- Direct Addressing Capability 128 Bit of Memory
- Range of Clock Rates:
  3.7962×10^{-2} Hz for 8086
- Architecture Designed for Powerful Assembly Language and Efficient High-Level Languages
- Available in EXPRESS
  - Standard Temperature Range
  - Extended Temperature Range
- Digital & Analog IO
- 8 and 16-Bit Signed and Unsigned Arithmetic in Binary or Decimal

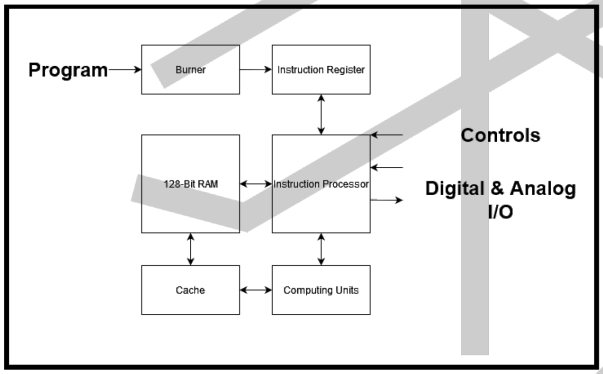


Figure 1. 8086 CPU Block Diagram

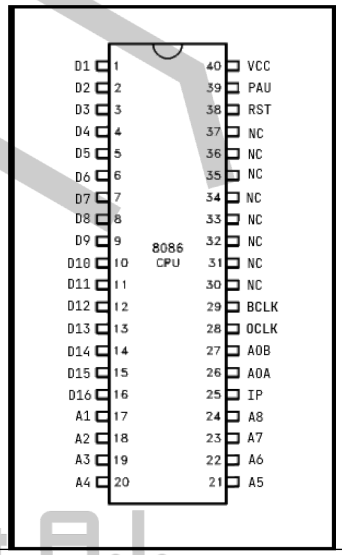


Figure 2. 8086 Pin Configuration

1. Program: Manual compiled shulker box input
2. Controls:
   a) IP
   b) BCLK, OCLK
   c) RST
   d) PAU
   e) VCC
3. Digital & Analog I/O:
   a) D1~D16
   b) A1~A8
   c) AOA, AOB
   d) IP

