Yintaru 8-bit Microprocessor

- Direct Addressing Capability 128 Bit of Memory
- Range of Clock Rates:
  $60 \times 2.7 \times 10^{-4}$ 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

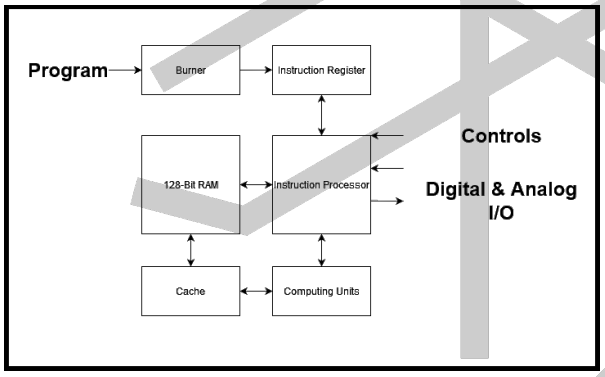


Figure 1. 8086 CPU Block Diagram

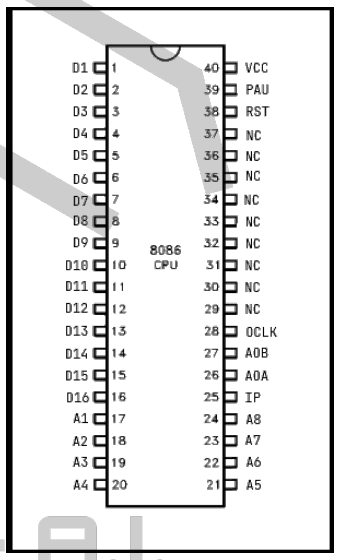


Figure 2. 8086 Pin Configuration

1. Program: Manual compiled shulker box input
2. Controls:
   a) IP
   b) 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

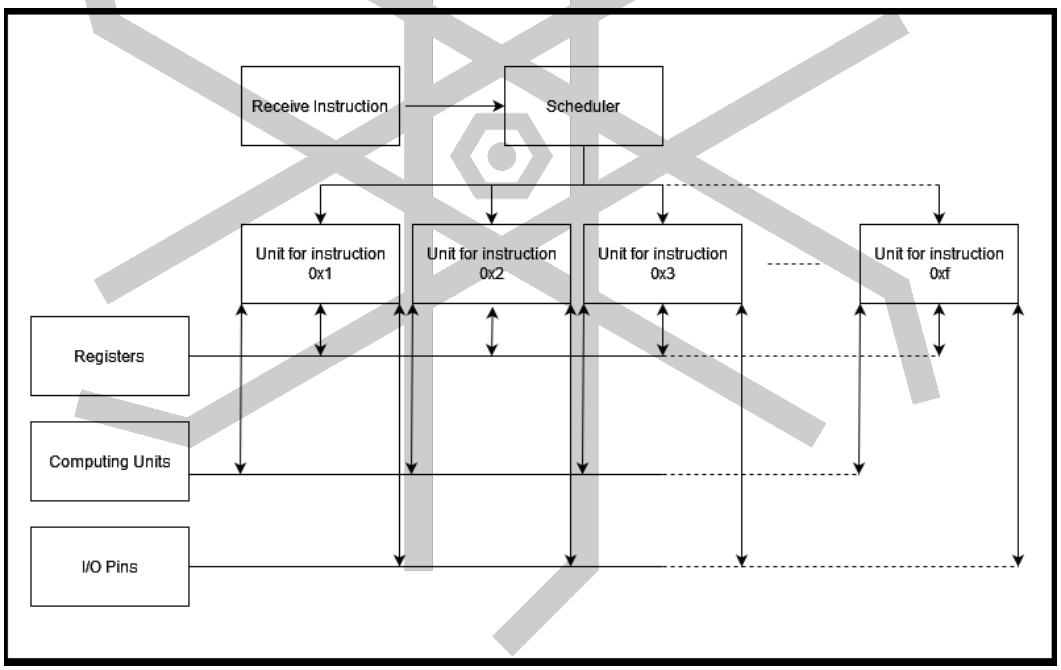
Symbol	Pin No.	Type	Name and Function
D1~D16	1~16	O	**Data digital output :** These lines output the data when executed instruction **"prt(.)"** 2(0010). Old data will be flushed or cleared when executed a new prt instruction, when program starting or system reset.
A1-A8	17~24	O	**Data analog output :** Unconditional output all the register from No1 to No8, only output value between 0~15, bigger value than 15 or smaller value than 0 will be output as 15.
IP	25	I	**Data analog input :** Send analog data inputed to the specified register when executed instruction "**get(,)**" 1(0001)
AOA	26	O	**EAX Data analog output :** Unconditional output eax data, ouly output value between 0~15, biggerf value than 15 or smaller value than 0 will be output as 15.
AOB	27	O	**EBX Data analog output :** Unconditional output ebx data, ouly output value between 0~15, biggerf value than 15 or smaller value than 0 will be output as 15.
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.
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.
PAU	39	I	**Pause :** Pause the internal clock in order to pause the processing progress.
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	N/A

FUNCTIONAL DESCRIPTION

Processor architecture

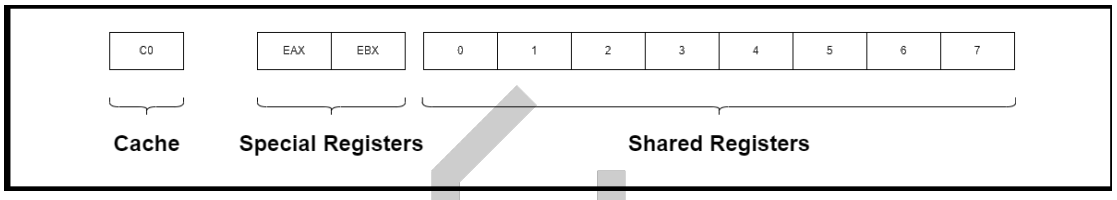
The internal functions of the 8086 processor are processed by the main processing unit, which in charge of processing all the instructions and communicating with register units and caches.

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 subtraced 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 EAX and EBX 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 EAX or “mvb(b)[10(1010)]”, and copy value back to targeting shared register from EAX by using instruction “cpy(p)[6(0110)]”.

Cache

Cache (register C0) is used for shared register calculations. Actually, when you use instruction “inc(+)[3(0100)]” or “dec(-)[4(0100)]”, the targeting register was copied to cache at first, then use the adder or the subtractor calculate, and the finally copied back to the targeting register. This is just an example, nearly all the instructions require to copy the data from shared register to cache before executing. Luckily, It’s automatically managed by system, you don’t need to think it too much.

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]
10	1010	mvb	move to ebx	b	Move value from targeting shared register to EBX	EBX=R[i]
11	1011	add	add	j	Add EAX and EBX, storage result to targeting register	R[i]=EAX+EBX
12	1100	sub	subtract	s	Subtrack EAX and EBX, storage result to targeting register	R[i]=EAX-EBX
13	1101	tag	tag	[	Set here a tag, overwrite previous one(If exist)	do while{
14	1110	jmp	jump	}	Unconditional jump back to the tag previously set	}(true)
15	1111	jnz	jump if not zero	]	Jump back to the tag previously set if EAX is not 0	}(EAX!=0)