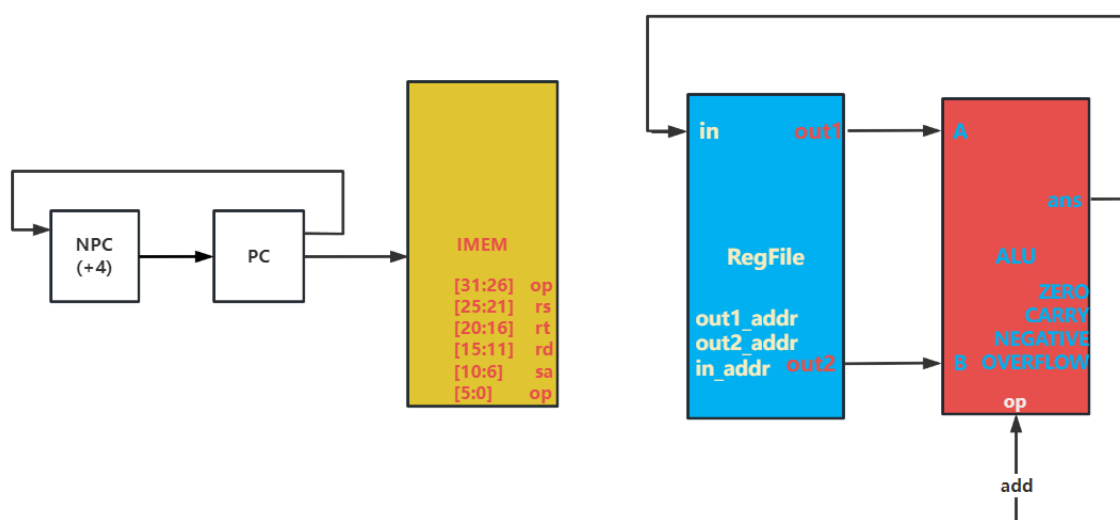
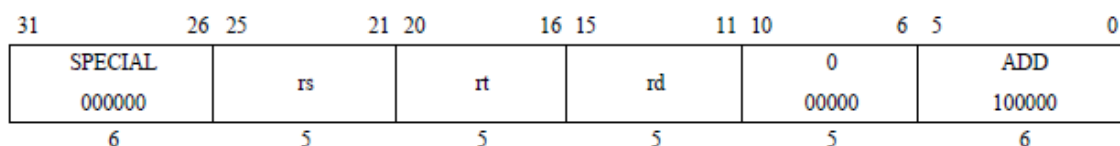


R 型指令

通路类型1

1.ADD



格式: ADD rd, rs, rt

目的: 32位数相加

描述: $rd \leftarrow rs + rt$

将通用寄存器中存的32位数据rs与rt相加产生一个32位数据存入目标寄存器rd。

- 如果发生了溢出，则rd不改变并且产生一个溢出的异常。
- 如果相加不溢出，则产生的32位数据直接存入目标寄存器rd。

操作:

$temp \leftarrow (GPR[rs]_{31} || GPR[rs]_{31..0}) + (GPR[rt]_{31} || GPR[rt]_{31..0})$

if $temp_{32} \neq temp_{31}$ then

SignalException(IntegerOverflow)

else

$GPR[rd] \leftarrow temp$

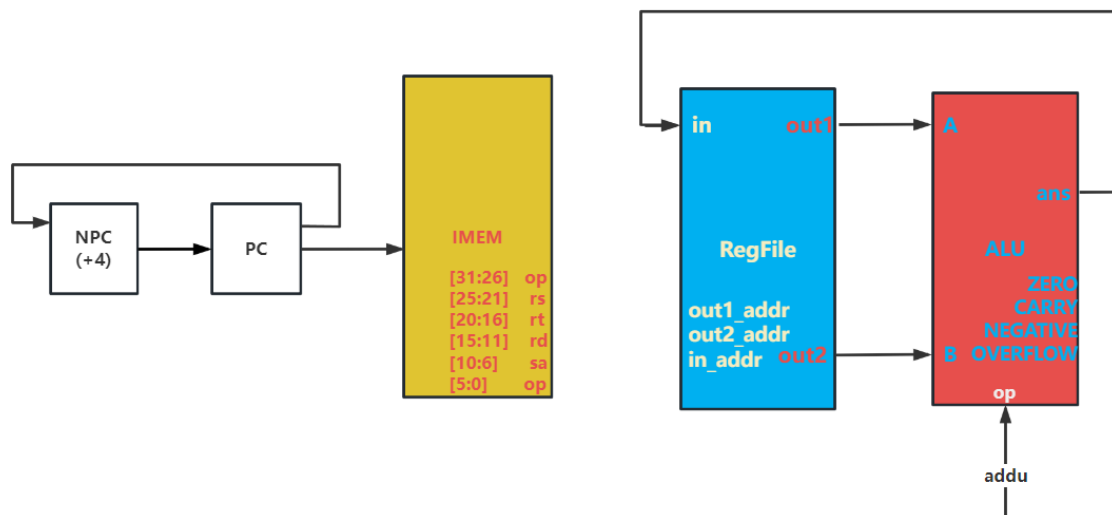
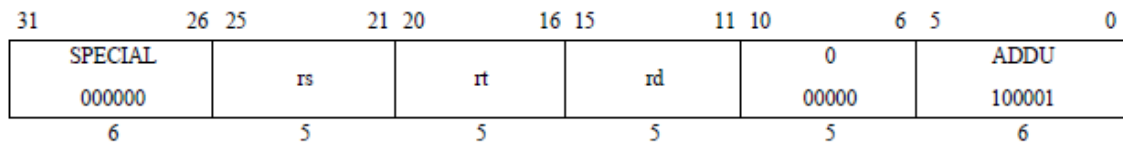
Endif

```

1 PC -> IMEM
2 PC + 4 -> NPC
3 NPC -> PC
4 out1 -> A , out2 -> B
5 (A + B -> RES)
6 ans -> in

```

2.ADDU



格式: ADDU rd, rs, rt

目的: 32位数据相加

描述: $rd \leftarrow rs + rt$

将通用寄存器中存的32位数据rs与rt相加产生一个32位数据存入目标寄存器rd。

在任何情况下都不会有溢出的异常。

操作:

$temp \leftarrow GPR[rs] + GPR[rt]$

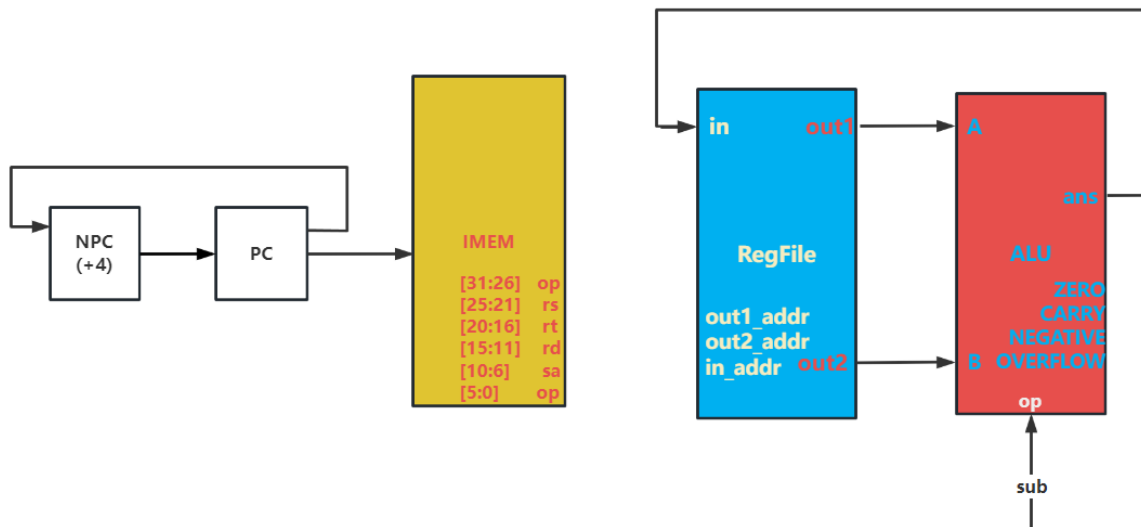
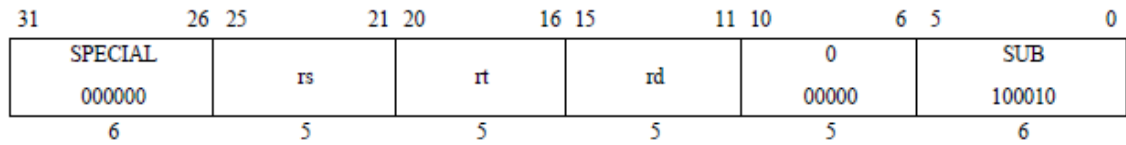
$GPR[rd] \leftarrow temp$

```

1 PC -> IMEM
2 PC + 4 -> NPC
3 NPC -> PC
4 out1 -> A , out2 -> B
5 (A + B -> RES)
6 ans -> in

```

3.SUB



格式: SUB rd, rs, rt

目的: 与32位数相减

描述: $rd \leftarrow rs - rt$

将通用寄存器中存的32位数据rs与rt相减产生一个32位数据存入目标寄存器rd。

- 如果发生了溢出，则rd不改变并且产生一个溢出的异常。
- 如果不溢出，则产生的32位数据直接存入目标寄存器rd。

操作:

$temp \leftarrow (GPR[rs]_{31} || GPR[rs]_{31..0}) - (GPR[rt]_{31} || GPR[rt]_{31..0})$

if $temp_{32} \neq temp_{31}$ then

SignalException(IntegerOverflow)

else

$GPR[rd] \leftarrow temp_{31..0}$

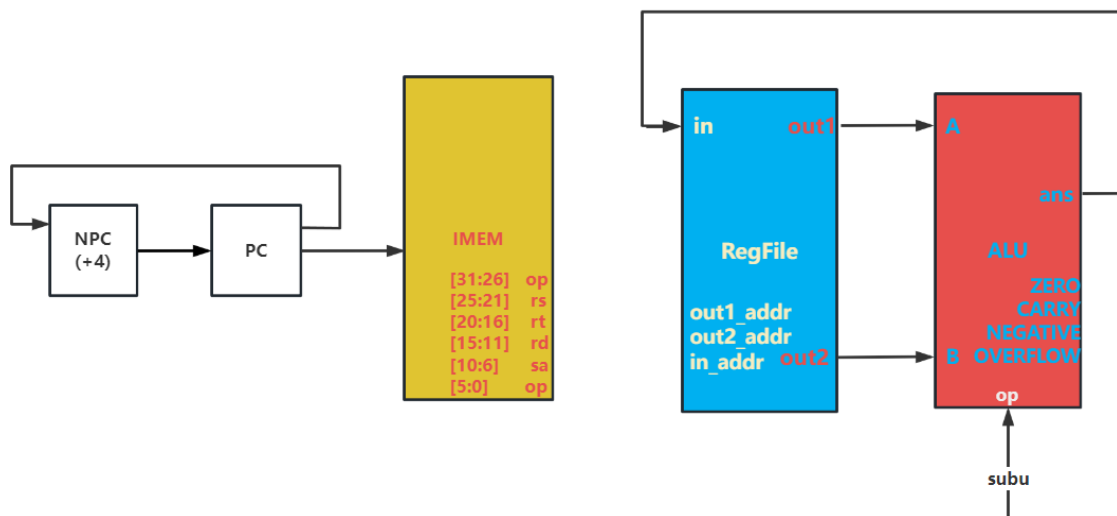
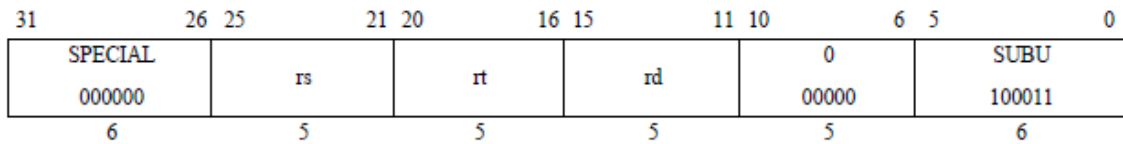
endif

```

1 PC -> IMEM
2 PC + 4 -> NPC
3 NPC -> PC
4 out1 -> A , out2 -> B
5 (A - B -> RES)
6 ans -> in

```

4.SUBU



格式: SUBU rd, rs, rt

目的: 32位数据相减

描述: $rd \leftarrow rs - rt$

将通用寄存器中存的32位数据rs与rt相减产生一个32位数据存入目标寄存器rd。

在任何情况下都不会有溢出的异常。

操作:

$temp \leftarrow GPR[rs] - GPR[rt]$

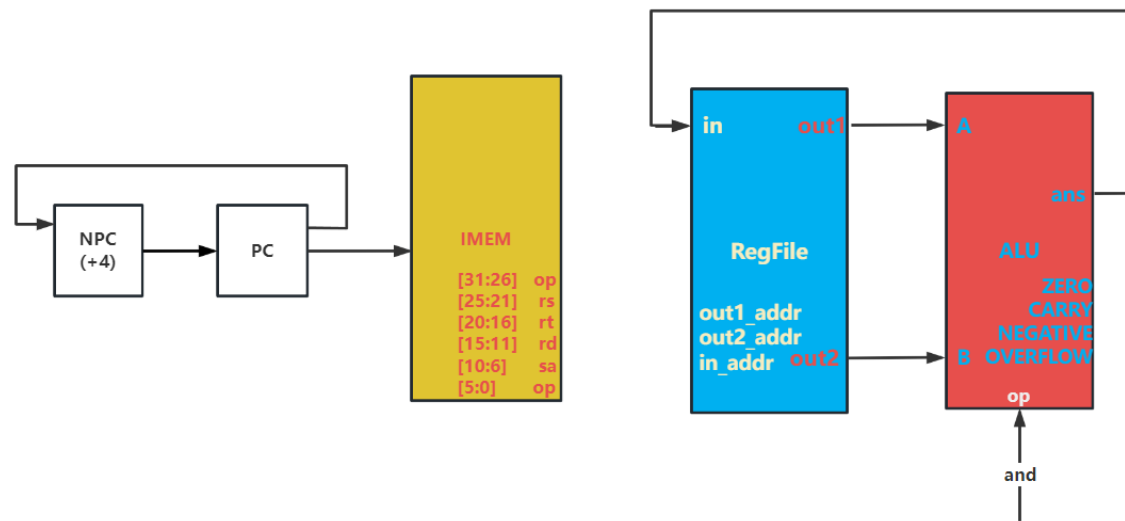
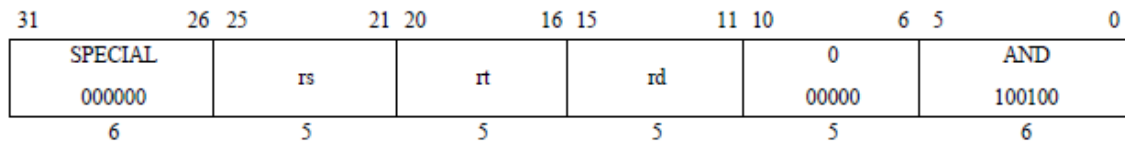
$GPR[rd] \leftarrow temp$

```

1 PC -> IMEM
2 PC + 4 -> NPC
3 NPC -> PC
4 out1 -> A , out2 -> B
5 (A - B -> RES)
6 ans -> in

```

5.AND



格式: AND rd, rs, rt

目的: 按位逻辑与

描述: $rd \leftarrow rs \text{ AND } rt$

将通用寄存器rs和rd中的数据每一位做按位与操作，将结果存入目标寄存器rd中。

操作:

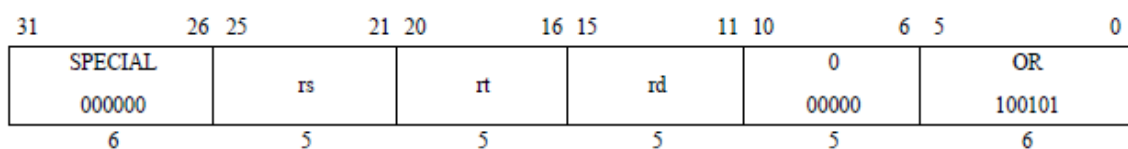
$GPR[rd] \leftarrow GPR[rs] \text{ and } GPR[rt]$

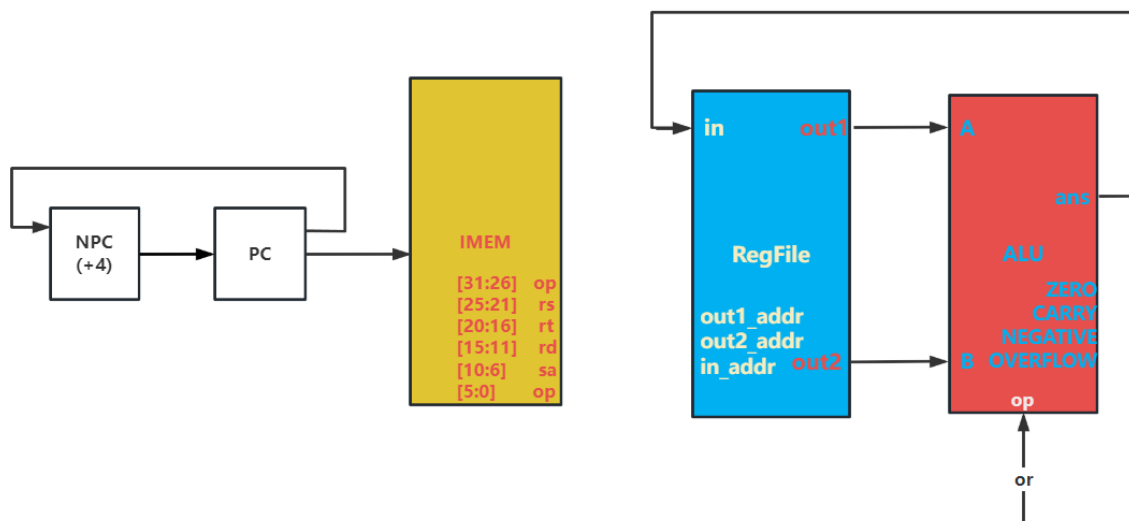
```

1 | PC -> IMEM
2 | PC + 4 -> NPC
3 | NPC -> PC
4 | out1 -> A , out2 -> B
5 | (A & B -> RES)
6 | ans -> in

```

6.OR





格式: OR rd, rs, rt

目的: 按位逻辑或

描述: $rd \leftarrow rs \text{ or } rt$

将通用寄存器rs和rt中的数据每一位做按位或操作，将结果存入目标寄存器rd中。

操作:

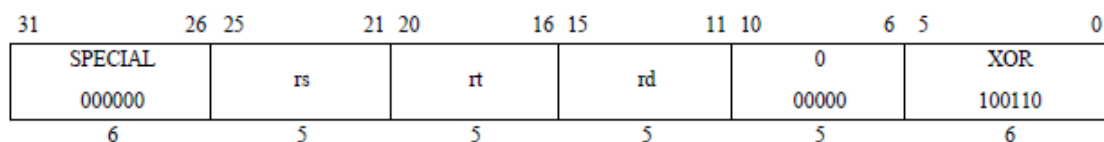
$GPR[rd] \leftarrow GPR[rs] \text{ or } GPR[rt]$

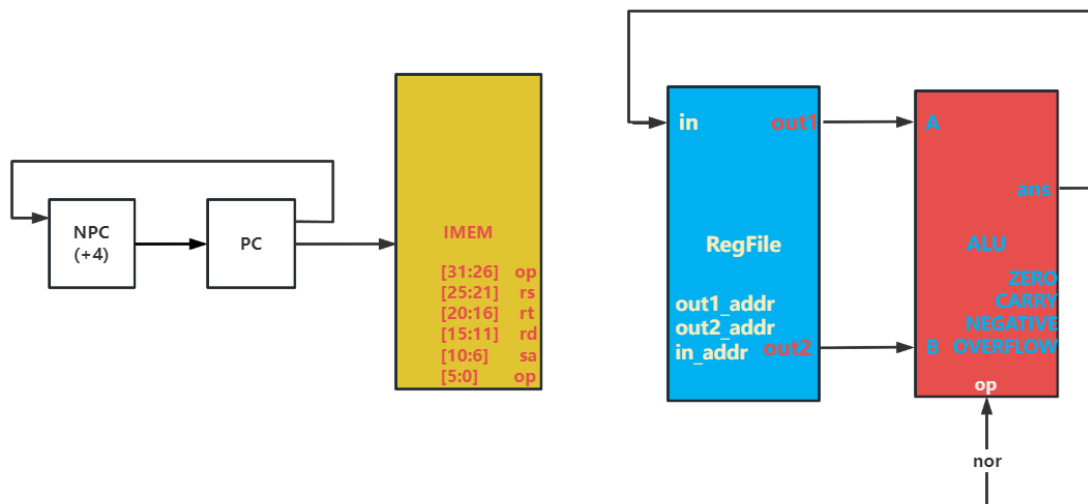
```

1 | PC -> IMEM
2 | PC + 4 -> NPC
3 | NPC -> PC
4 | out1 -> A , out2 -> B
5 | (A | B -> RES)
6 | ans -> in

```

7.XOR





格式: NOR rd, rs, rt

目的: 按位逻辑或非

描述: $rd \leftarrow rs \text{ NOR } rt$

将通用寄存器rs和rt中的数据每一位做按位或非操作，将结果存入目标寄存器rd中。

操作:

$GPR[rd] \leftarrow GPR[rs] \text{ nor } GPR[rt]$

```

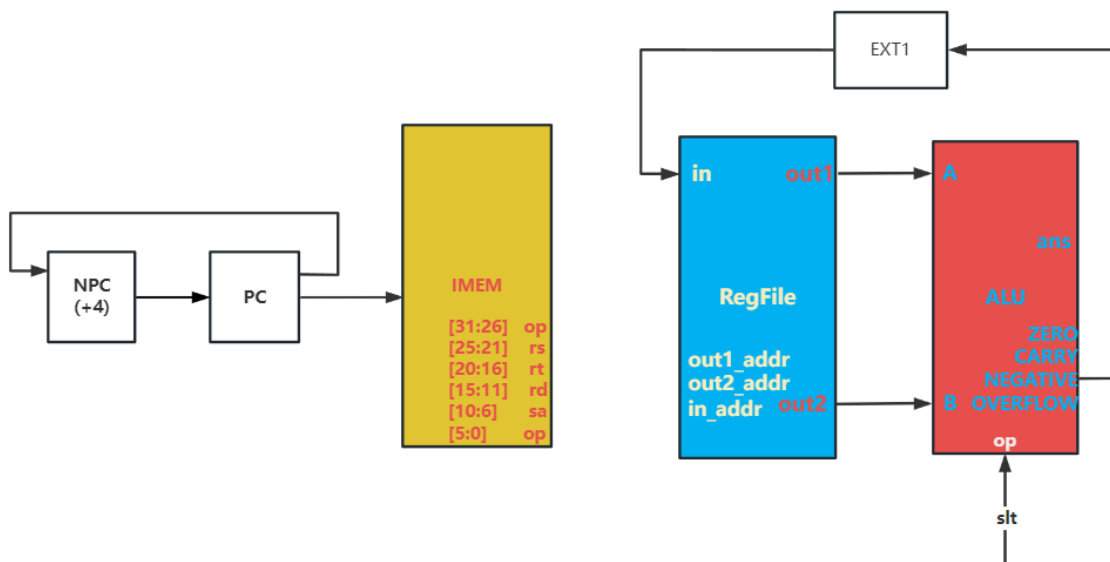
1 | PC -> IMEM
2 | PC + 4 -> NPC
3 | NPC -> PC
4 | out1 -> A , out2 -> B
5 | (A  $\odot$  B -> RES)
6 | ans -> in

```

通路类型2

9.SLT

31	26	25	21	20	16	15	11	10	6	5	0
SPECIAL	rs		rt		rd		0		SLT		
000000							00000		101010		
6	5		5		5		5		6		



