

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

– مقدمه

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

نمایش اعداد به صورت مکمل دو میباشد .

نمایش کلی پروژه همراه با تمام جزئیات داخل گیت میباشد

-پردازنده

همانطور که میدانید هر پردازنده از قطعات مختلفی تشکیل شده که تجميع آن ها به کمک یک data path درست ميتوانيد در نهايت باعث اينجاد یک پردازنده کامل و درست شود.

که در آن از LMP FF , INSTRUCTION MEMORY , PC ADDER , BUS MUX ,
, MULTIPLEXER , REGISTER ARRAY , SHIFT , INSTRUCTION INTERPRETER
JUMP ADDER , ALU , MAIN MEMORY که بعضی از این کامپوننت ها در ويزارد
ميباشد و حال به توضيح مختصر آنهایی که خودمان ايجاد کردم

شكل کلی پروژه :

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

حال به توضيح هر کدام از قطعات ميپردازيم :

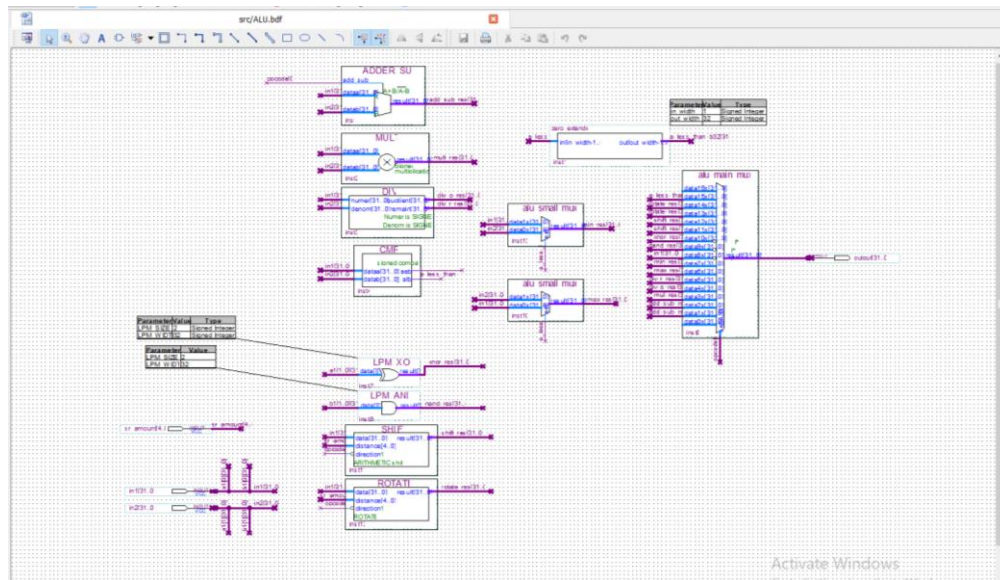
• ALU

همانطور که در تصوير مشاهده خواهيد کرد ما در این بخش تماما از بخش ويزارد کوارتوس استفاده کردیم که از قطعات زیر تشکیل شده است :

