پروژه درس معماری کامپیوتر

نیم سال اول 99-00

عنوان پروژه : طراحی پردازنده

استاد درس : دکتر حمید سربازی آزاد

اعضای تیم :

ساعی سعادت , مهدی اصمع , علیرضا حسین پور , محمد مهدی به نصر

ـ مقدمه

به صورت کلی پردازنده طراحی شده از بعضی جهات بسیار شبیه به به پردازنده mips میباشید که از نظر طول کلمه معماری تعداد ثبات عمومی یکسان و ازنظر نوع دستورات تفاوت هایی دارند هر دو نوع میشود گفت که از نوع معماری RISK میباشند .

نمایش اعداد به صورت مکمل دو میباشد .

نمایش کلی پروژه همراه با تمام جزعیات داخل گیت میباشد

-پردازنده

همانطور که میدانید هر پردازنده از قطعات مخلتفی تشکیل شده که تجمیع آن ها به کمک یک data path درست میتوانید در نهایت باعث اینجاد یک پردازنده کامل و درست شود.

که در آن از LMP FF , INSTRUCTION MEMORY , PC ADDER , BUS MUX ,

INSTRUCTION INTERPRETER , SHIFT , REGISTER ARRAY , MULTIPLEXER ,

JUMP ADDER , ALU , MAIN MEMORY که بعضی از این کامپوننت ها در ویزارد میباشد و حال به توضیح مختصر آنهایی که خودمان ایجاد کردم

شکل کلی پروژه :

که در فایلی جدا داخل همین زیپ ارسال شده است که شماتیک کلی پردازنده میباشد .

حال به توضیح هرکدام از قطعات میپردازیم :

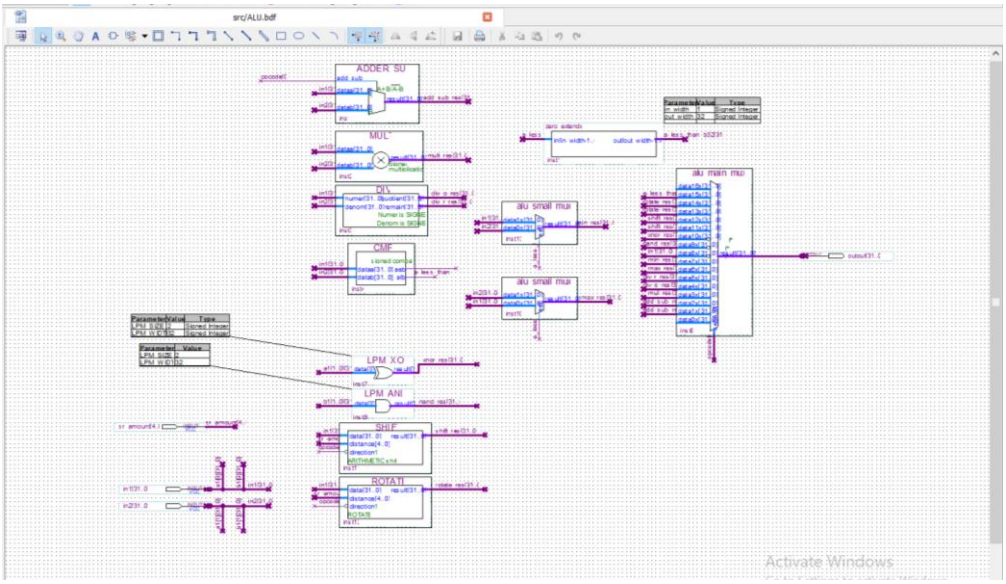
- ALU

همانطور که در تصویر مشاهده خواهید کرد ما در این بخش تماما از بخش ویزارد کوارتوس استفاده کردیم که از قطعات زیر تشکیل شده است :

ADDER MUL DIV LMP_XOR SHIFT ZERO_EXTENDES LMP_AND ROTATE

که همانطور از تصویر مشاهده میکنید پارامتر ها و خروجی و ورودی های مد نظر مشخص شده اند .

