计算机组成原理

指令集设计

徐逸辰
2018213555
计算机学院
yichen.xu@monad.email

摘要. 本文简单地描述了一种参考了 MIPS32 的指令集设计。本指令集选用了 MIPS32 中一部分常用指令，最大程度地保留了 MIPS32 原有的指令格式，并在其基础上作出拓展，使指令集能够支持更多种寻址方式。本文首先描述了指令集的设计宗旨，随后呈现了指令集所假设的机器架构，最后对指令集中的每一条指令进行了解释说明。

目录

1 设计宗旨

本指令集的设计参考了十分经典的 MIPS32 指令集。选择了 MIPS32 中的一部分常用指令，并且添加了一些新设计的指令。本指令集设计的原则是：尽量多地保留 MIPS32 中指令的规格，在已有指令的基础上，设计符合 MIPS32 指令格式的新指令，使得指令集能够支持更多种寻址方式。这样以来，产生的指令集将更加规整，并且很好地拓展了 MIPS32，实现了更多寻址方式。

指令集中的指令可以分为 4 类，分别为：传送与加载指令、栈操作指令、算数与逻辑运算指令、分支与判断指令。

2 机器架构

机器架构如图 1 所示。采用 32 位，单总线结构。有 32 个用户寄存器，也即 $R_0, R_1, \cdots, R_{31}$。其中，R_0 只读，不可写入，且其值恒为 0。除此以外，有 SP 寄存器位栈指针，IR 寄存器为指令寄存器，PC 为程序计数器，DR 为数据寄存器，SR 为状态寄存器，HI, LO, Z 皆为 ALU 的结果寄存器，X 为 ALU 的输入寄存器。

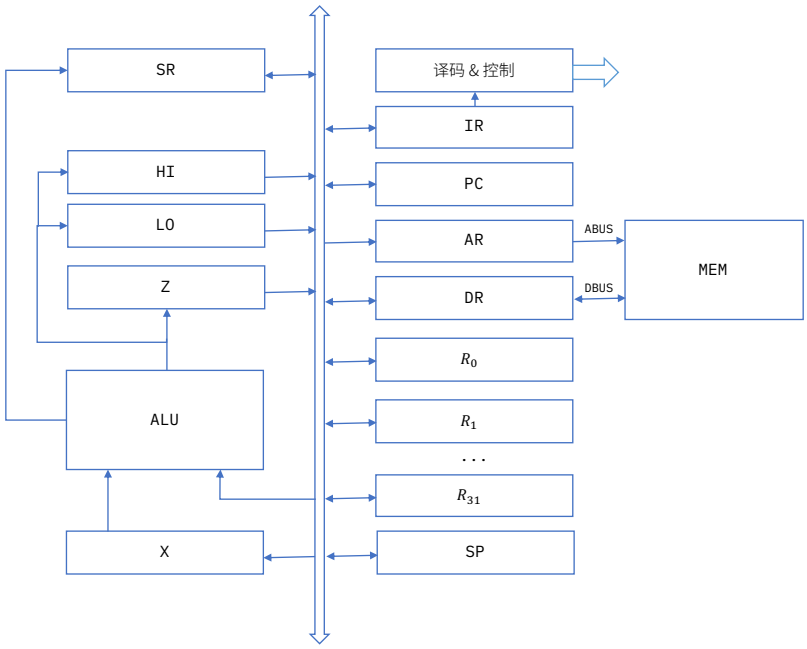


图 1: 机器架构图

3 指令设计

3.1 Mov

指令格式

`mov` r_d, r_s

指令功能

$r_d \leftarrow r_s$
将 r_s 的值送入 r_d。

限制

r_d 必须为 $R_1, R_2, \cdots, R_{31}$ 中的一个。

3.2 Lwi

指令格式

`lwi` r_d, Imm_{16}

指令功能

$r_d \leftarrow \text{signed\_extend}(Imm_{16})$
将 16 位立即数做符号扩展后送入 r_d。

限制

r_d 必须为 $R_1, R_2, \cdots, R_{31}$ 中的一个。

3.3 Lwda

指令格式

`lwda` $r_d, (Imm_{16})$

指令功能

$r_d \leftarrow mem[PC[31..18]||Imm_{16}||00]$
基于 PC 地址进行直接寻址。

限制

r_d 必须为 $R_1, R_2, \cdots, R_{31}$ 中的一个。

3.4 LWIA

指令格式

`lwia` $r_d, ((Imm_{16}))$

指令功能

$r_d \leftarrow mem[mem[PC[31..18]||Imm_{16}||00]]$
基于 PC 地址进行间接寻址。

限制

r_d 必须为 $R_1, R_2, \cdots, R_{31}$ 中的一个。

3.5 LWRIA

指令格式

`lwria` $r_d, (r_s)$

指令功能

$r_d \leftarrow mem[r_s]$
寄存器间接寻址。

限制

r_d 必须为 $R_1, R_2, \cdots, R_{31}$ 中的一个。

3.6 Lw

指令格式

`lw` $r_d, r_s, offset_{16}$

指令功能

$r_d \leftarrow mem[r_s + offset_{16}]$
基于寄存器 r_s 与立即数偏移量进行偏移寻址。

限制

r_d 必须为 $R_1, R_2, \cdots, R_{31}$ 中的一个。

3.7 Swda

指令格式

`swda` $r_s, (Imm_{16})$

指令功能

$mem[PC[31..18]||Imm_{16}||00] \leftarrow r_s$
基于 PC 和立即数进行直接寻址。

3.8 Swia

指令格式

`swia` $r_s, ((Imm_{16}))$

指令功能

$mem[mem[PC[31..18]||Imm_{16}||00]] \leftarrow r_s$
基于 PC 与立即数进行间接寻址。

3.9 SWRIA

指令格式

`swria` $r_s, (r_t)$

指令功能

$mem[r_t] \leftarrow r_s$
基于寄存器 r_t 进行间接寻址。

3.10 SW

指令格式

`sw` $r_s, r_t, offset_{16}$

指令功能

$mem[r_t + offset_{16}] \leftarrow r_s$
基于寄存器 r_t 与立即数偏移量进行偏移寻址。

3.11 PUSH

指令格式

`push` r_s

指令功能

$mem[SP] \leftarrow r_s$
$SP \leftarrow SP - 4$
将 r_s 中的数据压入栈顶。递减栈指针 SP。

3.12 POP

指令格式

`pop` r_d

指令功能

$r_d \leftarrow mem[SP]$
$SP \leftarrow SP + 4$
将栈顶数据打入 r_d。递增栈指针 SP。

限制

r_d 必须为 $R_1, R_2, \cdots, R_{31}$ 中的一个。

3.13 PUSHI

指令格式

`pushi` Imm_{28}

指令功能

$mem[SP] \leftarrow \text{signed\_extend}(Imm_{28})$
$SP \leftarrow SP - 4$
将 28 位立即数符号扩展至 32 位后压入栈中。递减栈指针 SP。

3.14 ADD

指令格式

`add` r_d, r_s, r_t

指令功能

$r_d \leftarrow r_s + r_t$
执行加法。考虑有符号溢出。

限制

r_d 必须为 $R_1, R_2, \cdots, R_{31}$ 中的一个。

3.15 ADDI

指令格式

`addi` r_d, r_s, Imm_{16}

指令功能

$r_d \leftarrow r_s + Imm_{16}^{\pm}$
将 16 位立即数有符号扩展至 32 位后执行加法。考虑有符号溢出。

限制

r_d 必须为 $R_1, R_2, \cdots, R_{31}$ 中的一个。

3.16 ADDU

指令格式

`addu` r_d, r_s, r_t

指令功能

$r_d \leftarrow r_s + r_t$
执行加法。不考虑有符号溢出。

限制

r_d 必须为 $R_1, R_2, \cdots, R_{31}$ 中的一个。

3.17 ADDIU

指令格式

`addi` r_d, r_s, Imm_{16}

指令功能

$r_d \leftarrow r_s + Imm_{16}^{\varnothing}$
将 16 位立即数零扩展至 32 位后执行加法。不考虑有符号溢出。

限制

r_d 必须为 $R_1, R_2, \cdots, R_{31}$ 中的一个。

3.18 SUB

指令格式

sub r_d, r_s, r_t

指令功能

$r_d \leftarrow r_s - r_t$
执行有符号减法。

限制

r_d 必须为 $R_1, R_2, \cdots, R_{31}$ 中的一个。

3.19 SUBU

指令格式

subu r_d, r_s, r_t

指令功能

$r_d \leftarrow r_s -_{\varnothing} r_t$
执行无符号减法。

限制

r_d 必须为 $R_1, R_2, \cdots, R_{31}$ 中的一个。

3.20 NEGU

指令格式

negu r_d, r_s

指令功能

$r_d \leftarrow -r_s$
求 r_s 的补，送入 r_d 中。

限制

r_d 必须为 $R_1, R_2, \cdots, R_{31}$ 中的一个。

3.21 DIV

指令格式

`div` r_d, r_s, r_t

指令功能

$LO \leftarrow r_s / r_t$
$HI \leftarrow r_s \% r_t$
执行有符号除法。

3.22 DIVU

指令格式

`divu` r_d, r_s, r_t

指令功能

$LO \leftarrow r_s^{\varnothing} / r_t^{\varnothing}$
$HI \leftarrow r_s^{\varnothing} \% r_t^{\varnothing}$
执行无符号除法。

3.23 MULT

指令格式

`mult` r_d, r_s, r_t

指令功能

$HI||LO \leftarrow r_s \times r_t$
执行有符号乘法。

3.24 MULTU

指令格式

`multu` r_d, r_s, r_t

指令功能

$HI||LO \leftarrow r_s^{\varnothing} \times r_t^{\varnothing}$
执行无符号乘法。

3.25 MFHI

指令格式

`mfhi` r_d

指令功能

$r_d \leftarrow HI$
将 HI 送入 r_d 中。

限制

r_d 必须为 $R_1, R_2, \cdots, R_{31}$ 中的一个。

3.26 MFLO

指令格式

`mflo` r_d

指令功能

$r_d \leftarrow LO$
将 LO 送入 r_d 中。

限制

r_d 必须为 $R_1, R_2, \cdots, R_{31}$ 中的一个。

3.27 AND

指令格式

and r_d, r_s, r_t

指令功能

$r_d \leftarrow r_s$ AND r_t
执行位与。

限制

r_d 必须为 $R_1, R_2, \cdots, R_{31}$ 中的一个。

3.28 ANDI

指令格式

and r_d, r_s, Imm_{16}

指令功能

$r_d \leftarrow r_s$ AND $Imm_{16}^{\varnothing}$
将 16 位立即数零扩展到 32 位后执行位与。

限制

r_d 必须为 $R_1, R_2, \cdots, R_{31}$ 中的一个。

3.29 OR

指令格式

or r_d, r_s, r_t

指令功能

$r_d \leftarrow r_s$ OR r_t
执行位或。

限制

r_d 必须为 $R_1, R_2, \cdots, R_{31}$ 中的一个。

3.30 ORI

指令格式

ori r_d, r_s, Imm_{16}

指令功能

$r_d \leftarrow r_s$ OR $Imm_{16}^{\varnothing}$
将 16 位立即数零扩展到 32 位后执行位或。

限制

r_d 必须为 $R_1, R_2, \cdots, R_{31}$ 中的一个。

3.31 XOR

指令格式

xor r_d, r_s, r_t

指令功能

$r_d \leftarrow r_s \oplus r_t$
执行位异或。

限制

r_d 必须为 $R_1, R_2, \cdots, R_{31}$ 中的一个。

3.32 XORI

指令格式

`xori` r_d, r_s, Imm_{16}

指令功能

$r_d \leftarrow r_s \oplus Imm_{16}^{\varnothing}$
将 16 位立即数零扩展到 32 位后执行位异或。

限制

r_d 必须为 $R_1, R_2, \cdots, R_{31}$ 中的一个。

3.33 NOT

指令格式

`not` r_d, r_s

指令功能

$r_d \leftarrow \neg r_s$
求 r_s 的按位非，送入 r_d 中。

限制

r_d 必须为 $R_1, R_2, \cdots, R_{31}$ 中的一个。

3.34 NOP

指令格式

`nop`

3.35 SLT

指令格式

`slt` r_d, r_s, r_t

指令功能

$r_d \leftarrow r_s < r_t$
比较 r_s 与 r_t 的大小，并将结果送入 r_d。

限制

r_d 必须为 $R_1, R_2, \cdots, R_{31}$ 中的一个。

3.36 SLTI

指令格式

`slti` r_d, r_s, Imm_{16}

指令功能

$r_d \leftarrow r_s < Imm_{16}^{\pm}$
将 16 位立即数符号扩展到 32 位后比较与 r_s 的大小，并将结果送入 r_d。

限制

r_d 必须为 $R_1, R_2, \cdots, R_{31}$ 中的一个。

3.37 SLTU

指令格式

`sltu` r_d, r_s, r_t

指令功能

$r_d \leftarrow r_s^{\varnothing} < r_t^{\varnothing}$
比较 $r_s^{\varnothing}$ 与 $r_t^{\varnothing}$ 的大小，并将结果送入 r_d。

