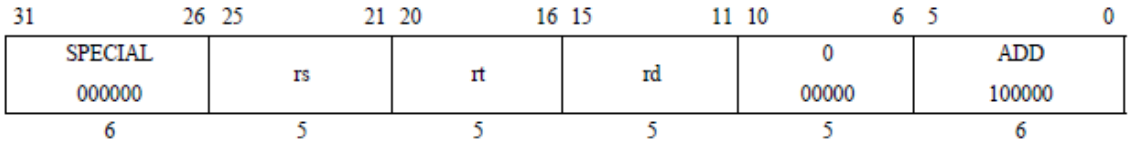
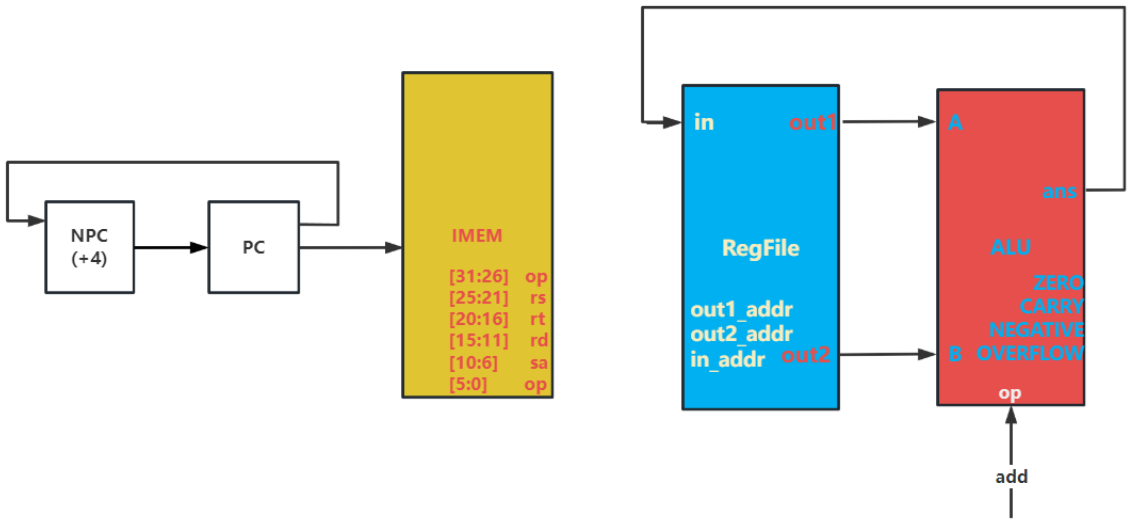
R 型指令

通路类型1

1.ADD





格式： ADD rd, rs, rt

目的：32位数相加

描述: rd ← 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 ←(GPR[rs]$_{31}$||GPR[rs]$_{31..0}$) + (GPR[rt]$_{31}$||GPR[rt]$_{31..0}$)

if temp$_{32}$ ≠temp$_{31}$ then

SignalException(IntegerOverflow)

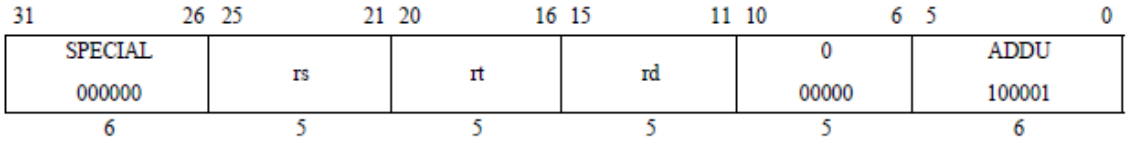
else

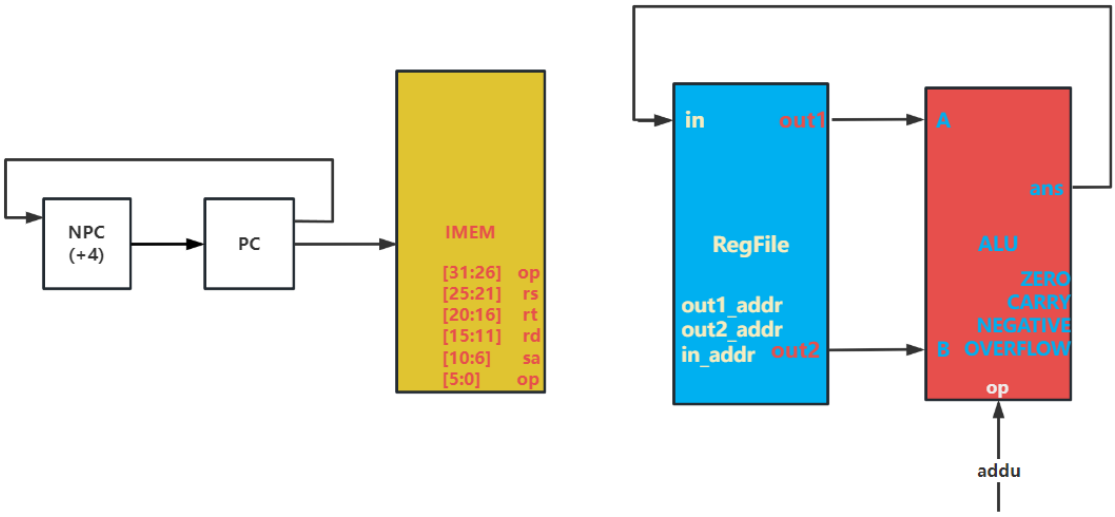
GPR[rd] ←temp

Endif

```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
out1 -> A , out2 -> B
(A + B -> RES)
ans -> in
```

2.ADDU





格式：ADDU rd, rs, rt

目的：32位数据相加

描述：rd ← 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。

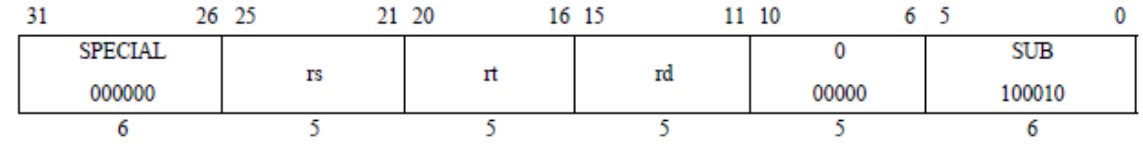
操作：

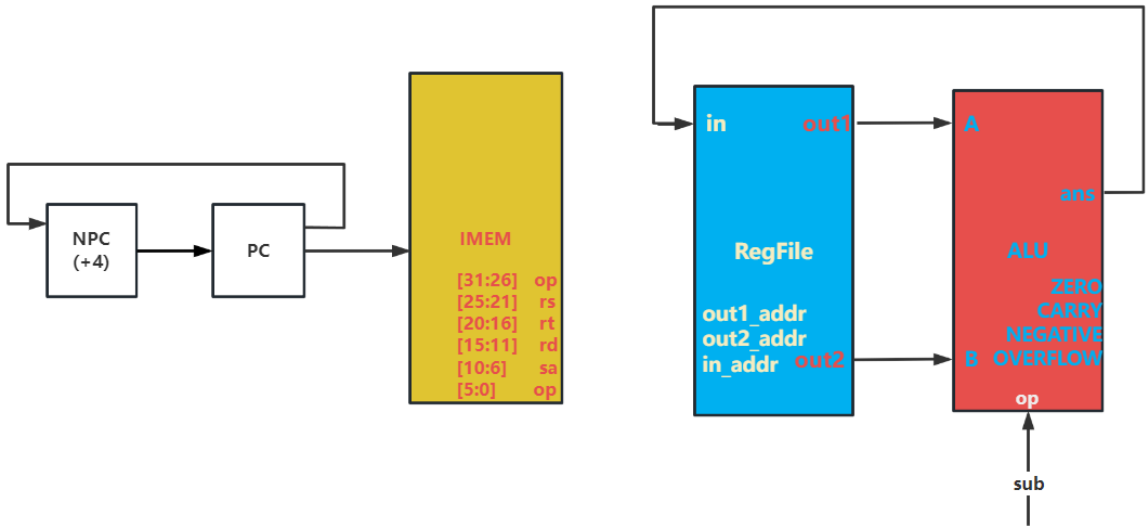
temp ← GPR[rs] + GPR[rt]

GPR[rd] ← temp

```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
out1 -> A , out2 -> B
(A + B -> RES)
ans -> in
```

3.SUB





格式：SUB rd, rs, rt

目的：与32位数相减

描述：rd ←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 ← (GPR[rs]$_{31}$||GPR[rs]$_{31..0}$) - (GPR[rt]$_{31}$||GPR[rt]$_{31..0}$)

if temp$_{32}$ ≠ temp$_{31}$ then

SignalException(IntegerOverflow)

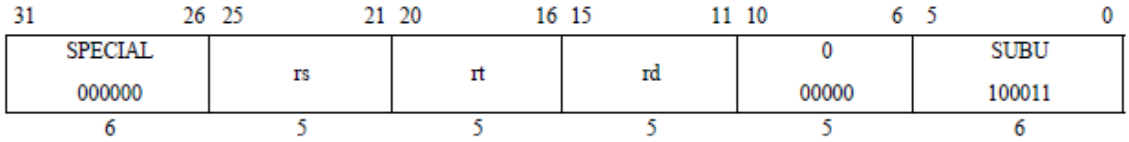
else

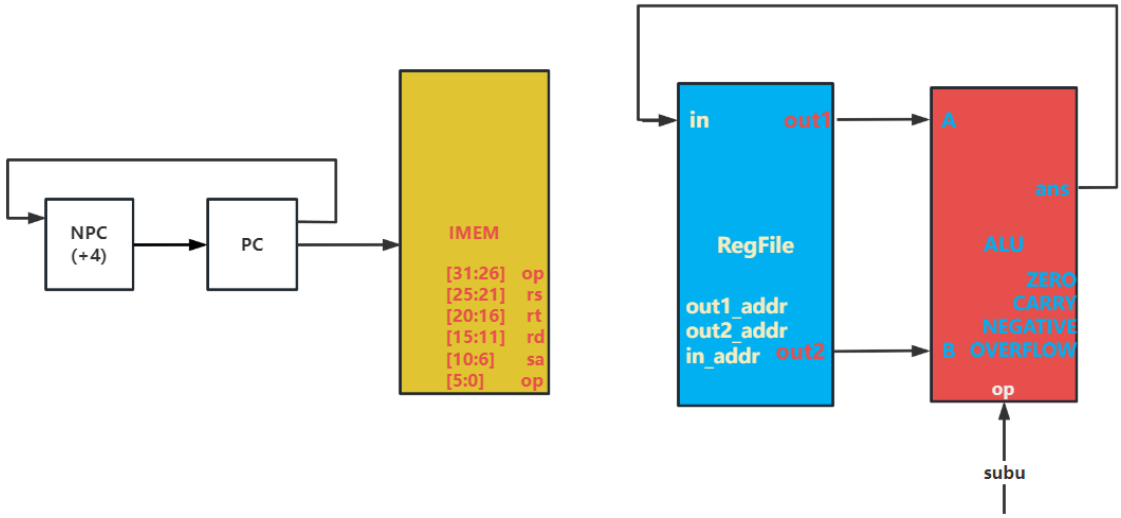
GPR[rd] ← temp$_{31..0}$

endif

```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
out1 -> A , out2 -> B
(A - B -> RES)
ans -> in
```

4.SUBU





格式：SUBU rd, rs, rt

目的：32位数据相减

描述：rd ←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。

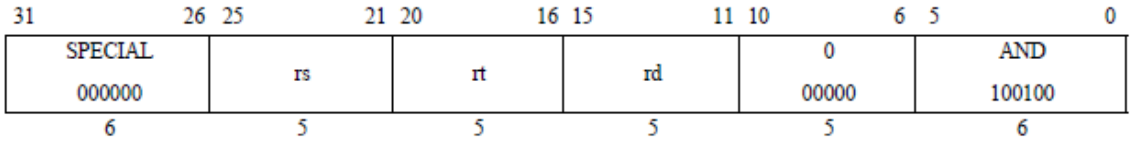
操作：

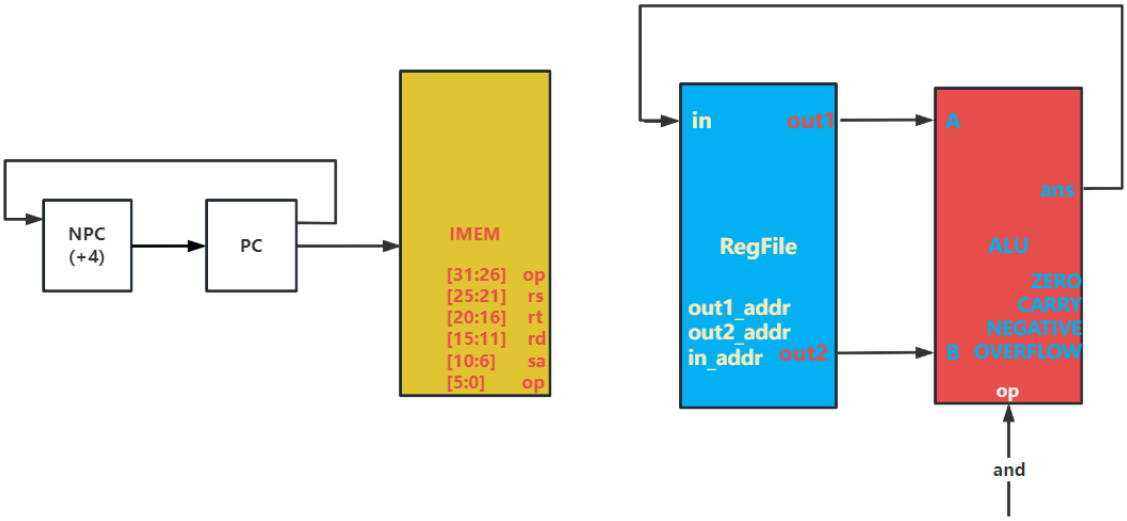
temp ← GPR[rs] - GPR[rt]

GPR[rd] ← temp

```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
out1 -> A , out2 -> B
(A - B -> RES)
ans -> in
```

5.AND





格式：AND rd, rs, rt

目的：按位逻辑与

描述：rd ← rs AND rt

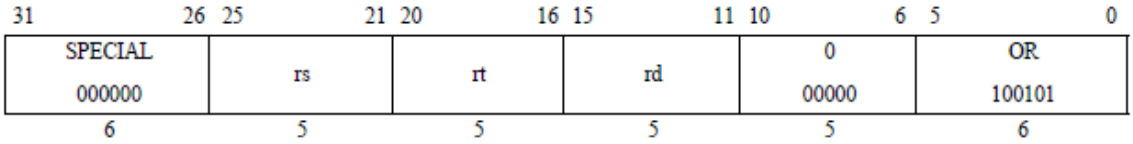
将通用寄存器rs和rd中的数据每一位做按位与操作，将结果存入目标寄存器rd中。

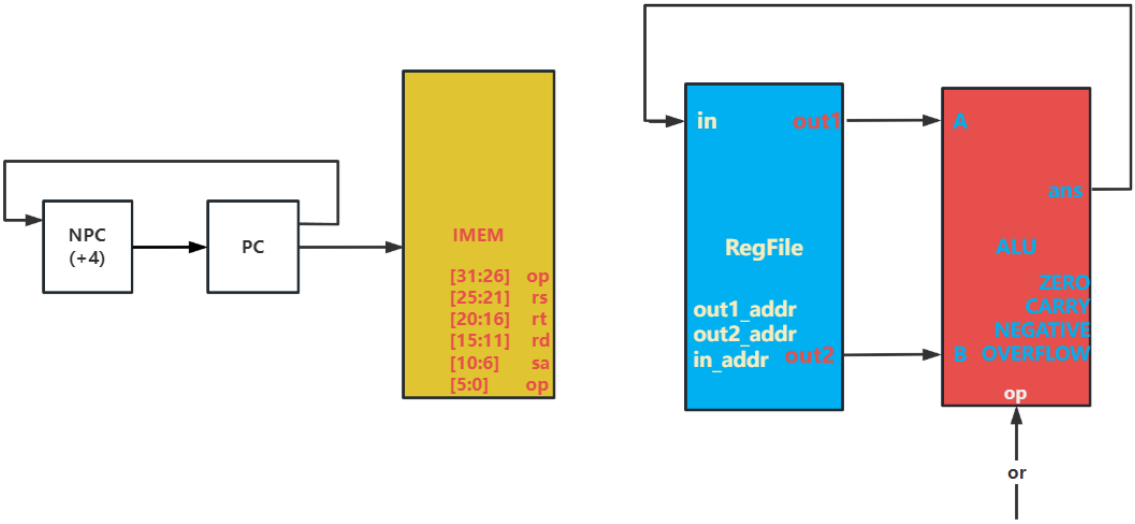
操作：

GPR[rd] ← GPR[rs] and GPR[rt]

```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
out1 -> A , out2 -> B
(A & B -> RES)
ans -> in
```

6.OR





格式：OR rd, rs, rt

目的：按位逻辑或

描述：rd ← rs 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。

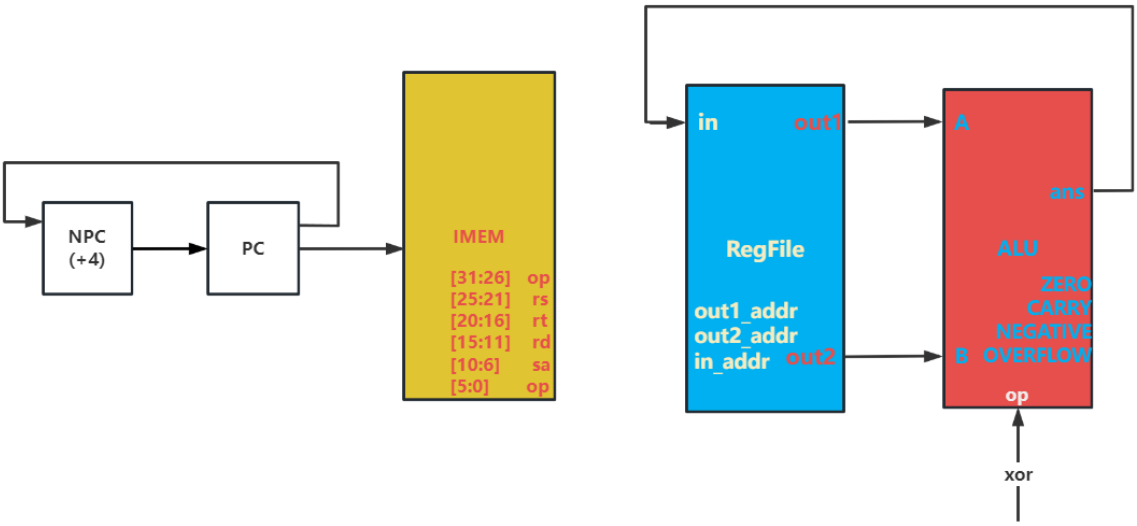
操作：

GPR[rd] ← GPR[rs] or GPR[rt]

```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
out1 -> A , out2 -> B
(A | B -> RES)
ans -> in
```

7.XOR

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 000000	rs	rt	rd	0 00000	XOR 100110
6	5	5	5	5	6