توجه : شماتیک ان در فایلی جدا داخل همین زیپ ارسال میشود که شماتیک کلی پردازنده میباشد



- INSTRUCTION MEMORY

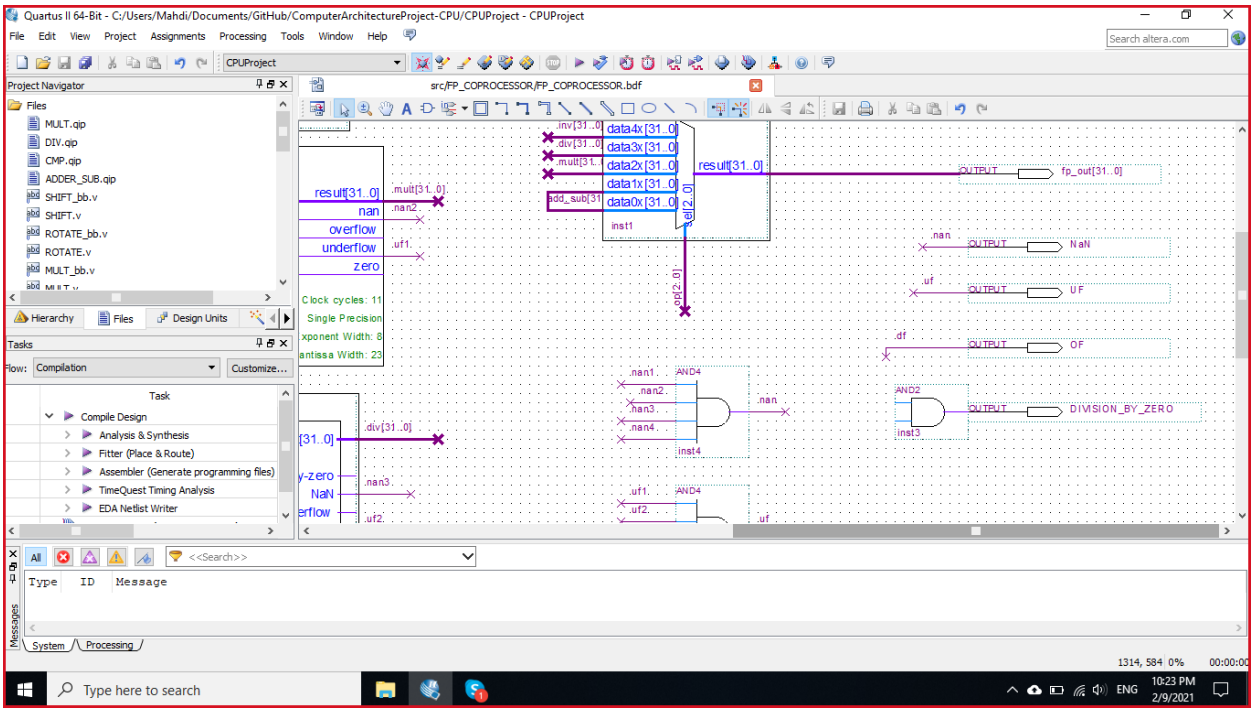
ما این بخش رو با وریلاگ که در پوشه پروژه میباشد نوشته شده که بخشی از بخش کنترل میباشد پیاده سازی شده است .