Table 1. Pin Description

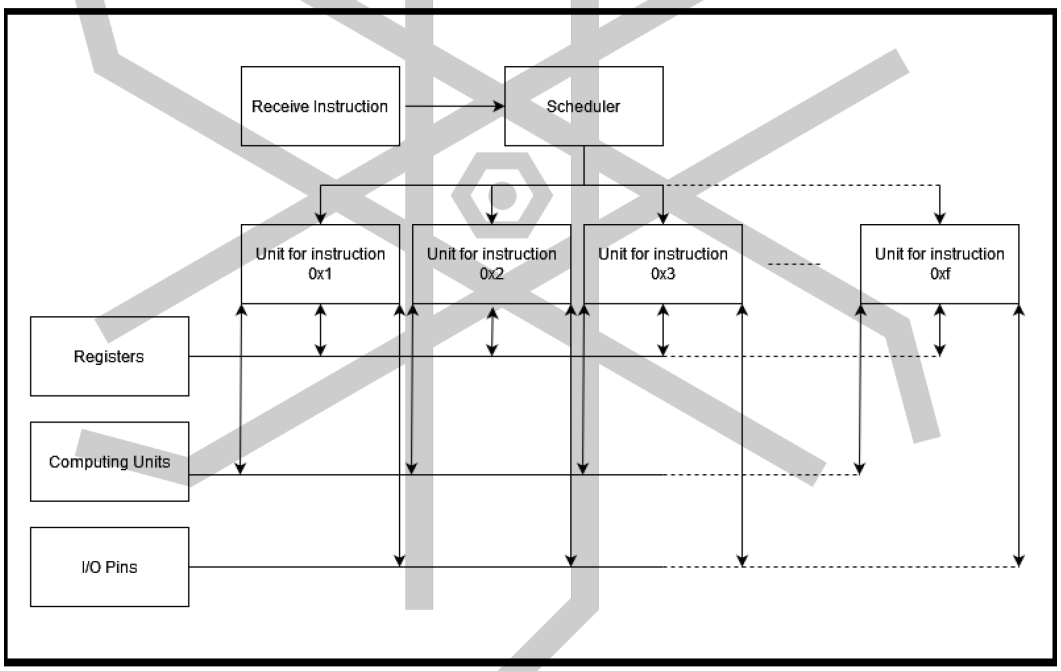
Symbol	Pin No.	Type	Name and Function	notes
D1~D16	1~16	O	**Data digital output :** These lines fix and output the data when executed instruction **"prt(.)"** 2(0010)**.** Old data will be flushed or cleared when executed a new prt instruction, when program starting or system will be reset.	
A1-A8	17~24	O	**Data analog output :** Unconditional output all the register from No1 to No8, only output value between 0~15, for bigger value than 15 or smaller value than 0, only the lower 4 bit will be output.	
IP	25	I	**Data analog input :** Send analog data inputed to the targeting register when executed instruction "**get(,)**" 1(0001)	
AOA	26	O	**EAX Data analog output :** Unconditional output eax data, ouly output value between 0~15, for bigger value than 15 or smaller value than 0, only the lower 4 bit will be output.	**Not installed yet**
AOB	27	O	**EBX Data analog output :** Unconditional output ebx data, ouly output value between 0~15, for bigger value than 15 or smaller value than 0, only the lower 4 bit will be output.	**Not installed yet**
OCLK	28	I	**External clock input :** Connect with external clock when need to overwrite internal clock. The external clock can only slower than internal clock in order to avoid hardware damage caused by hardware read&write conflect.	
BCLK	29	I	**External clock enabled input:** Use external clock input to overwrite internal clock. Set this to true to use external clock.	
RST	38	I	**Reset :** Reset the microprocessor when this line is true, when this line is true, the microprocessor will be unable to execute any instruction, ONLY work when VCC is false for at least two clock-tick.	
PAU	39	I	**Pause :** Pause the internal clock in order to pause the processing progress, ONLY work when using internal clock.	
VCC	40	I	**Power supply :** When this line is set to true, microprocessor will auto initalize and start executing the instructions. If not, the microprocessor will stop at once and all the data in registers will lose.	
NC	29~37	N/A	**No function**	

FUNCTIONAL DESCRIPTION

Processor architecture

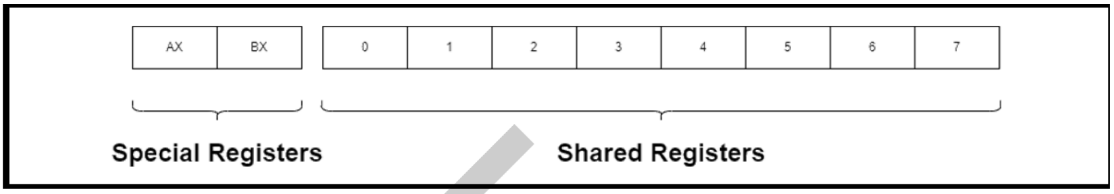
The internal functions of the 8086 processor are processed by the main processing unit, which in charge of processing all the instructions and communicating with register units and caches.

These units can interact directly in theory but the main processing unit was designed with 16 single units which responsible for processing only one kind of instruction, and when the main processor received a instruction, it will be described by the to the unit that in charge of that instruction.



Registers organization

There are 8 shared registers, an eax register and an ebx register in the microprocessor.



Shared Registers

8 shared registers are directly operational for general programming, a pointer is used to control the targeting shared register.

At first the pointer points to register #0, after instruction “nxt(>)[7(0110)]” executed, the pointer points to next shared register.(e.g. at register #0, use nxt instruction, the pointer was added 1 and points to register #1).

And after instruction “pre(<)[8(1000)]” executed, the pointer points to previous shared register.(e.g. at register #4, use pre instruction, the pointer was subtracted 1 and points to register #3).

Attention, if you attempt use nxt instruction when the pointer points to 7 or use pre instruction when the pointer points to 0, the pointer will not change.

Special Registers

Two registers called AX and BX are special registers. It can’t be directly operational by general instructions, but it can be copied as or copied to shared registers.

These two registers will directly take part in “add(j)[11(1011)]”, “sub(s)[12(1100)]” and “jnz(])[15(1111)]” instruction.