ADDER MUL DIV LMP_XOR SHIFT ZERO_EXTENDES LMP_AND
ROTATE

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

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



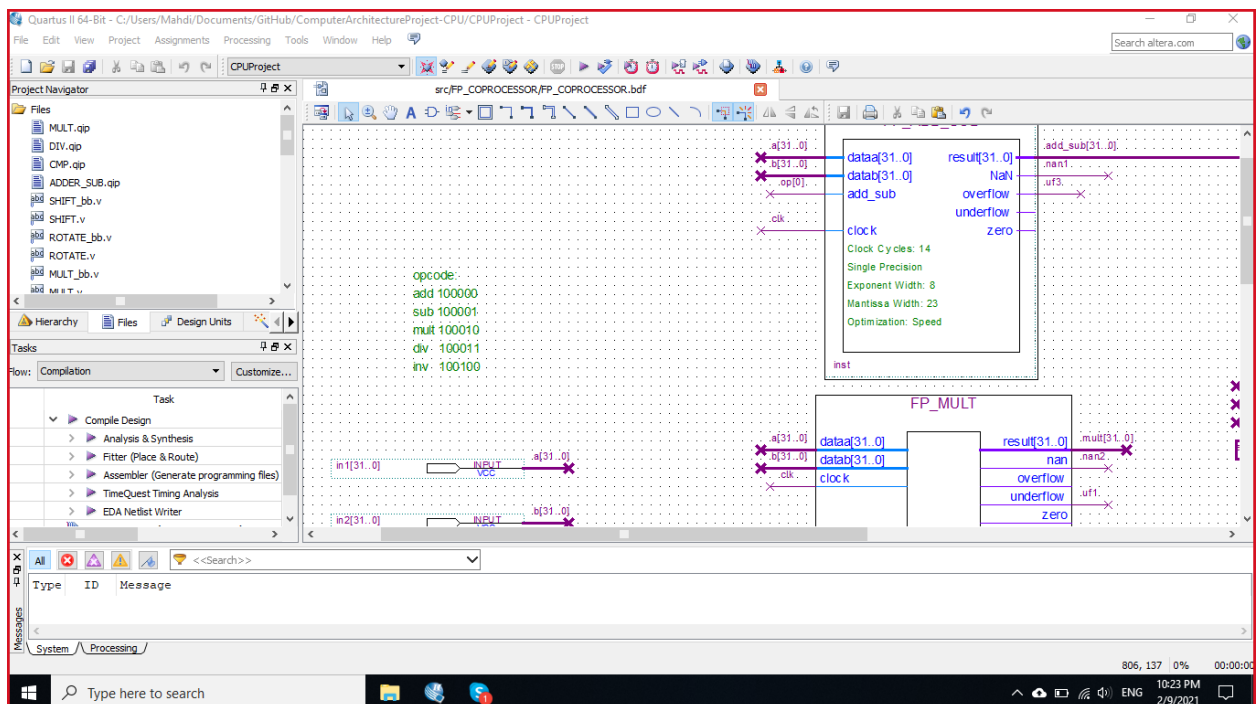
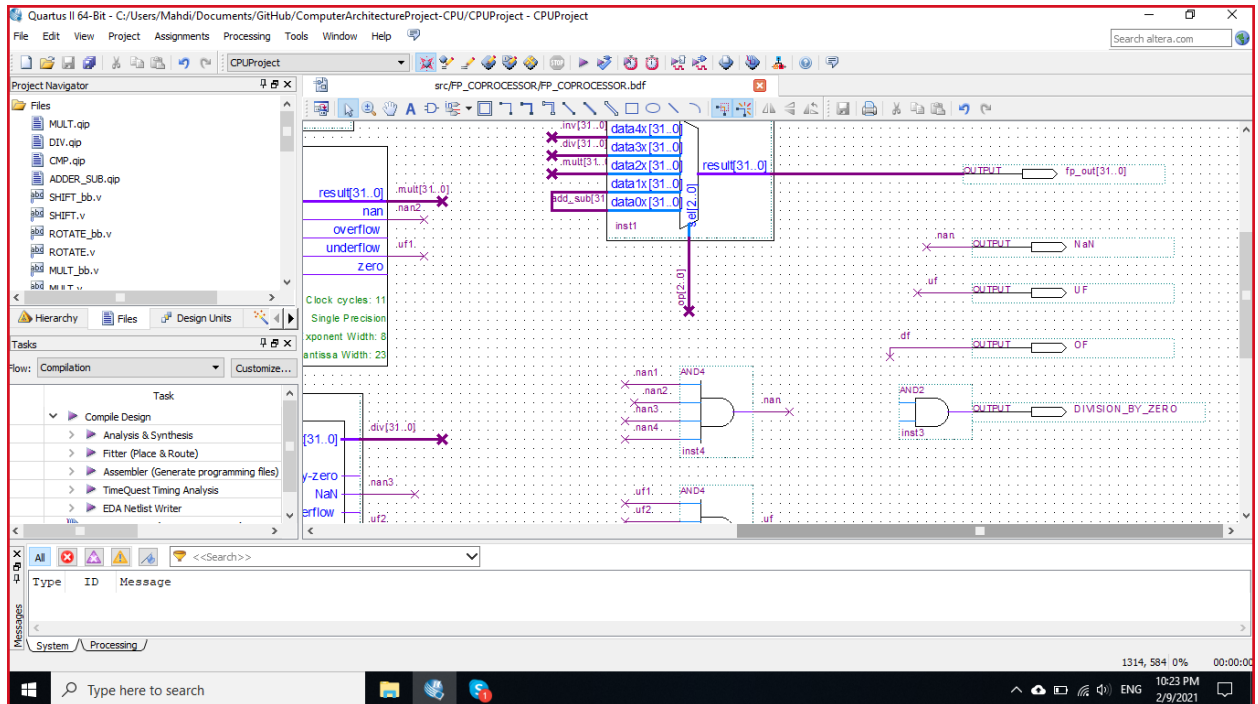
• INSTRUCTION MEMORY

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 : $data$ $address$ $wren$ $rden$:
 $data$ $b[7:0]$ $address$ $b[9:0]$ $clock$ asr
 $a[31:0]$ q و $a[7:0]$ q .