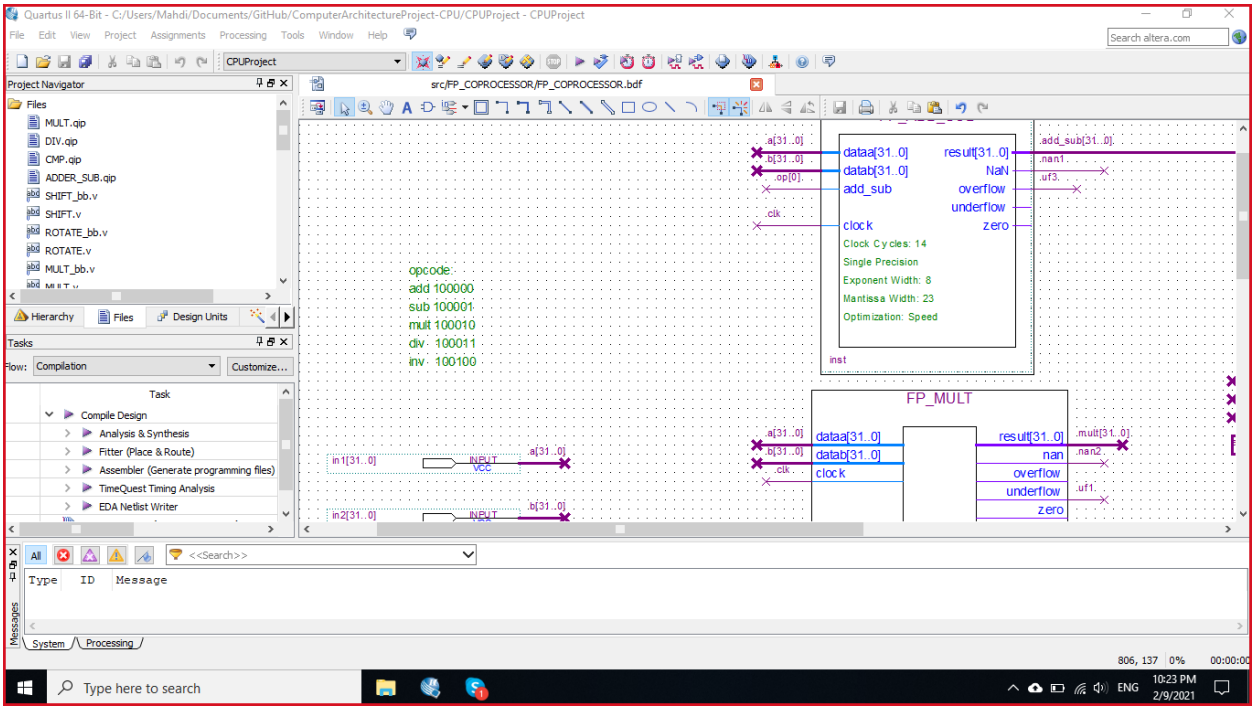
- MAIN MEMORY

یک رم 256 وردی میباشد که ورودی های آن : data address wren rden میباشید که خروجی آن ها data b[7:0] address b[9 :0] clock ascr

. q a[7:0] و q a[31:0]

COPROCESSOR •





توجه که شماتیک کلی ان در فایلی جدا داخل همین داک توضیح میباشد

PIPLINE -

بدون PIPLINE باید حساب کنیم که طولانی ترین عملیات کدام میباشد و با توجه به ان تایم کلاک رو مشخص کنیم.

حال برای پیاده سازی PIPLINE باید به اینگونه عمل کنیم که هر دفعه که وارد بخش کنترل یونیت میشویم مشخص کنیم که دستورات write and read فعال هستن یا نه اگه نبود که نباید pc را افزایش بدهیم .

یکی از مخاطراتی که ممکنه پیش بیاد این میباشید که در هنگام پیاده سازی به صورت PIPLINE ممکنه بعضی اوقات مقدار درستی در ریجستر ها ذخیره نشود .

ـ تست

ما برای تست تمامی دستورات اومدیم :

ابتدا با پایتون کدی نوشته شد که هر دستور پردازنده که نوشته میشود را کد hex آن را در اختیارمان قرار بدهد . سپس یک مجموعه دستورات متوالی همراه با داشتن جواب برای تست دقیق ماژول و data path نوشته شده که در یک فایل text.txt در کنار پروژه وجود دارد.