限制

r_d 必须为 $R_1, R_2, \cdots, R_{31}$ 中的一个。

3.38 SLTIU

指令格式

`sltiu` r_d, r_s, Imm_{16}

指令功能

$r_d \leftarrow r_s^{\varnothing} < Imm_{16}^{\varnothing}$
将 16 位立即数零扩展到 32 位后比较与 $r_s^{\varnothing}$ 的大小，并将结果送入 r_d。

限制

r_d 必须为 $R_1, R_2, \cdots, R_{31}$ 中的一个。

3.39 B

指令格式

`b` $offset_{18}$

指令功能

$PC \leftarrow PC + offset_{18}$
无条件跳转到 PC 寄存器之后 $offset_{18}$ 处继续执行指令。

限制

$offset_{18}$ 必须为 4 的倍数，也即最低 2 位为 0。

3.40 BAL

指令格式

`bal` $offset_{18}$

指令功能

$R_{31} \leftarrow PC + 4$

$PC \leftarrow PC + offset_{18}$

无条件跳转到 PC 寄存器之后 $offset_{18}$ 处继续执行指令。并且将返回地址写入 R_{31}，为当前 PC 的下一条指令。

限制

$offset_{18}$ 必须为 4 的倍数，也即最低 2 位为 0。

3.41 BEQ

指令格式

beq $r_s, r_t, offset_{18}$

指令功能

If $r_s = r_t$, $PC \leftarrow PC + offset_{18}$

若 $r_s = r_t$ 则到 PC 寄存器之后 $offset_{18}$ 处继续执行指令。

限制

$offset_{18}$ 必须为 4 的倍数，也即最低 2 位为 0。

3.42 BNE

指令格式

bne $r_s, r_t, offset_{18}$

指令功能

If $r_s \neq r_t$, $PC \leftarrow PC + offset_{18}$

若 $r_s \neq r_t$ 则到 PC 寄存器之后 $offset_{18}$ 处继续执行指令。

限制

$offset_{18}$ 必须为 4 的倍数，也即最低 2 位为 0。

3.43 BGEZ

指令格式

bgez $r_s, offset_{18}$

指令功能

If $r_s \geq 0$, $PC \leftarrow PC + offset_{18}$
若 $r_s \geq 0$ 则到 PC 寄存器之后 $offset_{18}$ 处继续执行指令。

限制

$offset_{18}$ 必须为 4 的倍数，也即最低 2 位为 0。

3.44 BGTZ

指令格式

bgtz $r_s, offset_{18}$

指令功能

If $r_s > 0$, $PC \leftarrow PC + offset_{18}$
若 $r_s > 0$ 则到 PC 寄存器之后 $offset_{18}$ 处继续执行指令。

限制

$offset_{18}$ 必须为 4 的倍数，也即最低 2 位为 0。

3.45 BLEZ

指令格式

blez $r_s, offset_{18}$

指令功能

If $r_s \leq 0$, $PC \leftarrow PC + offset_{18}$
若 $r_s \leq 0$ 则到 PC 寄存器之后 $offset_{18}$ 处继续执行指令。

限制

$offset_{18}$ 必须为 4 的倍数，也即最低 2 位为 0。

3.46 J

指令格式

j $offset_{28}$

指令功能

$PC \leftarrow PC[31..28]||offset_{28}$

根据 PC 寄存器当前值，跳转到 256MB 区域内的任一地方继续执行。

3.47 JAL

指令格式

jal $offset_{28}$

指令功能

$R_{31} \leftarrow PC + 4$

$PC \leftarrow PC[31..28]||offset_{28}$

根据 PC 寄存器当前值，跳转到 256MB 区域内的任一地方继续执行。并将返回地址写入 R_{31}，为当前 PC 的下一条指令。

3.48 JR

指令格式

jr r_s

指令功能

$PC \leftarrow r_s$

跳转到 r_s 中的值代表的地址所在位置继续执行。

3.49 JRAL

指令格式

jral r_s

指令功能

$R_{31} \leftarrow PC + 4$

$PC \leftarrow r_s$

跳转到 r_s 中的值代表的地址所在位置继续执行。并将返回地址写入 R_{31}，为当前 PC 的下一条指令。

3.50 BREAK

指令格式

break

指令功能

SignalException(Breakpoint)

触发断点异常。无条件地转移到异常处理。

3.51 SYSCALL

指令格式

syscall

指令功能

SignalException(SystemCall)

触发系统调用异常。无条件地转移到异常处理。

3.52 ERET

指令格式

eret

指令功能

$PC \leftarrow EPC$
$SR.EXL \leftarrow 0$
从异常返回。

3.53 MFC0

指令格式

`mfc0` r_d, r_t, sel

指令功能

$r_d \leftarrow SR.CP0[r_t, sel]$
读取 SR 中 CP0 的状态寄存器。

3.54 MTC0

指令格式

`mtc0` r_s, r_t, sel

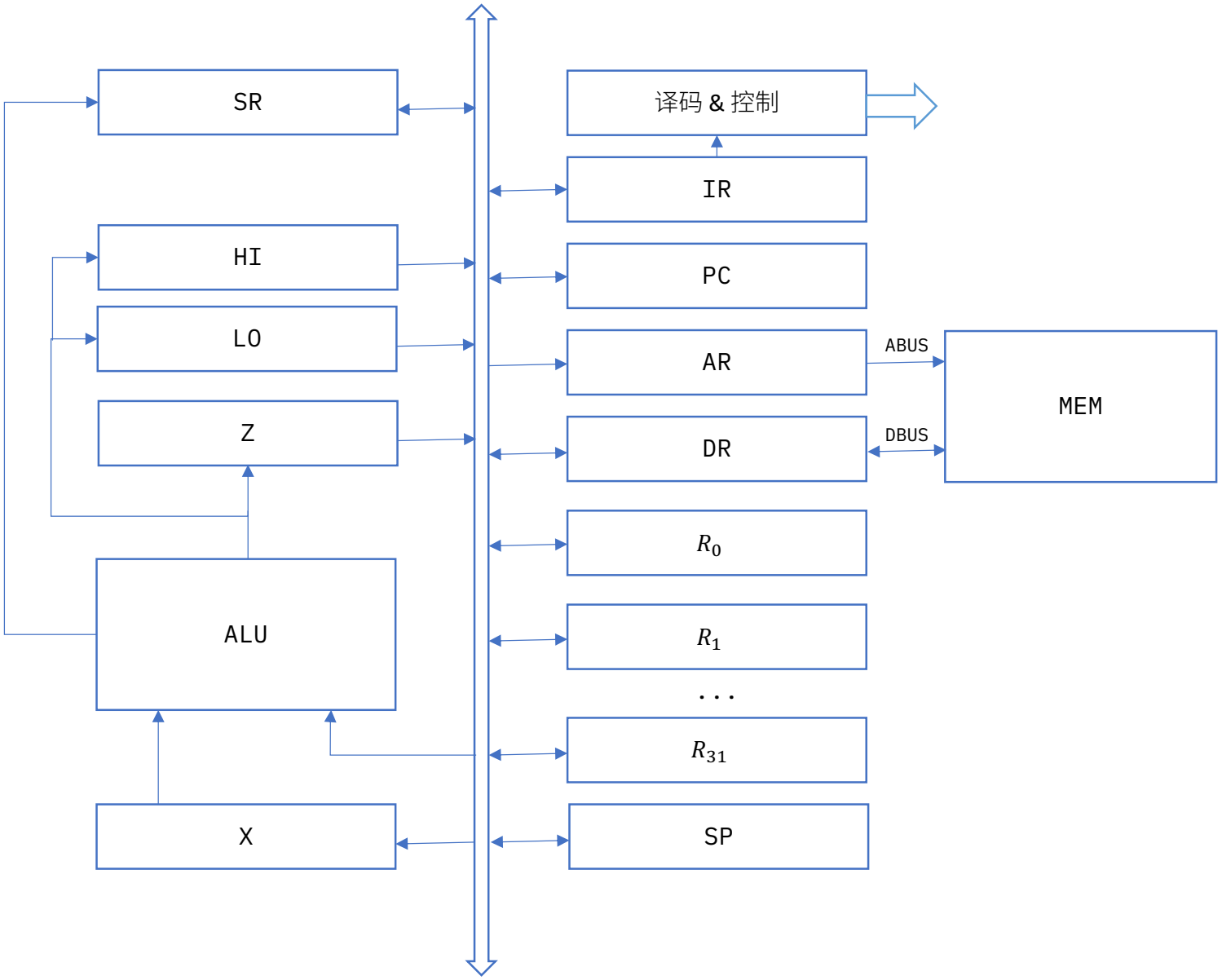
指令功能

$SR.CP0[r_t, sel] \leftarrow r_s$
写入 SR 中 CP0 的状态寄存器。

A 结构图与流程图

下文是机器的结构图与所有指令的执行流程图。

Architecture

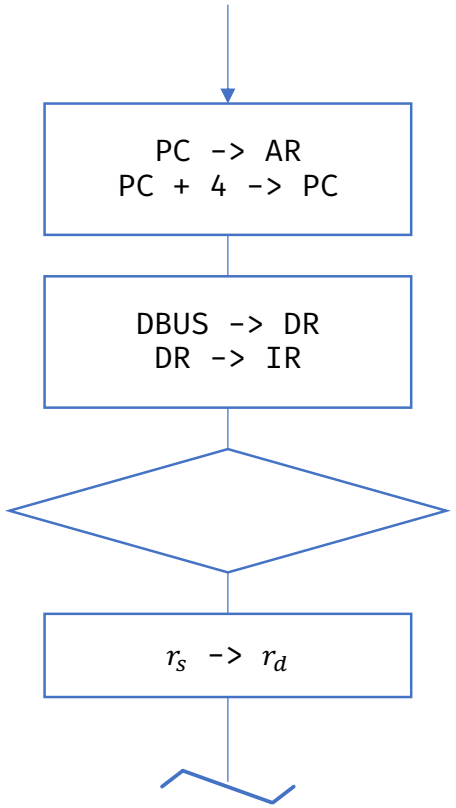


Instruction

mov r_d, r_s

Flow Chart of

MOV

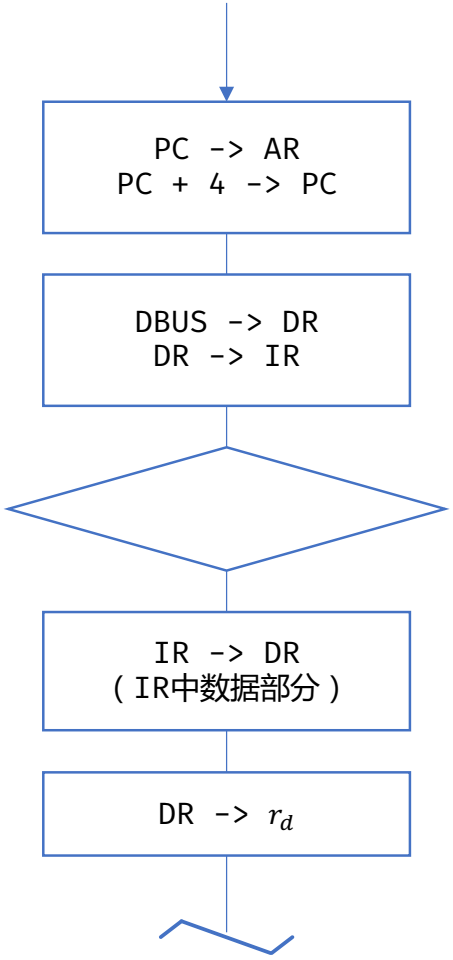


Instruction

`lwi` r_d, Imm_{16}

Flow Chart of

LWUI

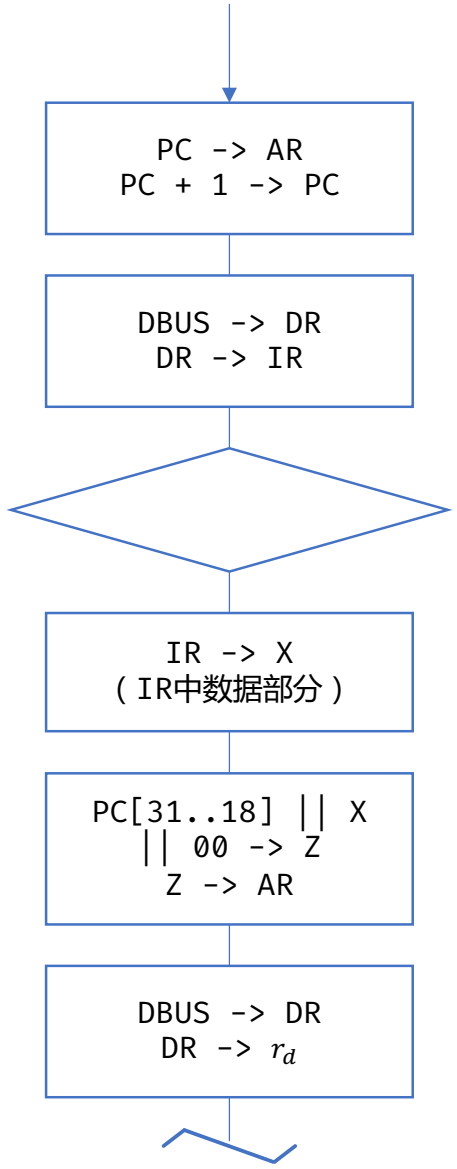


Instruction

`lwda` r_d, (Imm_{16})