COPROCESSOR •



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

PIPELINE -

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

حال برای پیاده سازی PIPELINE 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 میباشد که در هنگام پیاده سازی به صورت PIPELINE ممکنه بعضی اوقات مقدار درستی در رجیستر ها ذخیره نشود .

– 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',
    'XOR': '001010',
    'SHL': '001011',
    'SHR': '001100',
    'ROL': '001101',
    'ROR': '001110',
    'SLT': '001111'
}

I = {
    'LDI': '010000',
    'LUI': '010001',
    'ADI': '010010',
    'SUI': '010011',
    'MUL': '010100',
    'DIV': '010101',
    'MOD': '010110',
    'HLT': '010111'
}

```

1

```

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',
    'XOR': '001010',
    'SHL': '001011',
    'SHR': '001100',
    'ROL': '001101',
    'ROR': '001110',
    'SLT': '001111'
}

I = {
    'LDI': '010000',
    'LUI': '010001',
    'ADI': '010010',
    'SUI': '010011',
    'MUL': '010100',
    'DIV': '010101',
    'MOD': '010110',
    'HLT': '010111'
}

M = {
    'LM': '011000',
    'SM': '011001',
    'LB': '011010',
    'SB': '011011'
}

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

x = open("pasm.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(hex(int(binary, 2)))
        hexa = hex(int(binary, 2))
        print(hexa)

    if inp[0] in R:
        destination = int(inp[1])
        source1 = int(inp[2])
        source2 = int(inp[3])
        # print(destination, source1, source2)
        d = convert_to_binary(destination, 5)
        s1 = convert_to_binary(source1, 5)
        s2 = convert_to_binary(source2, 5)
        binary = R[inp[0]] + d + s1 + s2
        # print(binary)
        # print(hex(int(binary, 2)))
        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)
        # print(hex(int(binary, 2)))
        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)
        # print(hex(int(binary, 2)))
        hexa = hex(int(binary, 2))
        print(hexa)

    x.write(hexa + "\n")

x.close()

# if s[i] in I:

```

2

```

x = open("pasm.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(hex(int(binary, 2)))
        hexa = hex(int(binary, 2))
        print(hexa)

    if inp[0] in R:
        destination = int(inp[1])
        source1 = int(inp[2])
        source2 = int(inp[3])
        # print(destination, source1, source2)
        d = convert_to_binary(destination, 5)
        s1 = convert_to_binary(source1, 5)
        s2 = convert_to_binary(source2, 5)
        binary = R[inp[0]] + d + s1 + s2
        # print(binary)
        # print(hex(int(binary, 2)))
        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)
        # print(hex(int(binary, 2)))
        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)
        # print(hex(int(binary, 2)))
        hexa = hex(int(binary, 2))
        print(hexa)

    x.write(hexa + "\n")

x.close()

# if s[i] in I:

```

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 R:
    destination = int(inp[1])
    valueReg = int(inp[2])
    addressReg = int(inp[3])
    offset = int(inp[4])
    vr = convert_to_binary(valueReg, 5)
    ar = convert_to_binary(addressReg, 5)
    of = convert_to_binary(offset, 16)
    # print(valueReg, addressReg, offset)
    binary = R[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))
    print(hexa)

x.write(hexa + "\n")

x.close()

# if s[i] in I:

```

4

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

مجموعه تست های پردازنده :

```
Help
ToHexConverter.py  text.txt  x  pashm.txt
G:\Users\ASUS\OneDrive\Desktop> text.txt
1 SUB 0 0 0
2 SUB 1 1 1
3 SUB 2 2 2
4 SUB 3 3 3
5 SUB 4 4 4
6 SUB 5 5 5
7 SUB 6 6 6
8 SUB 7 7 7
9 SUB 8 8 8
10 SUB 9 9 9
11 SUB 10 10 10
12 SUB 11 11 11
13 SUB 12 12 12
14 SUB 13 13 13
15 SUB 14 14 14
16 SUB 15 15 15
17 SUB 16 16 16
18 SUB 17 17 17
19 SUB 18 18 18
20 SUB 19 19 19
21 SUB 20 20 20
22 SUB 21 21 21
23 SUB 22 22 22
24 SUB 23 23 23
25 SUB 24 24 24
26 SUB 25 25 25
27 SUB 26 26 26
28 SUB 27 27 27
29 SUB 28 28 28
30 SUB 29 29 29
31 SUB 30 30 30
32 SUB 31 31 31
33 ADDI 1 1 10 // 1 + 10
34 ADDI 2 1 2 // 2 + 2
35 ADDI 3 1 2 // 3 + 12
36 MUI 4 1 2 // 4 + 20
37 DIV 5 3 2 // 5 + 6
38 MOD 4 3 1 // 4 + 2
39 MAX 4 1 2 // 4 + 10
40 MIN 4 1 2 // 4 + 2
41 NOT 4 1 10 // 4 + 2^33 -11
42 NAND 4 1 2 // 4 + 2^33 -2
43 XOR 4 3 1 // 4 + 2^32 - 2 - 4
44 SHL 13 1 2 // 4 + 40
45 SHRL 4 13 2 // 4 + 10
46 ROL 4 1 2 // 4 + 40
47 ROR 4 1 2 // 4 + 2^31 +2
48 SLT 4 1 2 // 47 - 1
49 LDI 4 10 10 // 4 + 10
50 LUI 4 10 10 // 4 + 10 * 2^16
51 SUBI 4 1 6 // 4 + 4
52 MULI 4 4 4 // 4 + 16
53 DIVI 4 4 4 // 4 - 4
54 NANDI 4 4 4 // 2^32 -5
55 XORI 4 2 2 // 2^32 -1
56 LW 4 2 4 //4 - MEM[6]
57 SW 4 2 2
58 LB 4 2 4 //4 - MEM[6]
59 SB 4 2 2
60 JMP 10 10 2 //PC = PX[31:18]
61 JR 1 10 10
62 BEQ 4 2 3 // PC = PC + 4
63 BLT 4 2 3 // PC = PC + SIGN EXTEND ) Address | "00"
64 HLT 10 10 10
```

1

```
Help
ToHexConverter.py  text.txt  x  pashm.txt
G:\Users\ASUS\OneDrive\Desktop> text.txt
1 SUB 49 49 49
2 SUB 21 21 21
3 SUB 22 22 22
4 SUB 23 23 23
5 SUB 24 24 24
6 SUB 25 25 25
7 SUB 26 26 26
8 SUB 27 27 27
9 SUB 28 28 28
10 SUB 29 29 29
11 SUB 30 30 30
12 SUB 31 31 31
13 ADDI 1 1 10 // 1 + 10
14 ADDI 2 1 2 // 2 + 2
15 ADDI 3 1 2 // 3 + 12
16 MUI 4 1 2 // 4 + 20
17 DIV 5 3 2 // 5 + 6
18 MOD 4 3 1 // 4 + 2
19 MAX 4 1 2 // 4 + 10
20 MIN 4 1 2 // 4 + 2
21 NOT 4 1 10 // 4 + 2^33 -11
22 NAND 4 1 2 // 4 + 2^33 -2
23 XOR 4 3 1 // 4 + 2^32 - 2 - 4
24 SHL 13 1 2 // 4 + 40
25 SHRL 4 13 2 // 4 + 10
26 ROL 4 1 2 // 4 + 40
27 ROR 4 1 2 // 4 + 2^31 +2
28 SLT 4 1 2 // 47 - 1
29 LDI 4 10 10 // 4 + 10
30 LUI 4 10 10 // 4 + 10 * 2^16
31 SUBI 4 1 6 // 4 + 4
32 MULI 4 4 4 // 4 + 16
33 DIVI 4 4 4 // 4 - 4
34 NANDI 4 4 4 // 2^32 -5
35 XORI 4 2 2 // 2^32 -1
36 LW 4 2 4 //4 - MEM[6]
37 SW 4 2 2
38 LB 4 2 4 //4 - MEM[6]
39 SB 4 2 2
40 JMP 10 10 2 //PC = PX[31:18]
41 JR 1 10 10
42 BEQ 4 2 3 // PC = PC + 4
43 BLT 4 2 3 // PC = PC + SIGN EXTEND ) Address | "00"
44 HLT 10 10 10
```