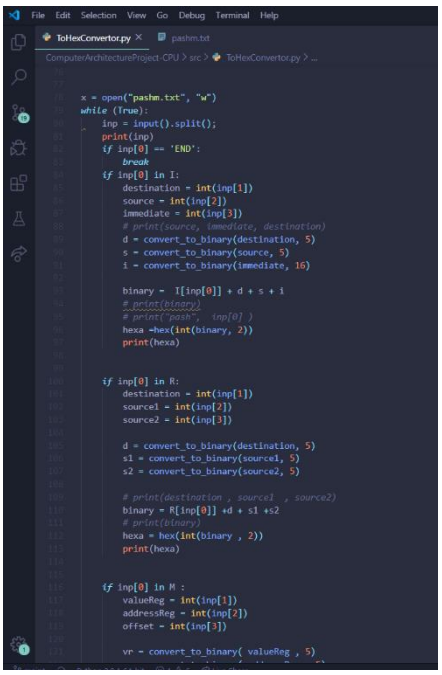
که hex تست بنچ رو گرفتیم .

```
def convert_to_hex(x):
    ans = ""
    for _ in range(8):···
    return ans

print(convert_to_hex(20))

def convert_to_binary(x, digit_count):
    ans = ''
    for i in range(digit_count):
        if ((x >> i) & 1) == 1:
            ans = '1' + ans;
            # print(ans)
        else:
            ans = '0' + ans;
            # print(ans)
    return ans


R = {
    'ADD' : '000001',
    'SUB' : '000010',
    'MUL' : '000011',
    'DIV' : '000100',
    'MOD' : '000101',
    'MAX' : '000110',
    'MIN' : '000111',
    'NOT' : '001000',
    'NAND': '001001',
    'XNOR': '001010',
    'SHL' : '001011',
    'SHRL': '001100',
    'ROL' : '001101',
    'ROR' : '001110',
    'SLT' : '001111'
}

I = {
    'LDI' : '010000',
    'LUI' : '010001',
    'ADDI': '010010',
    'SUBI': '010011',
    'MULI': '010100',
```

1

```
    'SLT' : '001111'
}


I = {
    'LDI' : '010000',
    'LUI' : '010001',
    'ADDI': '010010',
    'SUBI': '010011',
    'MULI': '010100',
    'DIVI': '010101',
    'NANDI': '010110',
    'XNORI': '010111'
}

M = {
    'LW' : '011000',
    'SW' : '011001',
    'LB' : '011010',
    'SB' : '011011'
}

J = {
    'JMP': '011100',
    'JR' : '011101',
    'BEQ': '011110',
    'BLT': '011111',
    'HLT': '000000'
}


x = open("pashm.txt", "w")
while (True):
    inp = input().split();
    print(inp)
    if inp[0] == 'END':
        break
    if inp[0] in I:
        destination = int(inp[1])
        source = int(inp[2])
        immediate = int(inp[3])
        # print(source, immediate, destination)
        d = convert_to_binary(destination, 5)
        s = convert_to_binary(source, 5)
        i = convert_to_binary(immediate, 16)
```

2



```
x = open("pashm.txt", "w")
while (True):
    inp = input().split();
    print(inp)
    if inp[0] == 'END':
        break
    if inp[0] in I:
        destination = int(inp[1])
        source = int(inp[2])
        immediate = int(inp[3])
        # print(source, immediate, destination)
        d = convert_to_binary(destination, 5)
        s = convert_to_binary(source, 5)
        i = convert_to_binary(immediate, 16)

        binary =  I[inp[0]] + d + s + i
        # print(binary)
        # print('pash',  inp[0] )
        hexa =hex(int(binary, 2))
        print(hexa)


    if inp[0] in R:
        destination = int(inp[1])
        source1 = int(inp[2])
        source2 = int(inp[3])

        d = convert_to_binary(destination, 5)
        s1 = convert_to_binary(source1, 5)
        s2 = convert_to_binary(source2, 5)

        # print(destination , source1  , source2)
        binary = R[inp[0]] +d + s1 +s2
        # print(binary)
        hexa = hex(int(binary , 2))
        print(hexa)


    if inp[0] in M :
        valueReg = int(inp[1])
        addressReg = int(inp[2])
        offset = int(inp[3])

        vr = convert_to_binary( valueReg , 5)
```