Flow Chart of

LWDA

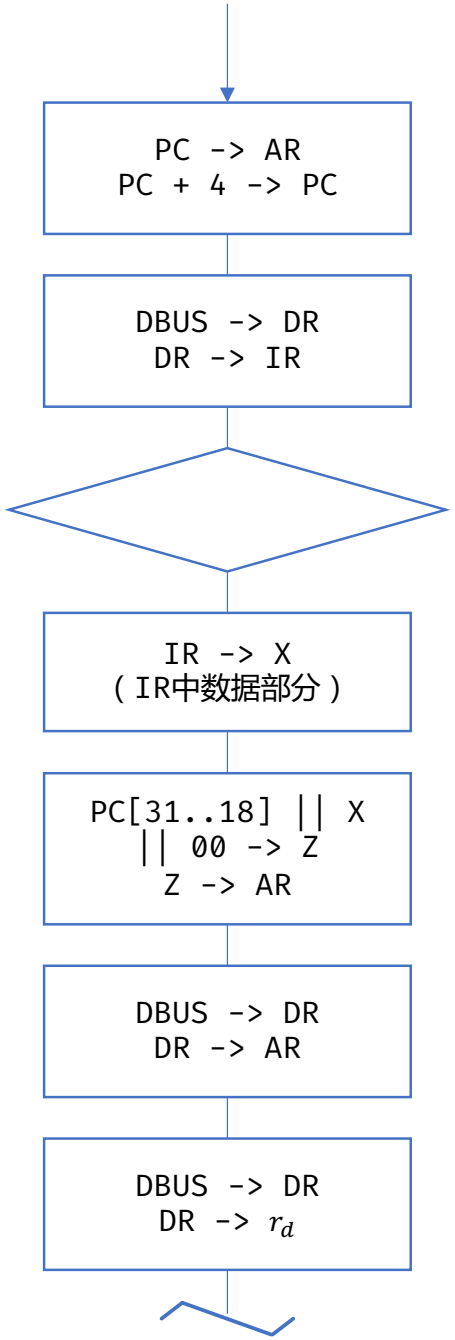


Instruction

`lwia` r_d, ((Imm_{16}))

Flow Chart of

LWIA

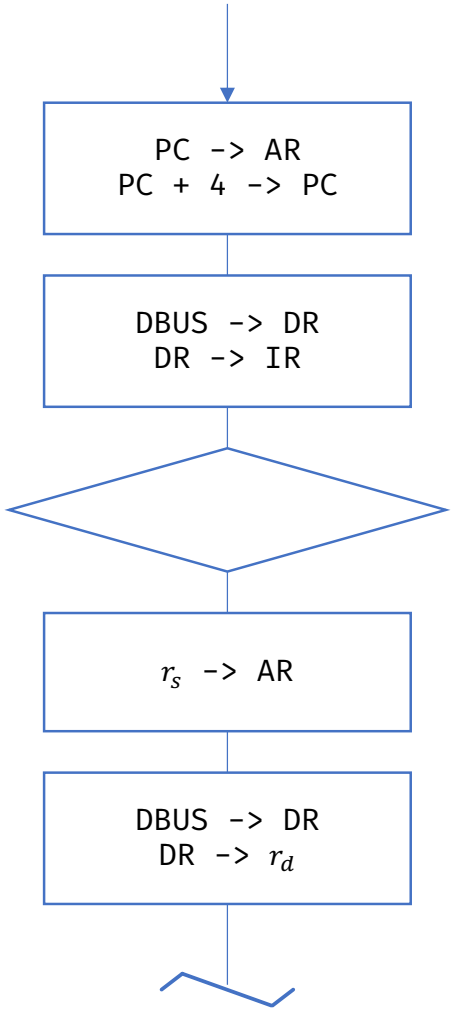


Instruction

`lwria` r_d`, (`r_s`)`

Flow Chart of

LWRIA

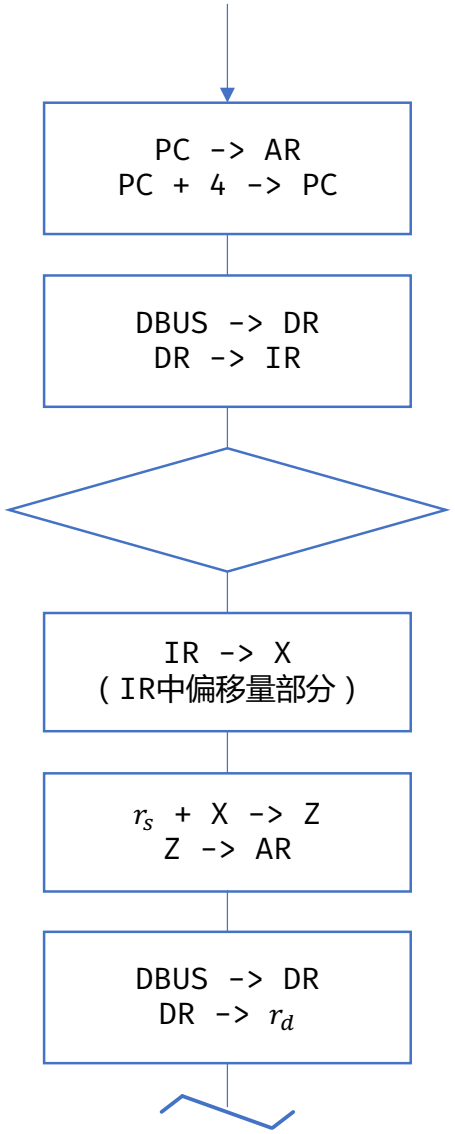


Instruction

lw r_d, r_s, $offset_{16}$

Flow Chart of

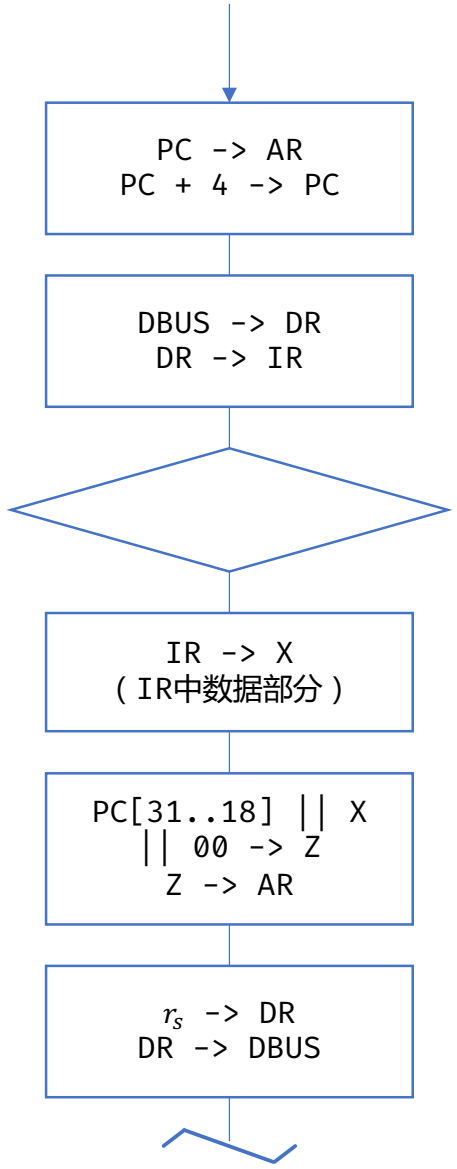
LW



Instruction

swda r_s, (Imm_{16})

Flow Chart of SWDA

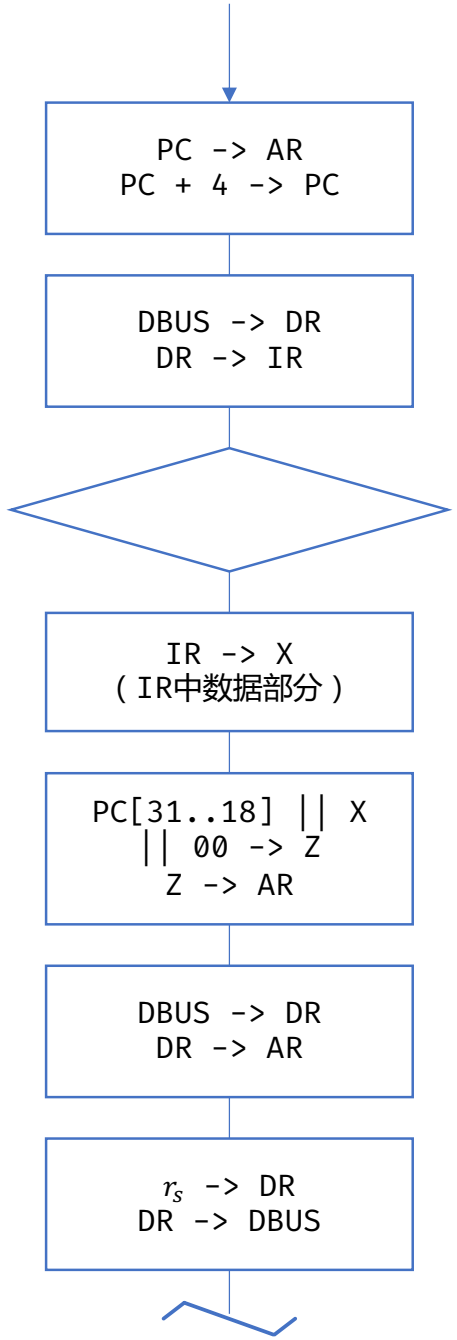


Instruction

swia r_s, ((Imm_{16}))

Flow Chart of

SWIA

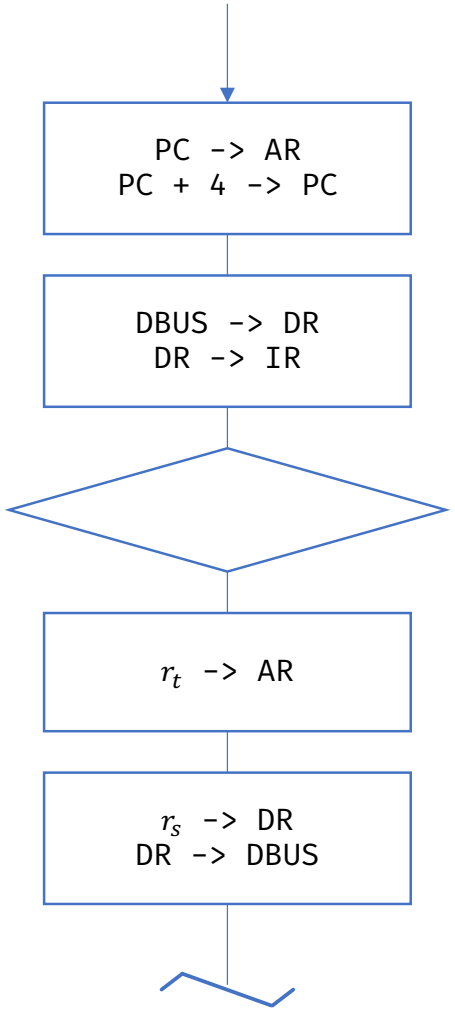


Instruction

`swria` r_s, (r_t)