You can copy shared registers to special registers by using instruction “mva(a)[9(1011)]” to move targeting shared register to AX or “mvb(b)[10(1010)]” to move targeting shared register to BX, and copy value back to targeting shared register from AX by using instruction “cpy(p)[6(0110)]”.

Table 2. Instruction Set Summary

HEX	BIN	ASM	EXT	ABB	MEAN	EQS
0	0000	ret	return	;	Program execution ended	/
1	0001	get	get	,	Get the value from input pin "IP"	/
2	0010	prt	print	.	Digital print the targeting shared register value to output D1-D16	/
3	0011	inc	increase	+	Increase the targeting shared register	R[i]=R[i]+1
4	0100	dec	decrease	-	Decrease the targeting shared register	R[i]=R[i]-1
5	0101	clr	clear	c	Set the targeting shared register to 0	R[i]=0
6	0110	cpy	copy	p	Copy value from EAX to targeting shared register	R[i]=EAX
7	0111	nxt	next	>	Switch to next shared register	i=i+1
8	1000	pre	previous	<	Switch to previous shared register	i=i-1
9	1001	mva	move to eax	a	Move value from targeting shared register to EAX	EAX=R[i]
a	1010	mvb	move to ebx	b	Move value from targeting shared register to EBX	EBX=R[i]
b	1011	add	add	j	Add EAX and EBX, storage result to targeting register	R[i]=EAX+EBX
c	1100	sub	subtract	s	Subtrack EAX and EBX, storage result to targeting register	R[i]=EAX-EBX
d	1101	tag	tag	[	Set here a tag, overwrite previous one(If exist)	do while{
e	1110	jmp	jump	}	Unconditional jump back to the tag previously set	}(true)
f	1111	jnz	jump if not zero	]	Jump back to the tag previously set if current shared register is not 0	}(EAX!=0)

Instructions

The instructions are the most important part. All the programming work basically use the following instructions, and they constructs the whole assemble language of the 8086 MCU.

0000-0-ret

"ret" : return.

"ret" is an instruction which used to stop the whole program, it can be exist anywhere in the program, but there must be one in the end of the program. So in someway you can understand the maxium size of the instruction register is not 64, it's 63 because there's always a "ret" at the end.

When "ret" is executed, there are only two ways to restart the program: send a pulse to <RST> pin or stop the power supply to VCC for at least two clock tick. **We recommend you use the second way in order to avoid any potential bugs caused by design defect.**

0001-1-get

"get" : get

"get" is an instruction which used to input an analog value from pin <IP>, it can only input value between 0 to 15.

When "get" is executed, the targeting shared register will be overwrite by the new value from <IP> pin.

0010-2-prt

"prt" : print

"prt" is an instruction which used to output the value of targeting shared register.

When "prt" is executed, the targeting shared register will be output at the output pins <D1~D16> and keep fixed.

The output pin will NOT be reset on restarting. This means you must reset it yourself by the following code if necessary.

```
clr
prt
```

0011-3-inc

"inc" : increase

"inc" is an instruction which used to increase the value of targeting shared register by 1.

When "inc" in executed, the targeting shared register will be increased by 1 and overwrite the old value.

0100-4-dec

"dec" : decrease

"dec" is an instruction which used to decrease the value of targeting shared register by 1.

When "dec" in executed, the targeting shared register will be decreased by 1 and overwrite the old value.

0101-5-clr

"clr" : clear

"clr" is an instruction which used to clear the targeting shared register.

When "clr" is executed, the targeting shared register will be overwritten with 0.

0110-6-cpy

"cpy" : copy

"cpy" is an instruction which used to copy the value from AX to targeting shared register.

When "cpy" is executed, the targeting shared register will be overwritten with the value of AX.

0111-7-nxt

"nxt" : next

"nxt" is an instruction which used to switch to next shared register.

When "nxt" is executed, it will select next shared register.

If next shared register does not exist(it's already the last shared register), the MCU will work without saving or loading data from any registers, NEVER make this happen or you will have to restart the program to get a recovery. (This is an undefined operation).

1000-8-pre

"pre" : previous

"pre" is an instruction which used to switch to previous shared register.