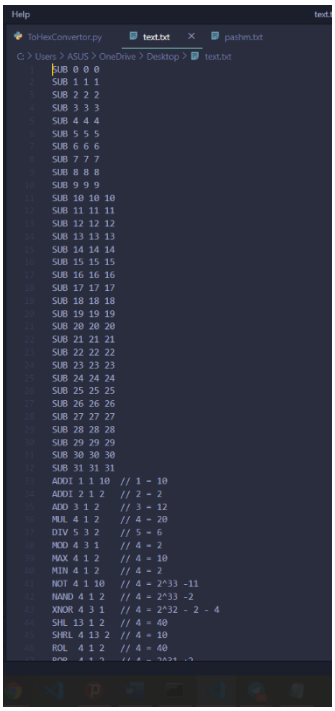
3

```
        # print(destination , source1  , source2)
        binary = R[inp[0]] +d + s1 +s2
        # print(binary)
        hexa = hex(int(binary , 2))
        print(hexa)


    if inp[0] in M :
        valueReg = int(inp[1])
        addressReg = int(inp[2])
        offset = int(inp[3])

        vr = convert_to_binary( valueReg , 5)
        ar = convert_to_binary( addressReg , 5)
        of = convert_to_binary ( offset ,  16 )

        # print(valueReg , addressReg , offset)
        binary = M[inp[0]] + vr + ar + of
        # print(binary)
        hexa = hex(int(binary , 2))
        print(hexa)

    if inp[0] in J :
        reg1 = int(inp[1])
        reg2 = int(inp[2])
        address = int (inp[3])

        r1 = convert_to_binary ( reg1 , 5)
        r2 = convert_to_binary ( reg2 , 5)
        addr = convert_to_binary ( address , 16)

        # print(reg1 , reg2 , address)
        binary = J[inp[0]] + r1 + r2 + addr
        # print(binary)
        hexa = hex(int(binary , 2))

        x.write(hexa + "\n")

x.close()

    # if s[1] in I:
```

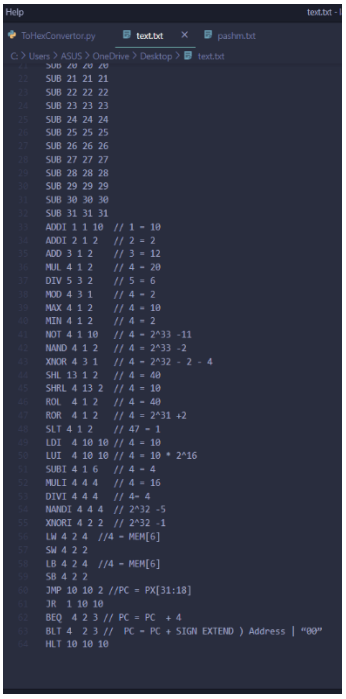
4

مجوعه کد نوشته شده برای تبدیل کد های اسمبلی پردازنده به هگزادسیمال :

مجموعه تست های پردازنده :