Flow Chart of

SWRIA

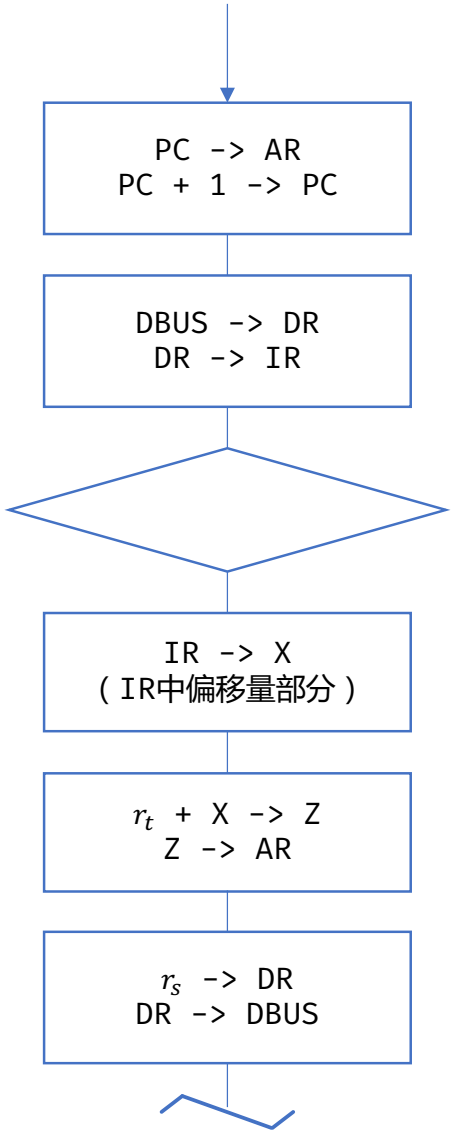


Instruction

sw r_s, r_t, $offset_{16}$

Flow Chart of

Sw

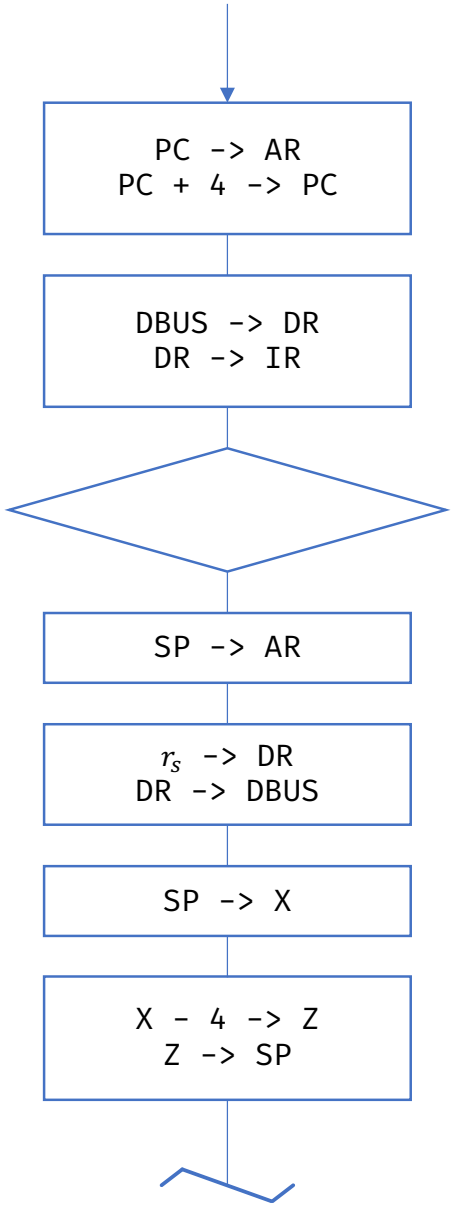


Instruction

`push` r_s

Flow Chart of

PUSH

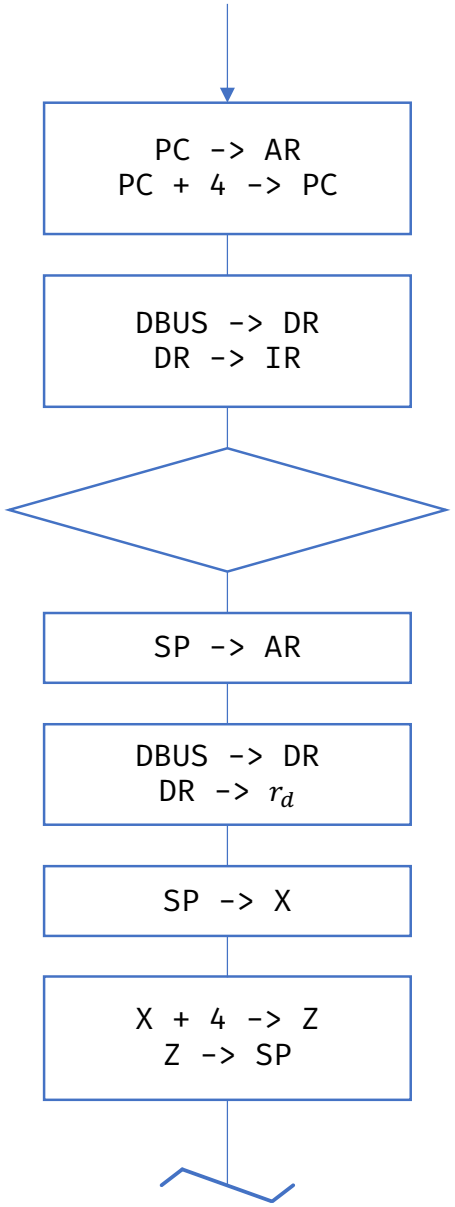


Instruction

pop r_d

Flow Chart of

POP

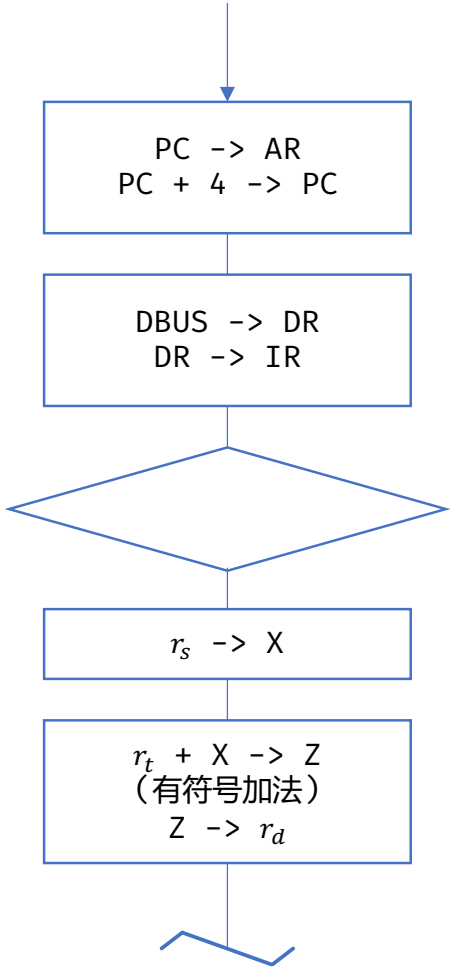


Instruction

add r_d, r_s, r_t

Flow Chart of

ADD

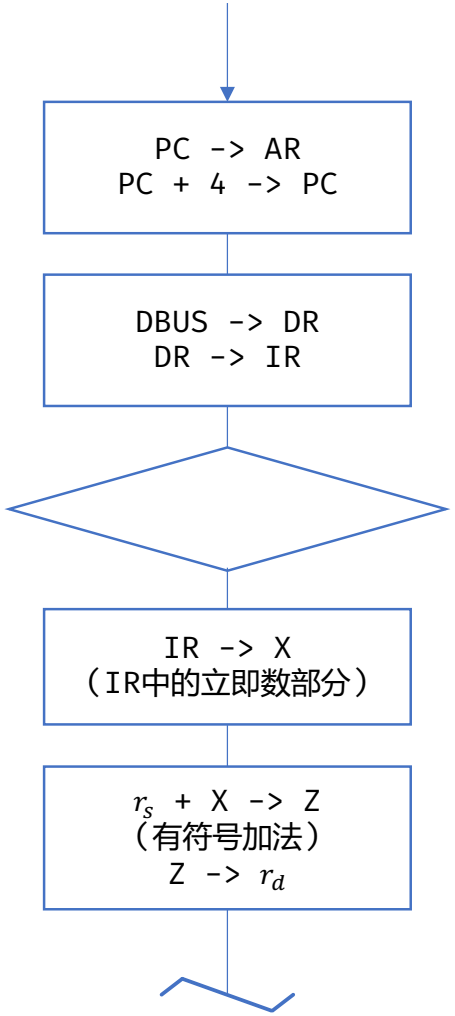


Instruction

addi r_d, r_s, Imm_{16}

Flow Chart of

ADDI

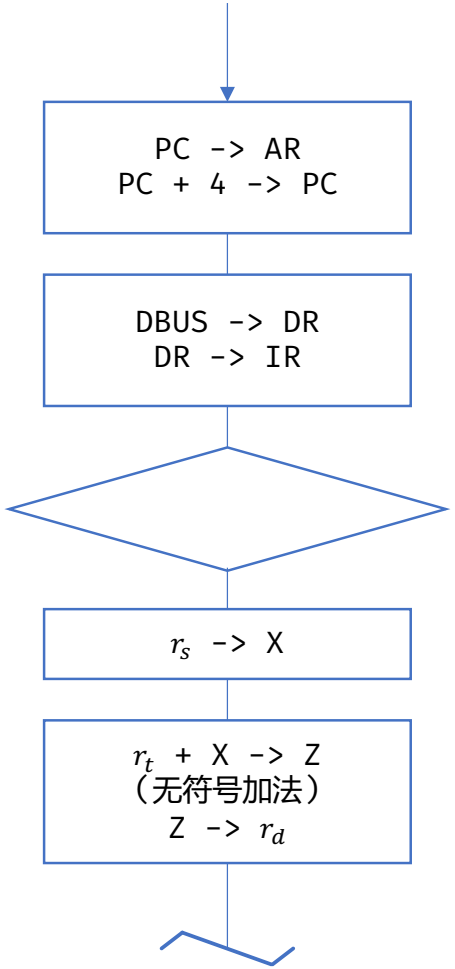


Instruction

addu r_d, r_s, r_t

Flow Chart of

ADDU

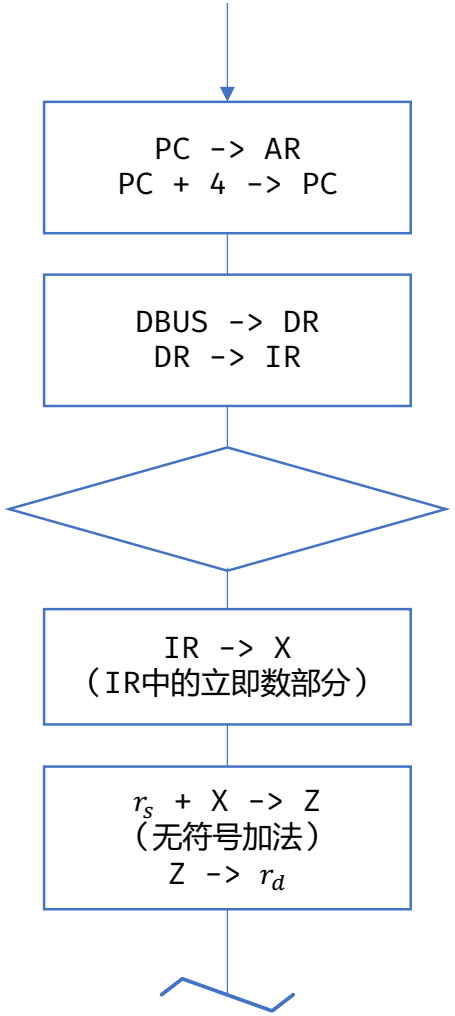


Instruction

addiu r_d, r_s, Imm_{16}

Flow Chart of

ADDIU

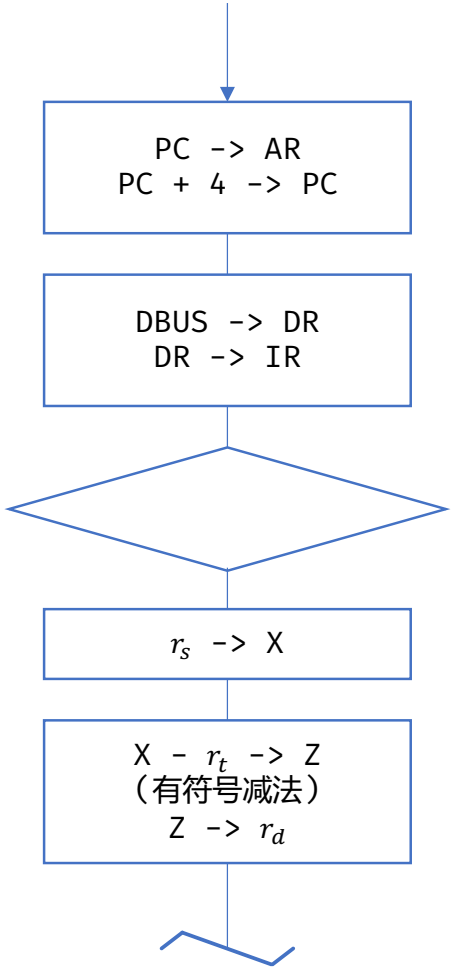


Instruction

sub r_d, r_s, r_t

Flow Chart of

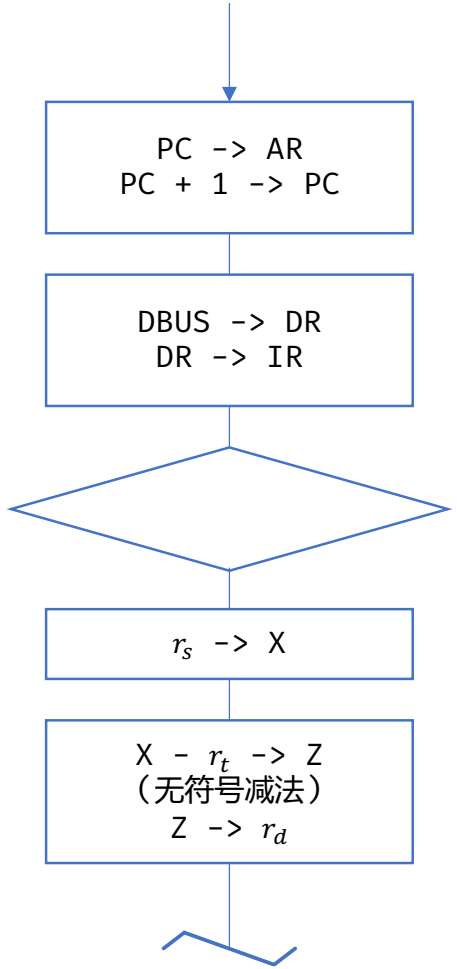
SUB



Instruction