格式：XOR rd, rs, rt

目的：按位逻辑异或

描述：rd ←rs XOR rt

将通用寄存器rs和rt中的内容按位进行异或操作，将结果存入rd中。

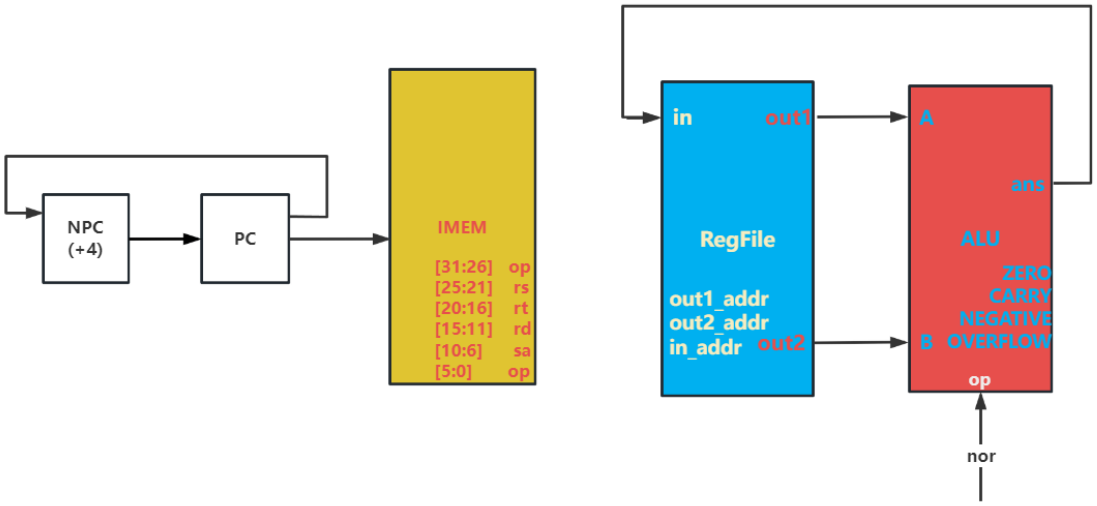
操作：

GPR[rd] ← GPR[rs] xor GPR[rt]

```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
out1 -> A , out2 -> B
(A ⊕ B -> RES)
ans -> in
```

8. NOR

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 000000	rs	rt	rd	0 00000	NOR 100111
6	5	5	5	5	6



格式：NOR rd, rs, rt

目的：按位逻辑或非

描述：rd ← rs 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：

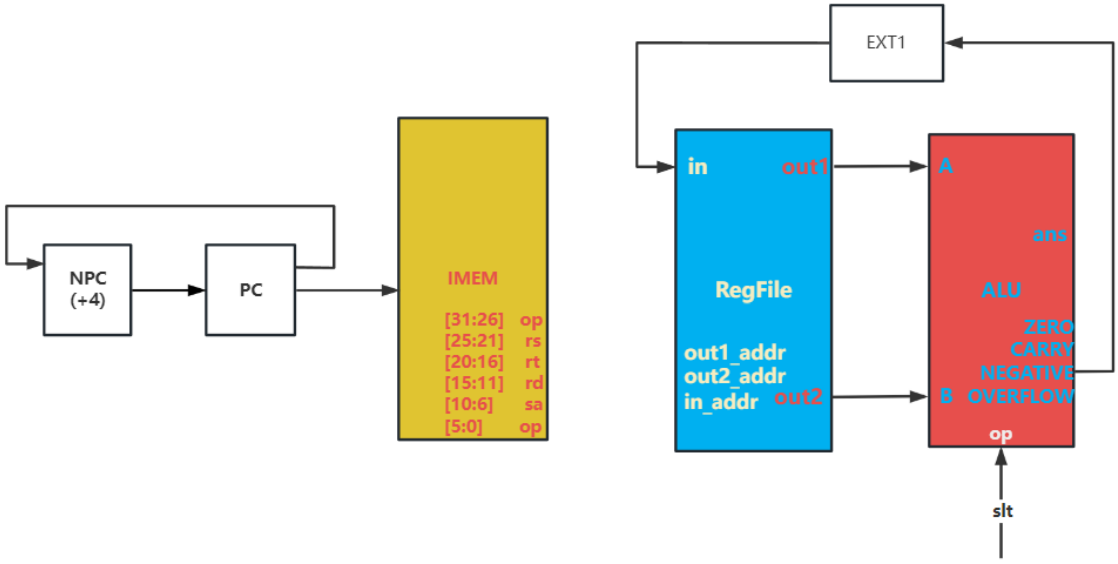
GPR[rd] ← GPR[rs] nor GPR[rt]

```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
out1 -> A , out2 -> B
(A ⊙ B -> RES)
ans -> in
```

通路类型2

9.SLT

31–26	25–21	20–16	15–11	10–6	5–0
SPECIAL 000000	rs	rt	rd	0 00000	SLT 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 ←(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] ← $0^{GPRLEN-1}$ || 1

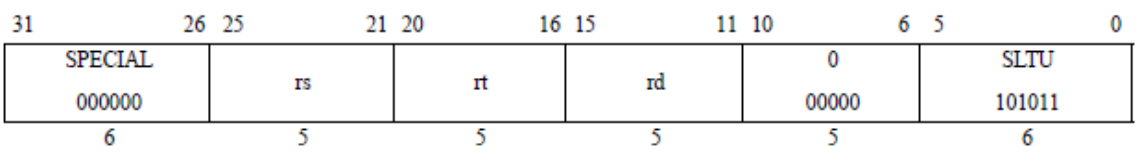
else

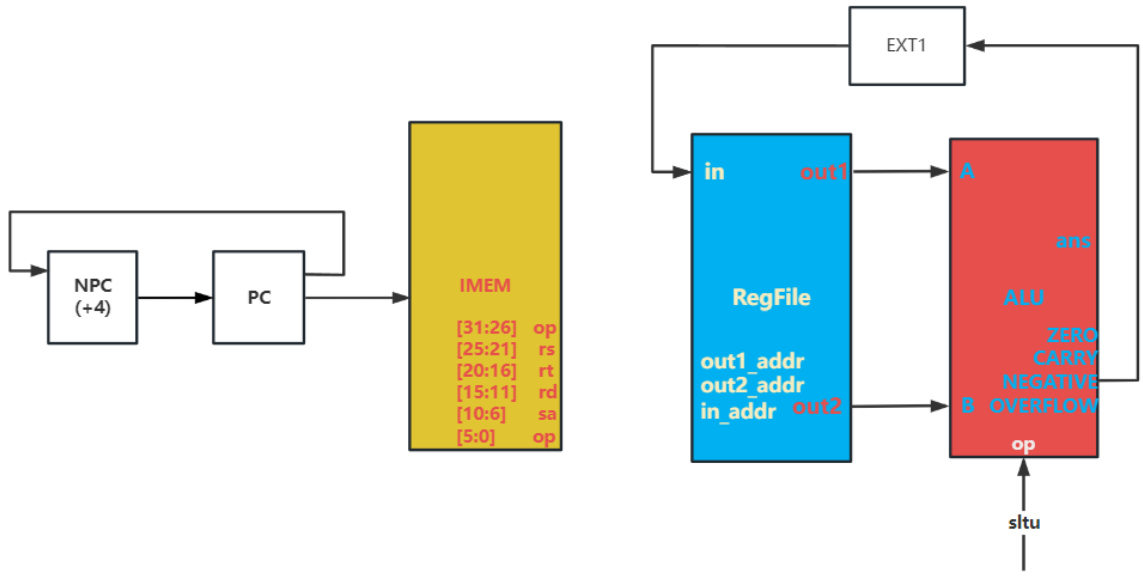
GPR[rd] ← 0^{GPRLEN}

endif

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

10.SLTU





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 ←(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] ← $0^{GPRLEN-1}$ || 1

else

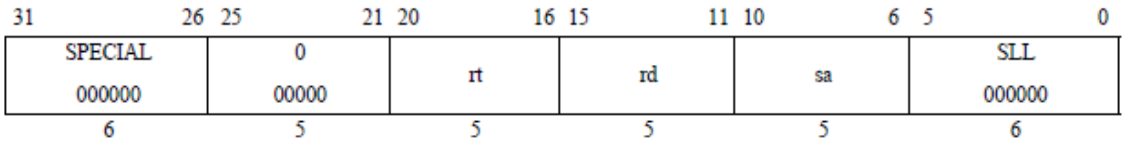
GPR[rd] ← 0^{GPRLEN}

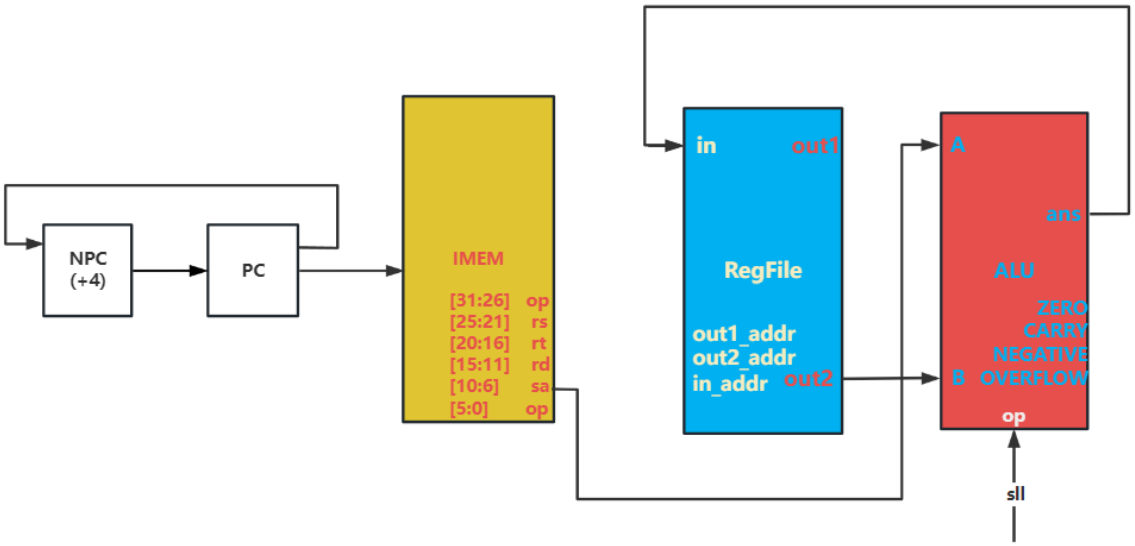
endif

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

通路类型3

11.SLL





格式：SLL rd, rt, sa

目的：通过数字填充逻辑左移

描述：rd ← rt << 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：

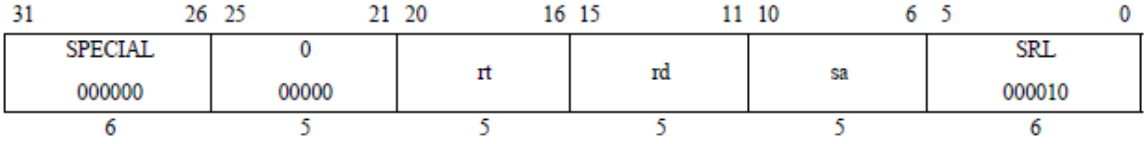
s ← sa

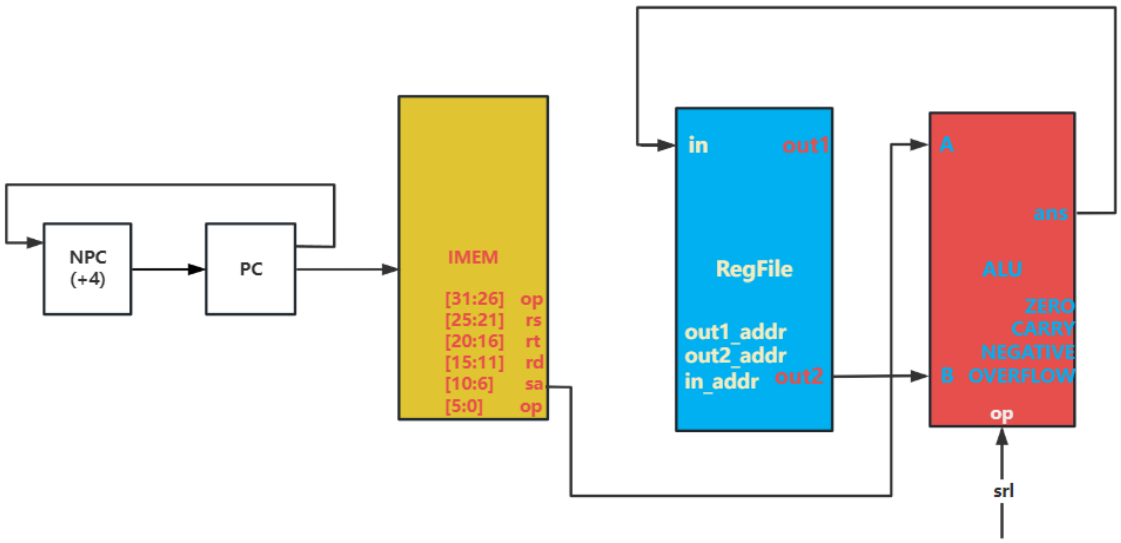
temp ← GPR[rt]$_{(31-s)..0}$ || 0^s

GPR[rd] ← temp

```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
IMEM[10:6] -> EXT5
EXT5 -> A
out2 -> B
(B<<A -> ans)
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 ←rt >> 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：

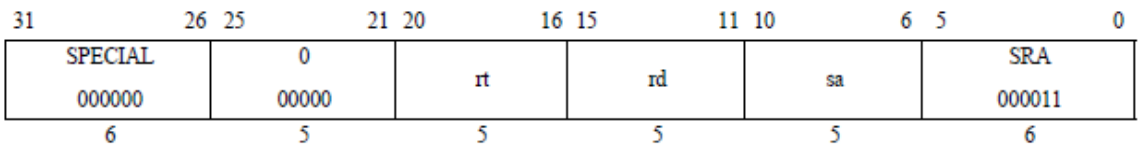
s ← sa

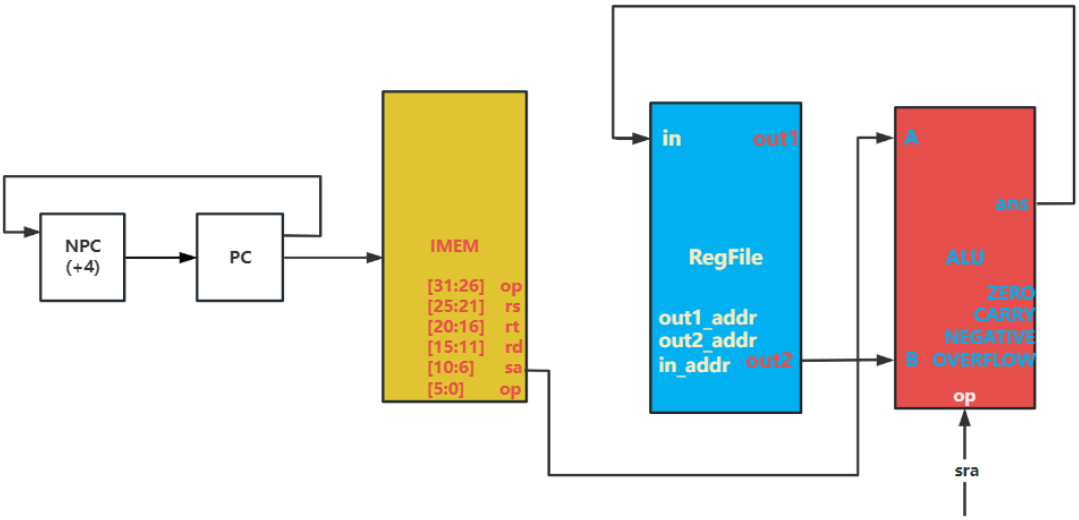
temp ← 0^s || GPR[rt]$_{31..s}$

GPR[rd] ← temp

```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
IMEM[10:6] -> EXT5
EXT5 -> A
out2 -> B
(B>>A -> ans)
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 ←rt >> 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 ← sa

temp ← $(GPR[rt]31)^s$ || $GPR[rt]_{31..s}$

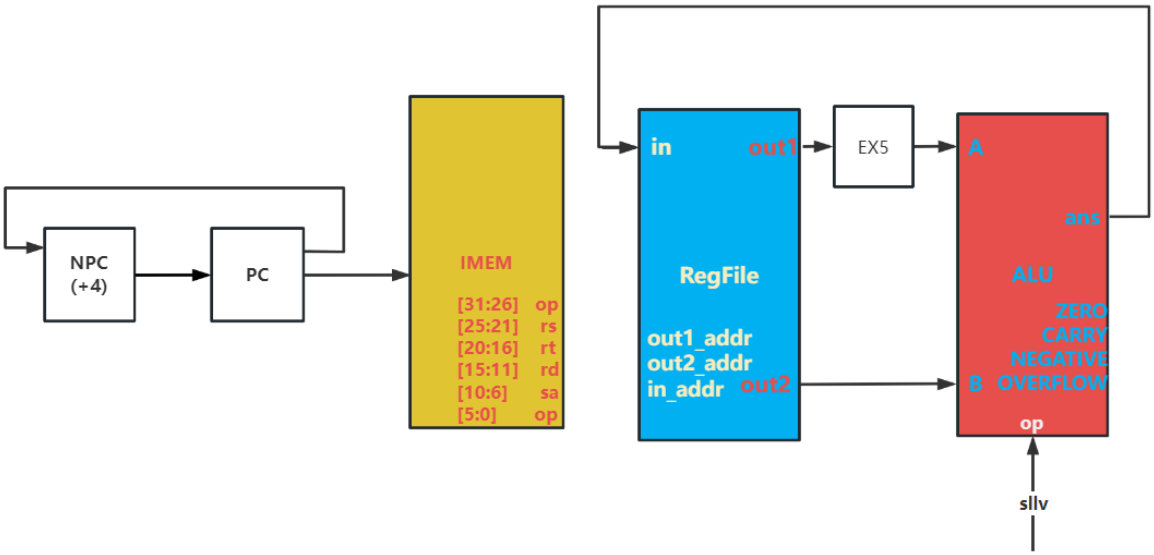
GPR[rd] ← temp

```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
IMEM[10:6] -> EXT5
EXT5 -> A
out2 -> B
(B>>A -> ans)
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 ← rt << 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：

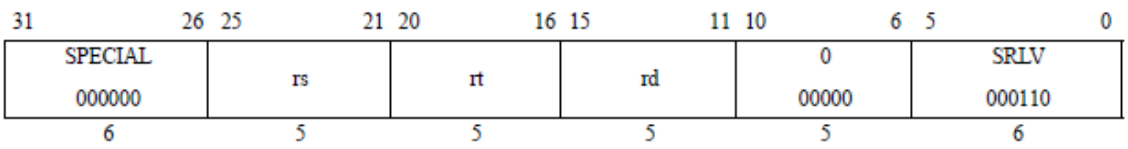
$s \leftarrow GPR[rs]_{4..0}$

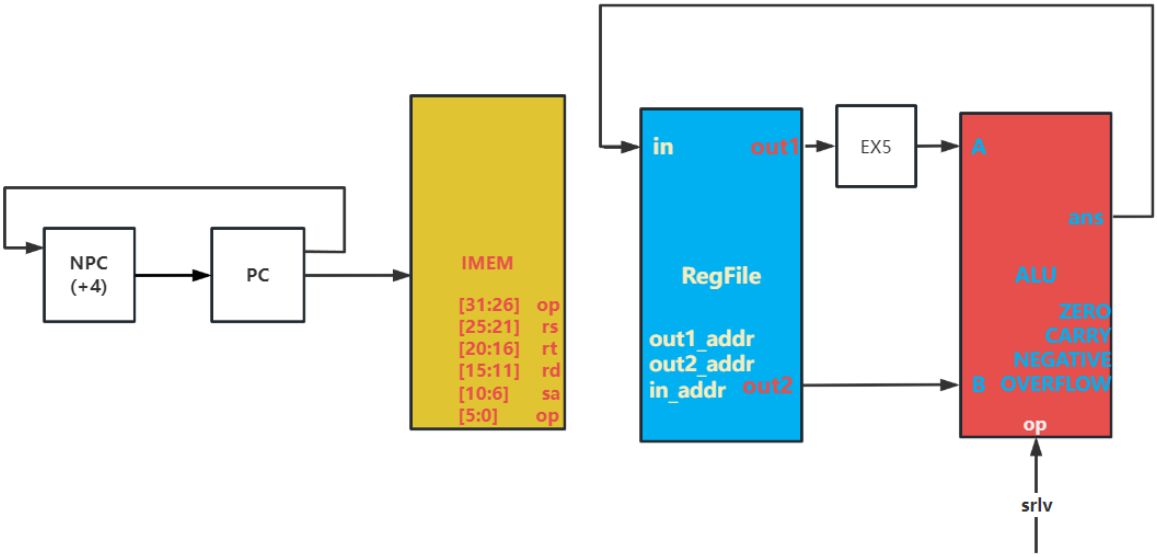
$temp \leftarrow GPR[rt]_{(31-s)..0} \,||\, 0^s$

$GPR[rd] \leftarrow temp$