When "pre" is executed, it will select previous shared register.

If previous shared register does not exist(it's already the first shared register), the address will not be changed, the targeting register will be kept as Number 0 register, NEVER use this characteristics in programming (This is an undefined operation).

1001-9-mva

“mva” : move to AX

“mva” is an instruction which used to copy the value from targeting shared register to AX.

When “mva” is executed, the value of AX will be overwritten with the value of targeting shared register.

Combine 1001-9-mva with 0110-6-cpy when programming can make you move value between shared register quickly.

1010-a-mvb

“mvb” : move to BX

“mvb” is an instruction which used to copy the value from targeting shared register to BX.

When “mvb” is executed, the value of BX will be overwritten with the value of targeting shared register.

1011-b-add

“add” : add

“add” is an instruction which used to add two values together.

When “add” is executed, the targeting shared register will be overwritten with the sum of AX and BX. (AX+BX)

1100-c-sub

“sub” : subtract

“sub” is an instruction used to subtract two values together.

When “sub” is executed, the targeting shared register will be overwritten with the difference of AX and BX. (AX-BX)

1101-d-tag

“tag” : tag

“tag” is an instruction which used to temporary save current instruction register address and prepare to use in “jmp” or “jnz”

When “tag” is executed, the instruction register address will be saved.

The initial position of tag is not specified, DO NOT jmp or jnz before any “tag” was executed in this program!

1110-e-jmp

"jmp" : jump

"jmp" is an instruction which used to jump to last tag address.

When "jmp" is executed, the program will jump to last position tagged and continue processing.

The initial position of tag is not specified, DO NOT jmp or jnz before any "tag" was executed in this program!

1111-f-jnz

"jnz" : jump if not zero

"jmz" is an instruction which used to jump to last tag address when the value of targeting shared register is not zero.

When "jnz" is executed, if the value of targeting shared register is not zero, the program will jump to last position tagged and continue processing.

The initial position of tag is not specified, DO NOT jmp or jnz before any "tag" was executed in this program!

Pins

The pins are the most basic way for the MCU communicating with other devices in the world.

D1~D16

D1~D16 are used to output data. See instruction 0010-2-prt for more info.

A1~A8

A1~A8 unconditionally outputs 8 shared registers with analog signal.

IP

IP is used to get analog data input from outside. See instruction 0001-1-get for more info.

OCLK

OCLK is used to send custom clock signal to the MCU. See the clock section for more info.

BCLK

BCLK is used to switch between custom clock signal and built-in clock signal. When this is high level, the MCU will work according to custom clock signal.

DO NOT switch this when the MCU is working!

RST

RST is used to reset the MCU.

Deprecated function, just switch VCC off, wait at least two clock cycle and switch VCC on again instead.

PAU

Pause the built-in clock.

Only work when using the internal clock.

VCC

Power supply:

High level: on,

Low level: off.

Clock cycle

The clock system is the heart of 8086 MCU.

Internal Clock System

The internal clock system, AKA built-in clock system, provides the most basic, stable and fast clock timing cycle to the 8086 MCU. There are also some function designed for it.

When Pin <BCLK> is at low level(default), the 8086 MCU will follow the internal clock system, the main frequency of internal clock system is 3.7962×10^{-2} Hz.

When Pin <BCLK> is at low level(default), the Pin <PAU> (pause function) will be available. When Pin <PAU> is at high level, the internal clock will be paused and the 8086 MCU will stop computing any instruction after completed current instruction. After the whole system being paused, it will be safe to unload the chunks of the 8086 MCU.

External Clock System

The external clock system, AKA custom clock system, is fully configurable by user, but there are still many things you need to pay attention to use external clock system.