格式: SLT rd, rs, rt

目的: 通过小于的比较来记录结果

描述: $rd \leftarrow (rs < rt)$

比较在rs和rt寄存器中保存的有符号数，用boolean值保存结果到rd寄存器中。如果rs小于rt，则结果为1，反之结果为0。算数比较不会引起溢出异常。

操作:

if GPR[rs] < GPR[rt] then

GPR[rd] $\leftarrow 0^{GPRLEN-1} || 1$

else

GPR[rd] $\leftarrow 0^{GPRLEN}$

endif

```

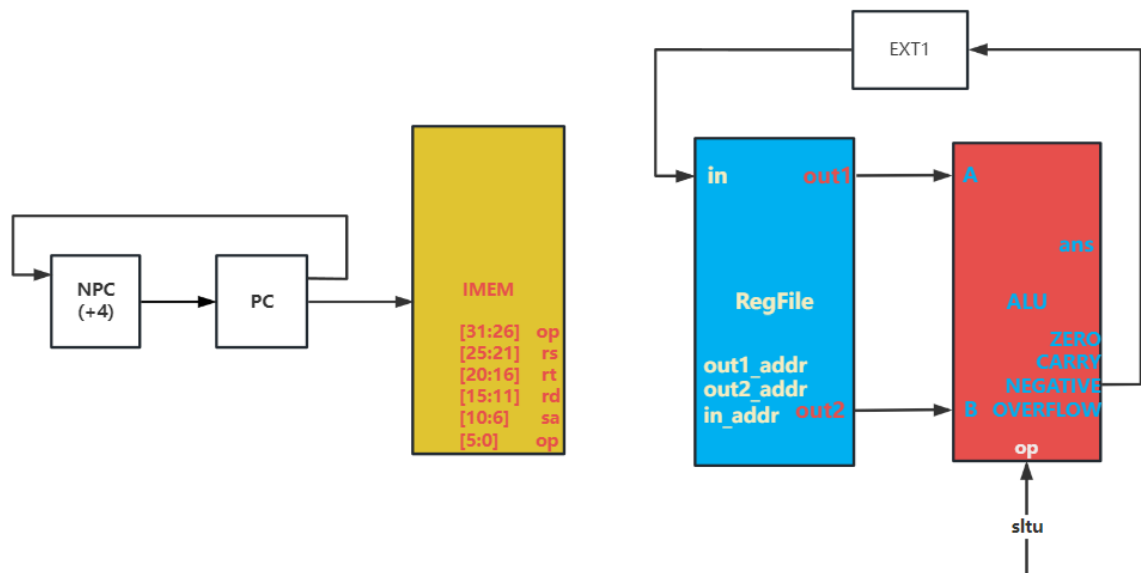
1 PC -> IMEM
2 PC + 4 -> NPC
3 NPC -> PC
4 out1 -> A , out2 -> B
5 (A - B -> ans) //相减判断，负数则为RS中数小
6 negative -> EXT1 //注意要做扩展
7 EXT1 -> in

```

10.SLTU

31	26	25	21	20	16	15	11	10	6	5	0
SPECIAL						rs		rt		rd	
000000											
6						5		5		5	

0						SLTU	
00000						101011	
5						6	



格式: SLTU rd, rs, rt

目的: 通过无符号小于的比较来记录结果

描述: $rd \leftarrow (rs < rt)$

比较在rs和rt寄存器中保存的无符号数，用boolean值保存结果到rd寄存器中。

如果rs小于rt，则结果为1，反之结果为0。算数比较不会引起溢出异常。

操作:

if (0 || GPR[rs]) < (0 || GPR[rt]) then

GPR[rd] $\leftarrow 0^{GPRLEN-1} || 1$

else

GPR[rd] $\leftarrow 0^{GPRLEN}$

endif

```

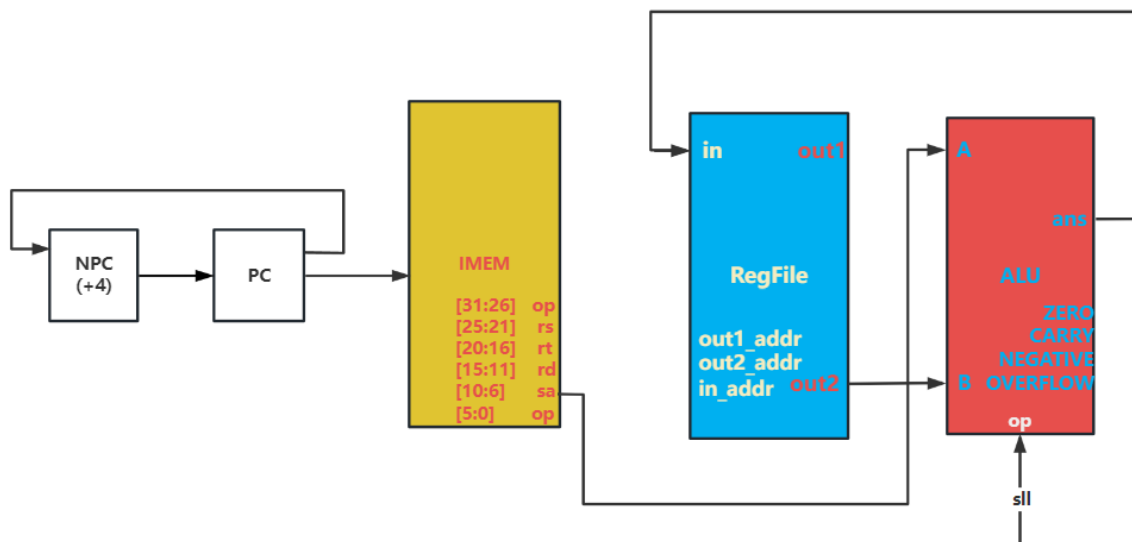
1  PC -> IMEM
2  PC + 4 -> NPC
3  NPC -> PC
4  out1 -> A , out2 -> B
5  (A - B -> ans) //相减判断，负数则为Rs中数小
6  negative -> EXT1 //注意要做扩展
7  EXT1 -> in

```

通路类型3

11.SLL

31	26	25	21	20	16	15	11	10	6	5	0
SPECIAL						SLL					
000000						000000					
6						6					



格式: SLL rd, rt, sa

目的: 通过数字填充逻辑左移

描述: $rd \leftarrow rt \ll sa$

将通用寄存器rt的内容左移sa位，空余出来的位置用0来填充，把结果存入rd寄存器。

操作:

$s \leftarrow sa$

$temp \leftarrow GPR[rt]_{(31-s)..0} || 0^s$

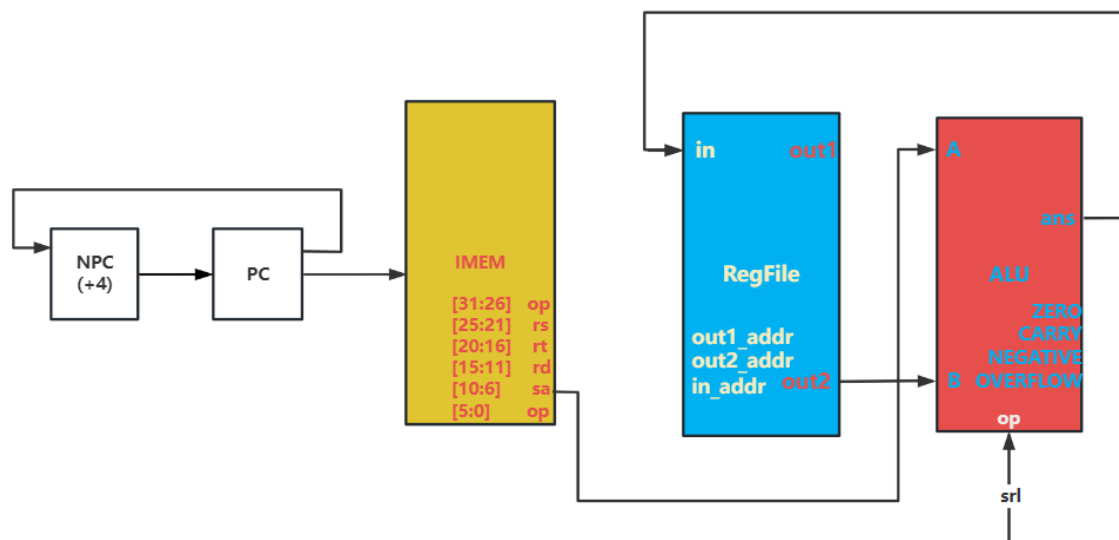
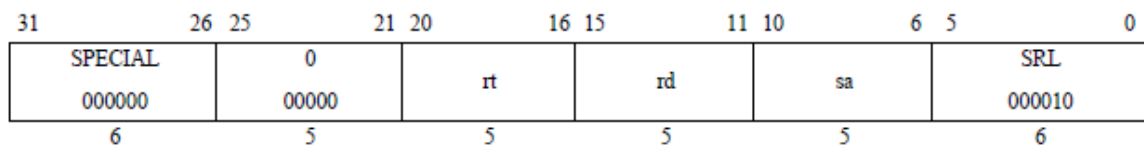
$GPR[rd] \leftarrow temp$

```

1 | PC -> IMEM
2 | PC + 4 -> NPC
3 | NPC -> PC
4 | IMEM[10:6] -> EXT5
5 | EXT5 -> A
6 | out2 -> B
7 | (B<<A -> ans)
8 | ans -> in

```

12.SRL



格式: SRL rd, rt, sa

目的: 通过数字填充逻辑右移

描述: $rd \leftarrow rt \gg sa$ (logical)

将通用寄存器rt中的32位内容右移sa位，高位用0来填充，结果存入通用寄存器rd。

操作:

$s \leftarrow sa$

$temp \leftarrow 0^s \parallel GPR[rt]_{31..s}$

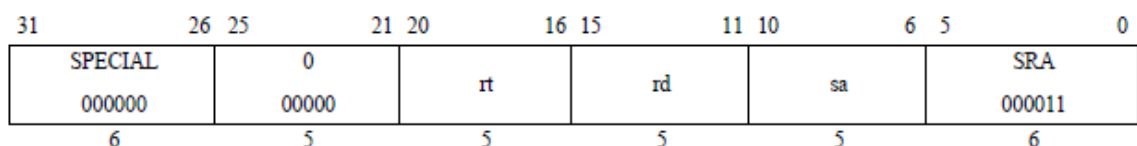
$GPR[rd] \leftarrow temp$

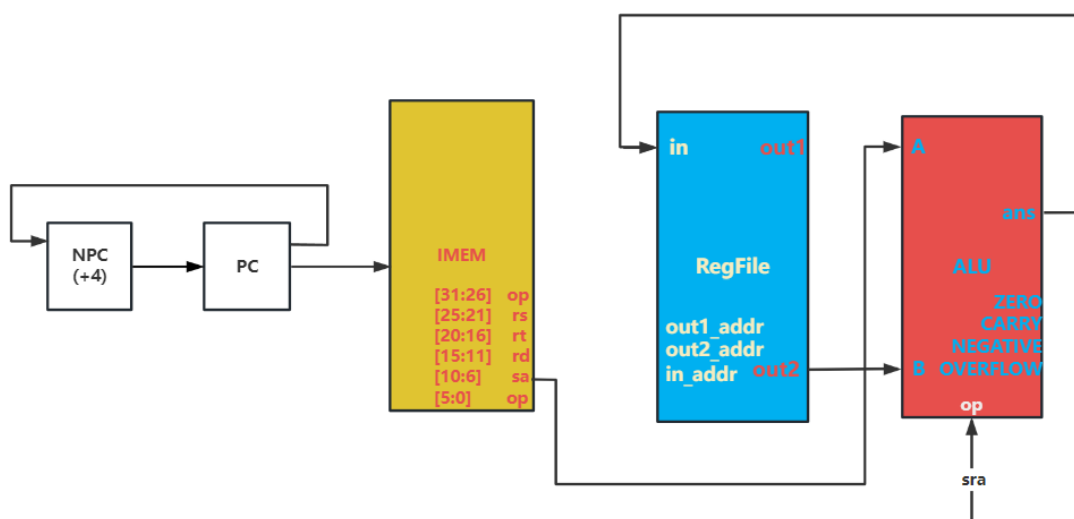
```

1 PC -> IMEM
2 PC + 4 -> NPC
3 NPC -> PC
4 IMEM[10:6] -> EXT5
5 EXT5 -> A
6 out2 -> B
7 (B>>A -> ans)
8 ans -> in

```

13.SRA





格式: SRA rd, rt, sa

目的: 通过数字填充算术右移

描述: $rd \leftarrow rt \gg sa$ (arithmetic)

将通用寄存器rt中的32位内容右移sa位，高位用rt[31]来填充，结果存入通用寄存器rd。

操作:

$s \leftarrow sa$

$temp \leftarrow (GPR[rt]31)^s \parallel GPR[rt]_{31..s}$

$GPR[rd] \leftarrow temp$

```

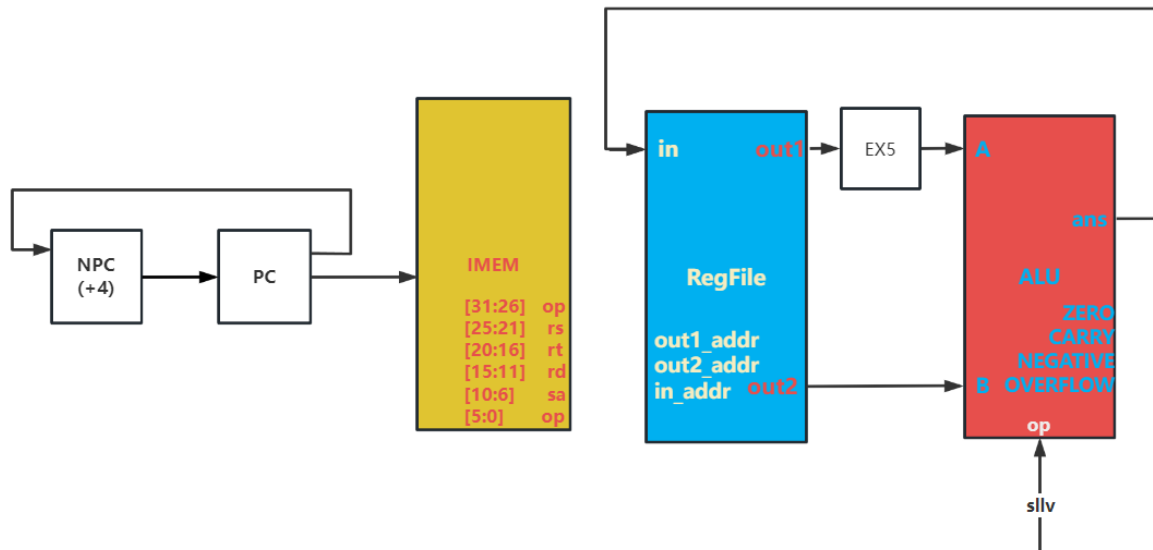
1 PC -> IMEM
2 PC + 4 -> NPC
3 NPC -> PC
4 IMEM[10:6] -> EXT5
5 EXT5 -> A
6 out2 -> B
7 (B>>A -> ans)
8 ans -> in

```

通路类型4

14.SLLV

31	26	25	21	20	16	15	11	10	6	5	0
SPECIAL 000000	rs	rt	rd	0 00000	SLLV 000100						
6	5	5	5	5	6						



格式: SLLV rd, rt, rs

目的: 通过数字填充逻辑左移

描述: $rd \leftarrow rt \ll rs$

将通用寄存器rt的内容逻辑左移，左移的位数保存在rs寄存器中，空余出来的位置用0来填充，把结果存入rd寄存器。

操作:

$s \leftarrow \text{GPR}[rs]_{4..0}$

$\text{temp} \leftarrow \text{GPR}[rt]_{(31-s)..0} \parallel 0^s$

$\text{GPR}[rd] \leftarrow \text{temp}$

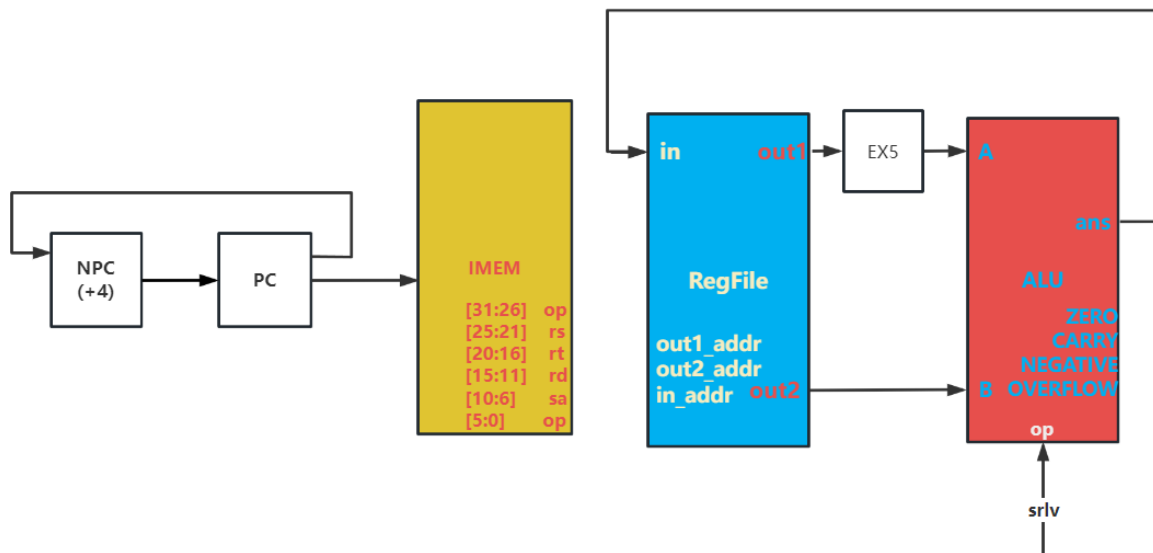
```

1  PC -> IMEM
2  PC + 4 -> NPC
3  NPC -> PC
4  out1[4:0] -> EXT5
5  EXT5 -> A
6  out2 -> B
7  (A<<B -> ans)
8  ans -> in

```

15.SRLV

31	26	25	21	20	16	15	11	10	6	5	0
SPECIAL						rs		rt		rd	
000000										0	
										00000	
										SRLV	
										000110	
6						5		5		5	



格式: SRLV rd, rt, rs

目的: 通过数字填充逻辑右移

描述: $rd \leftarrow rt \gg rs$ (logical)

将通用寄存器rt中的32位内容右移，高位用rt[31]来填充，结果存入通用寄存器rd。右移的位数由通用寄存器rs中的0-4bit确定。

操作:

$s \leftarrow \text{GPR}[rs]_{4..0}$

$\text{temp} \leftarrow 0^s \parallel \text{GPR}[rt]_{31..s}$

$\text{GPR}[rd] \leftarrow \text{temp}$

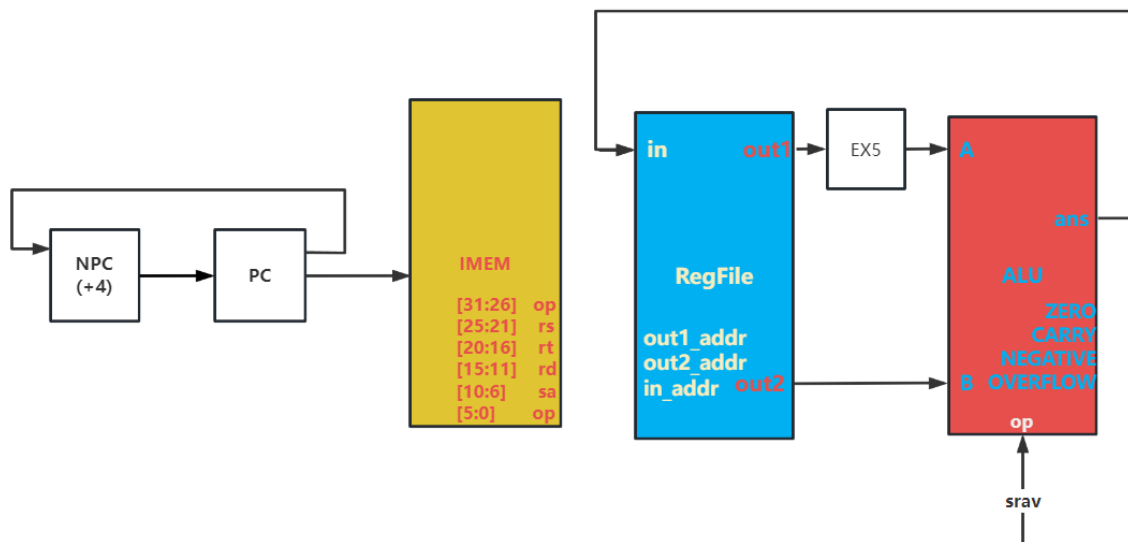
```

1 | PC -> IMEM
2 | PC + 4 -> NPC
3 | NPC -> PC
4 | out1[4:0] -> EXT5
5 | EXT5 -> A
6 | out2 -> B
7 | (A >> B -> ans)
8 | ans -> in

```

16.SRAV

31	26	25	21	20	16	15	11	10	6	5	0
SPECIAL						rs					
000000						rt					
						rd					
						0					
						00000					
						SRAV					
						000111					
6						5					



格式: SRAV rd, rt, rs

目的: 通过数字填充算术右移

描述: $rd \leftarrow rt \gg rs$ (arithmetic)

将通用寄存器rt中的32位内容右移，高位用rt[31]来填充，结果存入通用寄存器rd。右移的位数由通用寄存器rs中的0-4bit确定。

操作:

$s \leftarrow \text{GPR}[rs]_{4..0}$

$\text{temp} \leftarrow (\text{GPR}[rt]_{31})^s \parallel \text{GPR}[rt]_{31..s}$

$\text{GPR}[rd] \leftarrow \text{temp}$

```

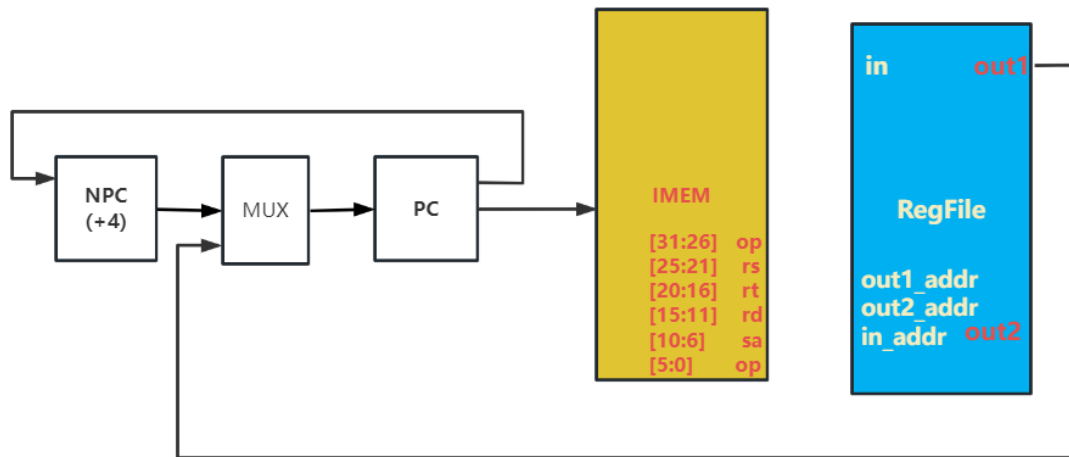
1 PC -> IMEM
2 PC + 4 -> NPC
3 NPC -> PC
4 out1[4:0] -> EXT5
5 EXT5 -> A
6 out2 -> B
7 (A >> B -> ans)
8 ans -> in

```

通路类型5

17.JR

31	26 25	21 20	11 10	6 5	0
SPECIAL 000000	rs	0 00 0000 0000	hint	JR 001000	
6	5	10	5	6	



格式: JR rs

目的: 使用寄存器的跳转指令

描述: $PC \leftarrow rs$

跳转地址存放在通用寄存器rs中，直接跳转到寄存器所存地址。

操作:

I: $temp \leftarrow GPR[rs]$

I+1: if Config1_{CA} = 0 then

$PC \leftarrow temp$

else

$PC \leftarrow temp_{GPRLEN-1..1} || 0$

ISAMode $\leftarrow temp_0$

endif

```

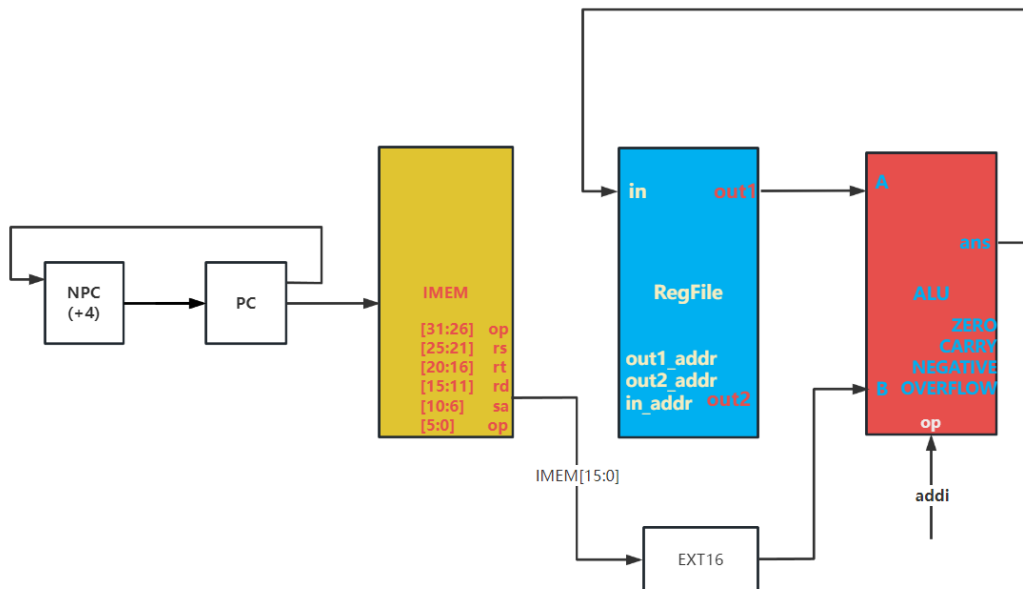
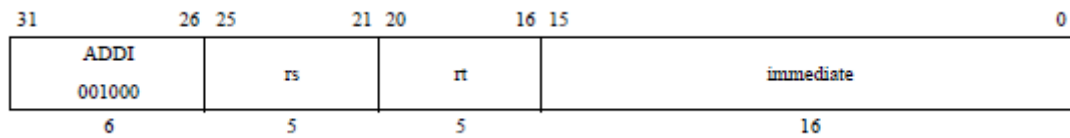
1 | PC -> IMEM
2 | PC + 4 -> NPC //无关指令
3 | out1 -> MUX
4 | MUX_OUT -> PC
5 | NPC -> MUX //无关指令

