Y86 simulator pipeline with Qt

汇报人： 陈杨栋 郭雨

Authors

郭雨

Mainly
design core

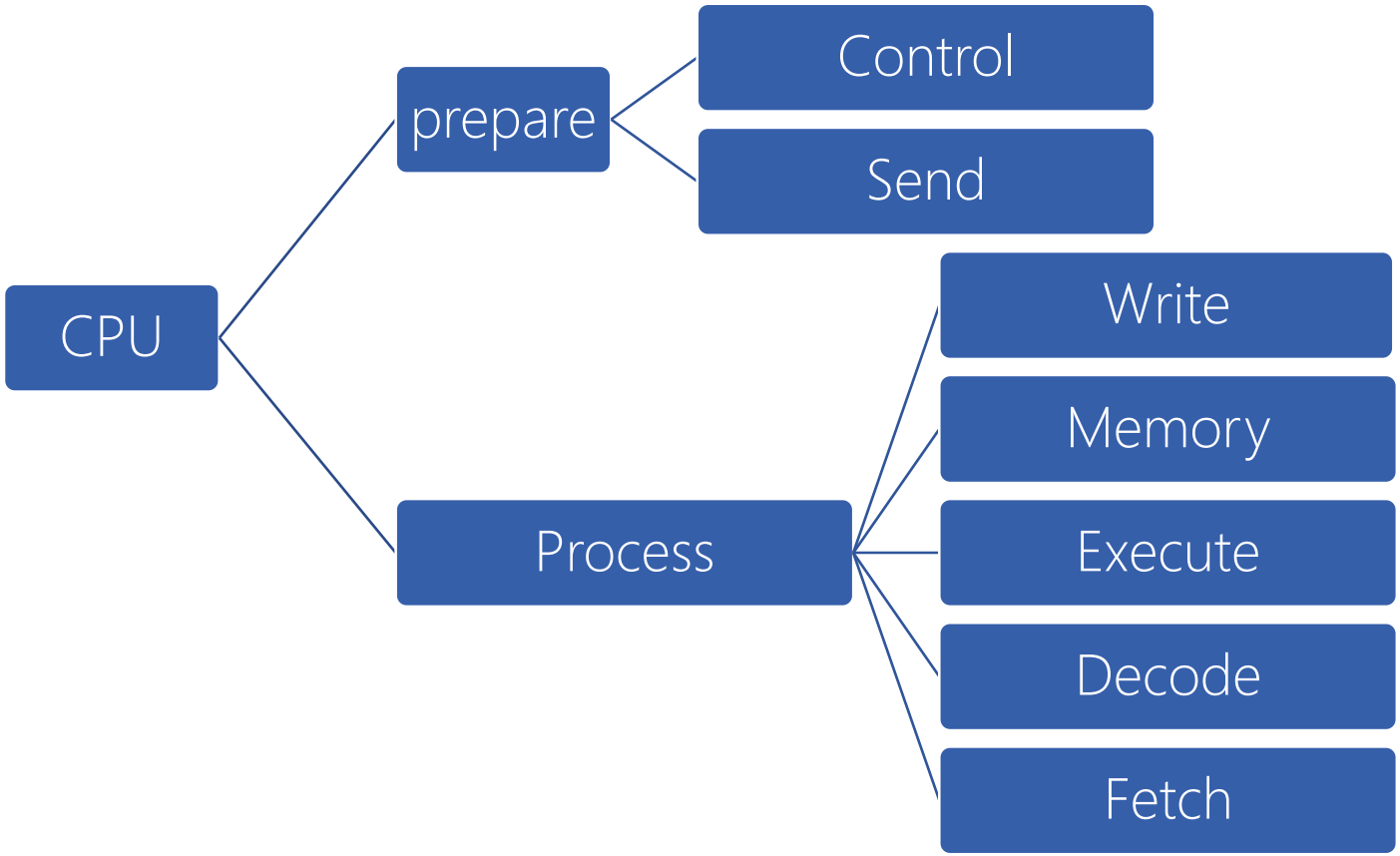
陈杨栋

Mainly
design ui

NO.1

内核结构

内核结构


CPU
prepare
Control
Send
Process
Write
Memory
Execute
Decode
Fetch

代码分层图

内核结构

Control

控制暂停与气泡信号

避免流水线冒险

……

Send

控制时钟上升沿的寄存器输入

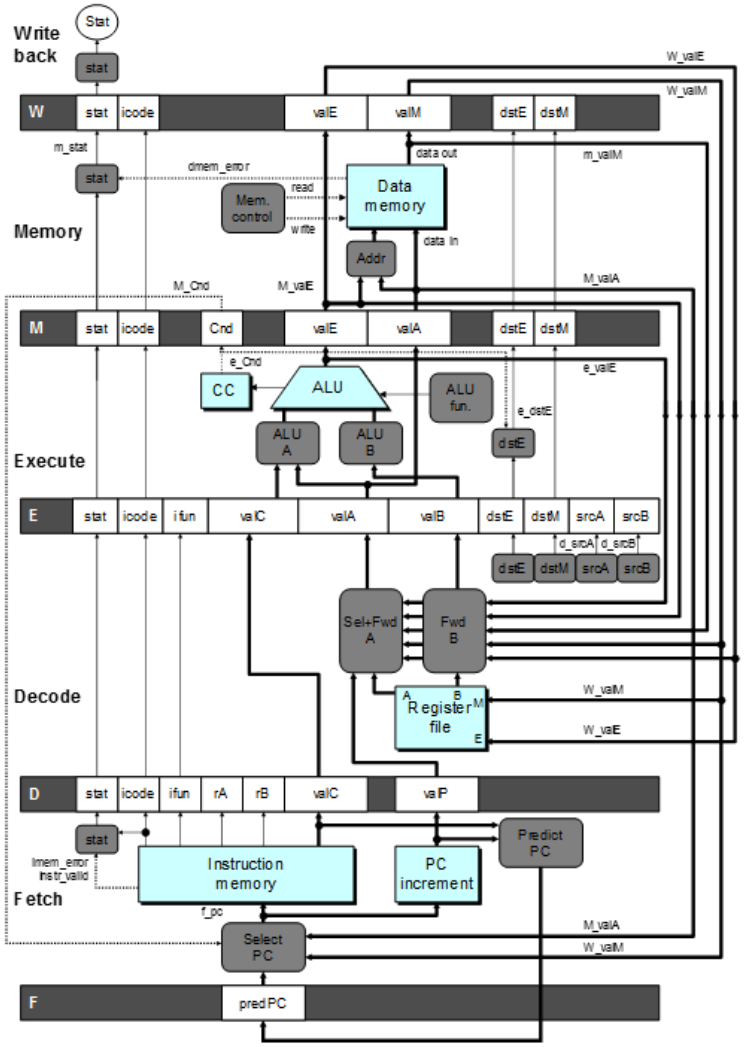
暂停：

- 关闭输入更新

气泡：

- 使用NOP填充
- 原先的输入更新被废弃

内核结构


Write back
Stat
stat
W
stat
icode
valE
valM
dstE
dstM
W_valE
W_valM
m_stat
stat
dmem_error
Memory
Mem. control
read
write
Data memory
data out
data in
m_valM
Addr
M_valA
M_Cnd
M_valE
M
stat
icode
Cnd
valE
valA
dstE
dstM
e_Cnd
e_valE
Execute
CC
ALU
ALU fun.
e_dstE
dstE
ALU A
ALU B
E
stat
icode
ifun
valC
valA
valB
dstE
dstM
srcA
srcB
d_srcA
d_srcB
dstE
dstM
srcA
srcB
Sel+Fwd A
Fwd B
Decode
A
B
Register file
M
E
W_valM
W_valE
D
stat
icode
ifun
rA
rB
valC
valP
stat
Predict PC
imem_error
instr_valid
Instruction memory
PC increment
Fetch
f_pc
M_valA
W_valM
Select PC
F
predPC

流水线图

NO.2

流水线冒险

流水线冒险

Load/use hazards

- Fetch stall
- Decode stall
- Execute bubble

流水线冒险

Load/use hazards

Condition：

(E_icode == IMRMOVL || E_icode == IPOPL)

&&

(E_dstM == d_srcA || E_dstM == d_srcB)