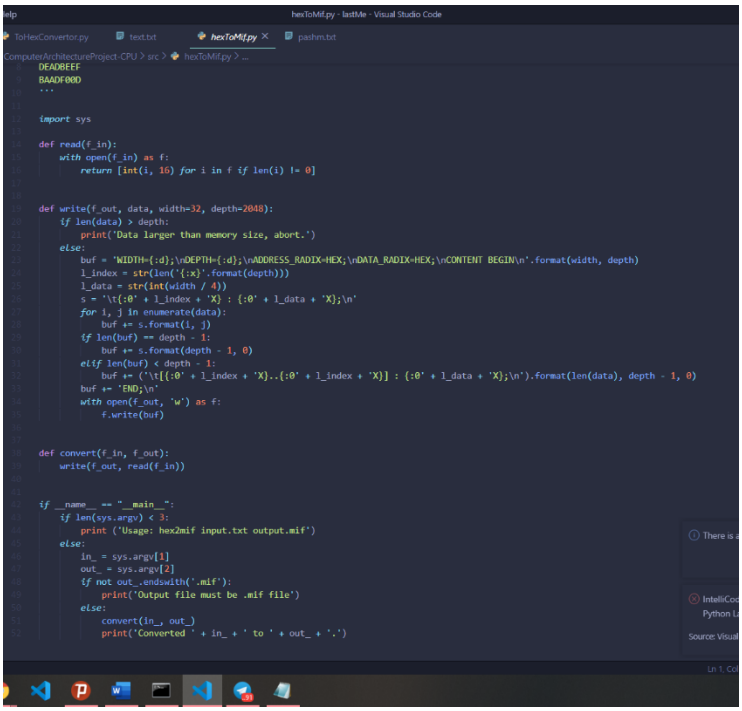
```
SUB 0 0 0
SUB 1 1 1
SUB 2 2 2
SUB 3 3 3
SUB 4 4 4
SUB 5 5 5
SUB 6 6 6
SUB 7 7 7
SUB 8 8 8
SUB 9 9 9
SUB 10 10 10
SUB 11 11 11
SUB 12 12 12
SUB 13 13 13
SUB 14 14 14
SUB 15 15 15
SUB 16 16 16
SUB 17 17 17
SUB 18 18 18
SUB 19 19 19
SUB 20 20 20
SUB 21 21 21
SUB 22 22 22
SUB 23 23 23
SUB 24 24 24
SUB 25 25 25
SUB 26 26 26
SUB 27 27 27
SUB 28 28 28
SUB 29 29 29
SUB 30 30 30
SUB 31 31 31
ADDI 1 1 10  // 1 = 10
ADDI 2 1 2   // 2 = 2
ADD 3 1 2    // 3 = 12
MUL 4 1 2    // 4 = 20
DIV 5 3 2    // 5 = 6
MOD 4 3 1    // 4 = 2
MAX 4 1 2    // 4 = 10
MIN 4 1 2    // 4 = 2
NOT 4 1 10   // 4 = 2^33 -11
NAND 4 1 2   // 4 = 2^33 -2
XNOR 4 3 1   // 4 = 2^32 - 2 - 4
SHL 13 1 2   // 4 = 40
SHRL 4 13 2  // 4 = 10
ROL  4 1 2   // 4 = 40
```