```

I 型指令

通路类型6

18. ADDI



格式: ADDI rt, rs, immediate

目的: 使32位数据与一个立即数相加

描述: $rt \leftarrow rs + \text{immediate}$

16位有符号立即数与通用寄存器rs中的32位数相加产生一个32位的数存入目标寄存器rt。

- 如果发生了溢出，则rt不改变并且产生一个溢出的异常。
- 如果相加不溢出，则结果存入目标寄存器rt。

操作:

$\text{temp} \leftarrow (\text{GPR}[rs]_{31} || \text{GPR}[rs]_{31..0}) + \text{sign_extend}(\text{immediate})$

if $\text{temp}_{32} \neq \text{temp}_{31}$ then

SignalException(IntegerOverflow)

else

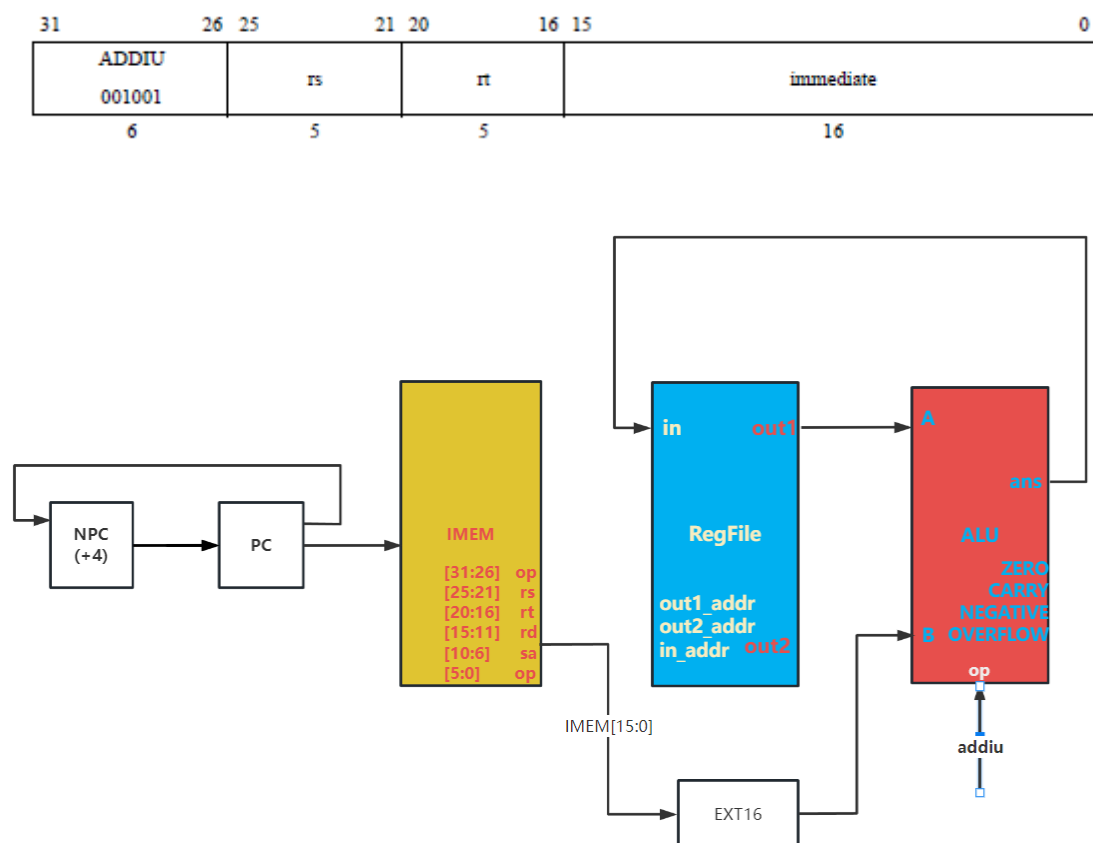
$\text{GPR}[rt] \leftarrow \text{temp}$

endif

```

1 PC -> IMEM
2 PC + 4 -> NPC
3 NPC -> PC
4 IMEM[15:0] -> EXT16
5 EXT16-> B
6 out1 -> A
7 (A + B -> RES)
8 ans -> Rd
    
```

19.addiu



格式: ADDIU rt, rs, immediate

目的: 使32位数据与一个立即数相加

描述: $rt \leftarrow rs + \text{immediate}$

一个16位有符号的立即数与通用寄存器rs中的32位数相加产生一个32位的数存入目标寄存器rt。

在任何情况下都不会有溢出的异常。

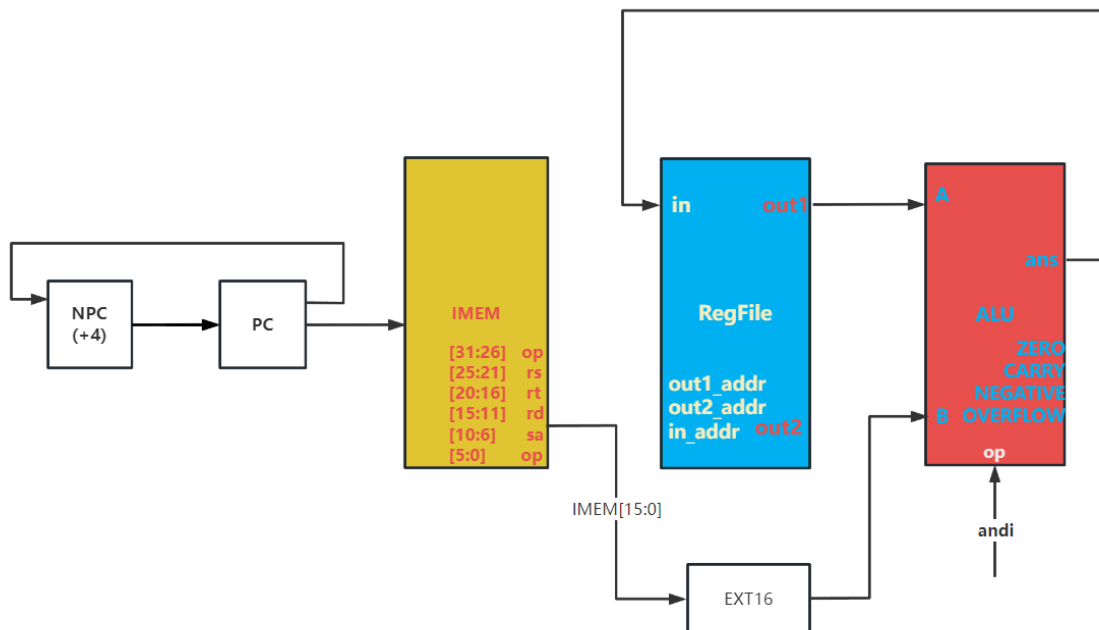
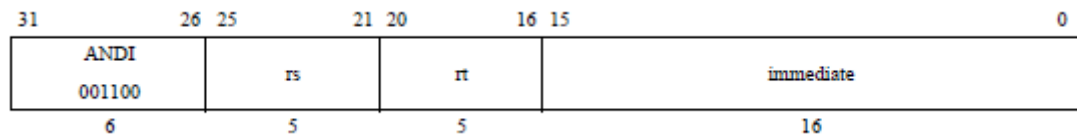
操作：

temp \leftarrow GPR[rs] + sign extend(immediate)

GPR[rt] ← temp

```
1 PC -> IMEM
2 PC + 4 -> NPC
3 NPC -> PC
4 IMEM[15:0] -> EXT16
5 EXT16-> B
6 out1 -> A
7 (A + B -> RES)
8 ans -> Rd
```

20.andi



格式: ANDI rt, rs, immediate

目的: 与一个常数做按位逻辑与

描述: $rt \leftarrow rs \text{ AND immediate}$

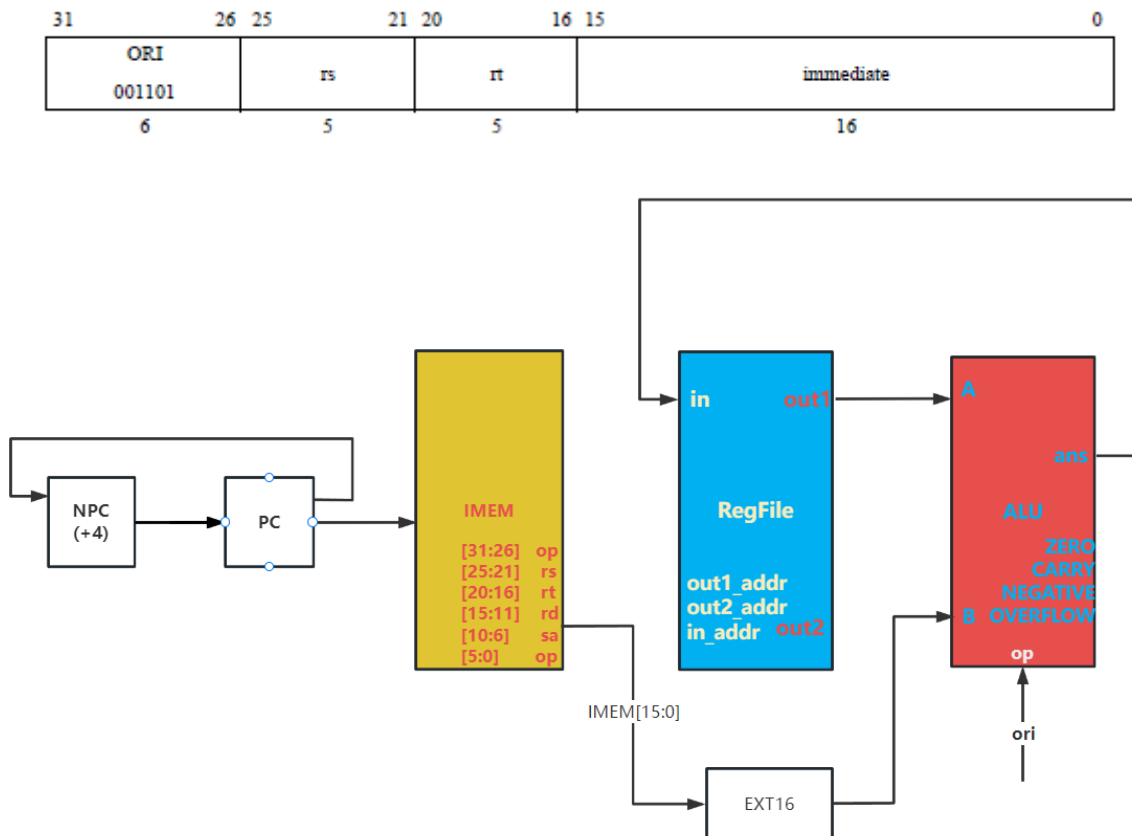
将16位立即数做0扩展后与通用寄存器rs中的32位数据做按位与，将结果存入目标寄存器rt。

操作:

$GPR[rt] \leftarrow GPR[rs] \text{ and zero_extend(immediate)}$

```
1 PC -> IMEM
2 PC + 4 -> NPC
3 NPC -> PC
4 IMEM[15:0] -> EXT16
5 EXT16-> B
6 out1 -> A
7 (A & B -> RES)
8 ans -> Rd
```