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.

```

help
hexToMif.py - lastMe - Visual Studio Code
TortoiseConverter.py  test.txt  hexToMif.py  pashm.txt
ComputerArchitectureProject-CPU > src > hexToMif.py 2...

DEADBEEF
BAADF00D
...

import sys

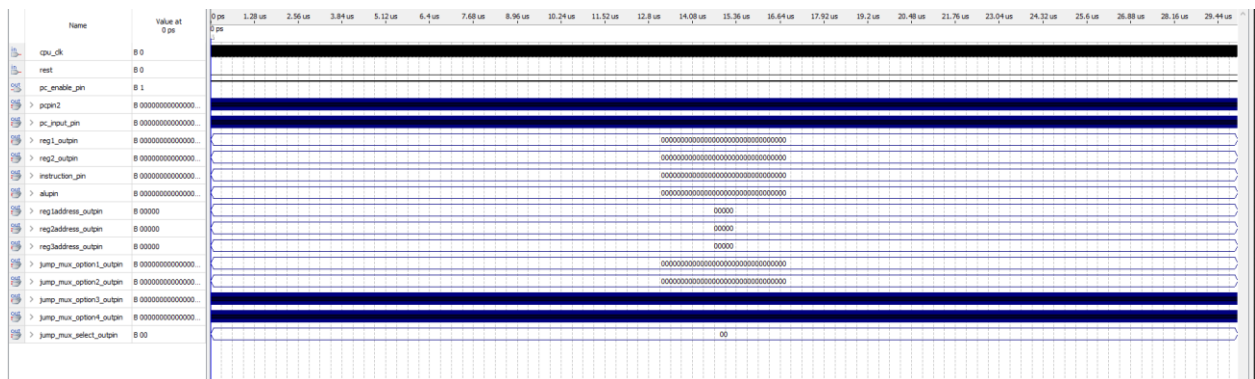
def read(f_in):
    with open(f_in) as f:
        return [int(i, 16) for i in f if len(i) != 0]

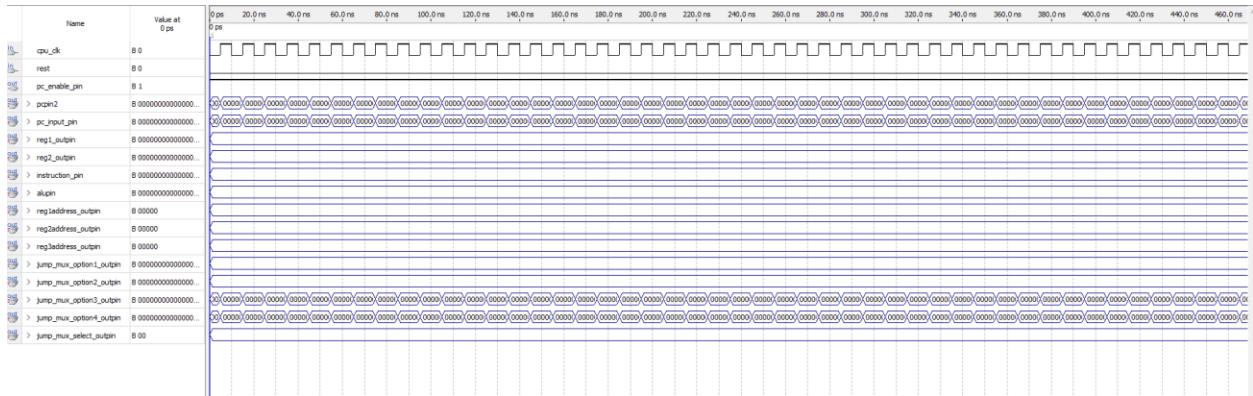
def write(f_out, data, width=32, depth=2048):
    if len(data) > depth:
        print('Data larger than memory size, abort..')
    else:
        buf = "WIDTH={:d};\nDEPTH={:d};\nADDRESS_RADIX=HEX;\nDATA_RADIX=HEX;\nCONTENT BEGIN\n".format(width, depth)
        l_index = str(len("{:x}".format(depth)))
        l_data = str(int(width / 4))
        s = "\t{:0} + l_index + 'X' : {:0} + l_data + 'X';\n"
        for i, j in enumerate(data):
            buf += s.format(i, j)
            if len(buf) == depth - 1:
                buf += s.format(depth - 1, 0)
            elif len(buf) < depth - 1:
                buf += "\t[{:0} + l_index + 'X'] : {:0} + l_data + 'X';\n".format(len(data), depth - 1, 0)
            buf += "\n\n"
        with open(f_out, 'w') as f:
            f.write(buf)

def convert(f_in, f_out):
    write(f_out, read(f_in))

if __name__ == "__main__":
    if len(sys.argv) < 3:
        print('Usage: hex2mif input.txt output.mif')
    else:
        in_ = sys.argv[1]
        out_ = sys.argv[2]
        if not out_.endswith('.mif'):
            print('Output file must be .mif file')
        else:
            convert(in_, out_)
            print('Converted ' + in_ + ' to ' + out_ + '.')
```

بعضی از مشاهدات پس از کامپایل پردازنده :





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