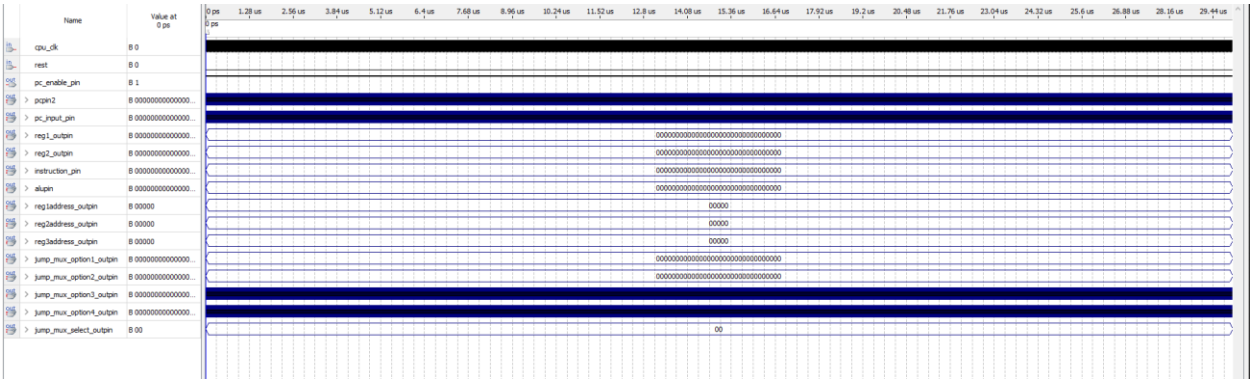
```
SUB 21 21 21
SUB 22 22 22
SUB 23 23 23
SUB 24 24 24
SUB 25 25 25
SUB 26 26 26
SUB 27 27 27
SUB 28 28 28
SUB 29 29 29
SUB 30 30 30
SUB 31 31 31
ADDI 1 1 10  // 1 = 10
ADDI 2 1 2   // 2 = 2
ADD 3 1 2    // 3 = 12
MUL 4 1 2    // 4 = 20
DIV 5 3 2    // 5 = 6
MOD 4 3 1    // 4 = 2
MAX 4 1 2    // 4 = 10
MIN 4 1 2    // 4 = 2
NOT 4 1 10   // 4 = 2^33 -11
NAND 4 1 2   // 4 = 2^33 -2
XNOR 4 3 1   // 4 = 2^32 - 2 - 4
SHL 13 1 2   // 4 = 40
SHRL 4 13 2  // 4 = 10
ROL  4 1 2   // 4 = 40
ROR  4 1 2   // 4 = 2^31 +2
SLT 4 1 2    // 47 = 1
LDI  4 10 10 // 4 = 10
LUI  4 10 10 // 4 = 10 * 2^16
SUBI 4 1 6   // 4 = 4
MULI 4 4 4   // 4 = 16
DIVI 4 4 4   // 4= 4
NANDI 4 4 4  // 2^32 -5
XNORI 4 2 2  // 2^32 -1
LW 4 2 4  //4 = MEM[6]
SW 4 2 2
LB 4 2 4  //4 = MEM[6]
SB 4 2 2
JMP 10 10 2 //PC = PX[31:18]
JR  1 10 10
BEQ  4 2 3 // PC = PC  + 4
BLT 4  2 3 //  PC = PC + SIGN EXTEND ) Address | "00"
HLT 10 10 10
```

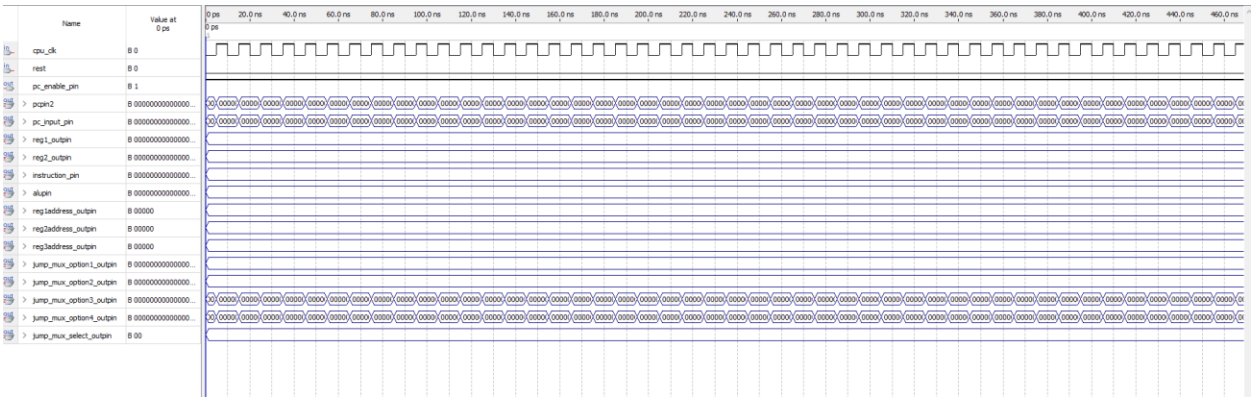
1 2

در نهایت برای استفاده از مجموعه دستورات ماشین و تبدیل دستورات از hex به پسوند .mif برای نمایش waveform یک مجموعه کدهای پایتون نوشته شد برای تبدیل فایل های با پسوند .txt به .mif سپس این فایل را در کوارتوس استفاده کردیم.



بعضی از مشاهدات پس از کامپایل پردازنده :





با توجه به گستردگی پروژه و زمان محدود توانستیم در بخش cacheکار خاصی انجام بدهیم