subu r_d, r_s, r_t

Flow Chart of SUBU

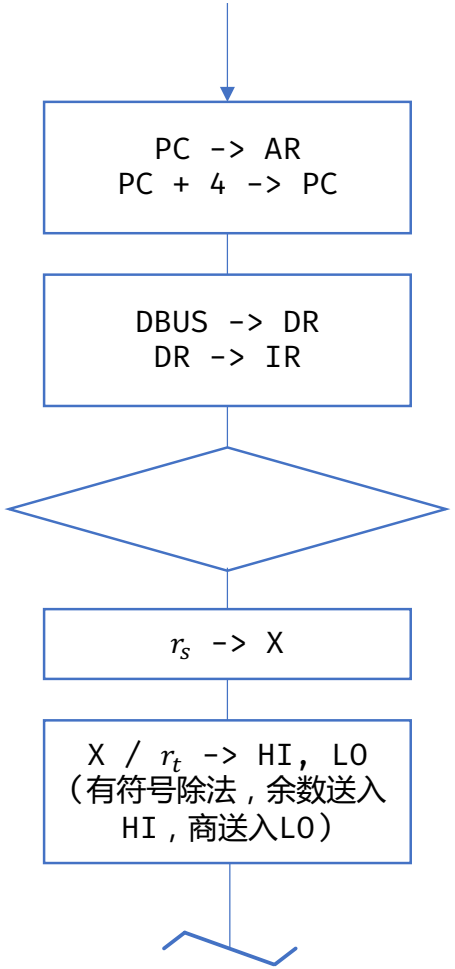


Instruction

div r_d, r_s, r_t

Flow Chart of

DIV

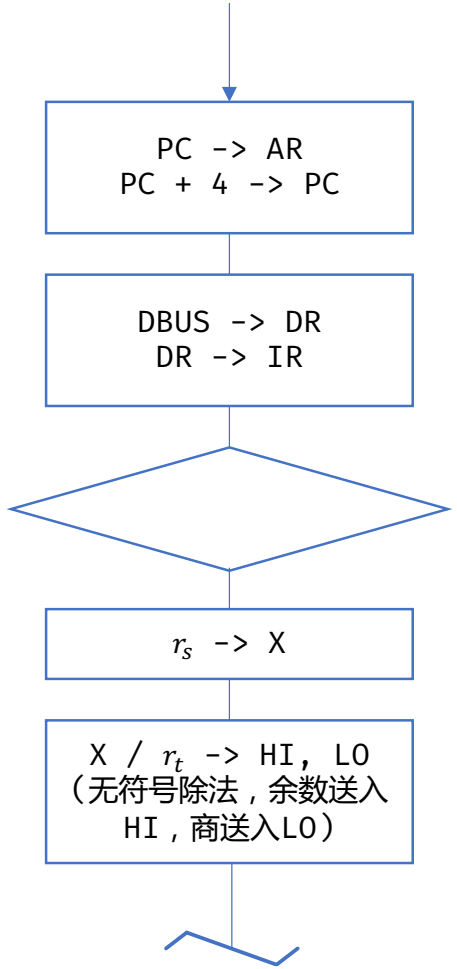


Instruction

divu r_d, r_s, r_t

Flow Chart of

DIVU

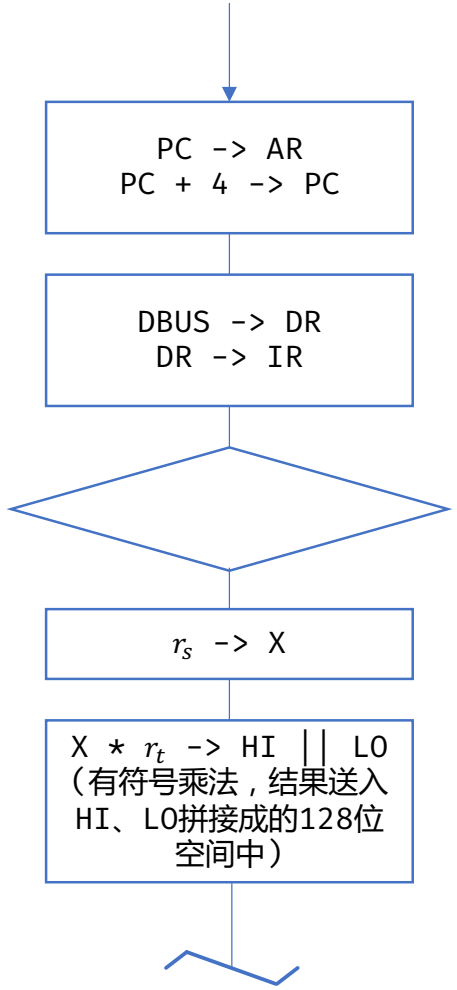


Instruction

mult r_d, r_s, r_t

Flow Chart of

MULT

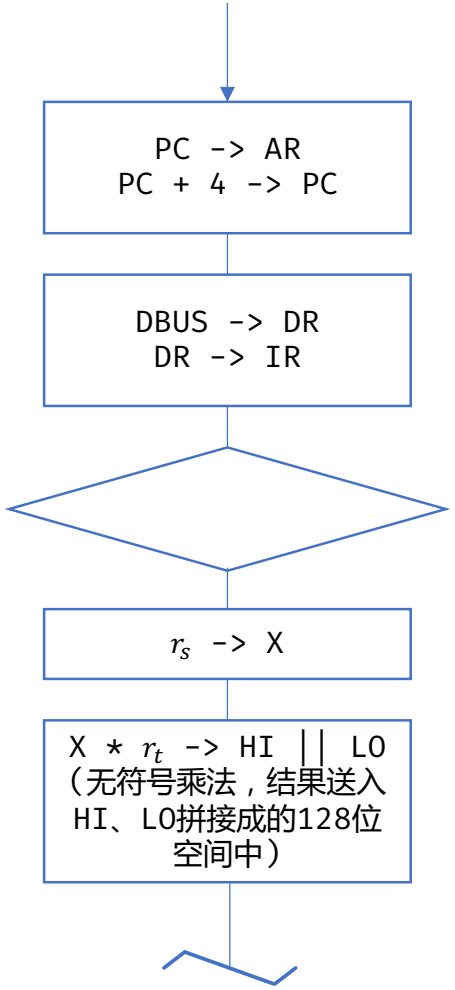


Instruction

multu r_d, r_s, r_t

Flow Chart of

MULTU

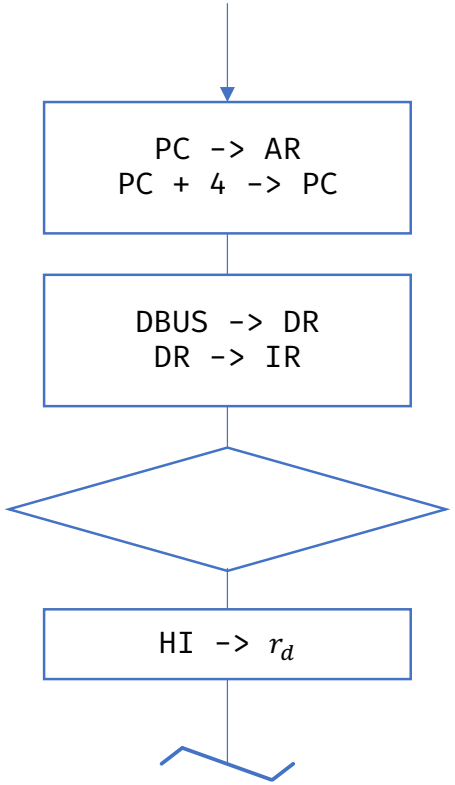


Instruction

`mfhi` r_d

Flow Chart of

MFHI

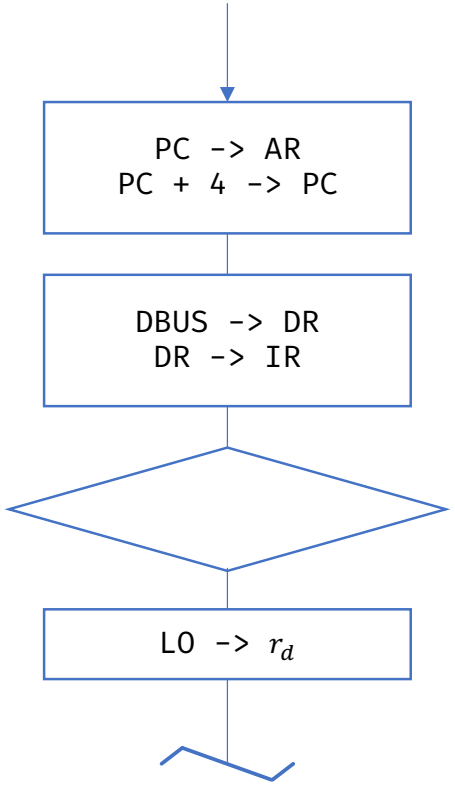


Instruction

`mflo` r_d

Flow Chart of

MFLO

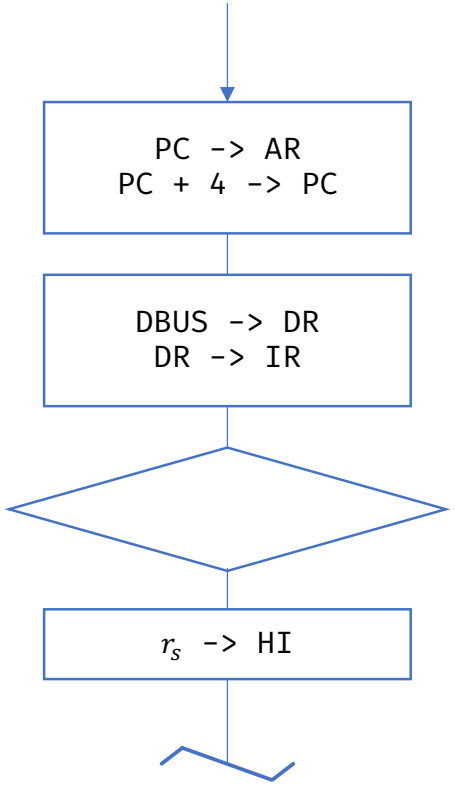


Instruction

`mthi` r_s

Flow Chart of

MTHI

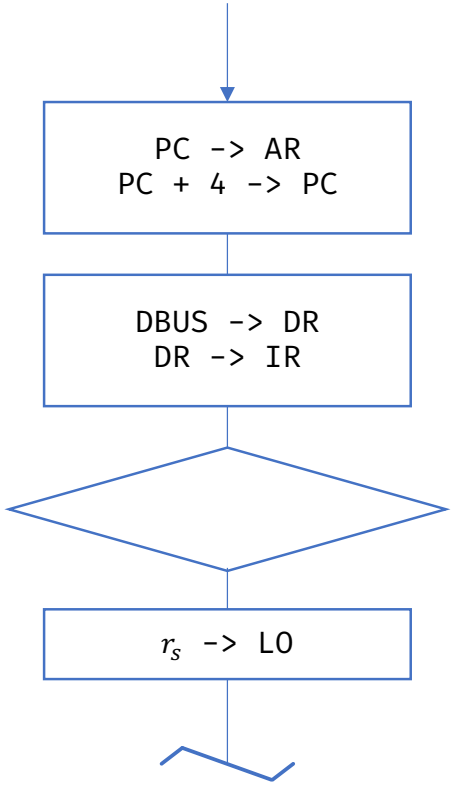


Instruction

`mtlo` r_s

Flow Chart of

MTLO

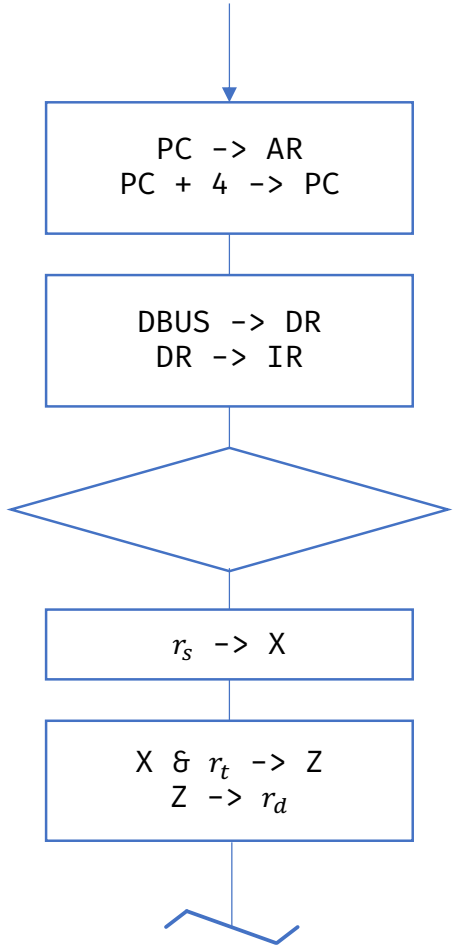


Instruction

and r_d, r_s, r_t