```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
out1[4:0] -> EXT5
EXT5 -> A
out2 -> B
(A<<B -> ans)
ans -> in
```

15.SRLV





格式：SRLV rd, rt, rs

目的：通过数字填充逻辑右移

描述：rd ←rt >> 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 ← GPR[rs]$_{4..0}$

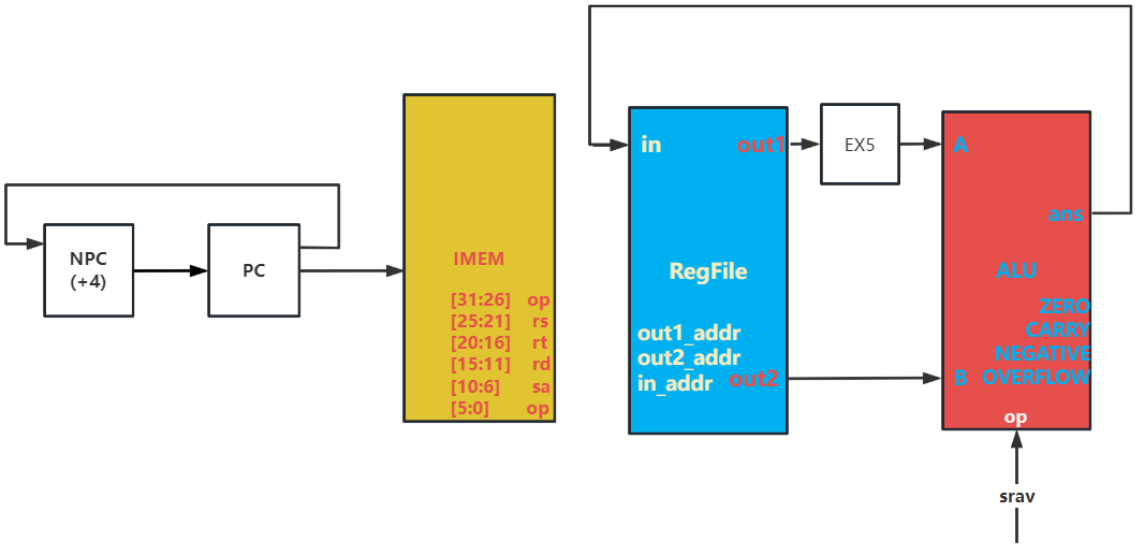
temp ← 0^s || GPR[rt]$_{31..s}$

GPR[rd] ← temp

```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
out1[4:0] -> EXT5
EXT5 -> A
out2 -> B
(A >> B -> ans)
ans -> in
```

16.SRAV

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



格式：SRAV rd, rt, rs

目的：通过数字填充算术右移

描述：rd ←rt >> 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 ← GPR[rs]$_{4..0}$

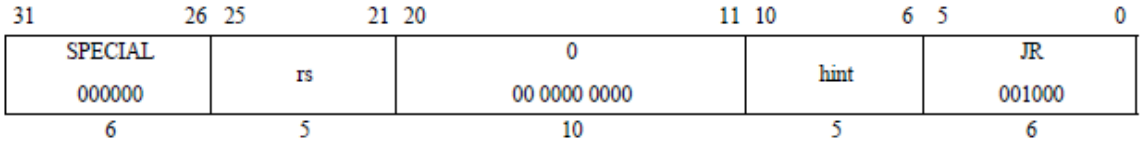
temp ← (GPR[rt]31)^{s} || GPR[rt]$_{31..s}$

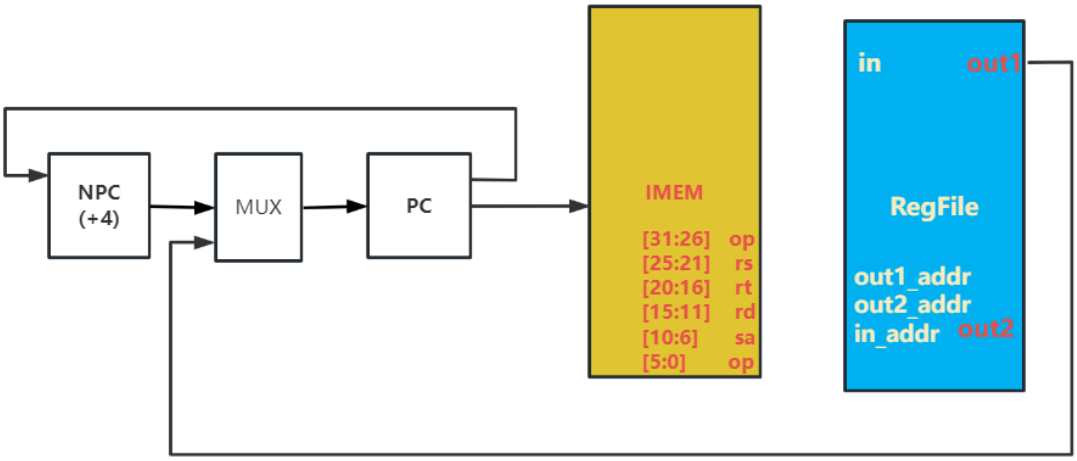
GPR[rd] ← temp

```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
out1[4:0] -> EXT5
EXT5 -> A
out2 -> B
(A >> B -> ans)
ans -> in
```

通路类型5

17.JR





格式：JR rs

目的：使用寄存器的跳转指令

描述：PC ← 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 ← GPR[rs]

I+1: if Config1$_{CA}$ = 0 then

PC ← temp

else

PC ← temp$_{GPRLEN-1..1}$ || 0

ISAMode ← temp$_0$

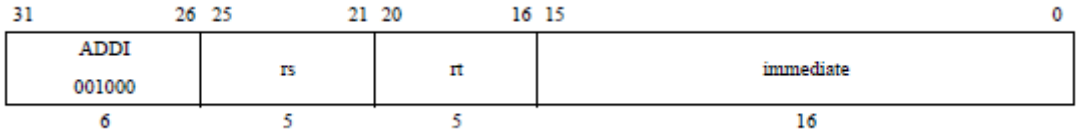
endif

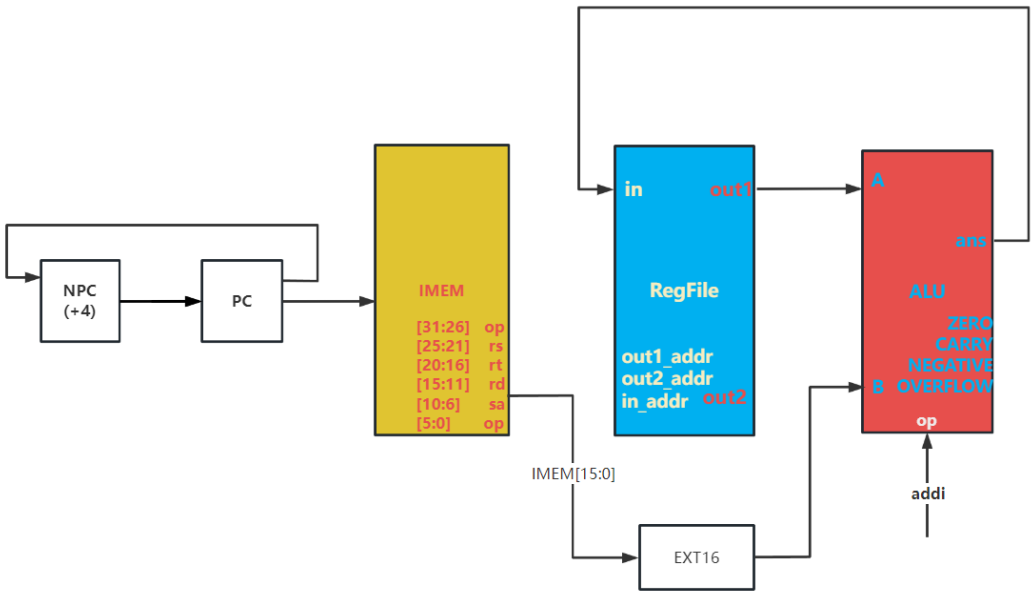
```
PC -> IMEM
PC + 4 -> NPC //无关指令
out1 -> MUX
MUX_OUT -> PC
NPC -> MUX //无关指令
```

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 ← rs + 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：

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

if temp$_{32}$ ≠ temp$_{31}$ then

SignalException(IntegerOverflow)

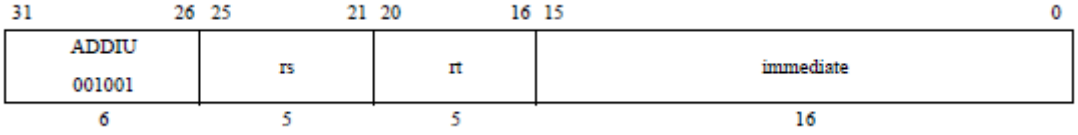
else

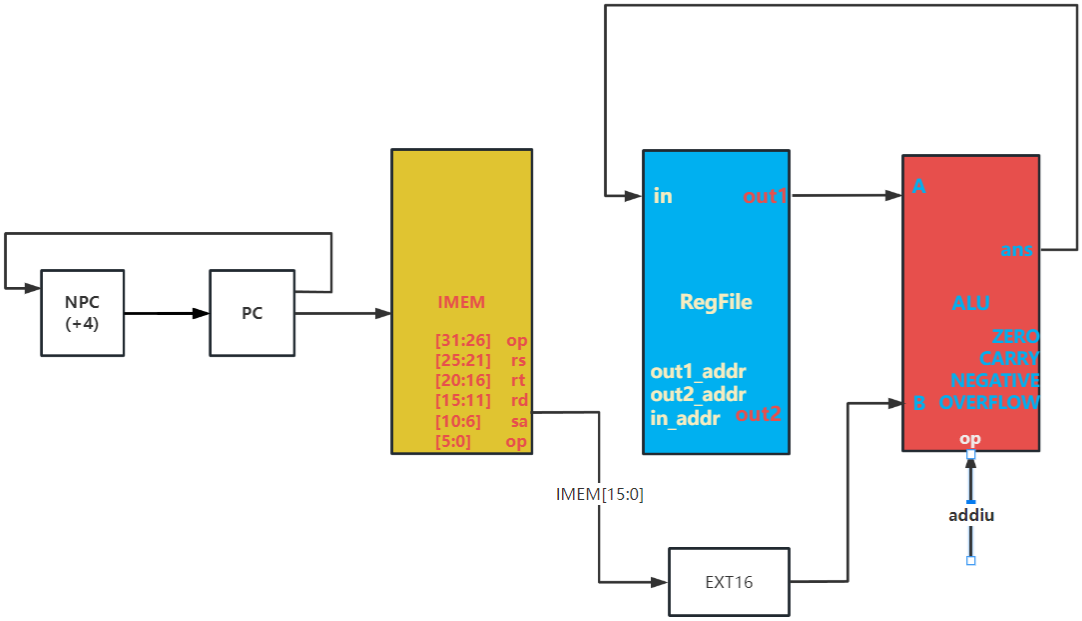
GPR[rt] ←temp

endif

```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
IMEM[15:0] -> EXT16
EXT16-> B
out1 -> A
(A + B -> RES)
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 ← rs + 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。

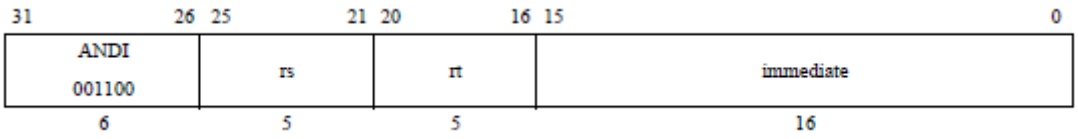
操作：

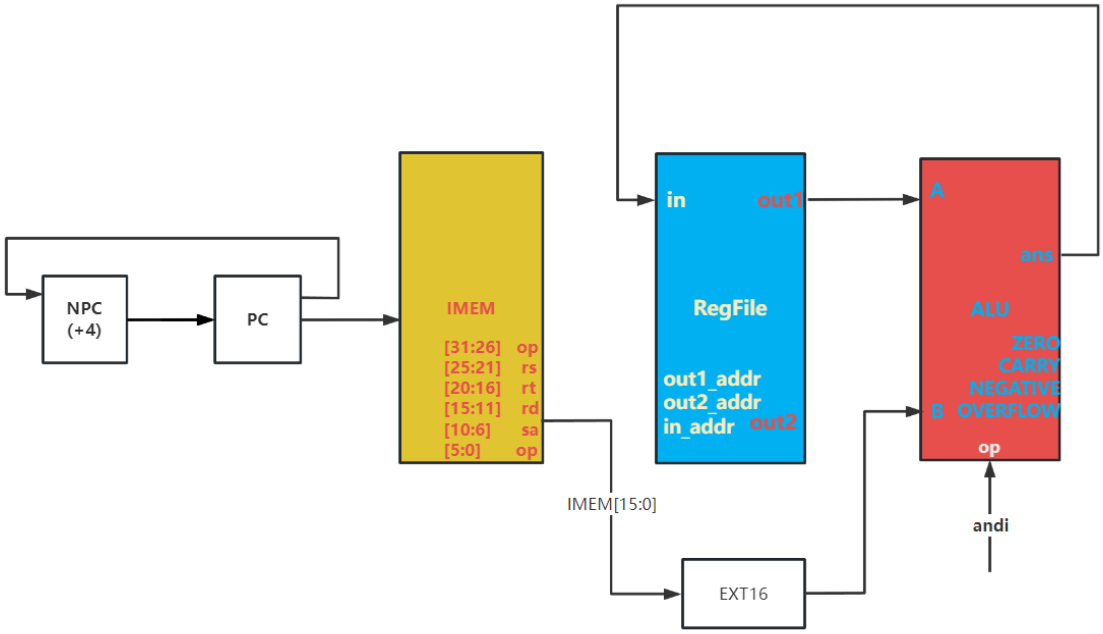
temp ←GPR[rs] + sign_extend(immediate)

GPR[rt] ← temp

```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
IMEM[15:0] -> EXT16
EXT16-> B
out1 -> A
(A + B -> RES)
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 ← rs AND immediate

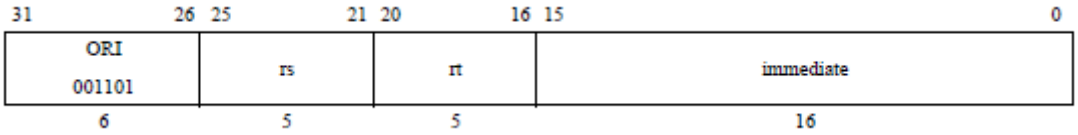
将16位立即数做0扩展后与通用寄存器rs中的32位数据做按位与，将结果存入目标寄存器rt。

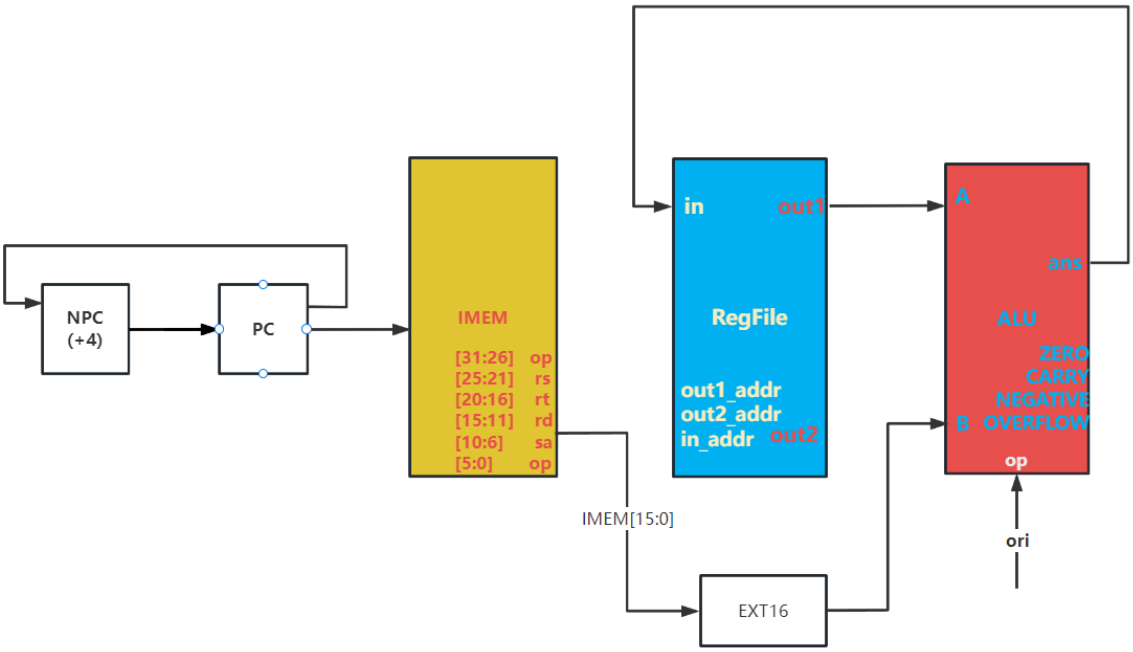
操作：

GPR[rt] ← GPR[rs] and zero_extend(immediate)

```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
IMEM[15:0] -> EXT16
EXT16-> B
out1 -> A
(A & B -> RES)
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 ← rs or immediate

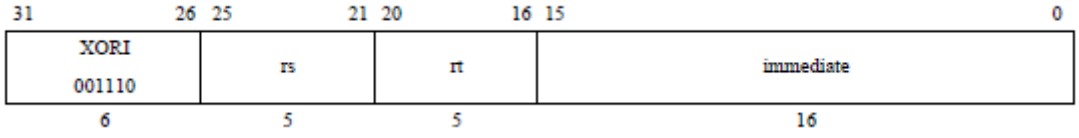
将通用寄存器rs和经过0扩展的立即数每一位做按位或操作，将结果存入目标寄存器rd中。