21.ori



格式: ORI rt, rs, immediate

目的: 和一个常数做按位逻辑或

描述: $rt \leftarrow rs \text{ or } \text{immediate}$

将通用寄存器rs和经过0扩展的立即数每一位做按位或操作，
将结果存入目标寄存器rd中。

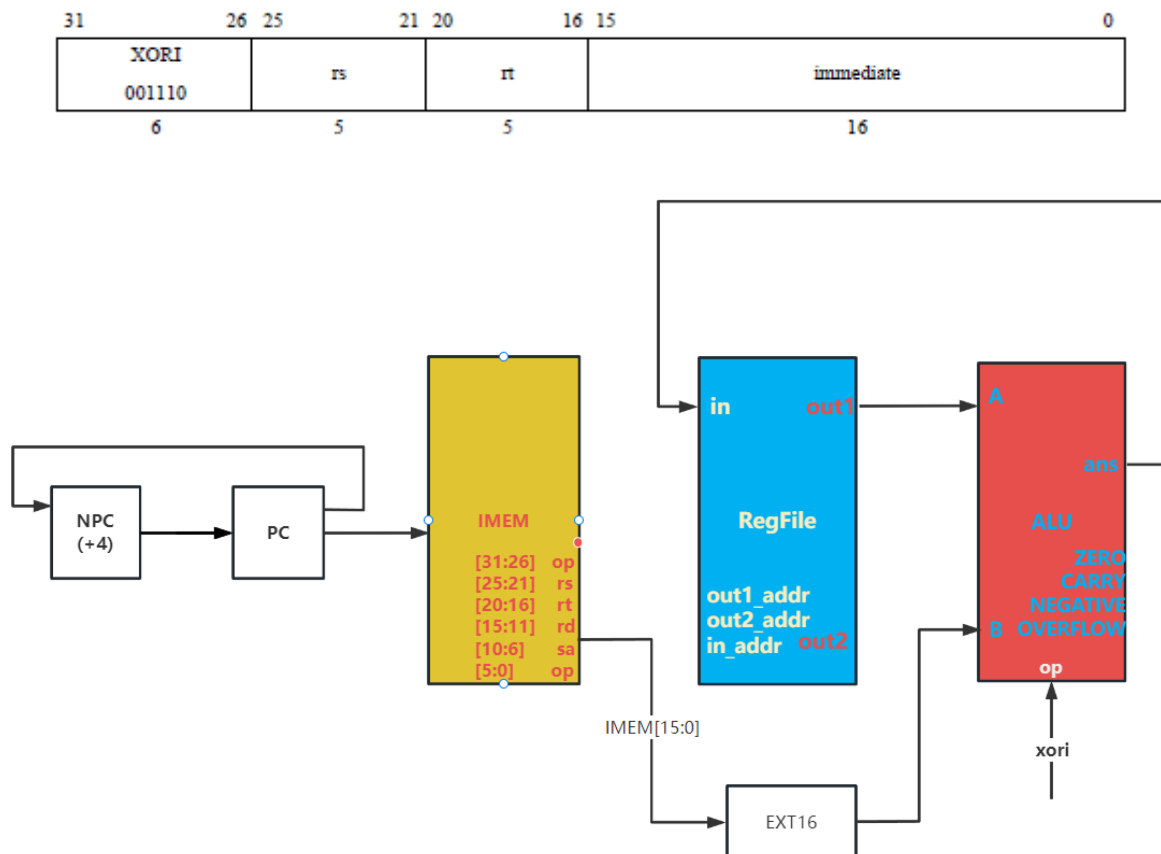
操作:

$GPR[rt] \leftarrow GPR[rs] \text{ or } \text{zero_extend}(\text{immediate})$

```

1 PC -> IMEM
2 PC + 4 -> NPC
3 NPC -> PC
4 IMEM[15:0] -> EXT16
5 EXT16-> B
6 out1 -> A
7 (A | B -> RES)
8 ans -> Rd
    
```

22.xori



格式: XORI rt, rs, immediate

目的: 和一个常数做按位逻辑异或

描述: $rt \leftarrow rs \text{ XOR immediate}$

将通用寄存器rs和经过0扩展的立即数每一位做按位异或操作，将结果存入目标寄存器rd中。

操作:

$GPR[rt] \leftarrow GPR[rs] \text{ xor zero_extend(immediate)}$

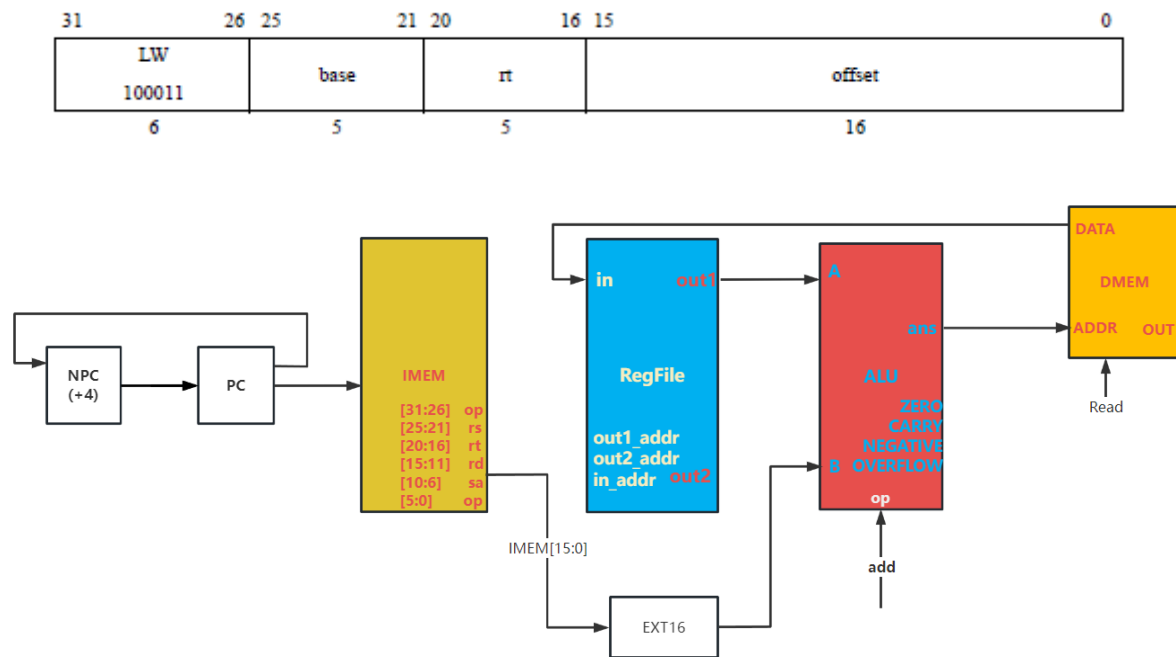
```

1 | PC -> IMEM
2 | PC + 4 -> NPC
3 | NPC -> PC
4 | IMEM[15:0] -> EXT16
5 | EXT16-> B
6 | out1 -> A
7 | (A ⊕ B -> RES)
8 | ans -> Rd

```

通路类型7

23.lw



格式: LW rt, offset(base)

目的: 从内存读取一个字的有符号数据

描述: $rt \leftarrow \text{memory}[\text{base} + \text{offset}]$

从内存中基地址加偏移量所得到的准确地址中的内容加载到通用寄存器rt中。

操作:

$vAddr \leftarrow \text{sign_extend}(\text{offset}) + \text{GPR}[\text{base}]$

if $vAddr_{1..0} \neq 0^2$ then

SignalException(AddressError)

endif

$(pAddr, CCA) \leftarrow \text{AddressTranslation}(vAddr, \text{DATA}, \text{LOAD})$

$\text{memword} \leftarrow \text{LoadMemory}(CCA, \text{WORD}, pAddr, vAddr, \text{DATA})$

$\text{GPR}[rt] \leftarrow \text{memword}$

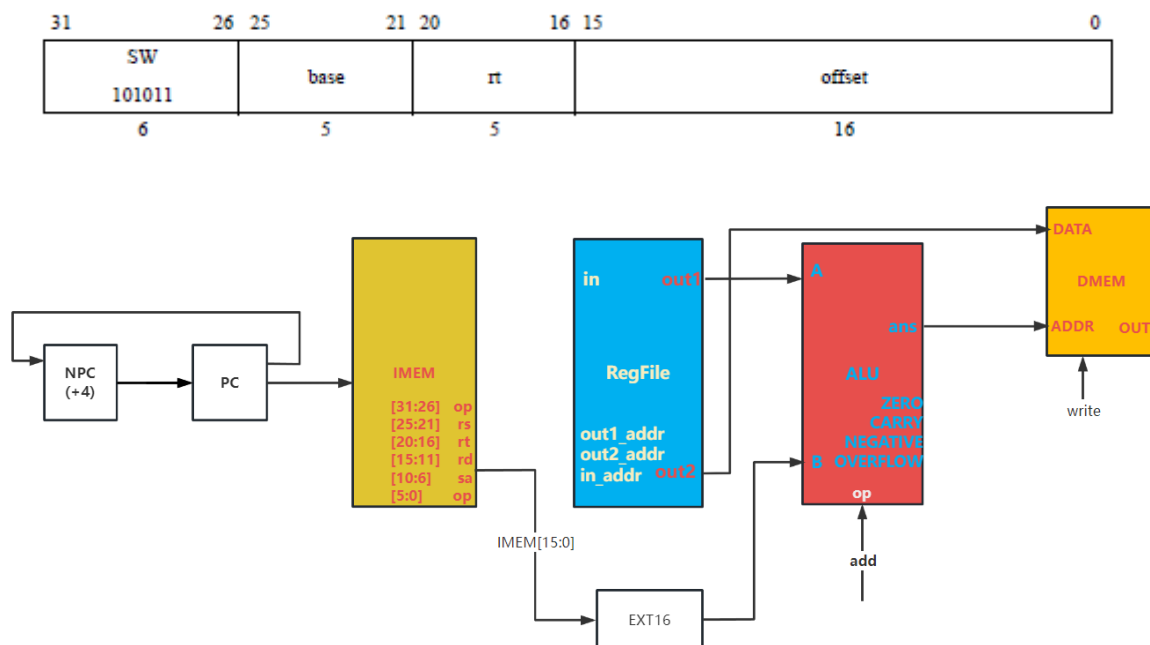
```

1  PC -> IMEM
2  PC + 4 -> NPC
3  NPC -> PC
4  IMEM[15:0] -> EXT16
5  EXT16 -> B
6  out1 -> A
7  (A + B -> ans)
8  ans -> DMEM_ADDR
9  data -> in

```

通路类型8

24.sw



格式: SW rt, offset(base)

目的: 存一个字到内存

描述: $\text{memory}[\text{base} + \text{offset}] \leftarrow \text{rt}$

将通用寄存器rt中的32位数据存入内存中的有效地址，有效地址由基地址和16位偏移量相加所得。

操作:

$\text{vAddr} \leftarrow \text{sign_extend}(\text{offset}) + \text{GPR}[\text{base}]$

if $\text{vAddr}_{1..0} \neq 0^2$ then

SignalException(AddressError)

endif

$(\text{pAddr}, \text{CCA}) \leftarrow \text{AddressTranslation}(\text{vAddr}, \text{DATA}, \text{STORE})$