Flow Chart of

AND

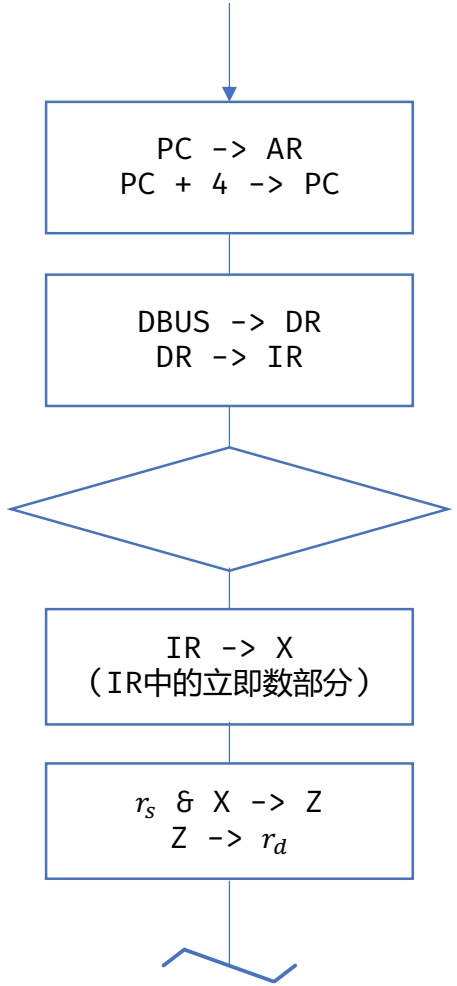


Instruction

andi r_d, r_s, Imm_{16}

Flow Chart of

ANDI

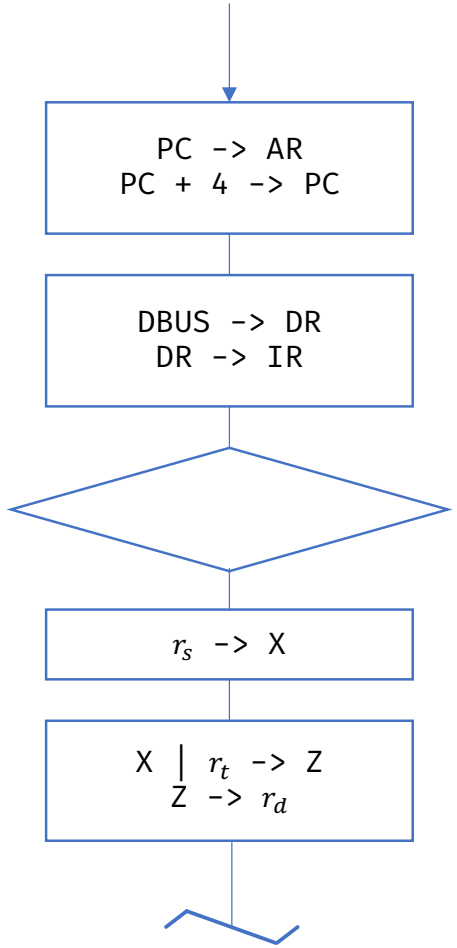


Instruction

or r_d, r_s, r_t

Flow Chart of

OR

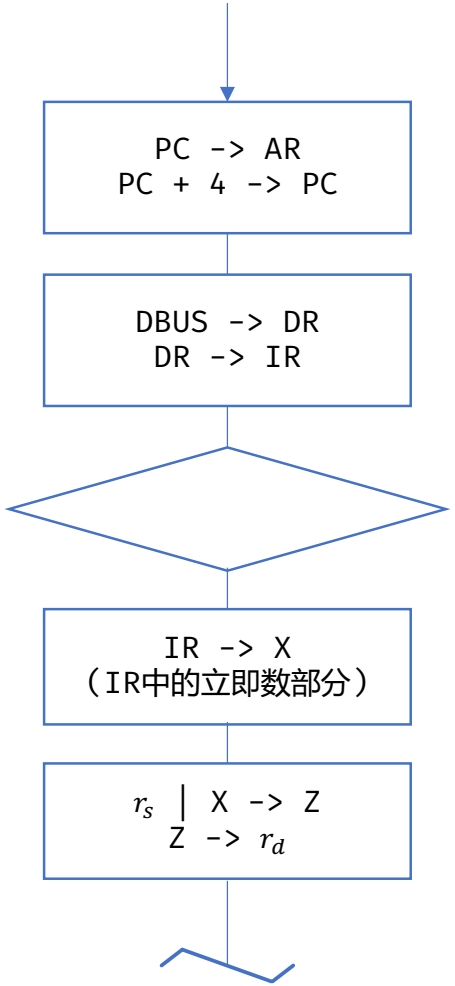


Instruction

ori r_d, r_s, Imm_{16}

Flow Chart of

ORI

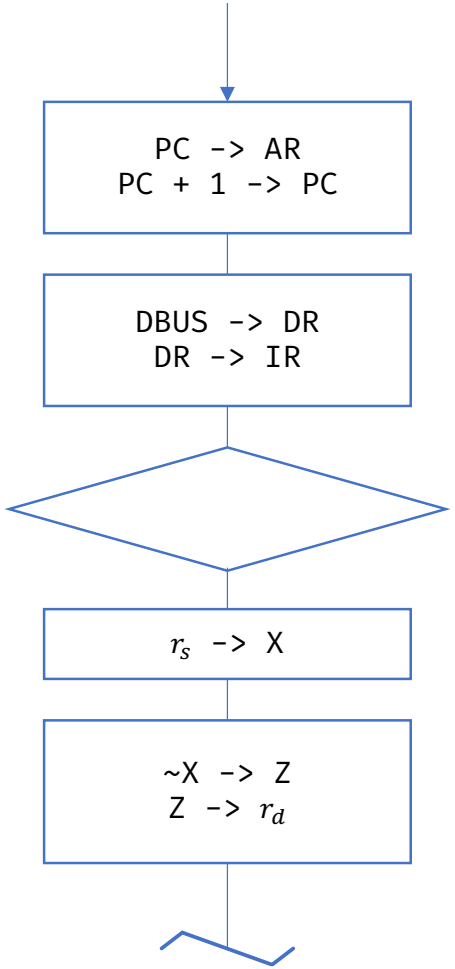


Instruction

not r_d, r_s

Flow Chart of

NOT

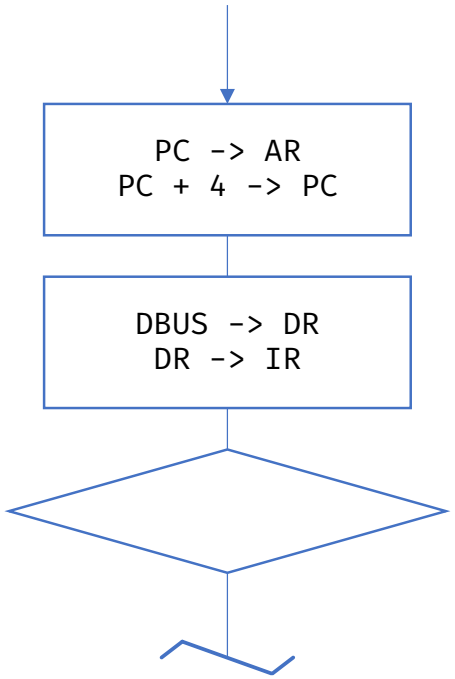


Instruction

`nop`

Flow Chart of

NOP

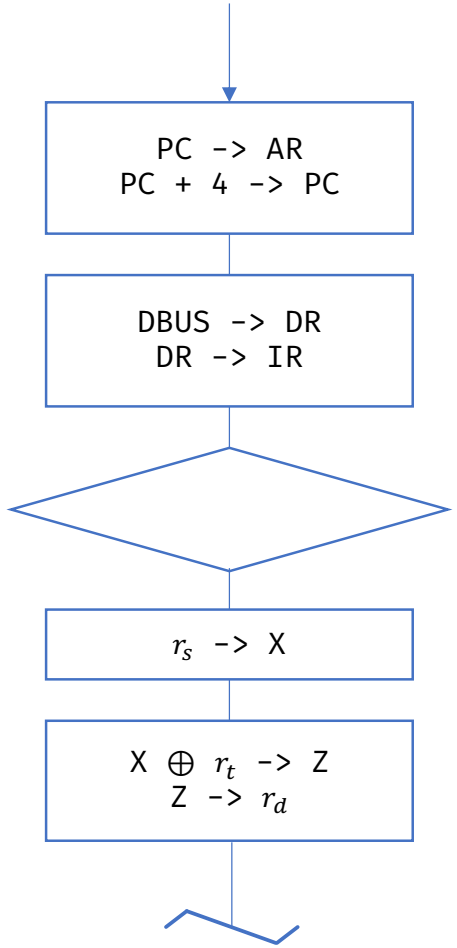


Instruction

xor r_d, r_s, r_t

Flow Chart of

XOR

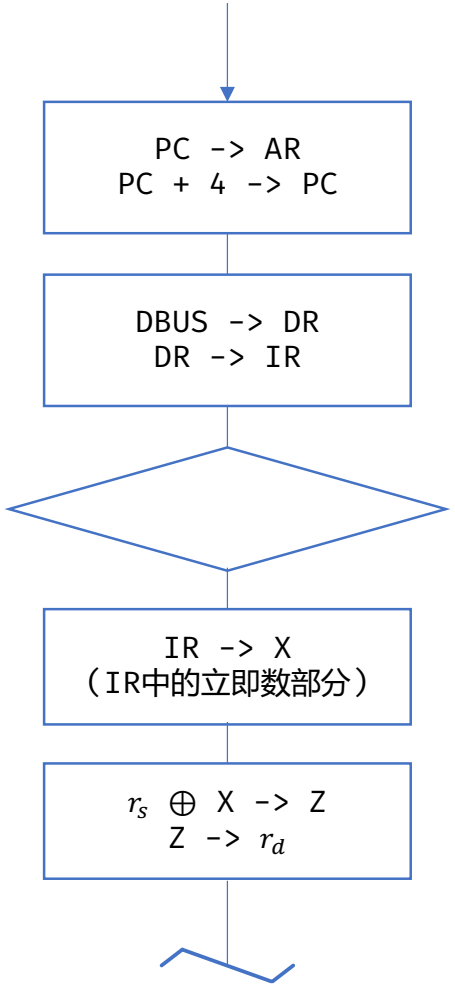


Instruction

xori r_d, r_s, Imm_{16}

Flow Chart of

XORI

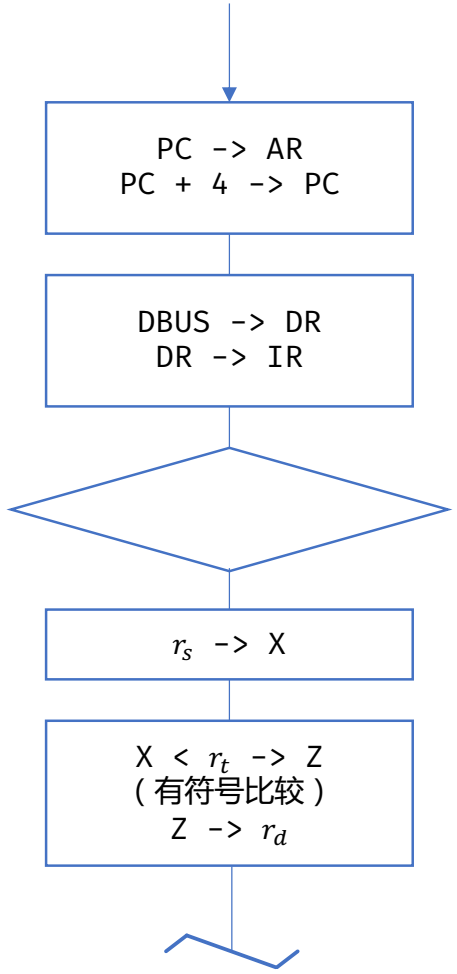


Instruction

slt r_d, r_s, r_t

Flow Chart of

SLT

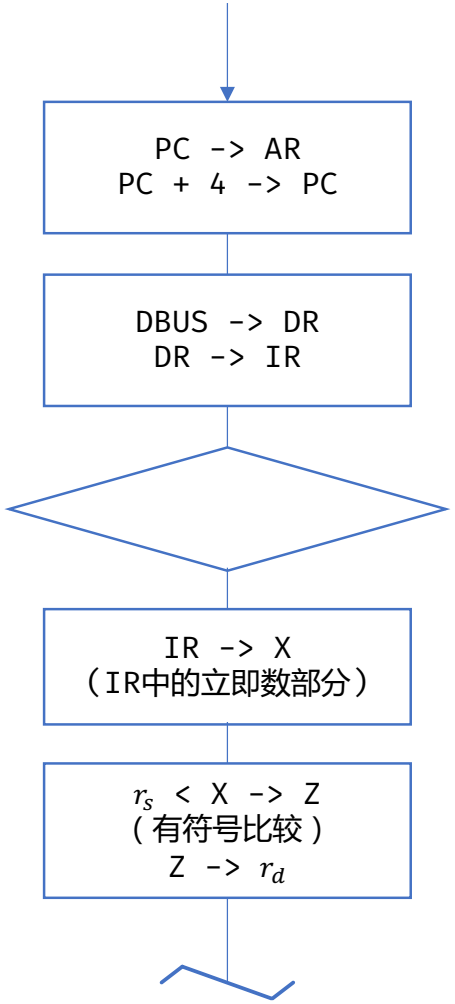


Instruction

slti r_d, r_s, Imm_{16}