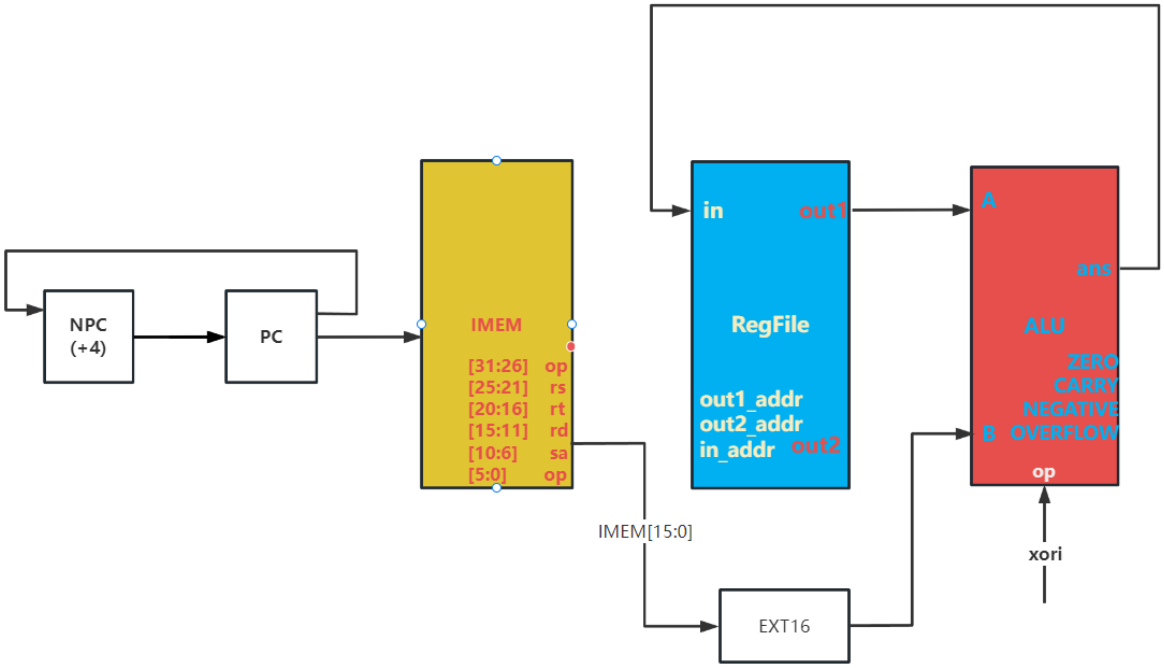
操作：

GPR[rt] ← GPR[rs] or zero_extend(immediate)

```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
IMEM[15:0] -> EXT16
EXT16-> B
out1 -> A
(A | B -> RES)
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 ←rs 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。

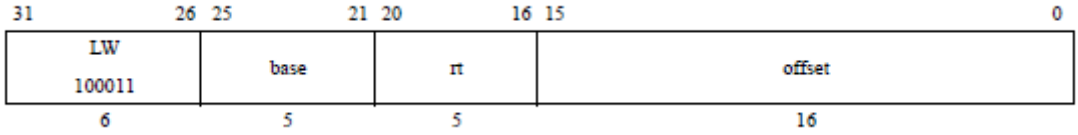
操作：

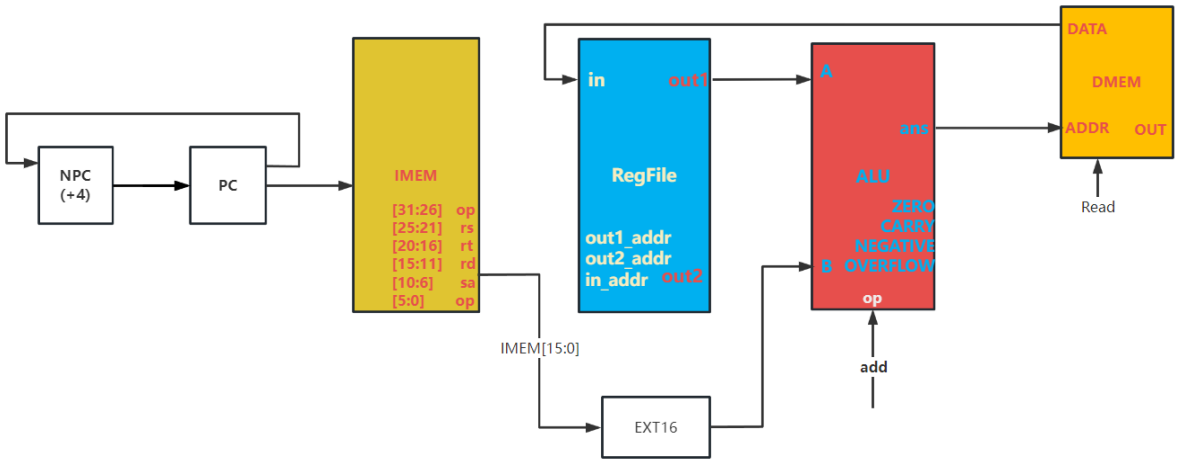
GPR[rt] ← GPR[rs] xor zero_extend(immediate)

```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
IMEM[15:0] -> EXT16
EXT16-> B
out1 -> A
(A ⊕ B -> RES)
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 ← memory[base+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 ← sign_extend(offset) + GPR[base]

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

SignalException(AddressError)

endif

(pAddr, CCA) ← AddressTranslation (vAddr, DATA, LOAD)

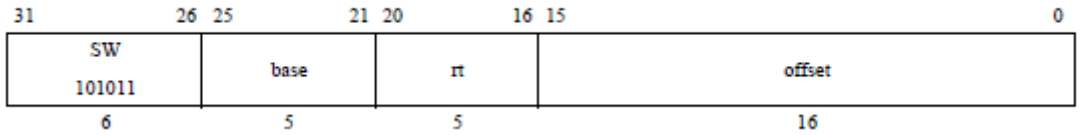
memword← LoadMemory (CCA, WORD, pAddr, vAddr, DATA)

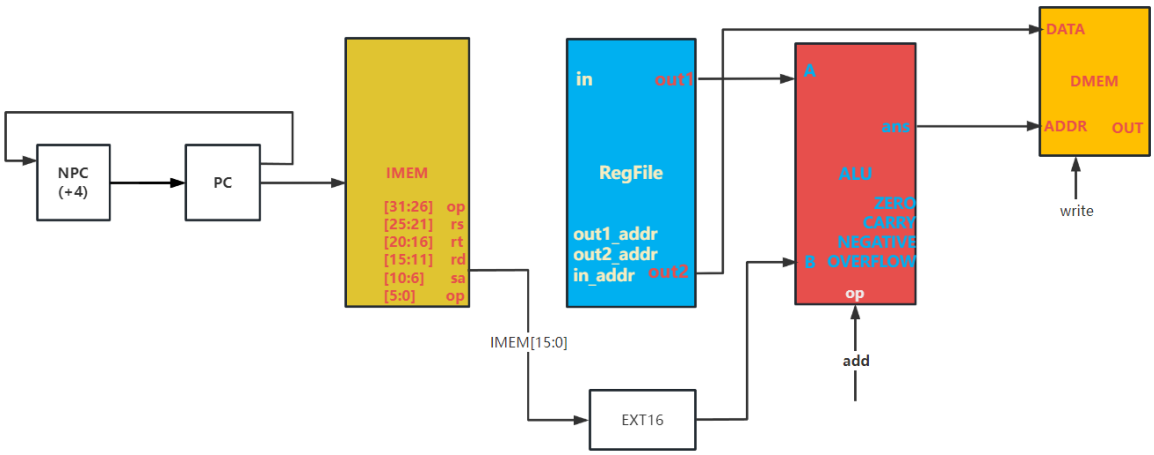
GPR[rt] ← memword

```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
IMEM[15:0] -> EXT16
EXT16 -> B
out1 -> A
(A + B -> ans)
ans -> DMEM_ADDR
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：memory[base+offset] ←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：

vAddr ← sign_extend(offset) + GPR[base]

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

SignalException(AddressError)

endif

(pAddr, CCA) ← AddressTranslation (vAddr, DATA, STORE)

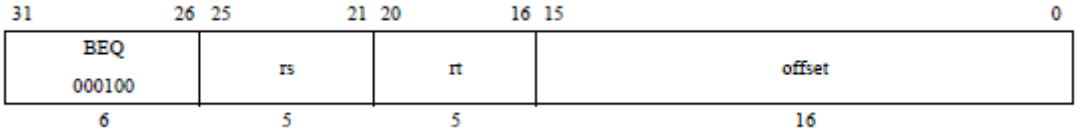
dataword← GPR[rt]

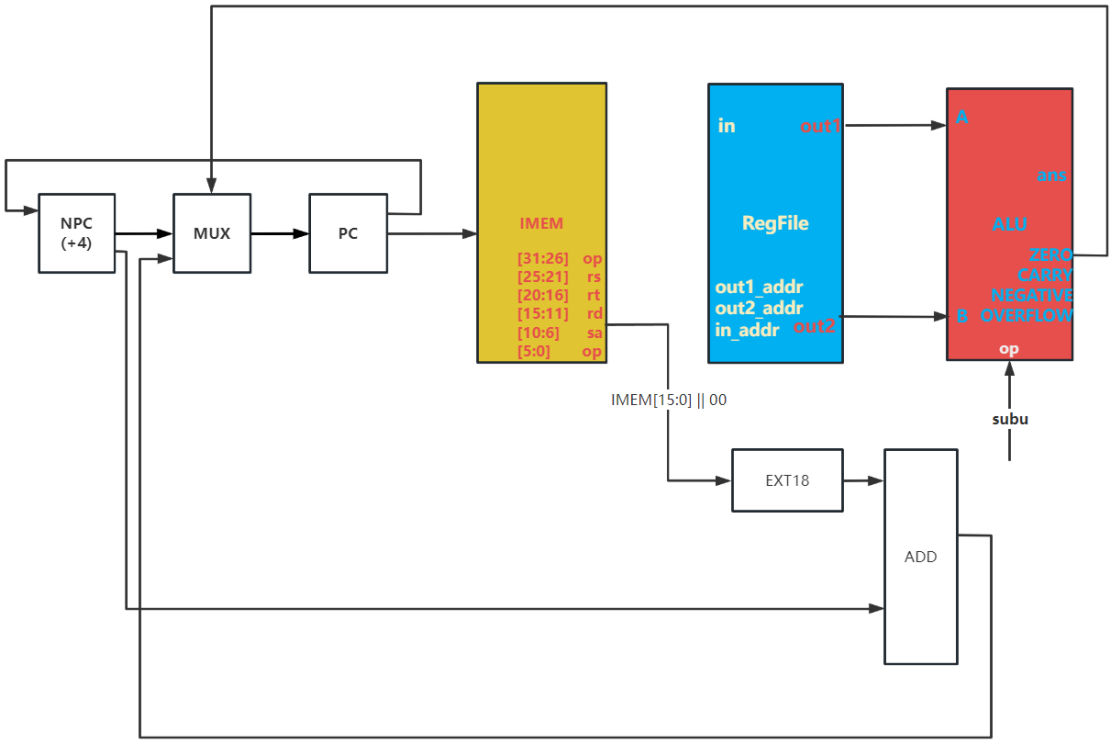
StoreMemory (CCA, WORD, dataword, pAddr, vAddr, DATA)

```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
IMEM[15:0] -> EXT16
EXT16_OUT -> B
out1 -> A
(A + B -> ans)
out2 -> DMEM
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 != rt，则继续执行下条指令。

操作：

I: target_offset ← sign_extend(offset || 0^2)

condition ←(GPR[rs] = GPR[rt])

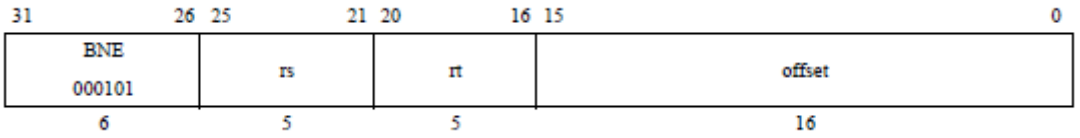
I+1: if condition then

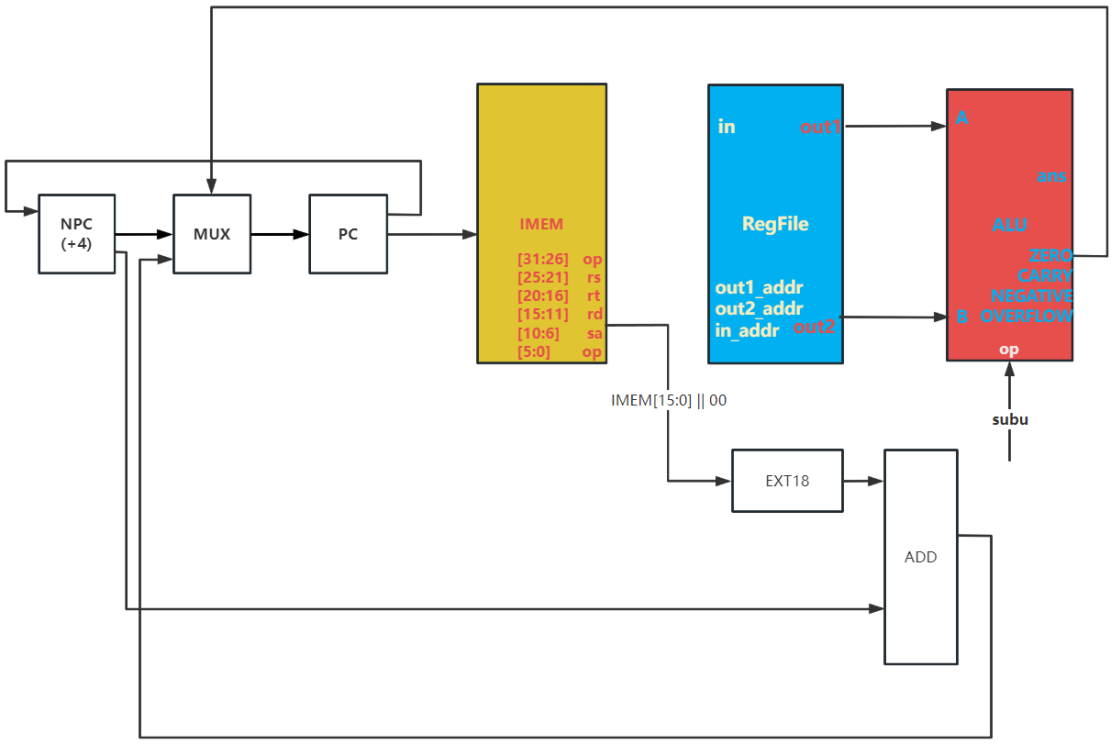
PC ←PC + target_offset

endif

```
PC -> IMEM
PC + 4 -> NPC
NPC -> MUX
IMEM[15:0] || 02 -> EXT18
EXT18_OUT -> ADD_A
NPC -> ADD_B
(ADD_A + ADD_B -> ADD_OUT)
ADD_OUT -> MUX
out1 -> A
out2 -> B
(A - B -> RES)
zero -> MUX
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 != 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 ← sign_extend(offset || 0^2)

condition ← (GPR[rs] ≠ GPR[rt])

I+1: if condition then

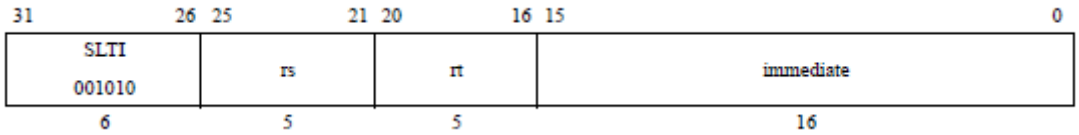
PC ← PC + target_offset

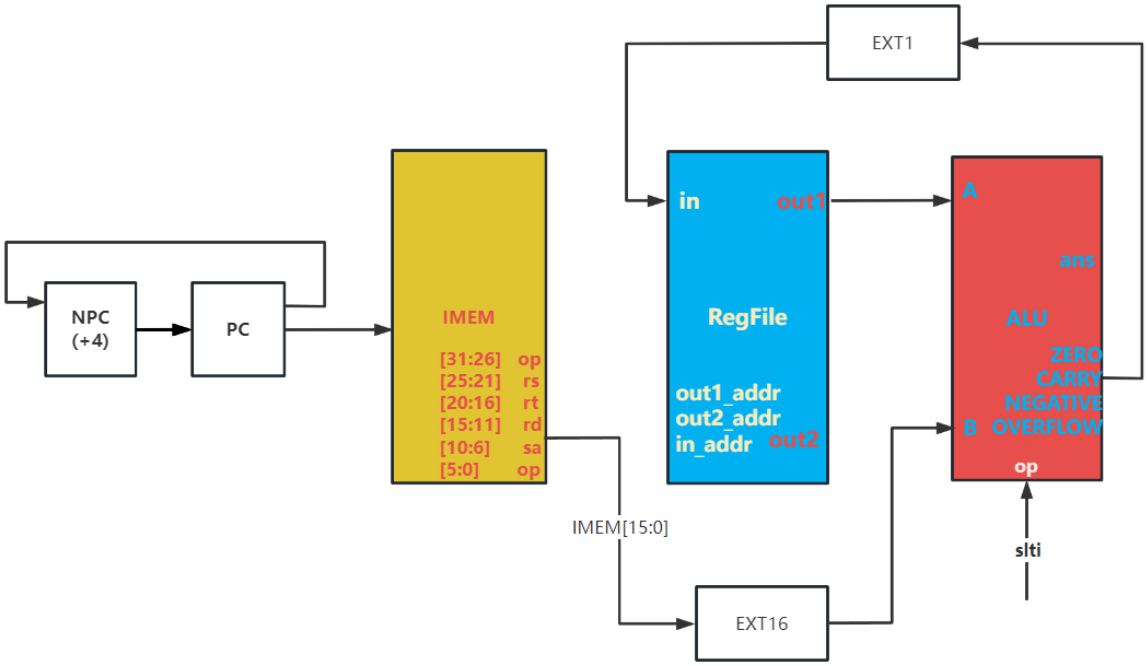
endif

```
PC -> IMEM
PC + 4 -> NPC
NPC -> MUX
IMEM[15:0] || 02 -> EXT18
EXT18_OUT -> ADD_A
NPC -> ADD_B
(ADD_A + ADD_B -> ADD_OUT)
ADD_OUT -> MUX
out1 -> A
out2 -> B
(A - B -> RES)
zero -> MUX
MUX -> PC
```

通路类型10

27.slti





格式：SLTI rt, rs, immediate

目的：通过跟立即数小于的比较来记录结果

描述：rt ←(rs < immediate)

比较rs中的数和经过符号扩展的16位立即数，用boolean值保存结果到rd寄存器中。如果rs中的数小于立即数，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 GPR[rs] < sign_extend(immediate) then

GPR[rd] ← $0^{GPRLEN-1}$|| 1

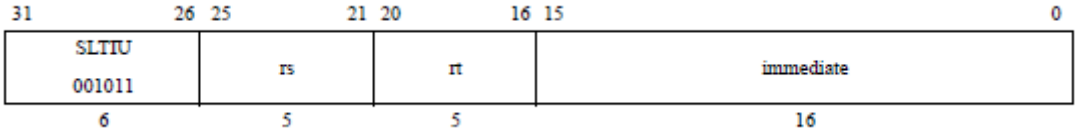
else

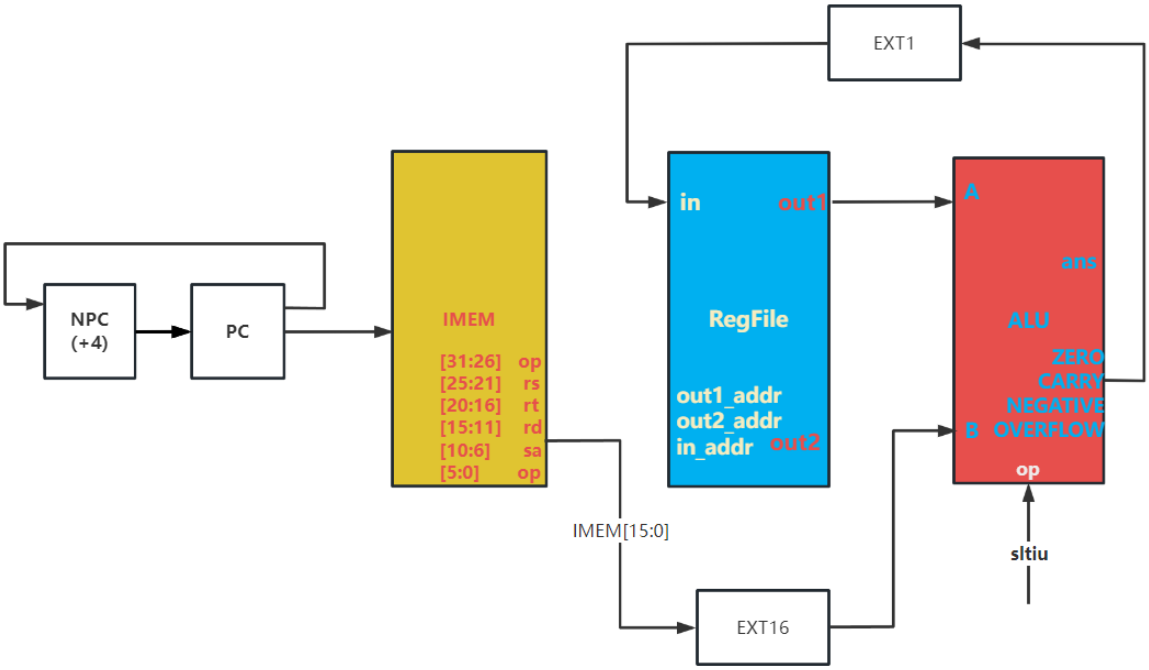
GPR[rd] ← 0^{GPRLEN}

endif

