1、

**在F阶段所做的修改**

## What address should instruction be fetched at

int f\_pc = [

# Mispredicted branch. Fetch at incremented PC

**M\_icode == IJXX && M\_Cnd : M\_valA; //现在如果进行条件跳转，就是Mispredicted**

# Completion of RET instruction.

W\_icode == IRET : W\_valM;

# Default: Use predicted value of PC

1 : F\_predPC;

];

# Predict next value of PC

int f\_predPC = [

# BNT: This is where you'll change the branch prediction rule

**f\_icode == IJXX && f\_ifun != UNCOND : f\_valP**

**#判断是否是条件跳转，如果是，那么就设为not taken**

f\_icode in { IJXX, ICALL } : f\_valC;

1 : f\_valP;

];

**在E阶段所作的修改**

## Generate valA in execute stage

int e\_valA = [

**E\_icode == IJXX && e\_Cnd : E\_valC**

**#判断是否是JXX（把JMP也算进去），如果需要跳转，就传E\_valC的值**

1: E\_valA; # Pass valA through stage

];

**在处理bubble的时候所作的修改**

bool D\_bubble =

# Mispredicted branch

**(E\_icode == IJXX && e\_Cnd) || //如果判断失误**

# Stalling at fetch while ret passes through pipeline

# but not condition for a load/use hazard

!(E\_icode in { IMRMOVL, IPOPL } && E\_dstM in { d\_srcA, d\_srcB }) &&

IRET in { D\_icode, E\_icode, M\_icode };

# Should I stall or inject a bubble into Pipeline Register E?

# At most one of these can be true.

bool E\_stall = 0;

bool E\_bubble =

# Mispredicted branch

**(E\_icode == IJXX && e\_Cnd) || //如果判断失误**

# Conditions for a load/use hazard

E\_icode in { IMRMOVL, IPOPL } &&

E\_dstM in { d\_srcA, d\_srcB};

2、

A.

Load/use hazard (E\_icode in {IMRMOVQ, IPOPQ}) && (E\_dstM in {d\_srcA, d\_srcB}) &&

**!(D\_icode in {IPUSHL,IRMMOVL} && E\_dstM == d\_srcA && E\_dstM != d\_srcB)**

B.

**在E阶段的forwarding**

## Generate valA in execute stage

## LB: With load forwarding, want to insert valM

## from memory stage when appropriate

## Here it is set to the default used in the normal pipeline

int e\_valA = [

**M\_icode in { IMRMOVL, IPOPL } &&**

**M\_dstM in { E\_srcA, E\_srcB } && E\_icode == IPUSHL : m\_valM; //如果发生了加载转发**

1 : E\_valA; # Use valA from stage pipe register

]

**在处理加载使用冒险的时候做的修改**

# Should I stall or inject a bubble into Pipeline Register F?

# At most one of these can be true.

bool F\_bubble = 0;

bool F\_stall =

# Conditions for a load/use hazard

## Set this to the new load/use condition

**E\_icode in { IMRMOVL, IPOPL } &&**

**E\_dstM in { d\_srcA, d\_srcB } && !(D\_icode in {IPUSHL,IRMMOVL} && E\_dstM == d\_srcA && E\_dstM != d\_srcB) || //如果出现加载使用冒险，并且不能使用加载转发**

# Stalling at fetch while ret passes through pipeline

IRET in { D\_icode, E\_icode, M\_icode };

# Should I stall or inject a bubble into Pipeline Register D?

# At most one of these can be true.

bool D\_stall =

# Conditions for a load/use hazard

## Set this to the new load/use condition

**E\_icode in { IMRMOVL, IPOPL } &&**

**E\_dstM in { d\_srcA, d\_srcB } && !(D\_icode in {IPUSHL,IRMMOVL} && E\_dstM == d\_srcA && E\_dstM != d\_srcB); //如果出现加载使用冒险，并且不能使用加载转发**

# Should I stall or inject a bubble into Pipeline Register E?

# At most one of these can be true.

bool E\_stall = 0;

bool E\_bubble =

# Mispredicted branch

(E\_icode == IJXX && !e\_Cnd) ||

# Conditions for a load/use hazard

## Set this to the new load/use condition

**E\_icode in { IMRMOVL, IPOPL } &&**

**E\_dstM in { d\_srcA, d\_srcB } && !(D\_icode in {IPUSHL,IRMMOVL} && E\_dstM == d\_srcA && E\_dstM != d\_srcB); //如果出现加载使用冒险，并且不能使用加载转发**

3、

A.见后图

B.CEP下界=3（受制于浮点数加法）

C.CEP下界=1（受制于整数加法）

D.因为浮点数乘法的capacity是2，所以可以并行计算两个，所以完全可以在6个周期内处理2个浮点数乘法

4、

void inner4\_unrolling(vec\_ptr u,vec\_ptr v,data\_t \*dest)

{

long i;

long length=vec\_length(u);

data\_t \*udata=get\_vec\_strat(u);

data\_t \*vdata=get\_vec\_strat(v);

data\_t sum=(data\_t) 0;

for(i=0;i<length-5;i+=6)

{

sum=sum+udata[i]\*vdata[i]+udata[i+1]\*vdata[i+1]+udata[i+2]\*vdata[i+2]

+udata[i+3]\*vdata[i+3]+udata[i+4]\*vdata[i+4]+udata[i+5]\*vdata[i+5];

}

for(;i<length;i++) sum=sum+udata[i]\*vdata[i];

\*dest=sum;

}

A.

因为整数运算的吞吐量被整数乘法限制在1，浮点数运算的吞吐量被浮点数加法限制在1。

B.

见后图