Flow Chart of

SLTI

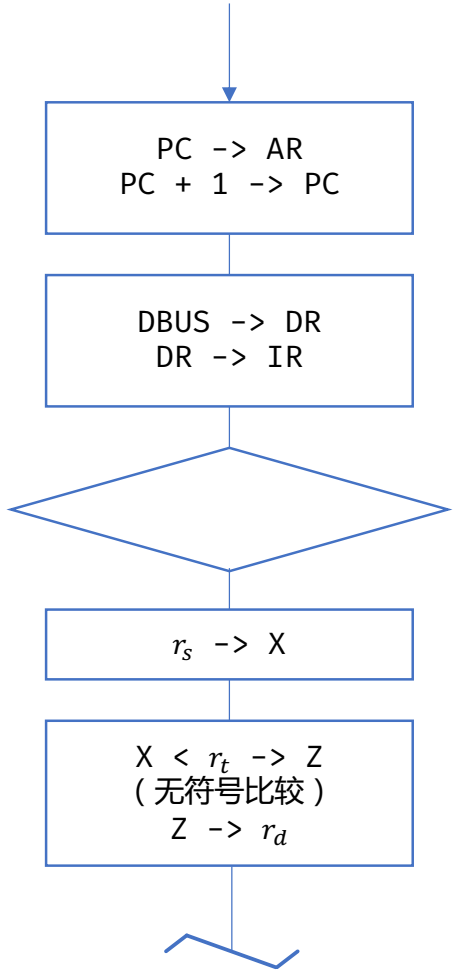


Instruction

sltu r_d, r_s, r_t

Flow Chart of

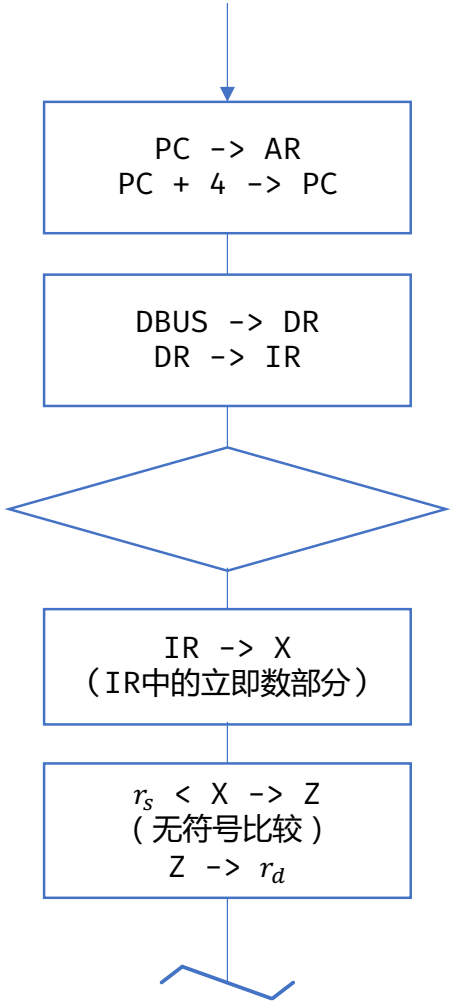
SLTU



Instruction

sltiu r_d, r_s, Imm_{16}

Flow Chart of SLTIU

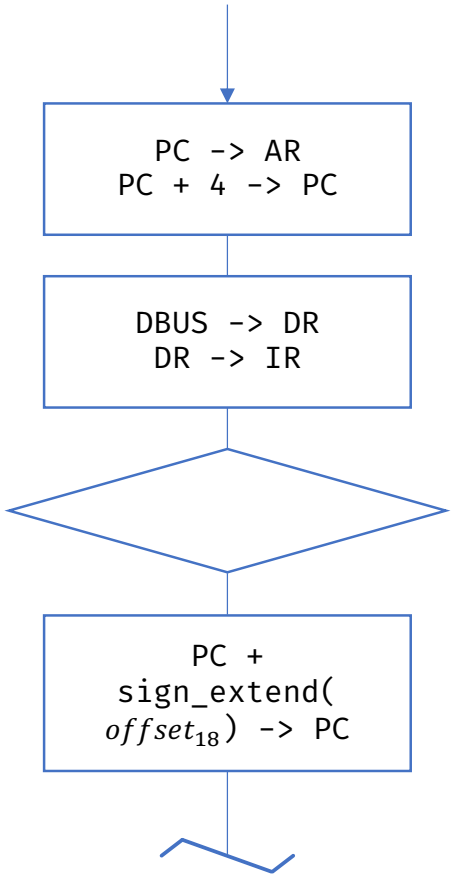


Instruction

b $offset_{18}$

Flow Chart of

B

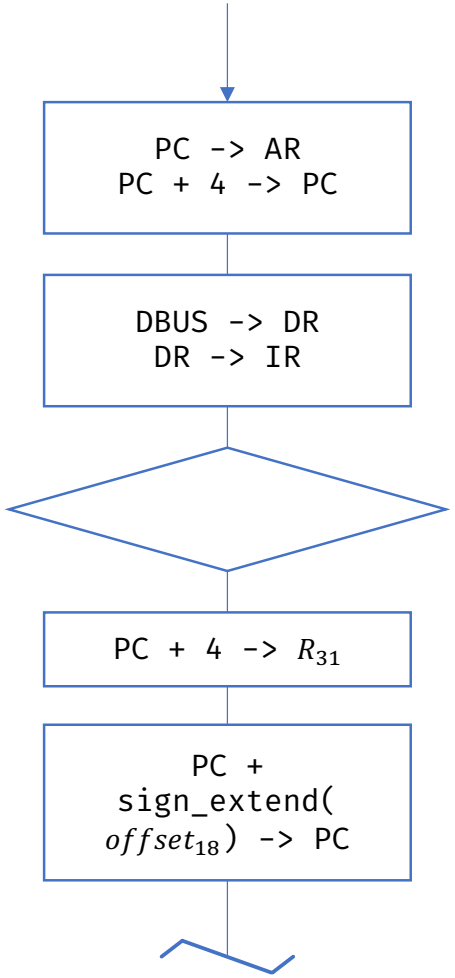


Instruction

`bal` $offset_{18}$

Flow Chart of

BAL

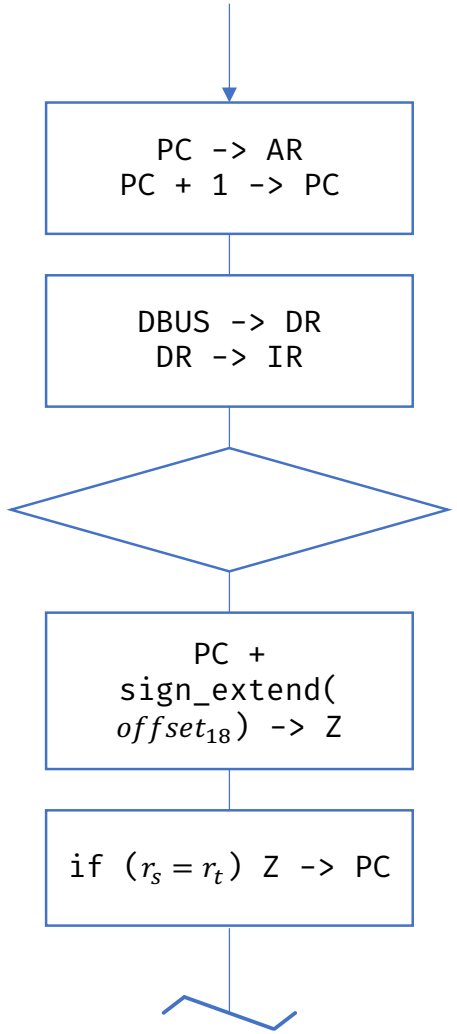


Instruction

beq $r_s, r_t, offset_{18}$

Flow Chart of

BEQ

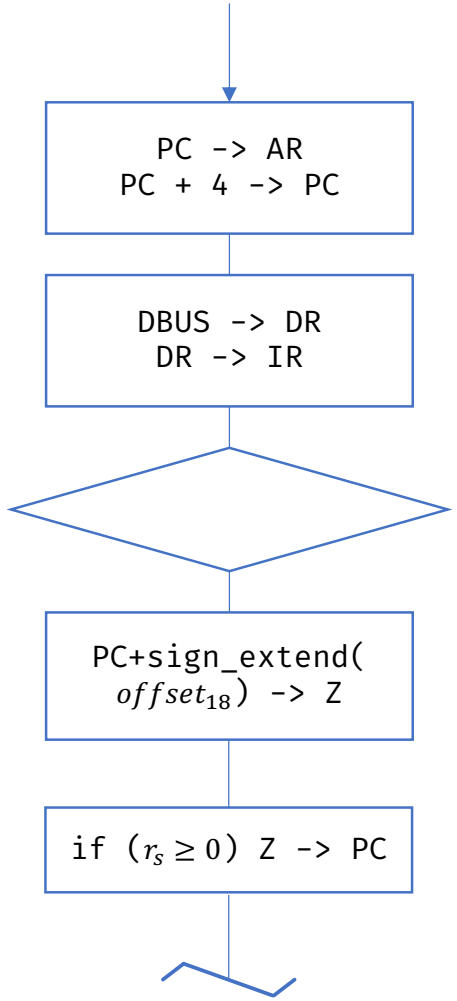


Instruction

bgez $r_s, offset_{18}$

Flow Chart of

BGEZ

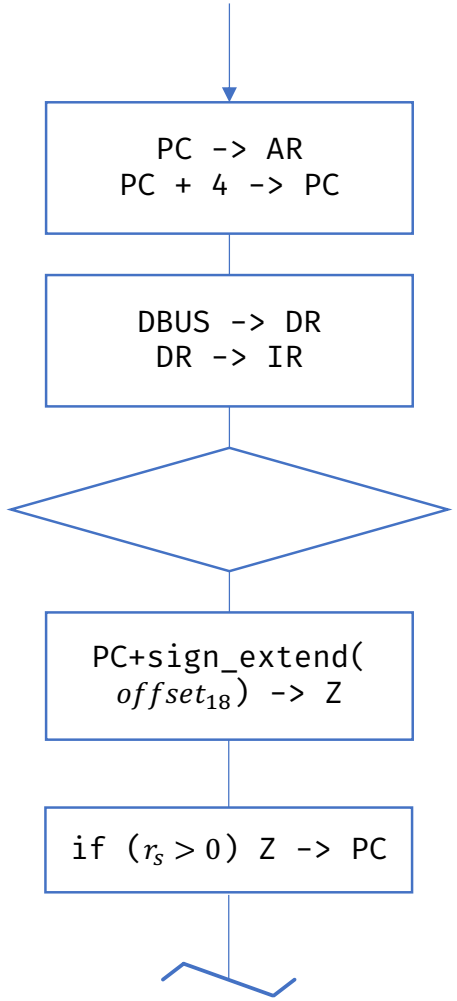


Instruction

bgtz $r_s, offset_{18}$

Flow Chart of

BGTZ

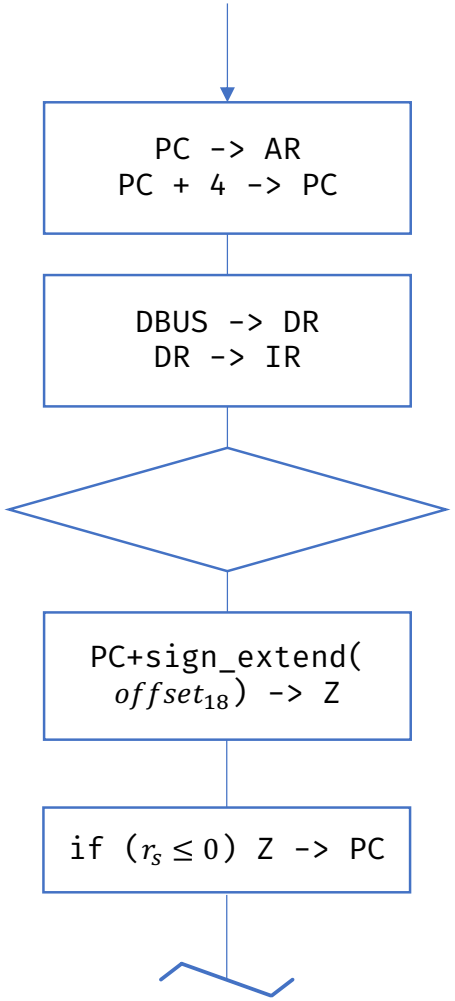


Instruction

`blez` $r_s, offset_{18}$

Flow Chart of

BLEZ

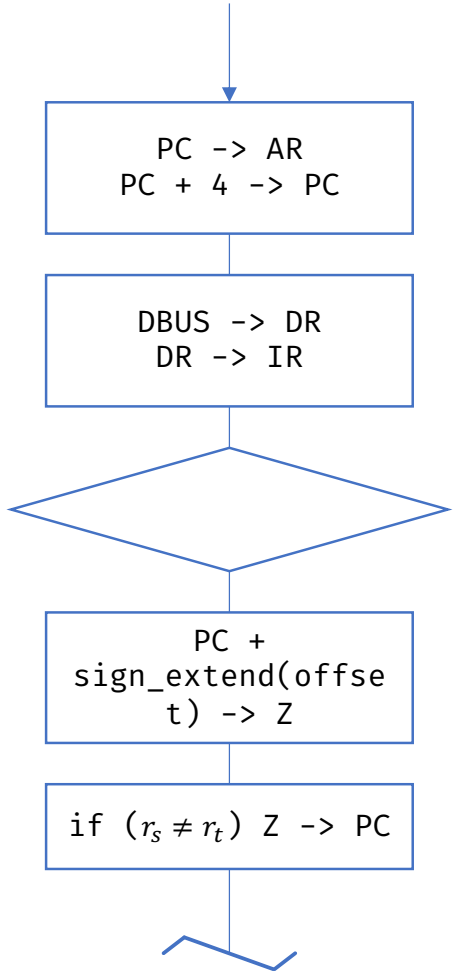