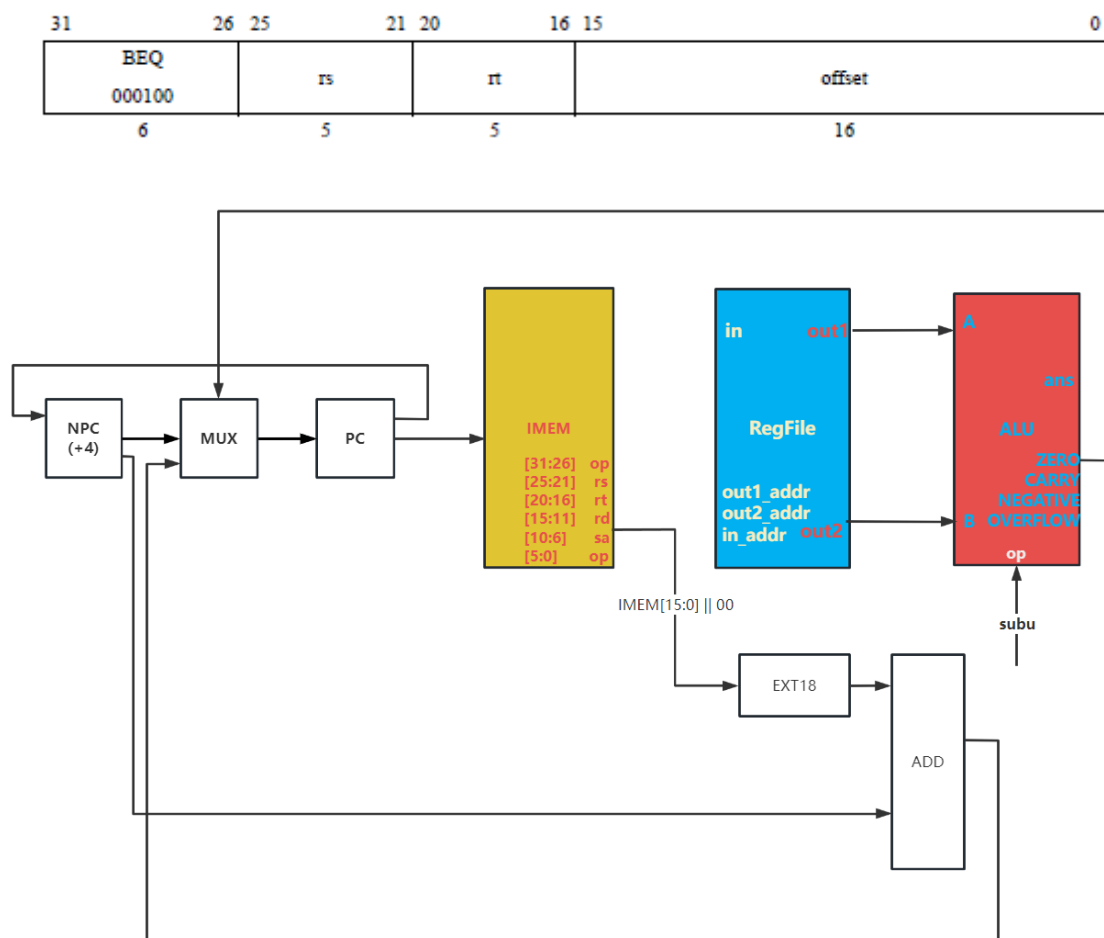
$\text{dataword} \leftarrow \text{GPR}[\text{rt}]$

StoreMemory (CCA, WORD, dataword, pAddr, vAddr, DATA)

```
1 PC -> IMEM
2 PC + 4 -> NPC
3 NPC -> PC
4 IMEM[15:0] -> EXT16
5 EXT16_OUT -> B
6 out1 -> A
7 (A + B -> ans)
8 out2 -> DMEM
9 ans -> DMEM_ADDR
```


通路类型9

25.beq



格式: BEQ rs, rt, offset

目的: 比较通用寄存器的值, 然后做pc相关的分支跳转

描述: 比较通用寄存器的值, 然后做pc相关的分支跳转。

如果 $rs = rt$, 那么将offset左移两位, 再进行符号扩展到32位与当前pc相加, 形成有效转移地址, 转到该地址。

如果 $rs \neq rt$, 则继续执行下条指令。

操作:

I: $target_offset \leftarrow sign_extend(offset \ll 2)$

$condition \leftarrow (GPR[rs] = GPR[rt])$

I+1: if condition then

$PC \leftarrow PC + target_offset$

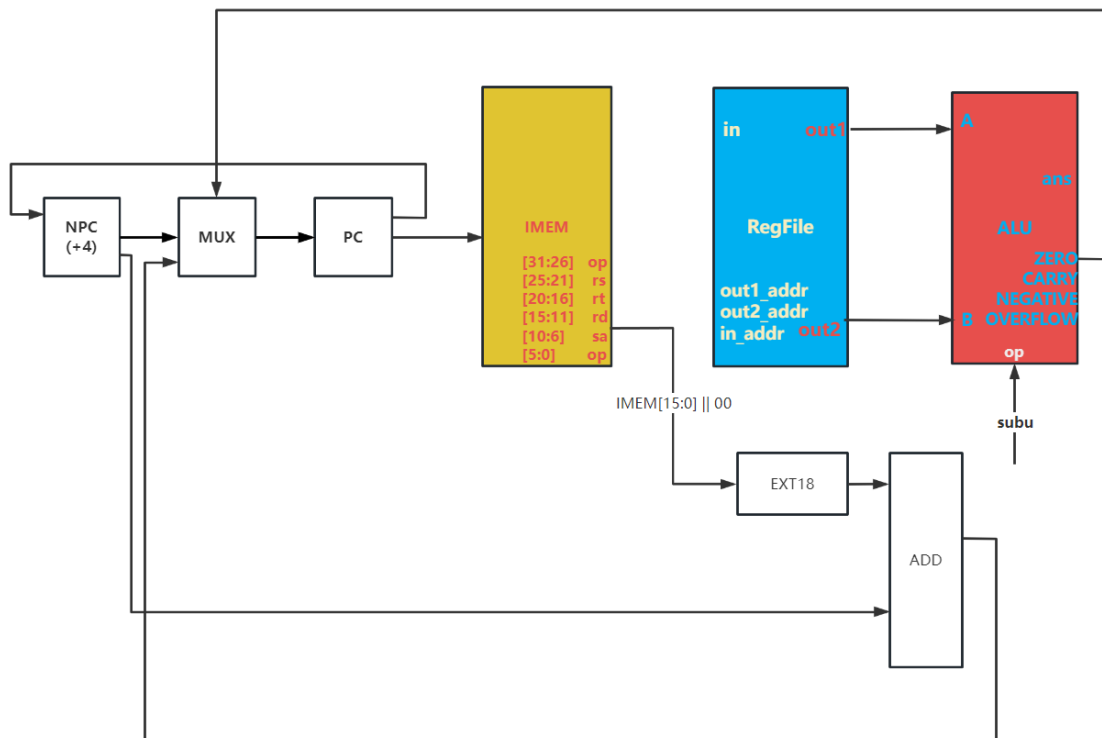
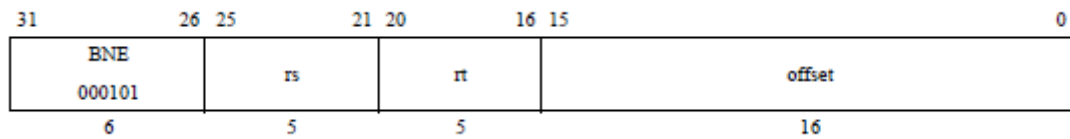
endif

```

1  PC -> IMEM
2  PC + 4 -> NPC
3  NPC -> MUX
4  IMEM[15:0] || 02 -> EXT18
5  EXT18_OUT -> ADD_A
6  NPC -> ADD_B
7  (ADD_A + ADD_B -> ADD_OUT)
8  ADD_OUT -> MUX
9  out1 -> A
10 out2 -> B
11 (A - B -> RES)
12 zero -> MUX
13 MUX -> PC

```

26.bne



格式: BNE rs, rt, offset

目的: 比较通用寄存器的值, 然后做pc相关的分支跳转

描述: 如果 $rs \neq rt$, 那么将会跳转到现在pc与偏移量offset (如果是16位需扩展到18位) 相加后所得的指令。

如果 $rs = rt$, 则继续执行。

操作:

I: $target_offset \leftarrow sign_extend(offset \parallel 0^2)$

condition $\leftarrow (GPR[rs] \neq GPR[rt])$

I+1: if condition then

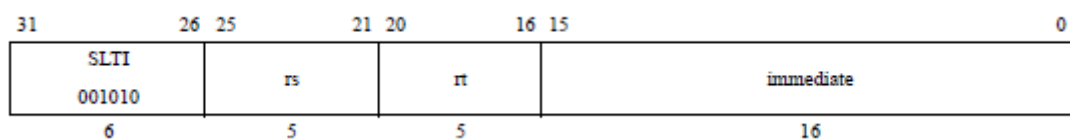
PC $\leftarrow PC + target_offset$

endif

```
1  PC -> IMEM
2  PC + 4 -> NPC
3  NPC -> MUX
4  IMEM[15:0] || 02 -> EXT18
5  EXT18_OUT -> ADD_A
6  NPC -> ADD_B
7  (ADD_A + ADD_B -> ADD_OUT)
8  ADD_OUT -> MUX
9  out1 -> A
10 out2 -> B
11 (A - B -> RES)
12 zero -> MUX
13 MUX -> PC
```

通路类型10

27.slti

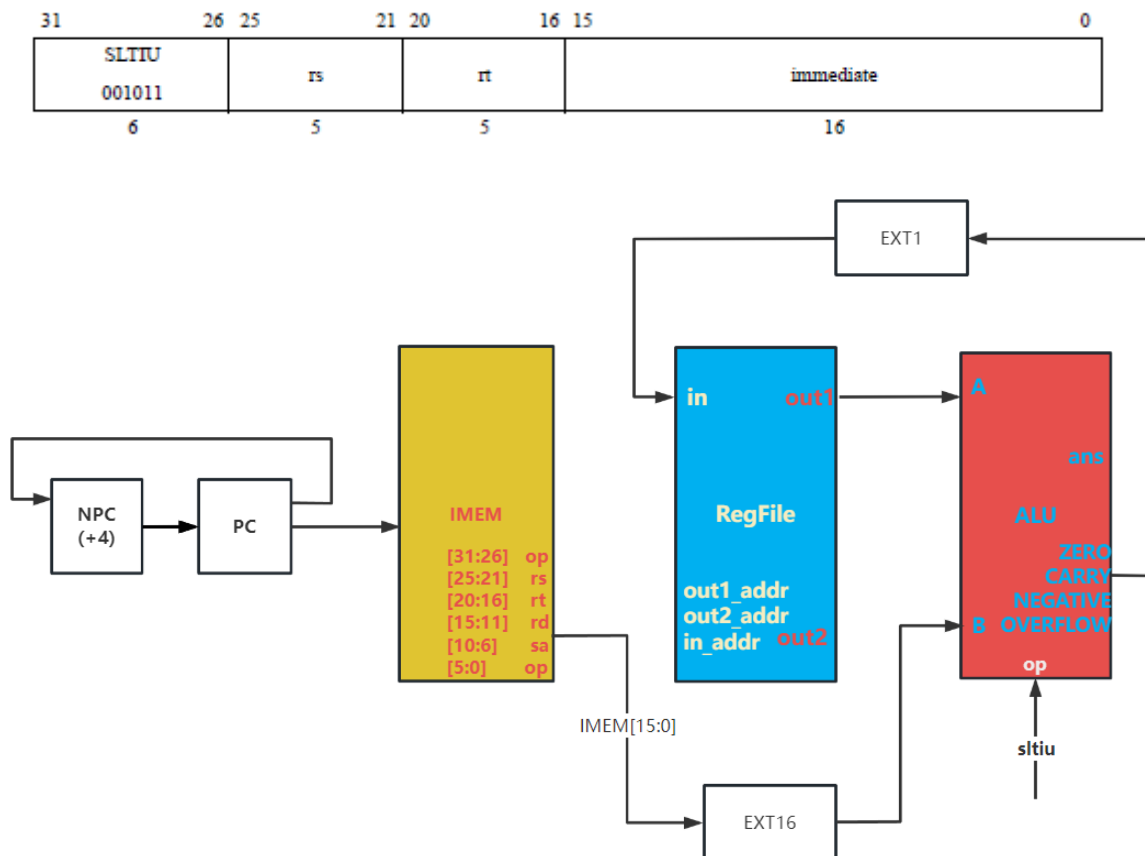


```

1 PC -> IMEM
2 PC + 4 -> NPC
3 NPC -> PC
4 IMEM[15:0] -> EXT16
5 EXT16 -> B
6 out1 -> A
7 (A - B -> RES)
8 Carry -> EXT1
9 EXT1 -> in

```

28.sltiu



格式: SLTIU rt, rs, immediate

目的: 通过跟立即数进行无符号小于的比较来记录结果

描述: $rt \leftarrow (rs < \text{immediate})$

把rs中的数和经过符号扩展的16位立即数进行无符号小于比较，用boolean值保存结果到rd寄存器中。如果rs小于符号扩展的立即数，则结果为1，反之结果为0。算数比较不会引起溢出异常。