```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
IMEM[15:0] -> EXT16
EXT16 -> B
out1 -> A
(A - B -> RES)
Carry -> EXT1
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 ←(rs < 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] ← $0^{GPRLEN-1}$ || 1

else

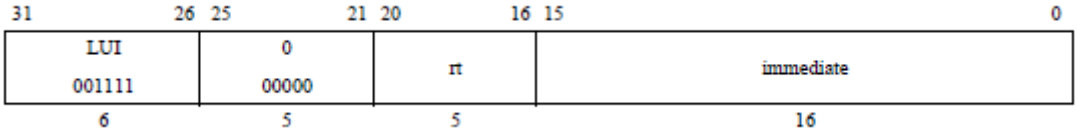
GPR[rd] ← 0^{GPRLEN}

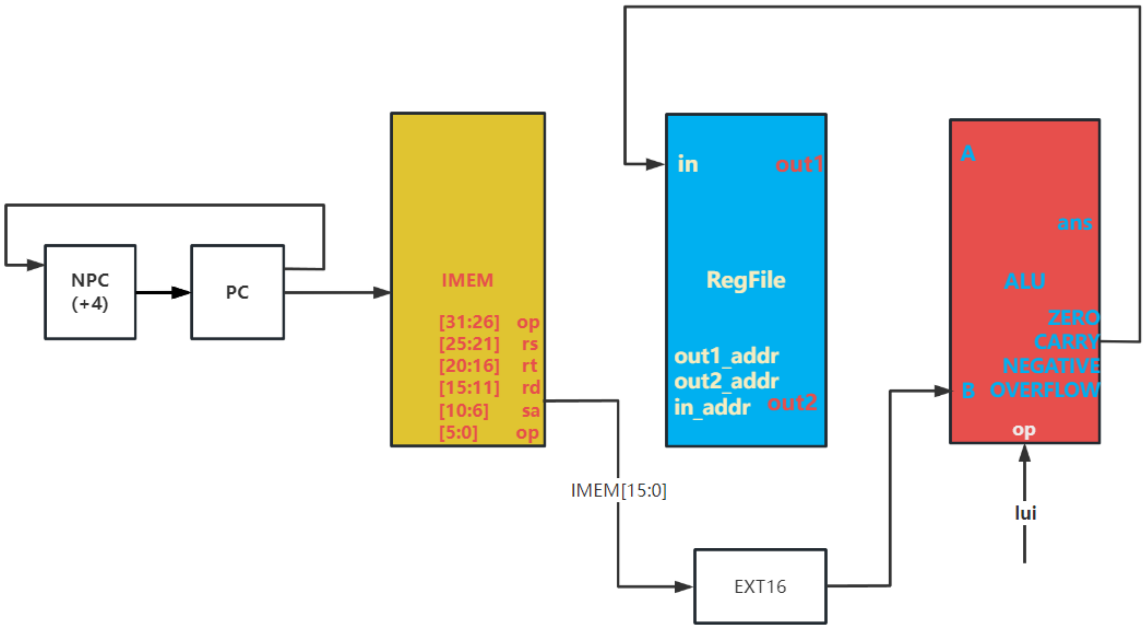
endif

```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
IMEM[15:0] -> EXT16
EXT16 -> B
out1 -> A
(A - B -> ans)
Carry -> EXT1
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 ← immediate || 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：

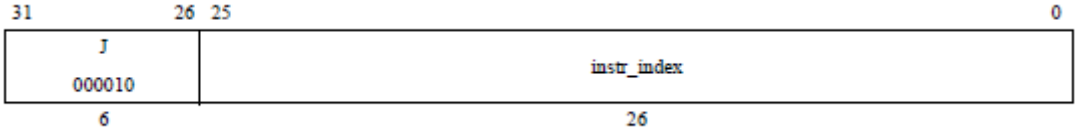
GPR[rt] ← immediate || 0^{16}

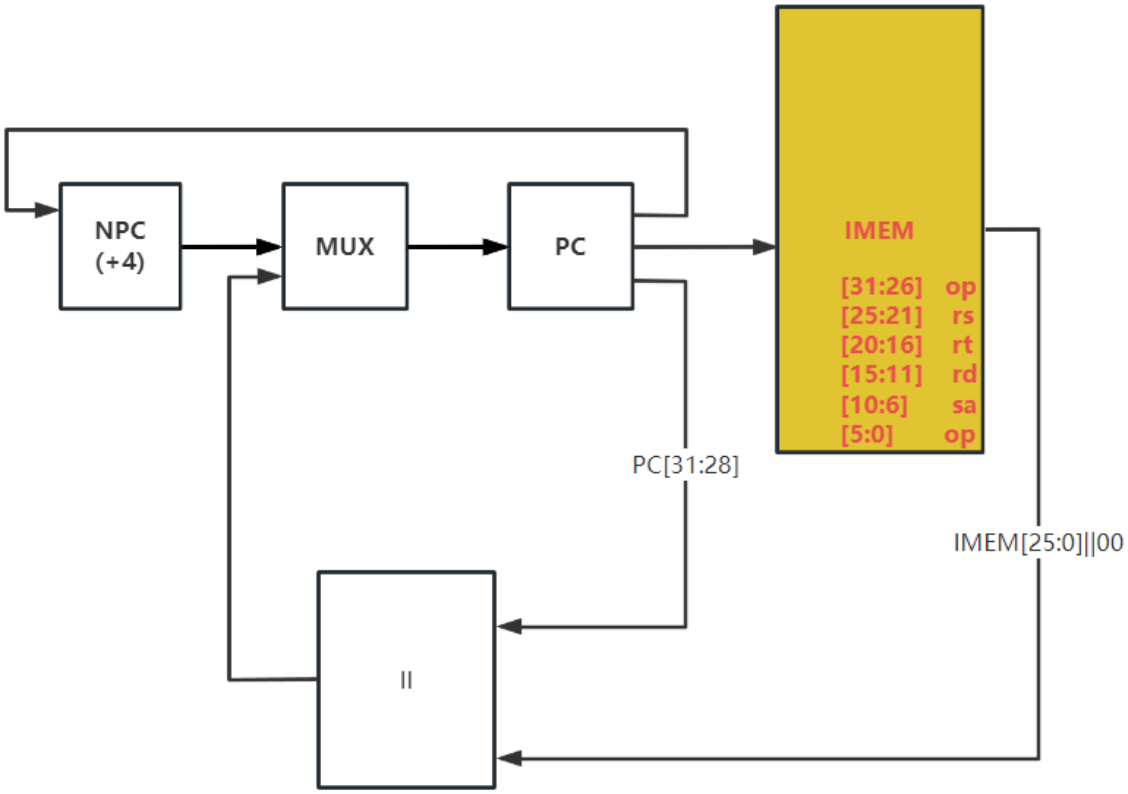
```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
IMEM[15:0] -> EXT16
EXT16 -> B
ans -> in(!!!!!!上面画错了!!!!!!)
```

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：

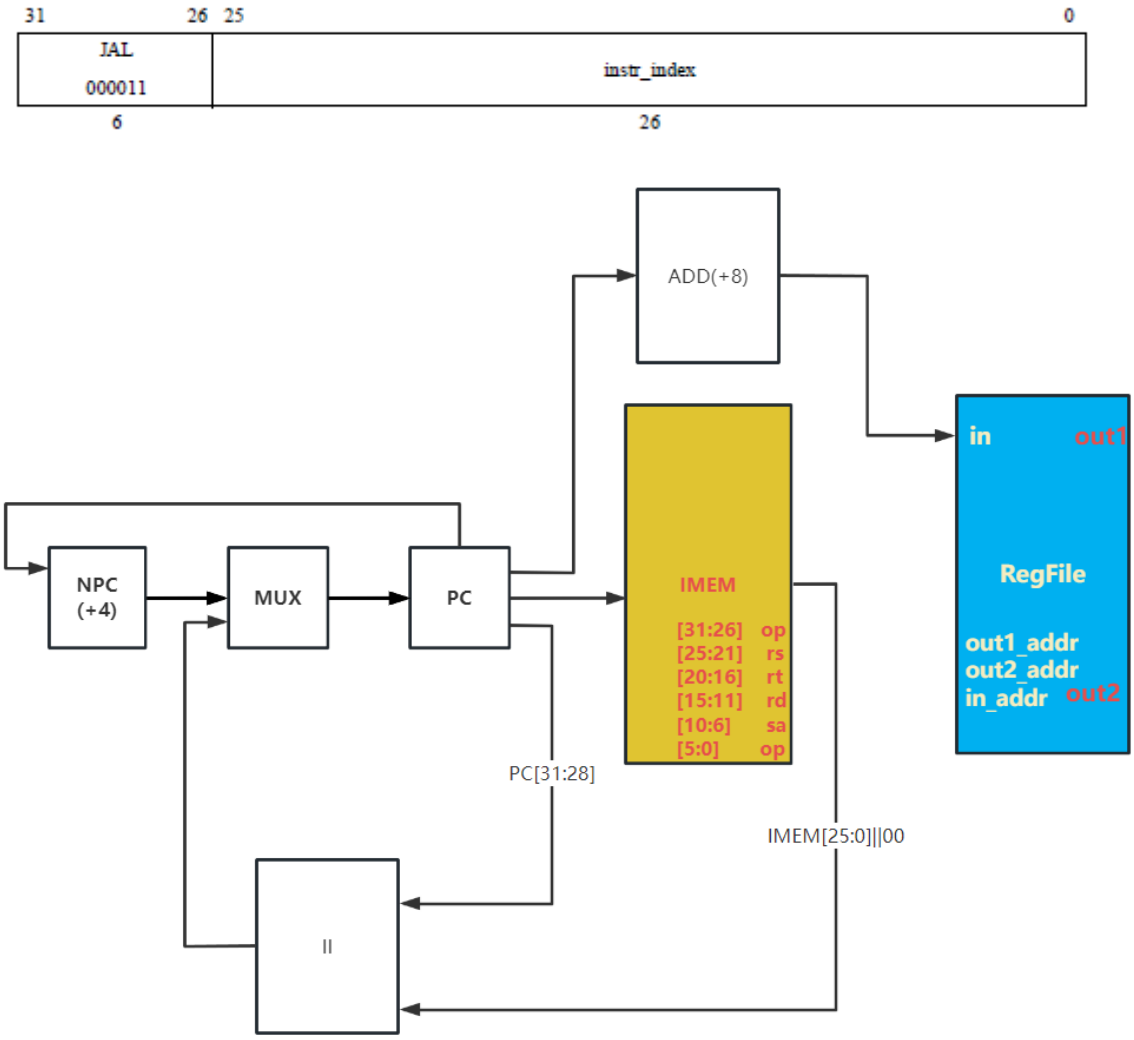
I:

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

```
PC -> IMEM
PC + 4 -> NPC
NPC -> MUX
PC[31:28] -> ||_A
IMEM[25,0] || 00 -> ||_B
||_OUT -> MUX
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] ←PC + 8

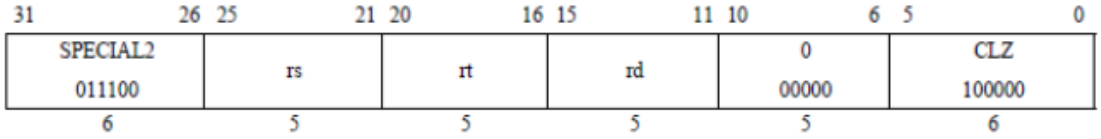
I+1：PC ←$PC_{GPRLEN-1..28}$ || instr_index || 0^2

```
PC -> IMEM
PC + 4 -> NPC
NPC -> MUX
8 -> ADD_A
PC -> ADD_B
(ADD_A + ADD_B -> ADD_OUT)
ADD_OUT -> Rd
PC[31:28] -> ||_A
IMEM[25,0] || 02 -> ||_B
||_OUT -> MUX
MUX_OUT -> PC
```

23条新增指令

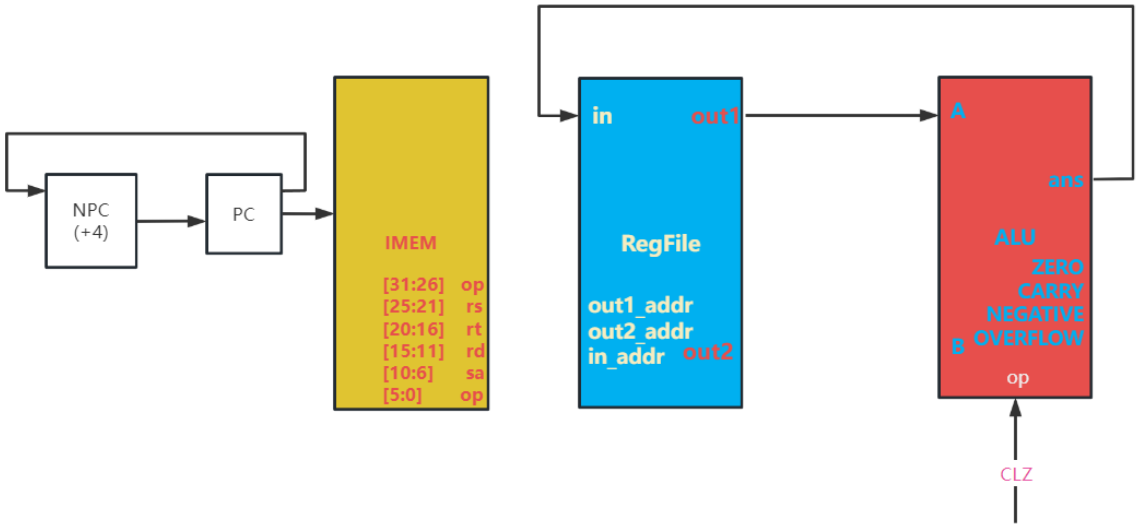
通路类型14（R指令）

32.CLZ




```
操作：
从RS里面拿数据，计算有多少个前导0，结果存放到rd里面去()
```

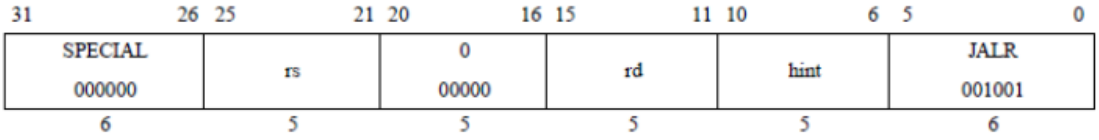
后面不是ALU 了,发现ALU单独实现这个计算有点麻烦,干脆用另一个部件了,就叫CLZ吧



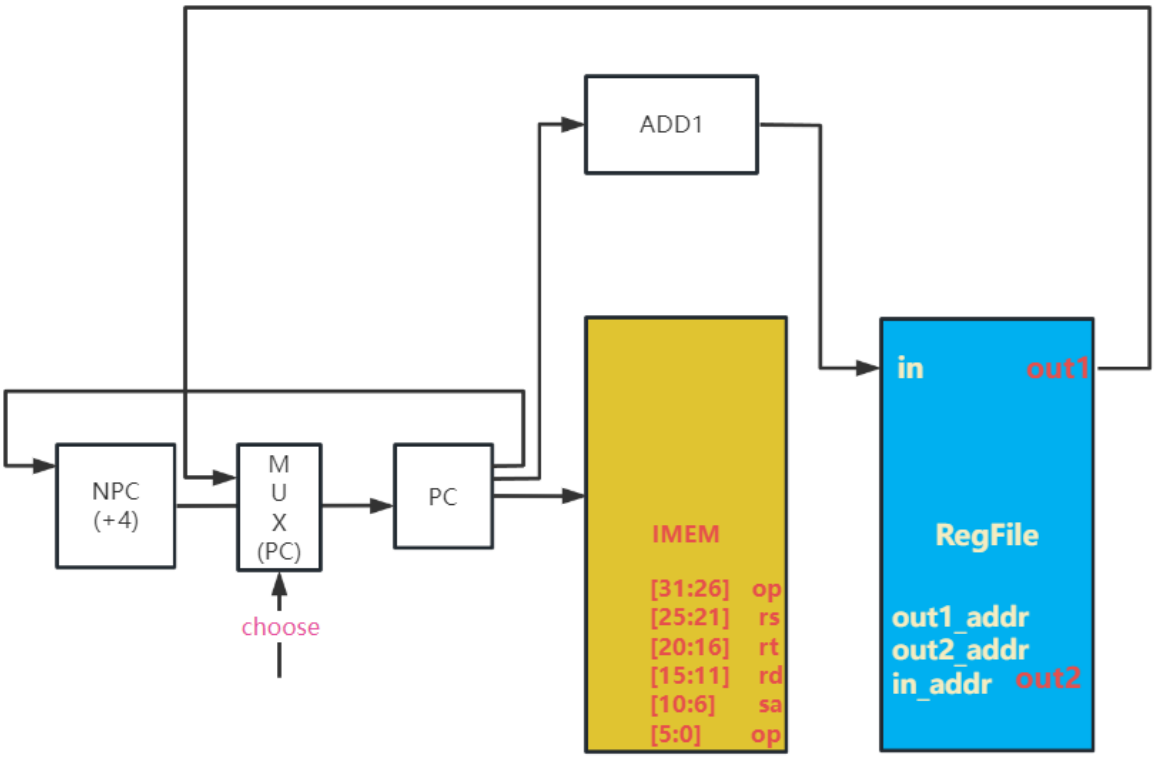

```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
IMEM[25:21] -> out1_addr
out1 -> A
(A -> ans)
ans -> in
```

通路类型15（R指令）

33.JALR




```
描述：
把返回地址放到rd里面，把rs寄存器里的地址放到pc里面
```

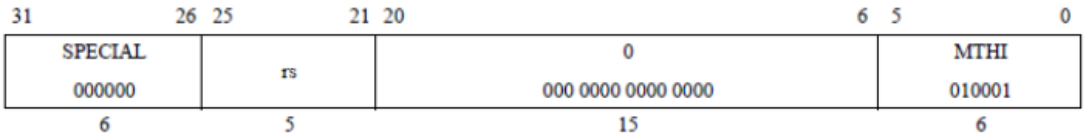



```
PC -> IMEM
PC -> ADD1 (+8)
ADD1 -> in
IMEM[25:21] -> out1_addr
out1 -> PC
```

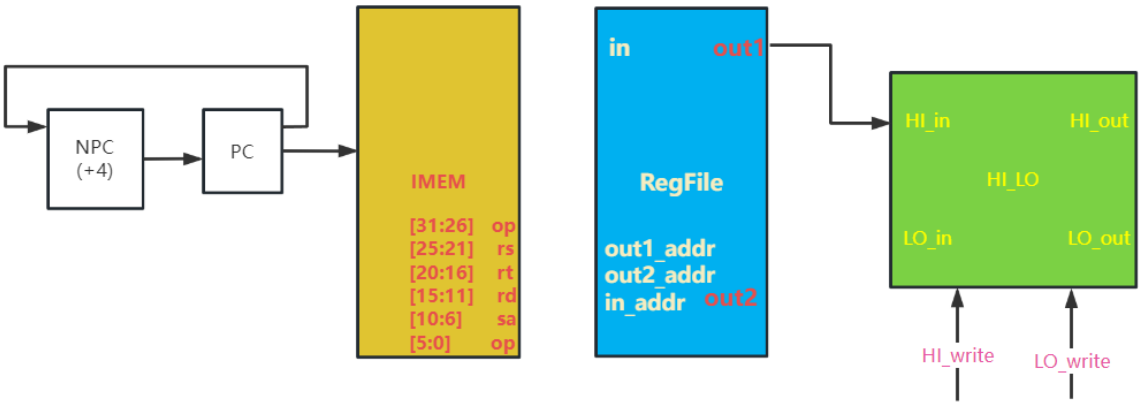
通路类型16（只用到RS）

说明：HI_LO相当于两个寄存器，用来放乘除法的结果，可以单独理解为两个特殊的pc寄存器，HI(high)后面会描述用来放乘法结果高32位，LO(low）放乘法低32位结果，没什么特殊的