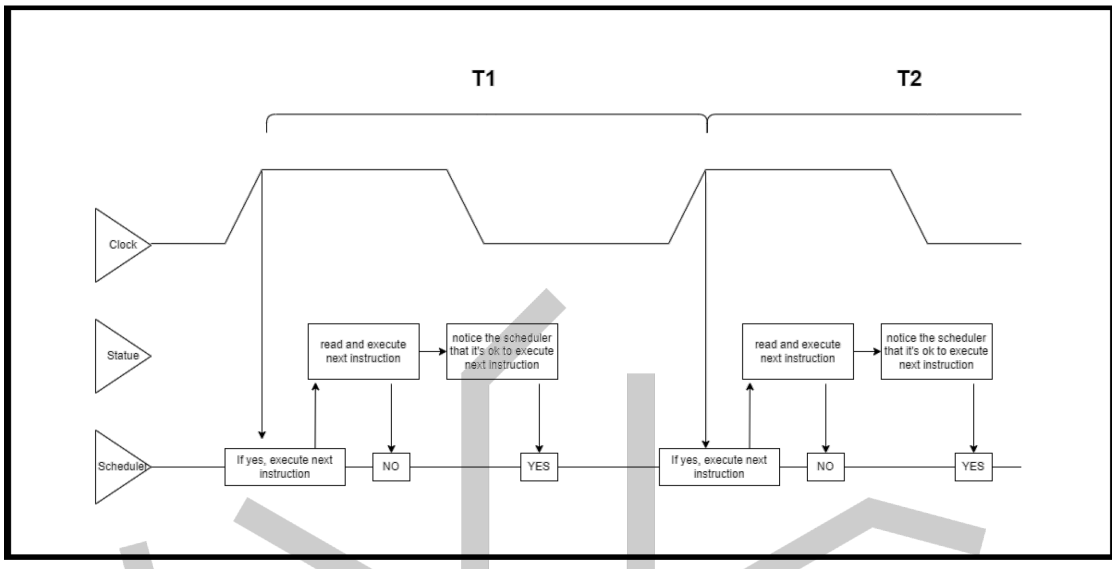
When Pin <BCLK> is at high level, the 8086 MCU will follow the external clock system, the main frequency of external clock system will be fully configurable by user, only the following regulations need to be followed, then it’s safe for switching to external clock system:

1. The frequency of external clock system MUST equals or slower than internal clock system.
2. The duty cycle of the clock should be at 50%.
3. The clock signal must not exhibit noise

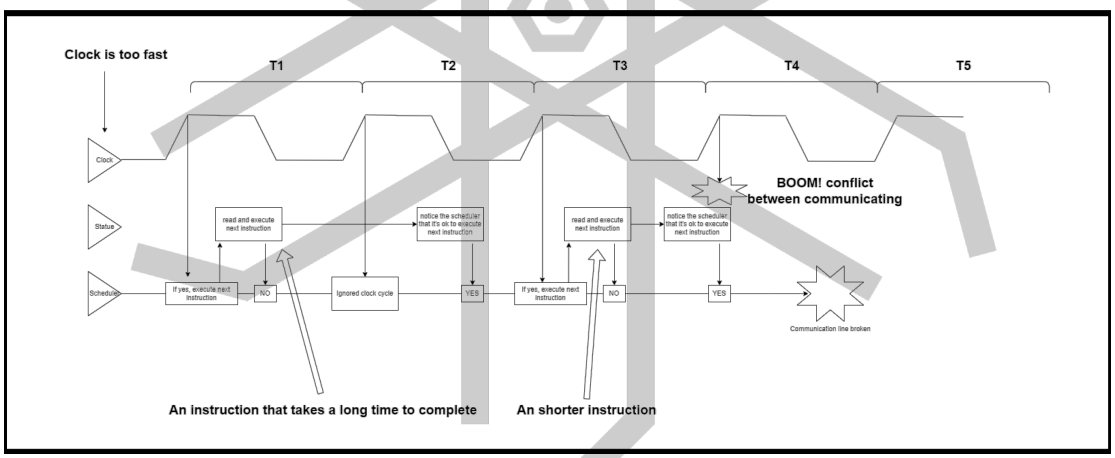
>1: If the clock is faster then internal system(the calculated maxium frequency that 8086 MCU can handle) , the 8086 MCU will work under an unknown integral multiple each instruction, and it’s completely possible to cause some permanently destruction of the main construction of 8086 MCU caused by chaotic or excessively fast clock signals.

>2: If the duty cycle isn’t 50%, the chaotic clock signals which Interval occurs will possibly destruct the internal construction of the 8086 MCU.

>3: The chaotic clock signals will cause destruction.



The relationship between clock cycle and program executing(normal)



The conflict between too fast clock cycle and normal program executing(abnormal)