操作:

if (0 || GPR[rs]) < (0 || sign_extend(immediate)) then

GPR[rd] \leftarrow 0^{GPREN-1} || 1

else

GPR[rd] \leftarrow 0^{GPREN}

endif

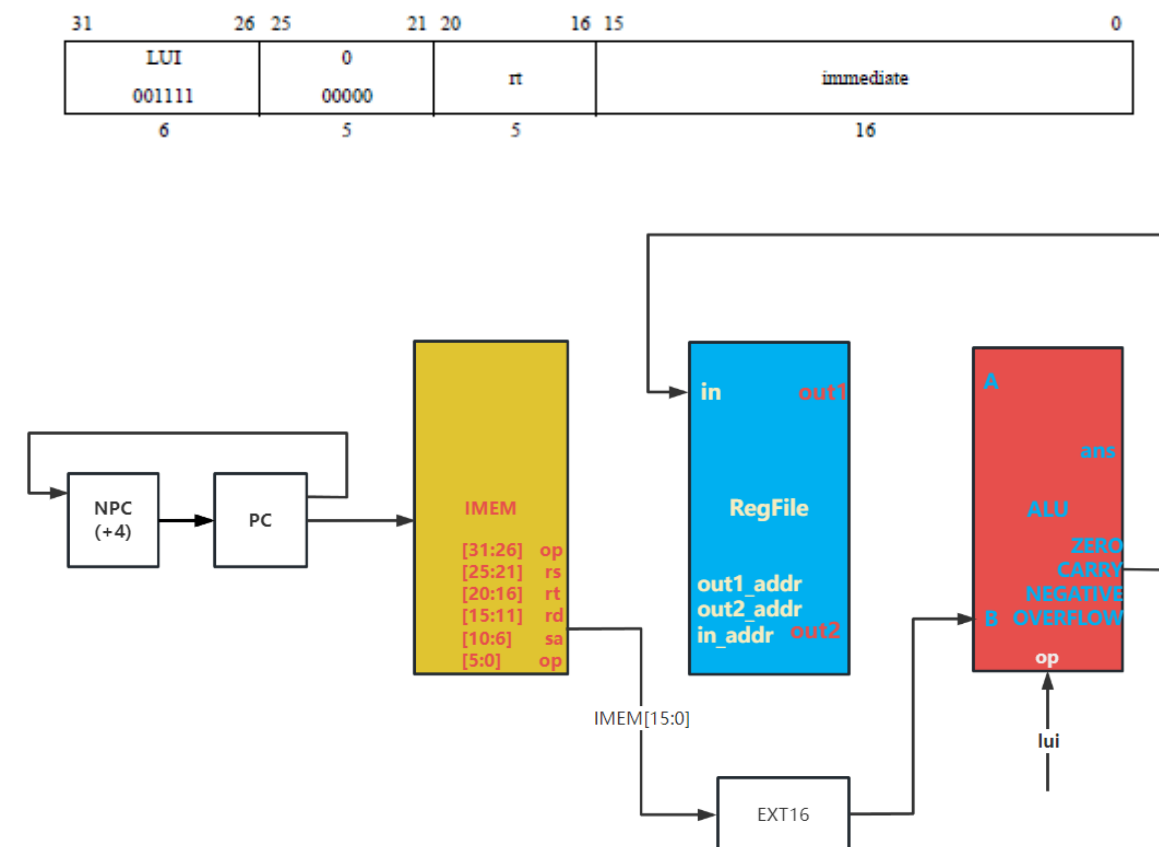
```

1 PC -> IMEM
2 PC + 4 -> NPC
3 NPC -> PC
4 IMEM[15:0] -> EXT16
5 EXT16 -> B
6 out1 -> A
7 (A - B -> ans)
8 Carry -> EXT1
9 EXT1 -> in

```

通路类型11

29.lui



格式: LUI rt, immediate

目的: 把一个立即数载入到寄存器的高位, 低位补0

描述: $rt \leftarrow \text{immediate} \parallel 0^{16}$

将一个16位的立即数载入到通用寄存器rt的高位, 低16位补0。

操作:

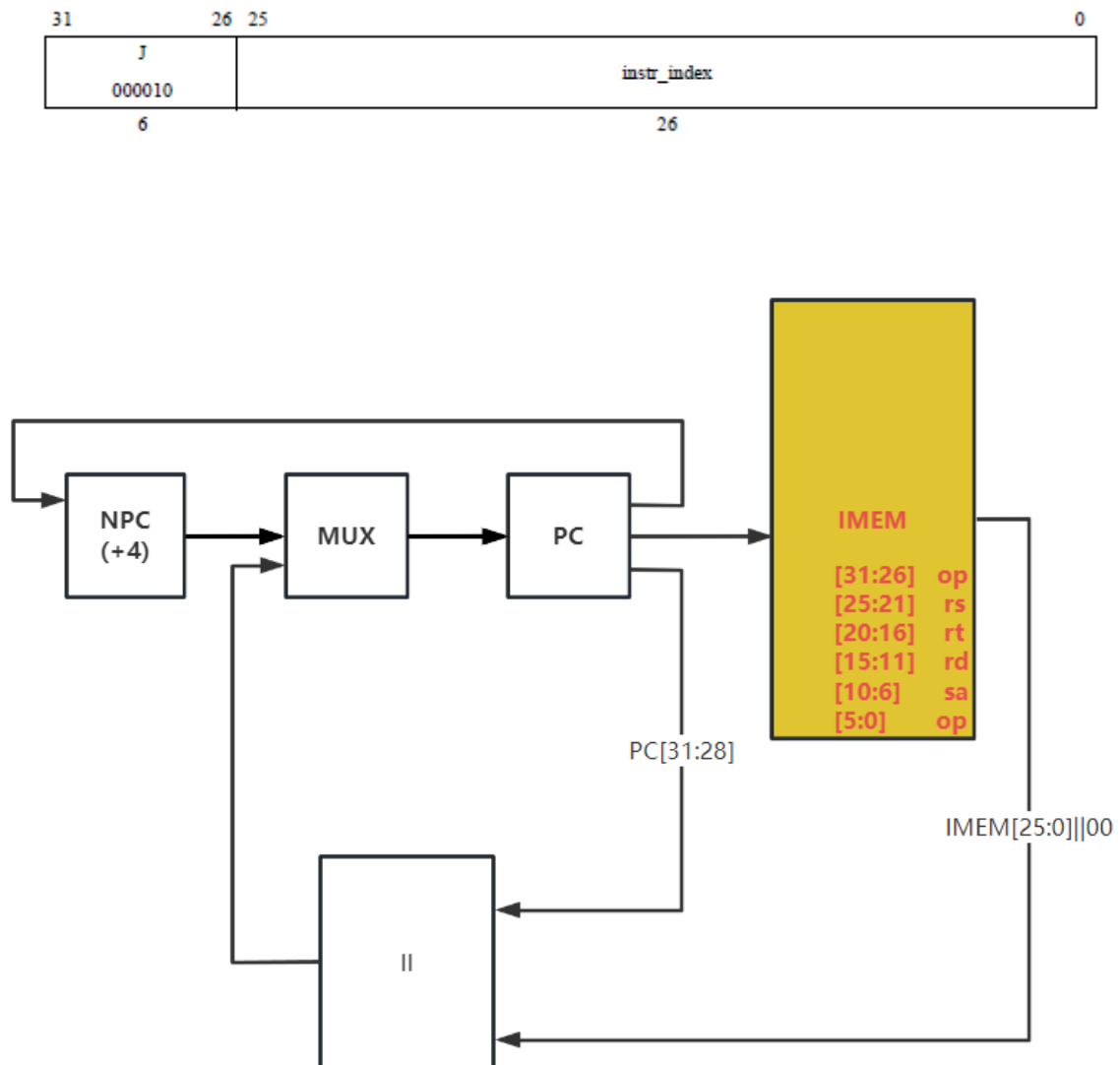
$\text{GPR}[rt] \leftarrow \text{immediate} \parallel 0^{16}$

```
1 PC -> IMEM
2 PC + 4 -> NPC
3 NPC -> PC
4 IMEM[15:0] -> EXT16
5 EXT16 -> B
6 res -> in
```

J 型指令

通路类型12

30.j



格式: J target

目的: 在256MB的范围内跳转

描述: 该指令无条件跳转到一个绝对地址, instr_index有26位, 在左移过后访问空间能达到256M。

操作:

I:

I+1: $PC \leftarrow PC_{\text{GPRLEN}-1..28} \parallel \text{instr_index} \parallel 0^2$

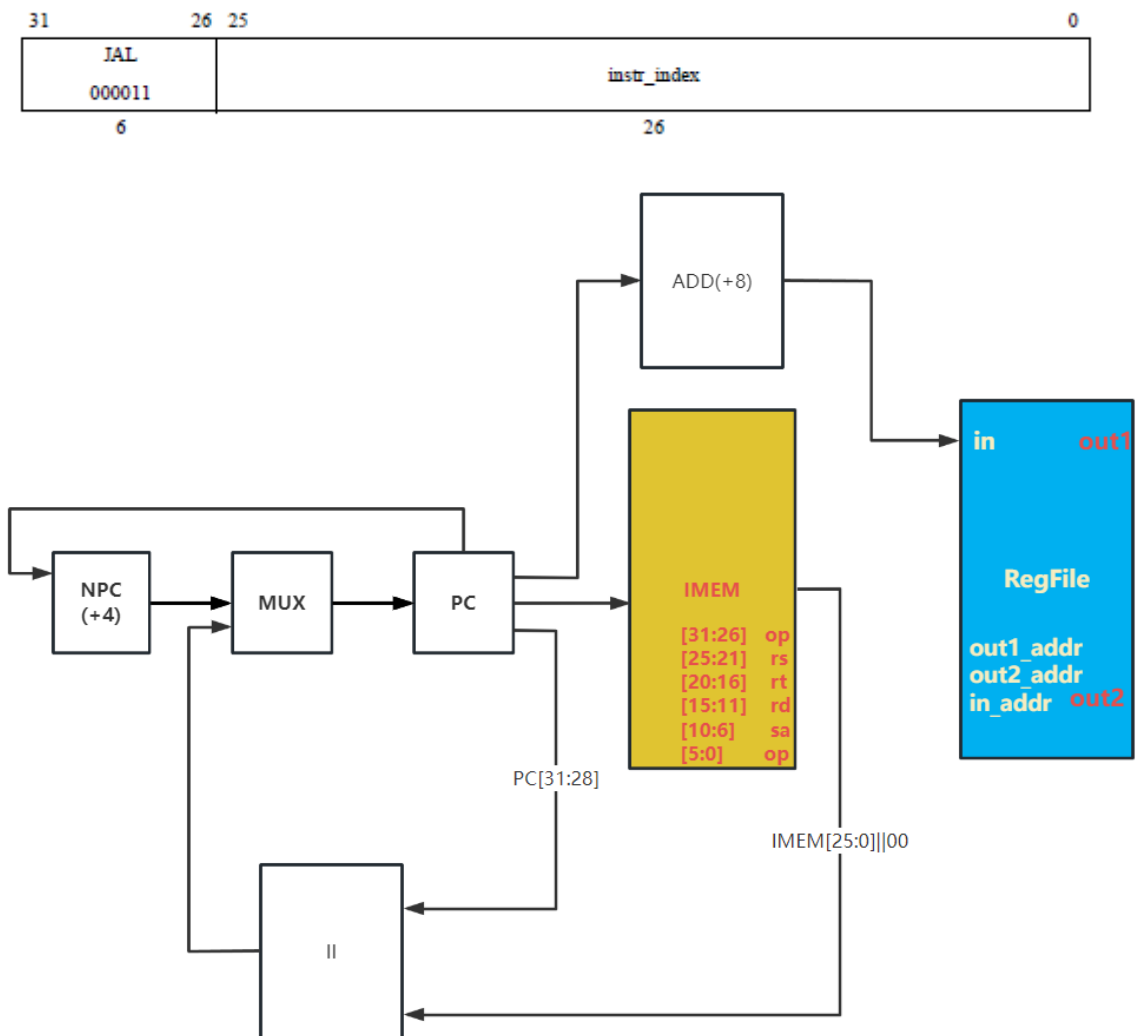
```

1 PC -> IMEM
2 PC + 4 -> NPC
3 NPC -> MUX
4 PC[31:28] -> ||_A
5 IMEM[25,0] || 00 -> ||_B
6 ||_OUT -> MUX
7 MUX_OUT -> PC

```

通路类型13

31.jal



格式: JAL target

目的: 在256MB范围内执行一个过程调用

描述: 在跳转到指定地址执行子程序调用的同时, 在31号寄存器中存放返回地址 (当前地址后的第二条指令地址)。

操作:

I: $GPR[31] \leftarrow PC + 8$

I+1: $PC \leftarrow PC_{GPRLEN-1..28} || instr_index || 0^2$

```
1 | PC -> IMEM
2 | PC + 4 -> NPC
3 | NPC -> MUX
4 | 8 -> ADD_A
5 | PC -> ADD_B
6 | (ADD_A + ADD_B -> ADD_OUT)
7 | ADD_OUT -> Rd
8 | PC[31:28] -> ||_A
9 | IMEM[25,0] || 02 -> ||_B
10 | ||_OUT -> MUX
11 | MUX_OUT -> PC
```