34.MTHI

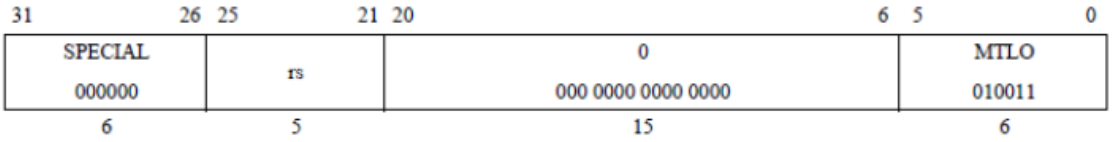



```
描述：
把rs寄存器里的值放到HI里面
```

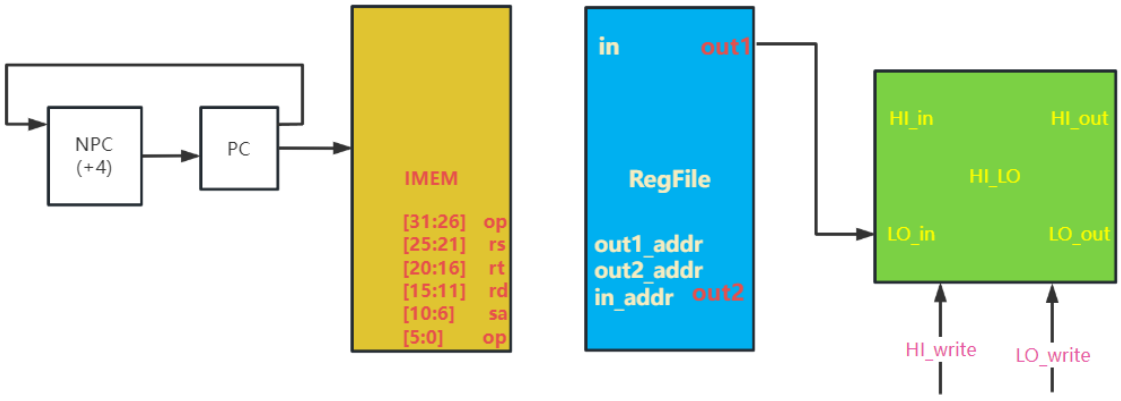



```
PC -> IMEM
PC +4 -> NPC
NPC -> PC
out1 -> HI_in
```

35.MTLO



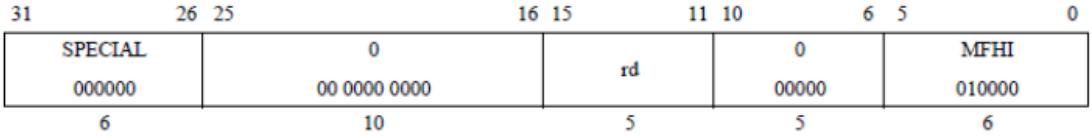

```
描述：
把rs寄存器里的值放到LO里面
```



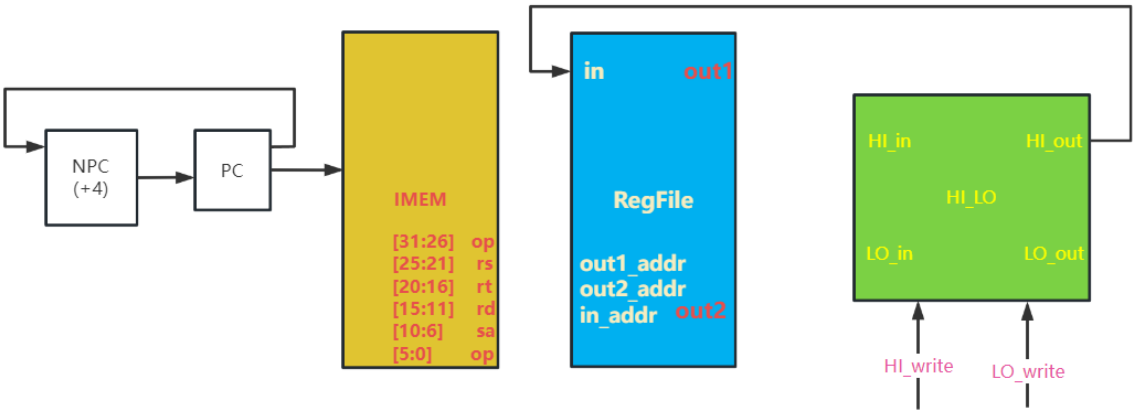

```
PC -> IMEM
PC +4 -> NPC
NPC -> PC
out1 -> LO_in
```

通路类型17（只用到Rd）

36.MFHI

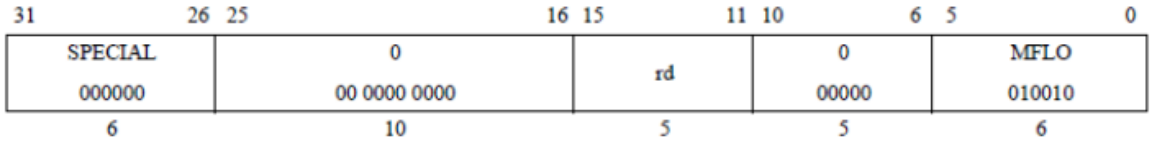



```
描述：
把HI里的值写会到Rd里面
```

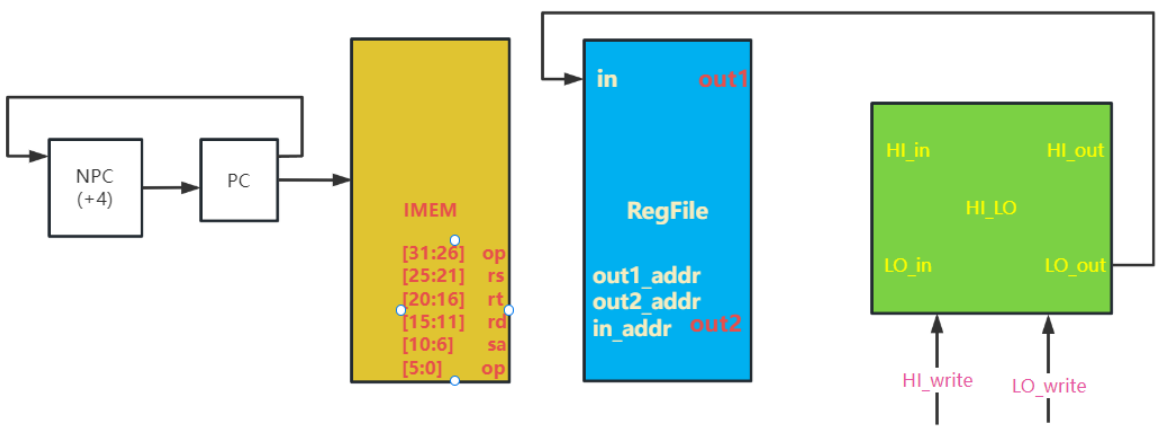



```
PC -> IMEM
PC +4 -> NPC
NPC -> PC
HI_out ->in
```

37.MFLO



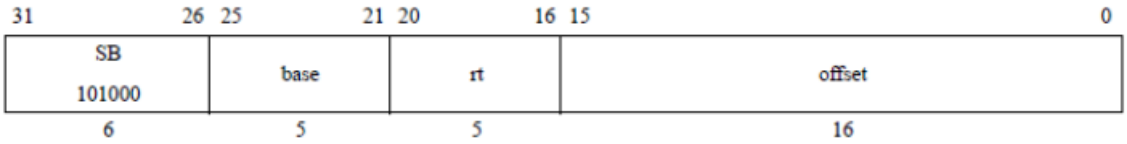

```
描述：
把LO里的值写回到Rd里面
```



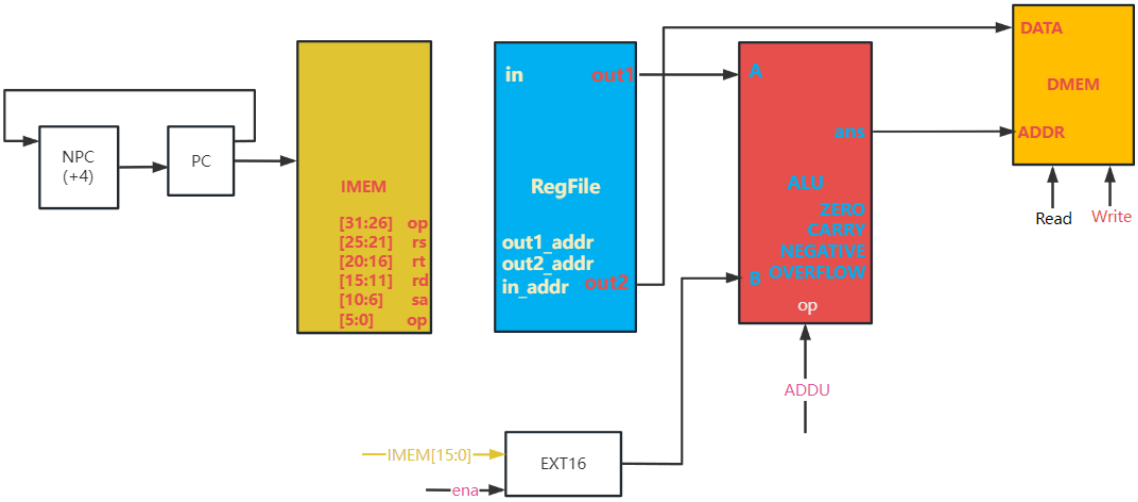
```
PC -> IMEM
PC +4 -> NPC
NPC -> PC
LO_out ->in
```

通路类型18(I指令 ,只不过out2还是Rt)

38.SB

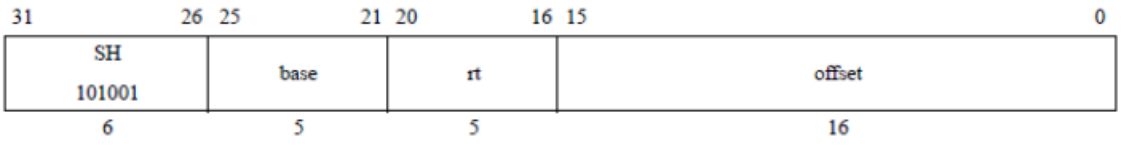



```
描述:
把Rt里的值放到DMEM 里的[(base) + offset]这个地址里面的低8位 ,立即数符号扩展,
```



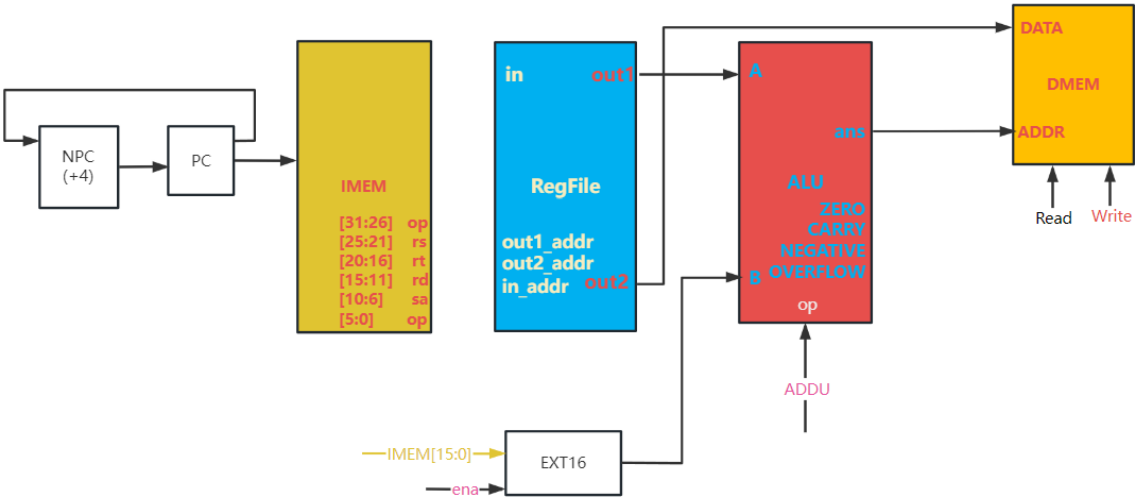

```
PC -> IMEM
PC +4 -> NPC
NPC -> PC
out1(base) -> A
IMEM[15:0](offset) -> EXT16 -> B
A+B ->ans
out2 -> DATA
```

39.SH




```
描述:
和SB一样,只不过只存半字,立即数符号扩展低16位

```

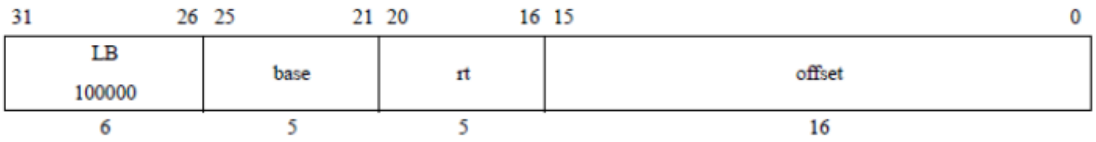



```
PC -> IMEM
PC +4 -> NPC
NPC -> PC
out1(base) -> A
IMEM[15:0](offset) -> EXT16 -> B
A+B ->ans
out2 -> DATA
```

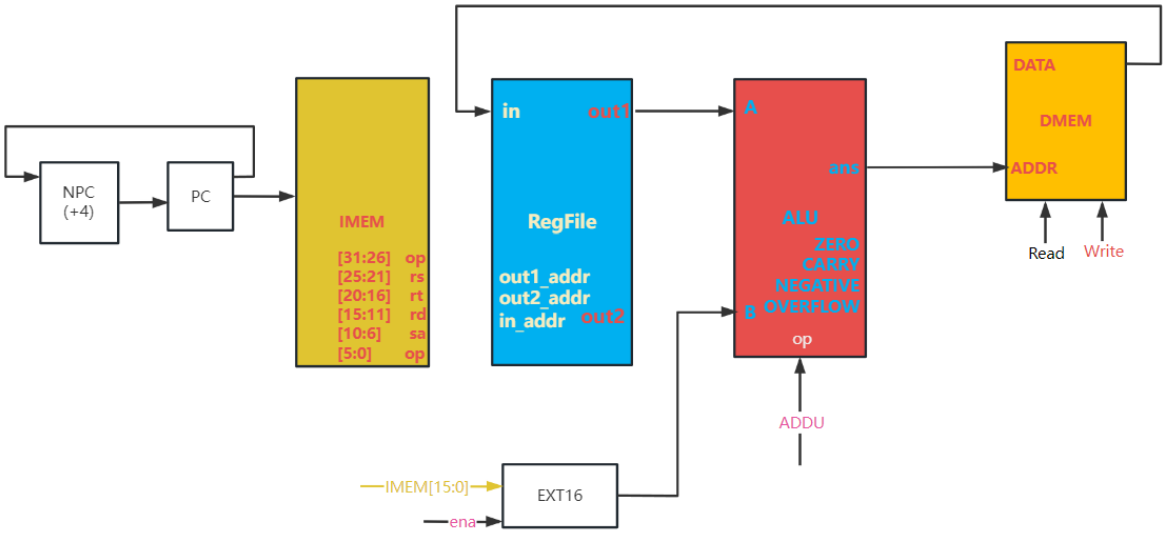
通路类型19(I指令)

这里的每个data线应该都有个符号或者无符号扩展器,图中没画出,但是有,看描述就知道了

40.LB

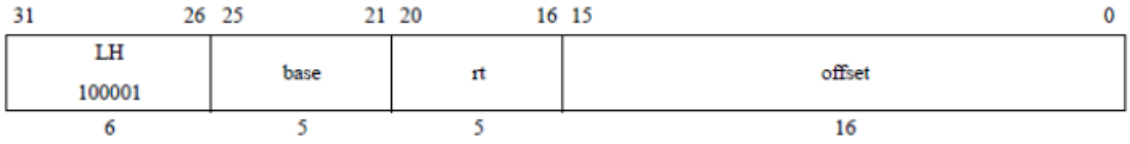



```
描述:
DMEM 里的[(base) + offset]这个地址里面的值写回到rt , 有符号,低8位符号扩展
```

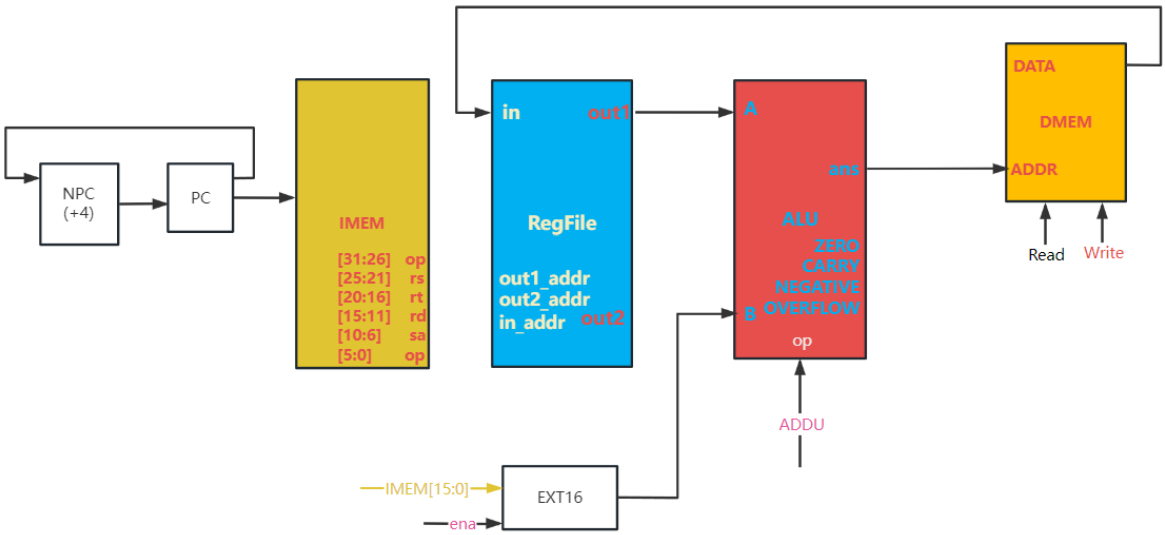



```
PC -> IMEM
PC +4 -> NPC
NPC -> PC
out1(base) -> A
IMEM[15:0](offset) -> EXT16 -> B
A+B ->ans
DATA -> in
```

41.LH

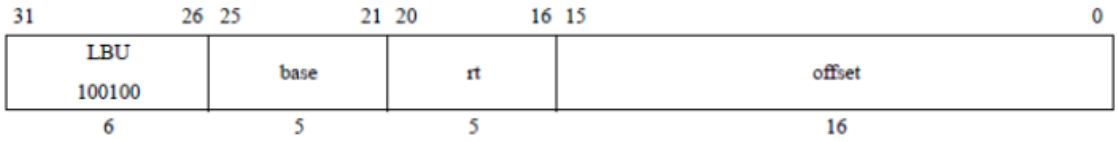



```
描述:
DMEM 里的[(base) + offset]这个地址里面的值写回到rt,只不过是半字,低16位符号扩展
```