Instruction

bne $r_s, r_t, offset_{18}$

Flow Chart of

BNE

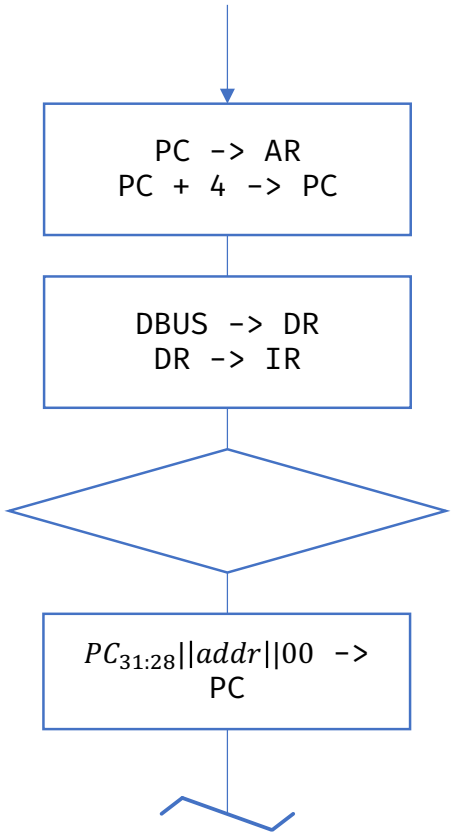


Instruction

j $addr_{28}$

Flow Chart of

J

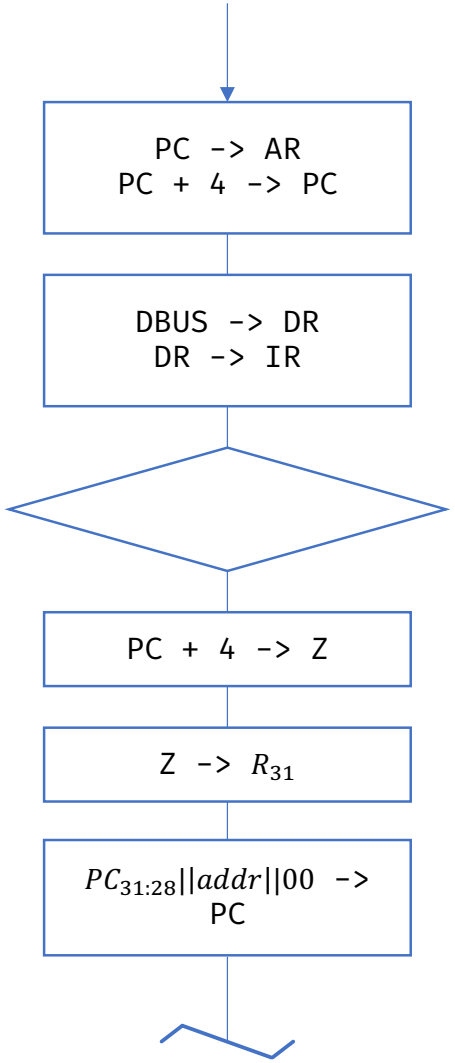


Instruction

`jal` $addr_{26}$

Flow Chart of

JAL

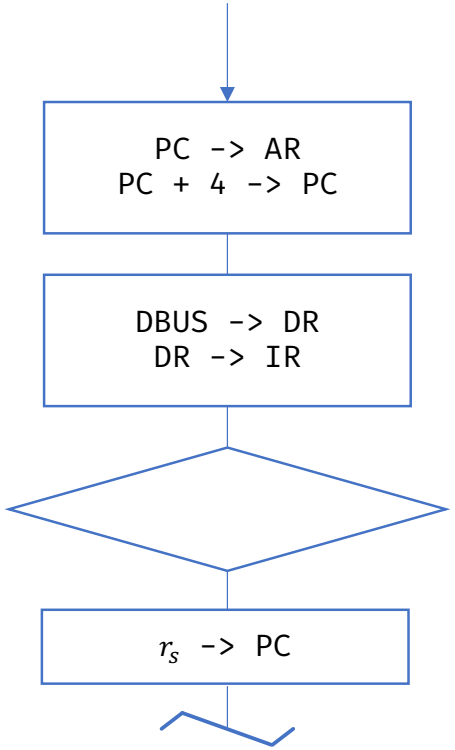


Instruction

`jr` r_s

Flow Chart of

JR

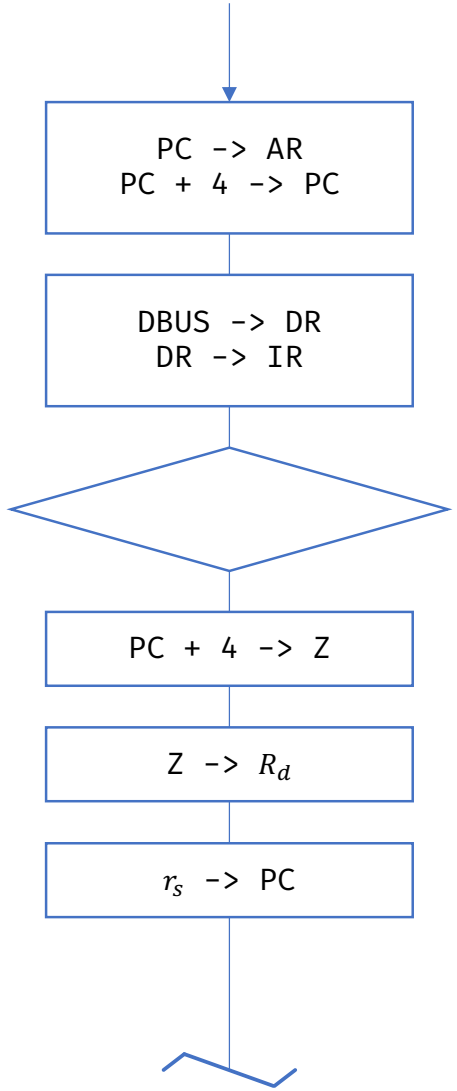


Instruction

`jalr` r_d, r_s

Flow Chart of

JALR

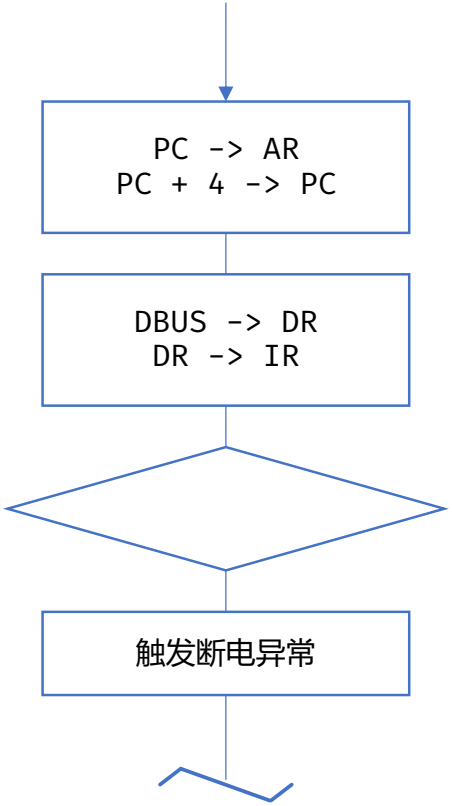


Instruction

`break`

Flow Chart of

BREAK

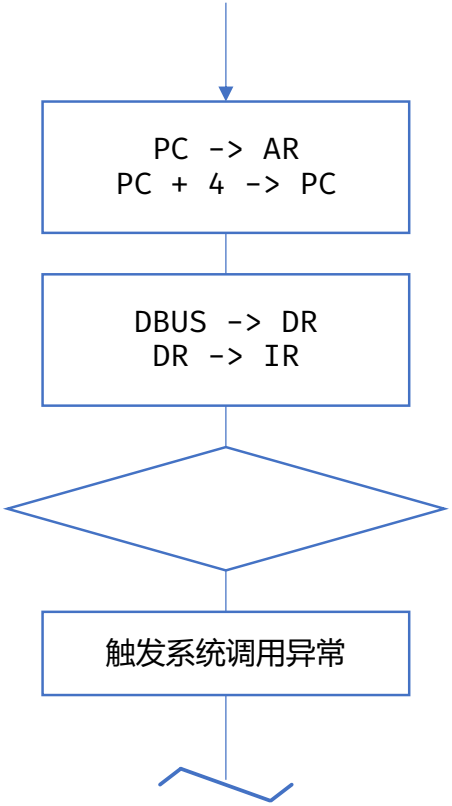


Instruction

`syscall`

Flow Chart of

SYSCALL

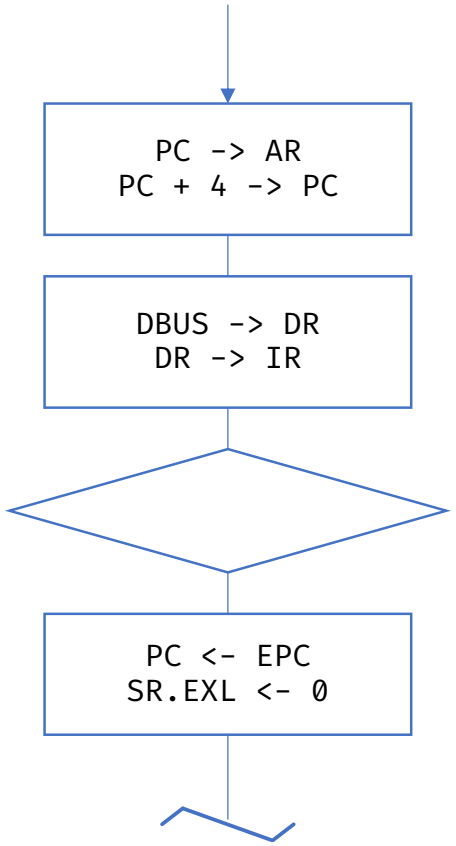


Instruction

```
eret
```

Flow Chart of

ERET

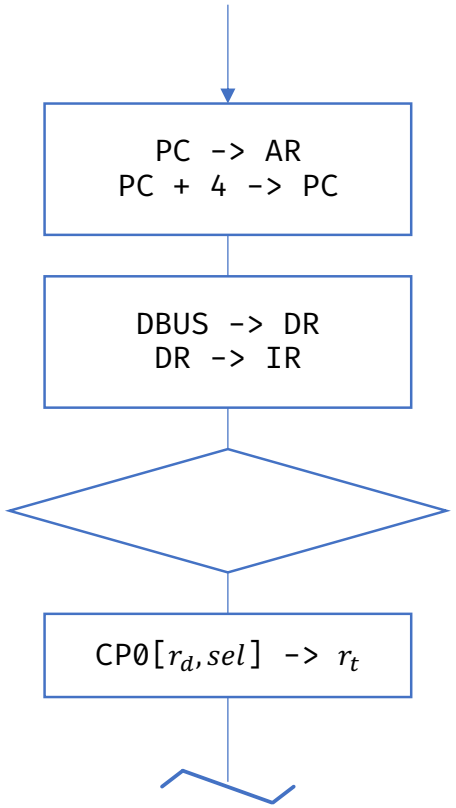


Instruction

mfc0 r_t, r_d, sel

Flow Chart of

MFC0



Instruction

mtc0 r_t, r_d, sel

Flow Chart of

MTC0