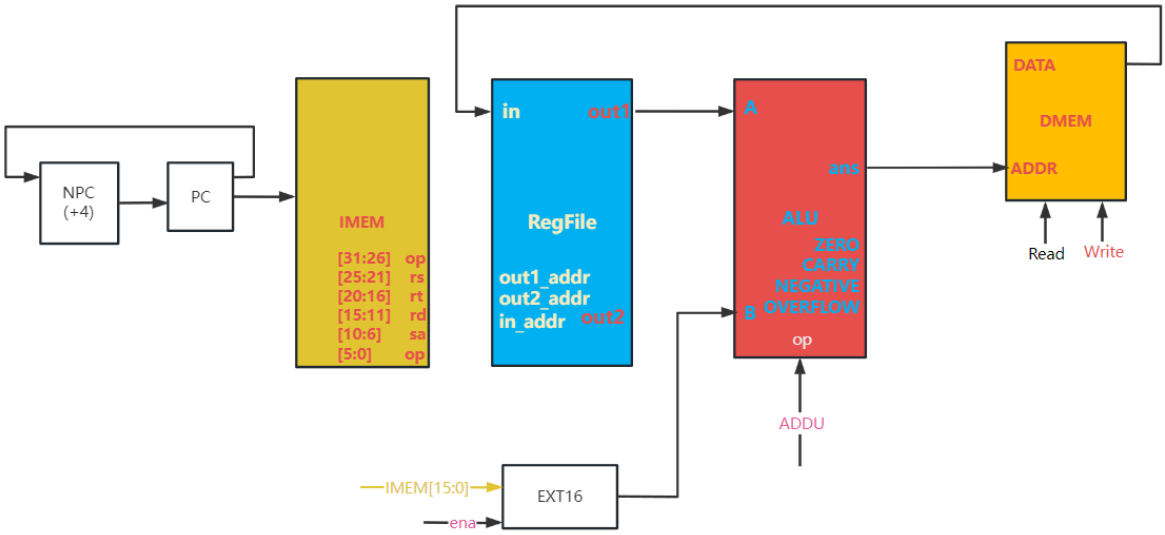


```
PC -> IMEM
PC +4 -> NPC
NPC -> PC
out1(base) -> A
IMEM[15:0](offset) -> EXT16 -> B
A+B ->ans
DATA -> in
```

42.LBU

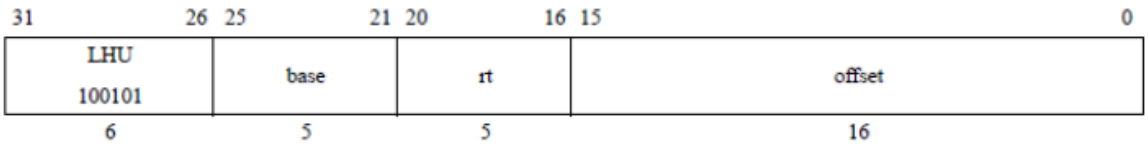



```
描述:
DMEM 里的[(base) + offset]这个地址里面的值写回到rt,无符号,低8位0扩展
```

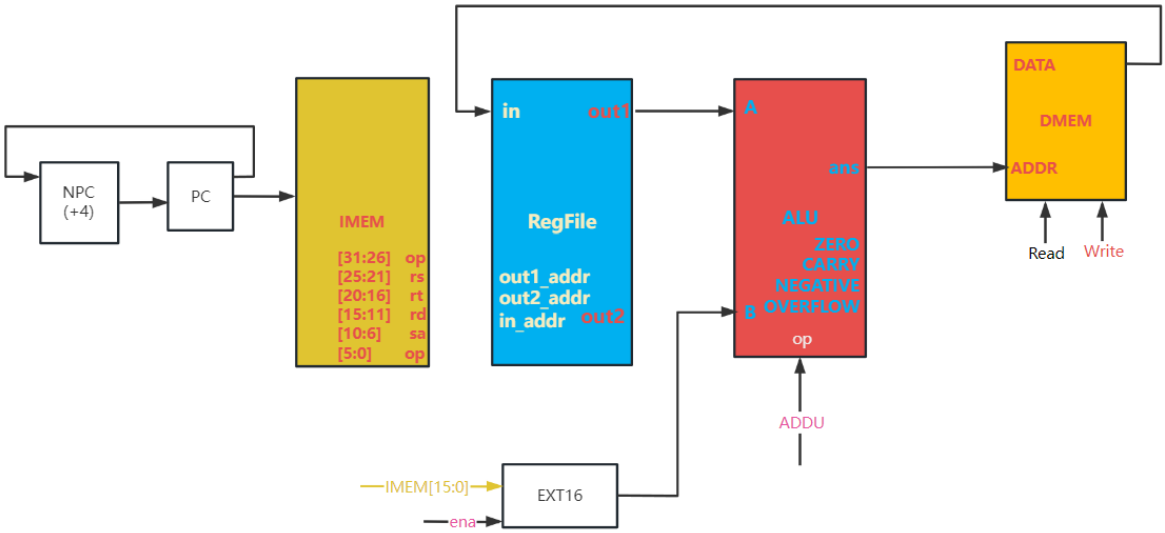



```
PC -> IMEM
PC +4 -> NPC
NPC -> PC
out1(base) -> A
IMEM[15:0](offset) -> EXT16 -> B
A+B ->ans
DATA -> in
```

43.LHU



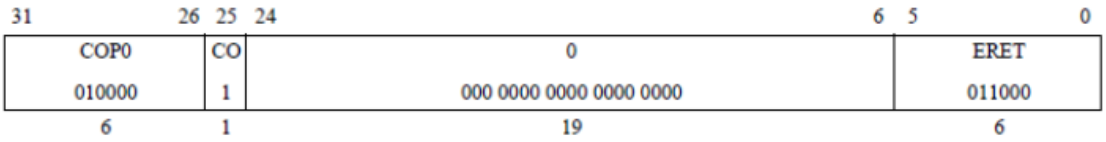

```
描述:
DMEM 里的[(base) + offset]这个地址里面的值写回到rt,只不过是半字,低16位0扩展,无符号
```



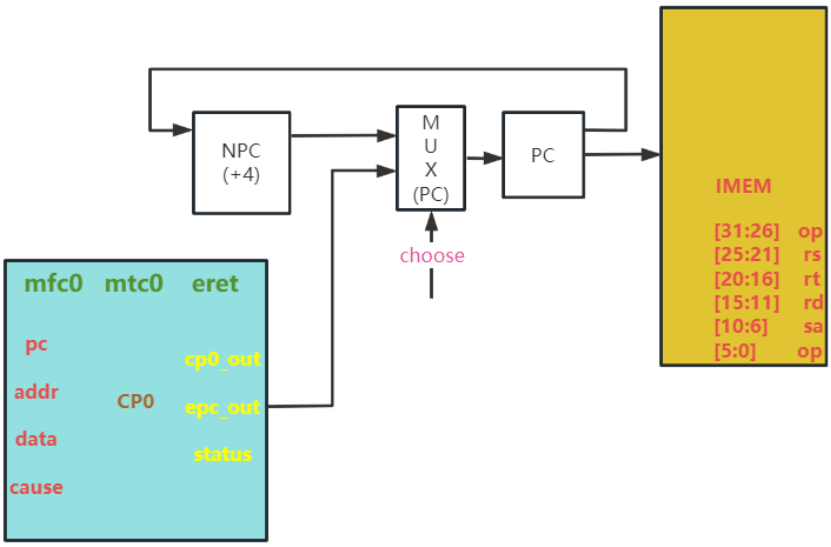

```
PC -> IMEM
PC +4 -> NPC
NPC -> PC
out1(base) -> A
IMEM[15:0](offset) -> EXT16 -> B
A+B ->ans
DATA -> in
```

通路类型20(中断类型)

44.ERET




```
描述:
pc ->npc ,imem
status >>5 -> status
epc_out ->pc
(就是把CP0的ert写回到PC,把保存的status再移动回来,那个英文文档真是答辩)
```

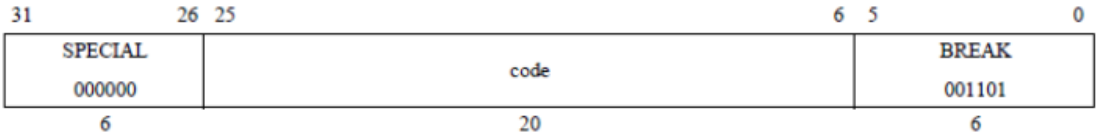



```
PC -> IMEM
PC +4 -> NPC
NPC -> PC
epc_out -> PC
```

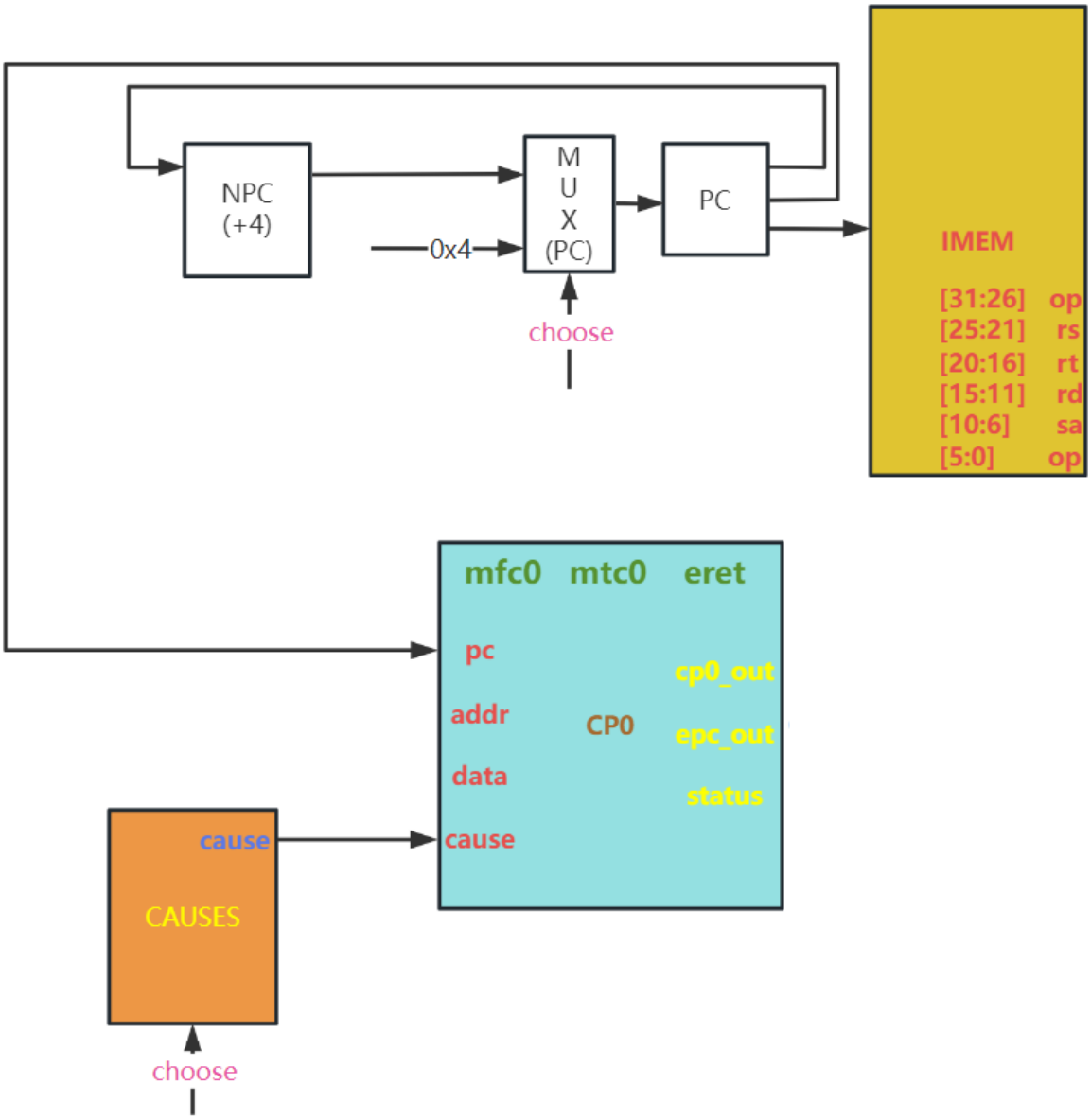
通路类型21中断类型)

45.BREAK

CAUSES是一个只读寄存器,里面存的中断原因,这个写死了,看模块就行了

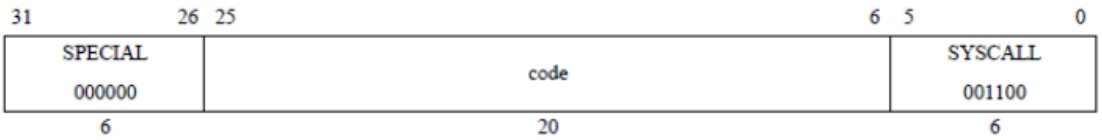



```
描述:
中断,把PC放到cp0里面去,
保存中断原因 cause ->cp0,
然后保存当前状态status<<5,
0X4 -> pc
```

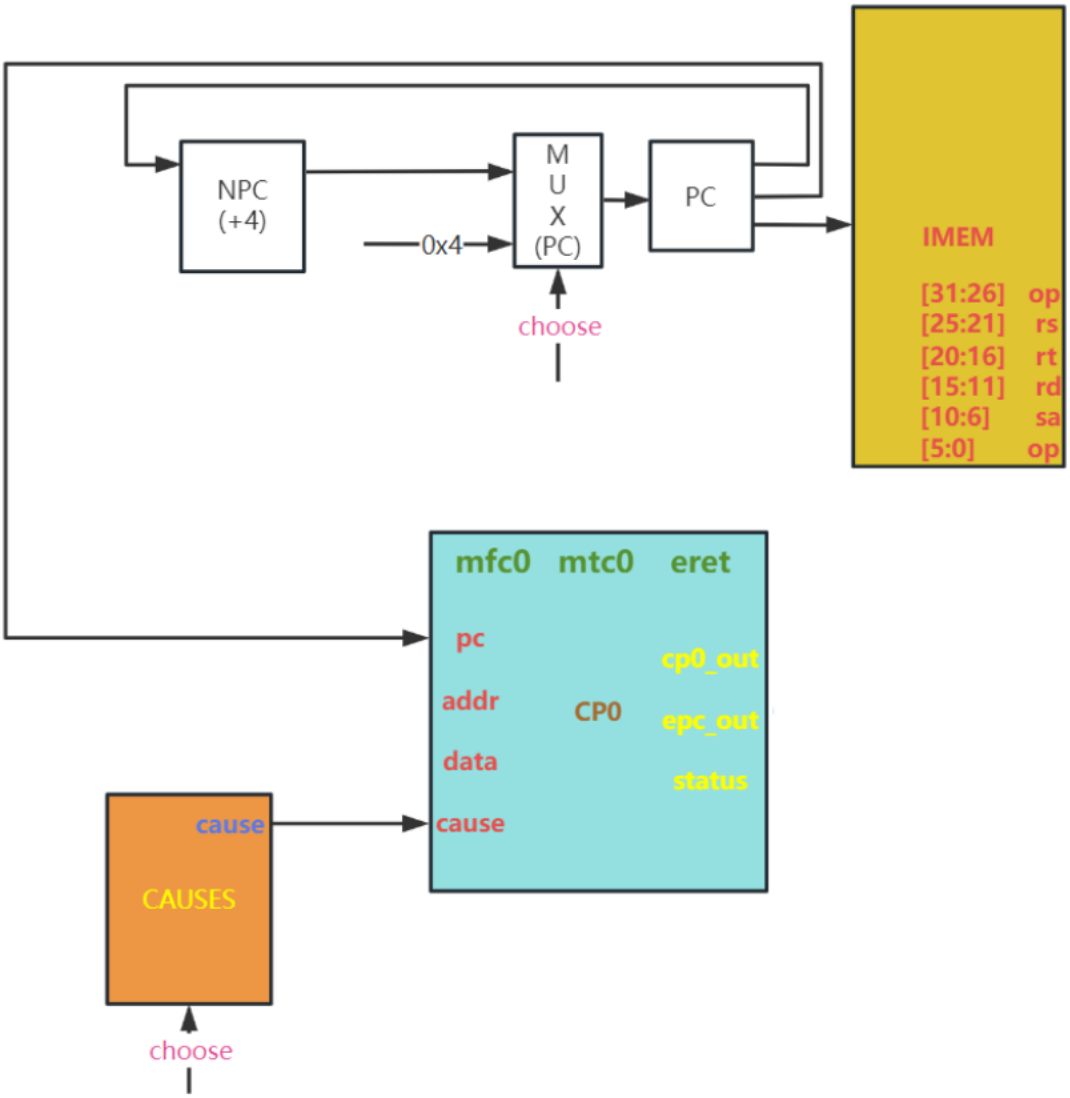



```
PC -> IMEM , cp0,
PC +4 -> NPC
CAUSES ->  cause(cp0)
0x4 -> pc
```

46.SYSCALL



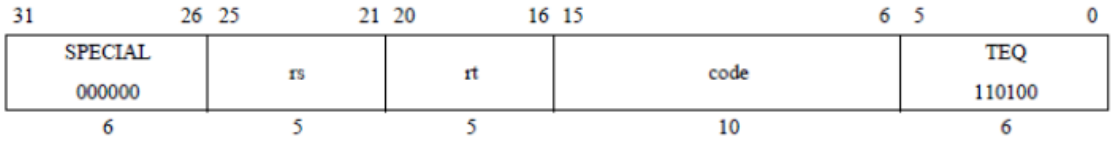

```
描述:
中断,把PC放到cp0里面去,
保存中断原因 cause ->cp0,
然后保存当前状态status<<5,
0X4 -> pc (这个和break一样的,只不过causes里面的原因不一样)
```




```
PC -> IMEM , cp0,
PC +4 -> NPC
CAUSES ->  cause(cp0)
0x4 -> pc
```

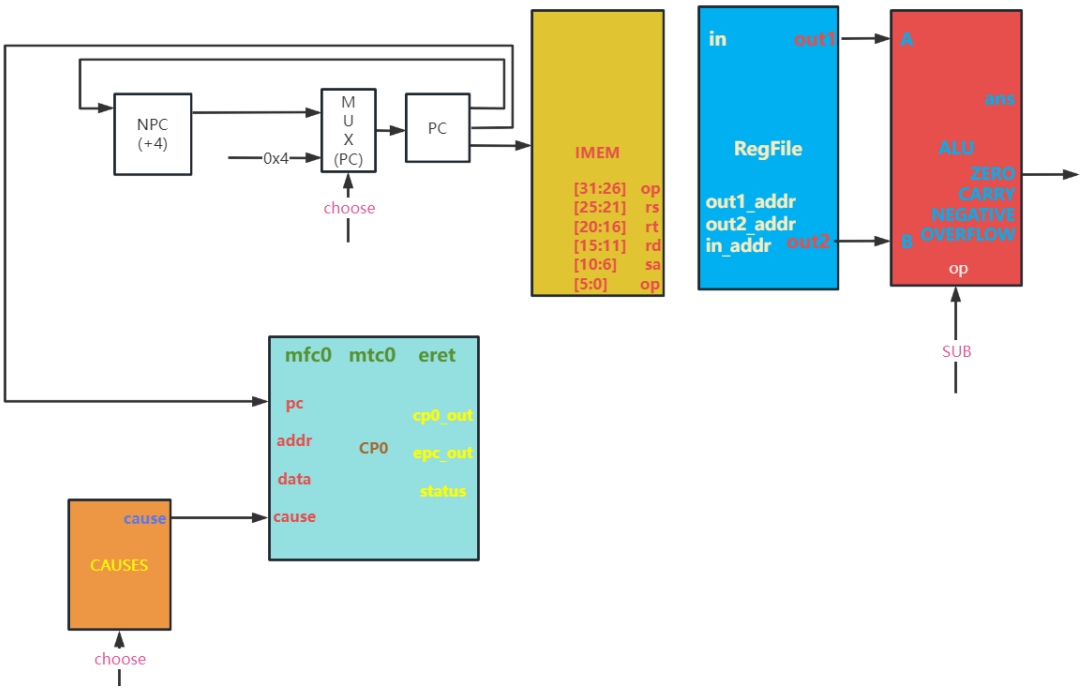
通路类型22(R指令,没有用rd)

47.TEQ



```
描述:
确定teq所取的操作,取指令
如果 rs == rt 然后进行中断操作
cause ->cp0
status<<5
0x4 ->pc
不然的话就正常
npc->pc
```

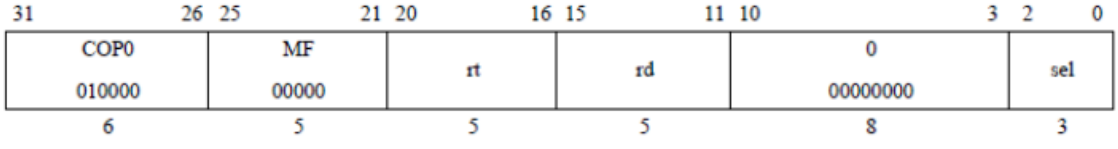
这里的zero要接到CP0的选择端和pc的数据选择器端,因为要判断中断类型看看是不是要保存中断信息,以及是不是要改变PC



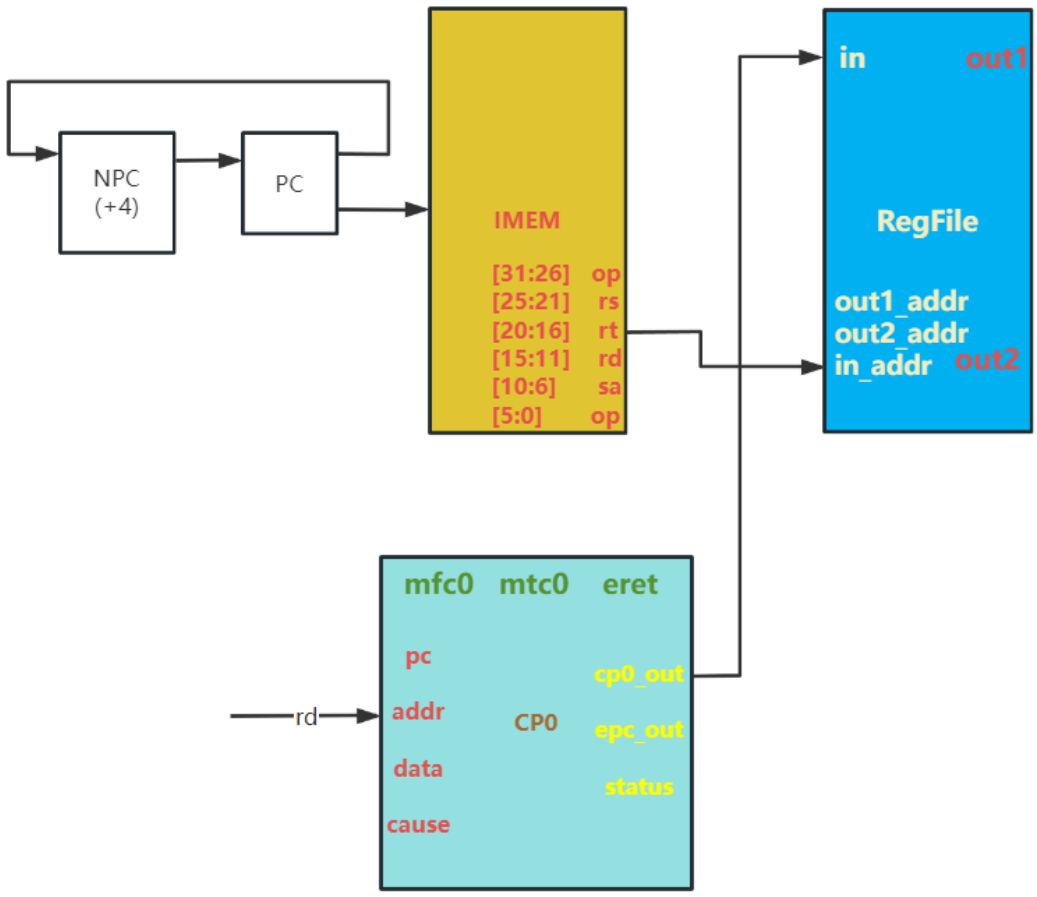

```
pc ->imem ,NPC
if out1 == out2
then
cause ->cp0
status <<5
0x4 -> pc
else
npc -> pc
```

通路类型23 (R指令,用到rt,rd)

48.MFC0

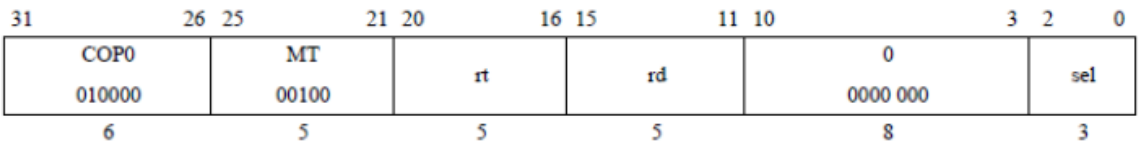


```
描述:
把cp0地址为rd里的东西写入Rt,
cp0_out (rd) ->  in (regfile / Rt)
```

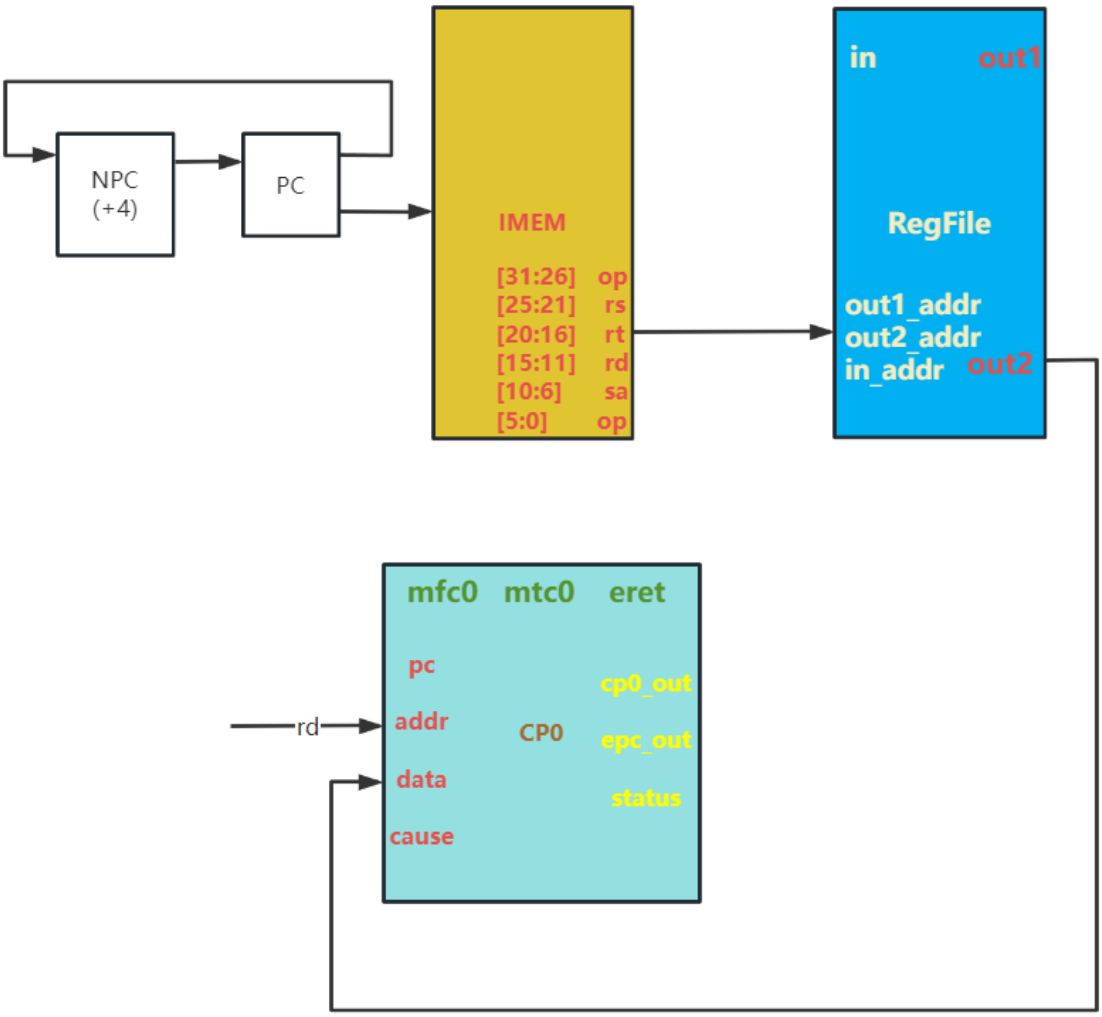



```
pc  + 4 -> npc
npc -> pc
cp0_out (rd) -> in(rt)
```

49.MTC0



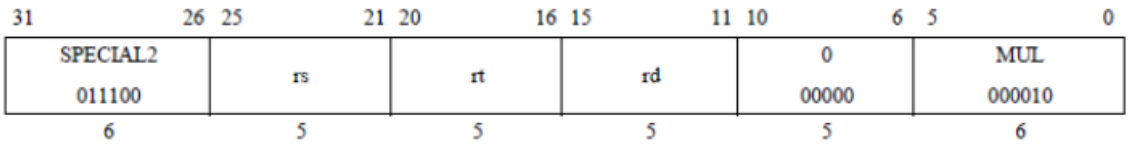

```
描述:
取指令,rt里面的东西写到cp0里的rd地址里面
```



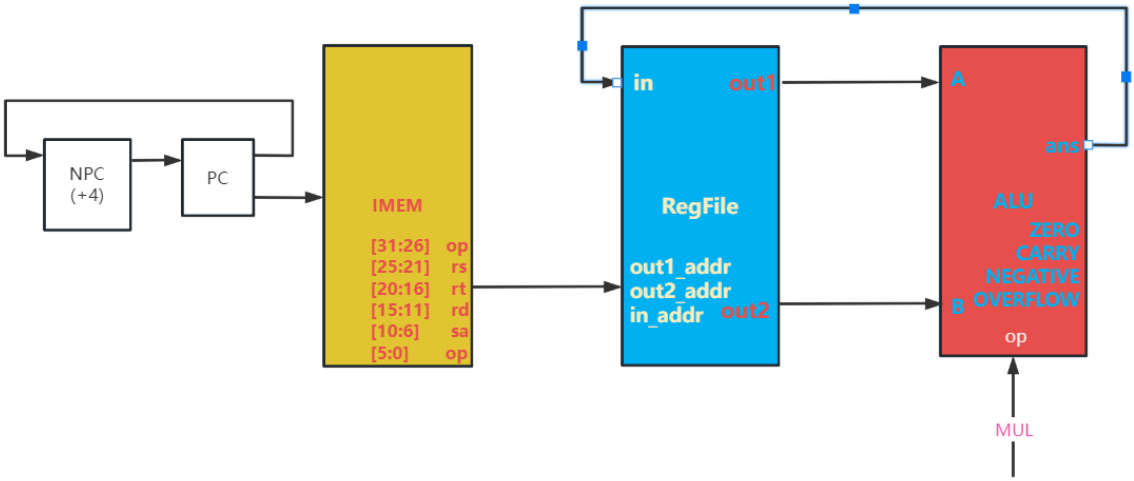

```
pc -> IMEM ,npc
rt ->out2_addr
rd ->cp0_addr
out2 -> data
```

通路类型24(乘除法指令)

50.MUL

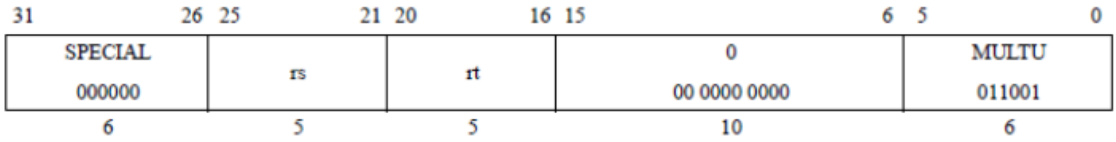



```
描述:
把rt * rs 的东西写入到rd(反正要截断,干脆直接写到ALU 里面了)
```

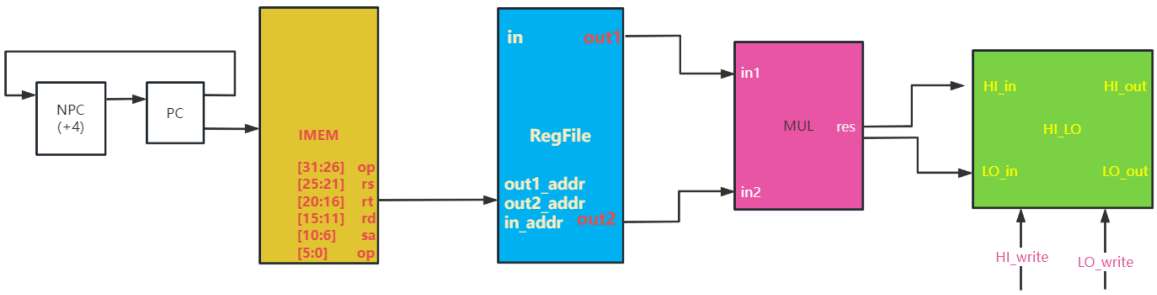



```
PC -> NPC ,IMEM
OUT1 * OUT2 ->ans
ans -> in
```

51.MULTU



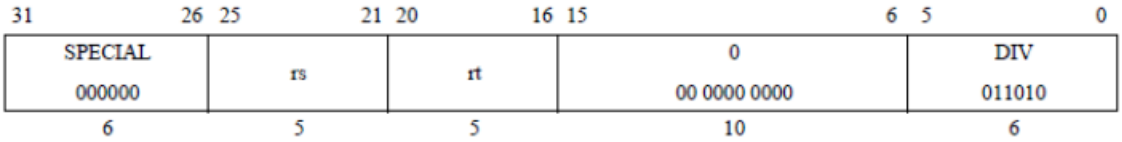

```
描述:
把rs*rt的结果存放到 HI  和   lo 里面
```



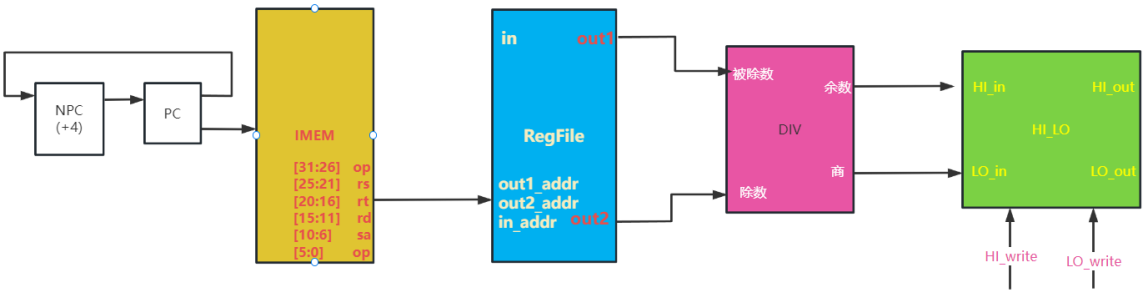

```
pc -> npc ,imem
out1 * out2 = res
res [63:32]-> hi
res [31:0] -> lo
```

通路类型25

52.DIV

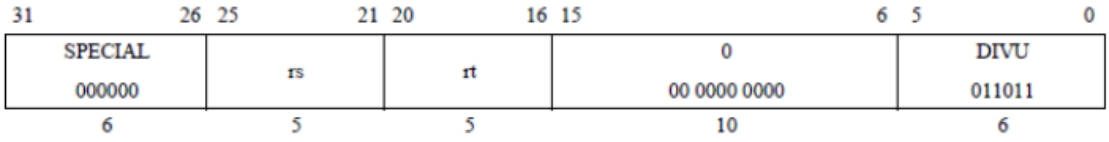



```
得到除法结果
HI = rt % ts
lo = rt /rs
```

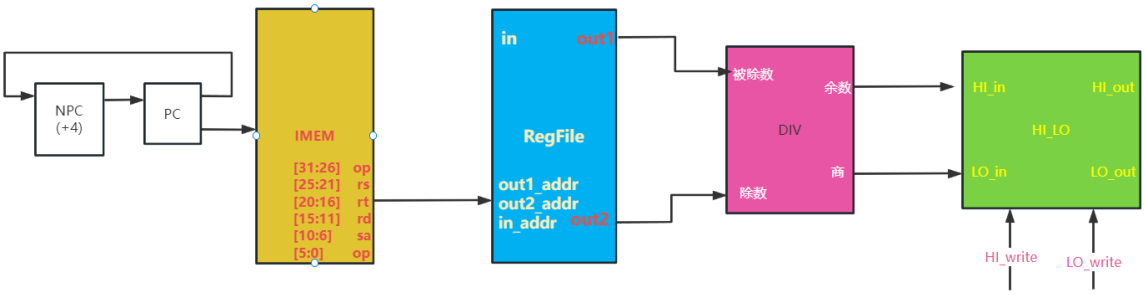



```
pc -> npc ,imem
rs ->out1
rd ->out2
out1 / out2 ->lo_in
out1%out2 ->hi_in
```

53.DIVU



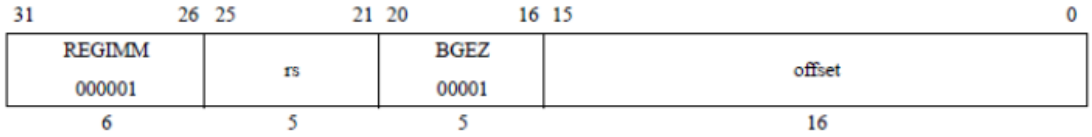

```
得到除法结果 (无符号结果)
HI = rt % ts
lo = rt /rs
```



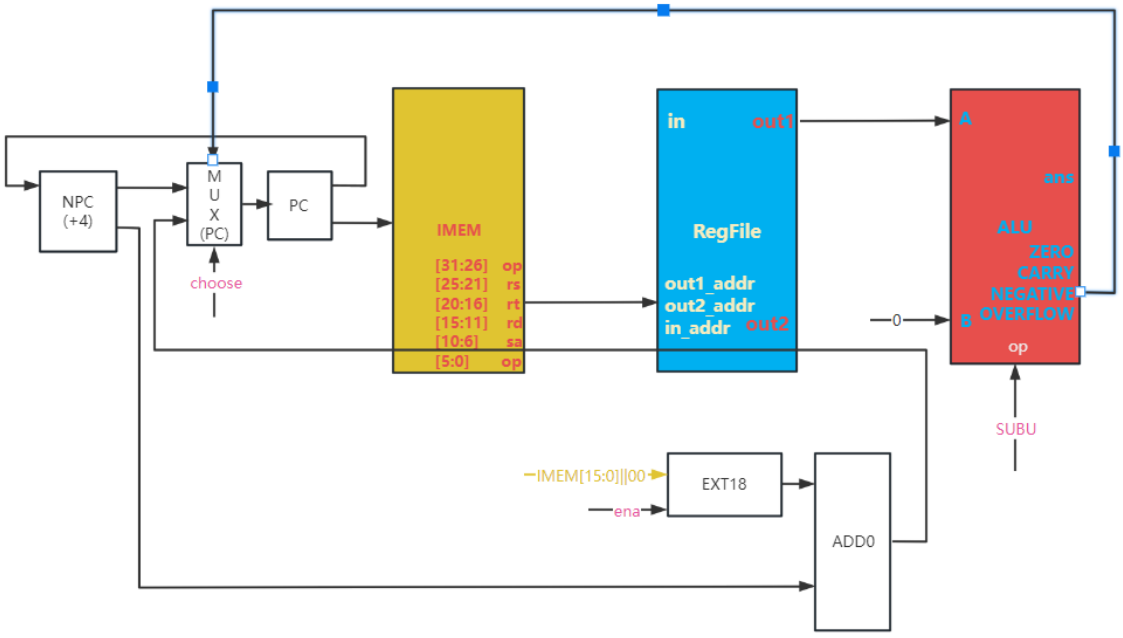

```
pc -> npc ,imem
rs ->out1
rd ->out2
out1 / out2 ->lo_in
out1%out2 ->hi_in
```

通路类型26

54.BGEZ




```
描述:
如果rs >= 0 则pc 跳转到  npc + (符号扩展offset||00)
否则npc = pc +4
```



```
pc -> npc ,imem
out1  -  0
if neg
pc = npc
else
pc = NPC  +  imem||00
```