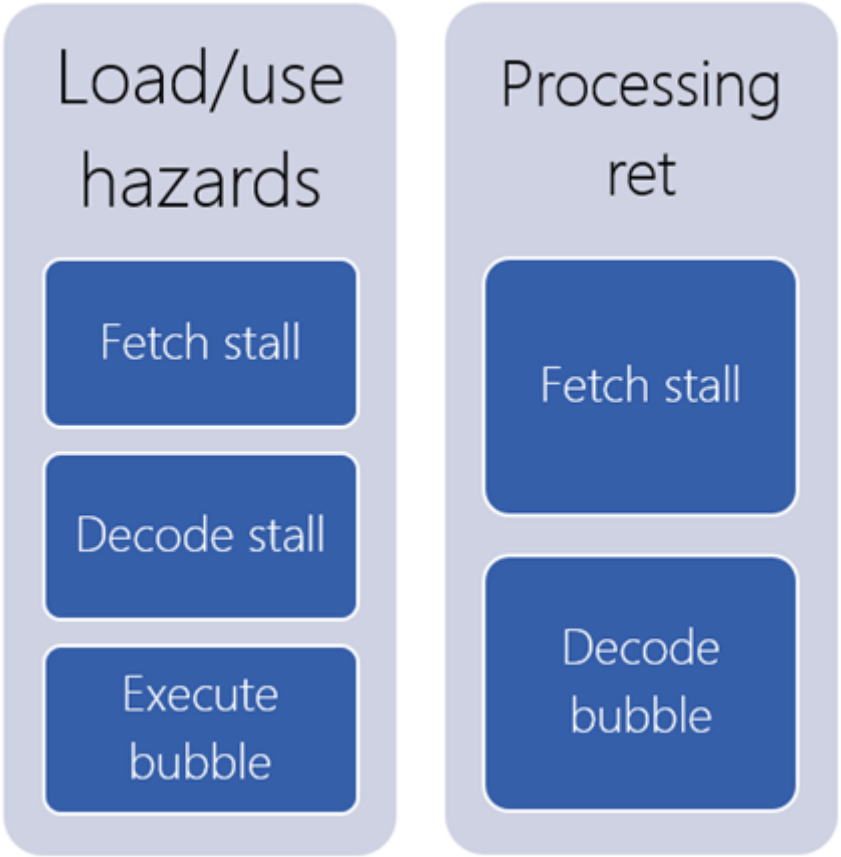
利用bubble & stall机制，在指令A前插入一个nop

Fetch：stall

Decode：stall

Execute：bubble

流水线冒险


Load/use hazards
Fetch stall
Decode stall
Execute bubble
Processing ret
Fetch stall
Decode bubble

分类

流水线冒险

Processing ret

Condition：

D_icode == IRET ||

E_icode == IRET ||

M_icode == IRET

第一次触发：

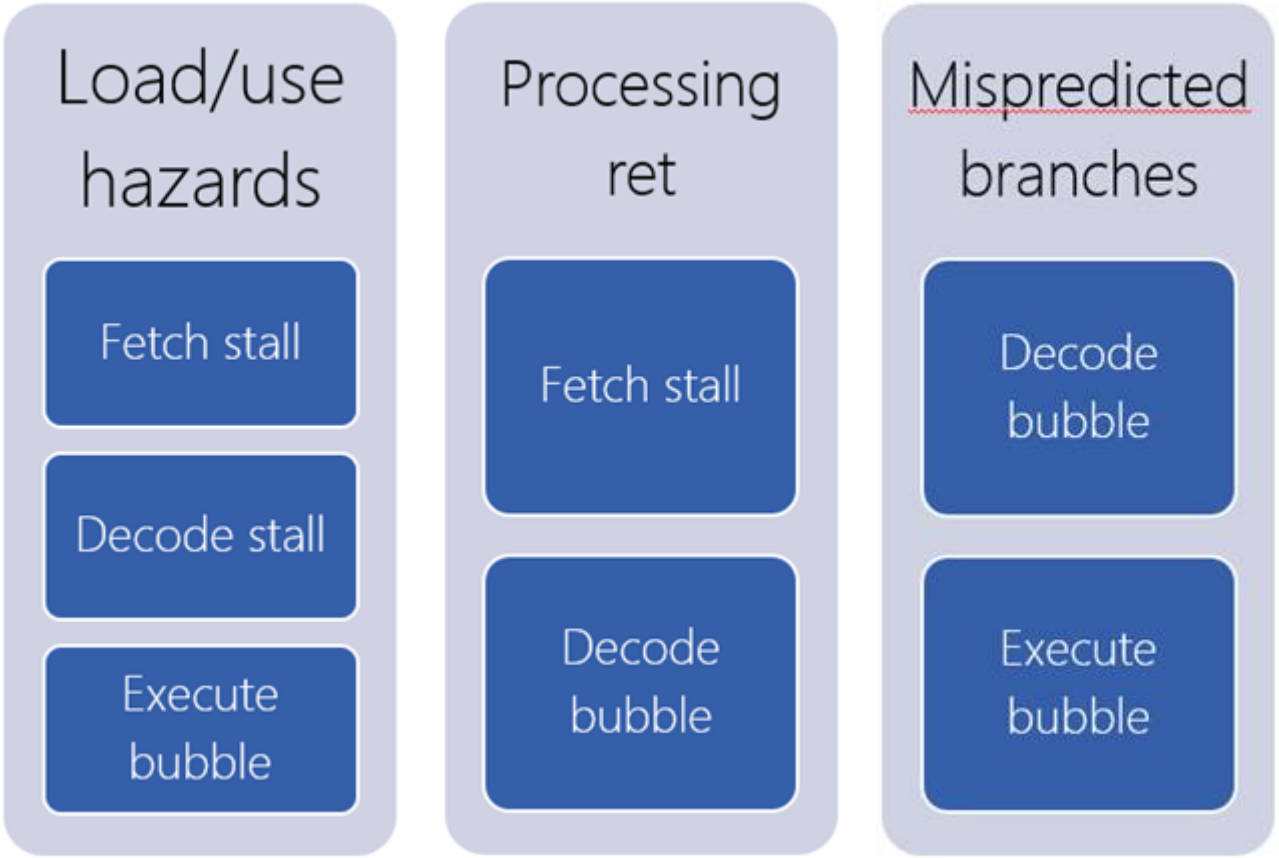
- Decode后的第一个时钟上升沿之前
- 直到ret进入Write-back阶段

Decode：bubble

Fetch：在同一位置取值

- 实质上相当于插入了三个nop语句

流水线冒险


Load/use hazards
Fetch stall
Decode stall
Execute bubble
Processing ret
Fetch stall
Decode bubble
Mispredicted branches
Decode bubble
Execute bubble

分类

流水线冒险

Mispredicted branches

Condition

E_icode == IJXX && !e_Cnd

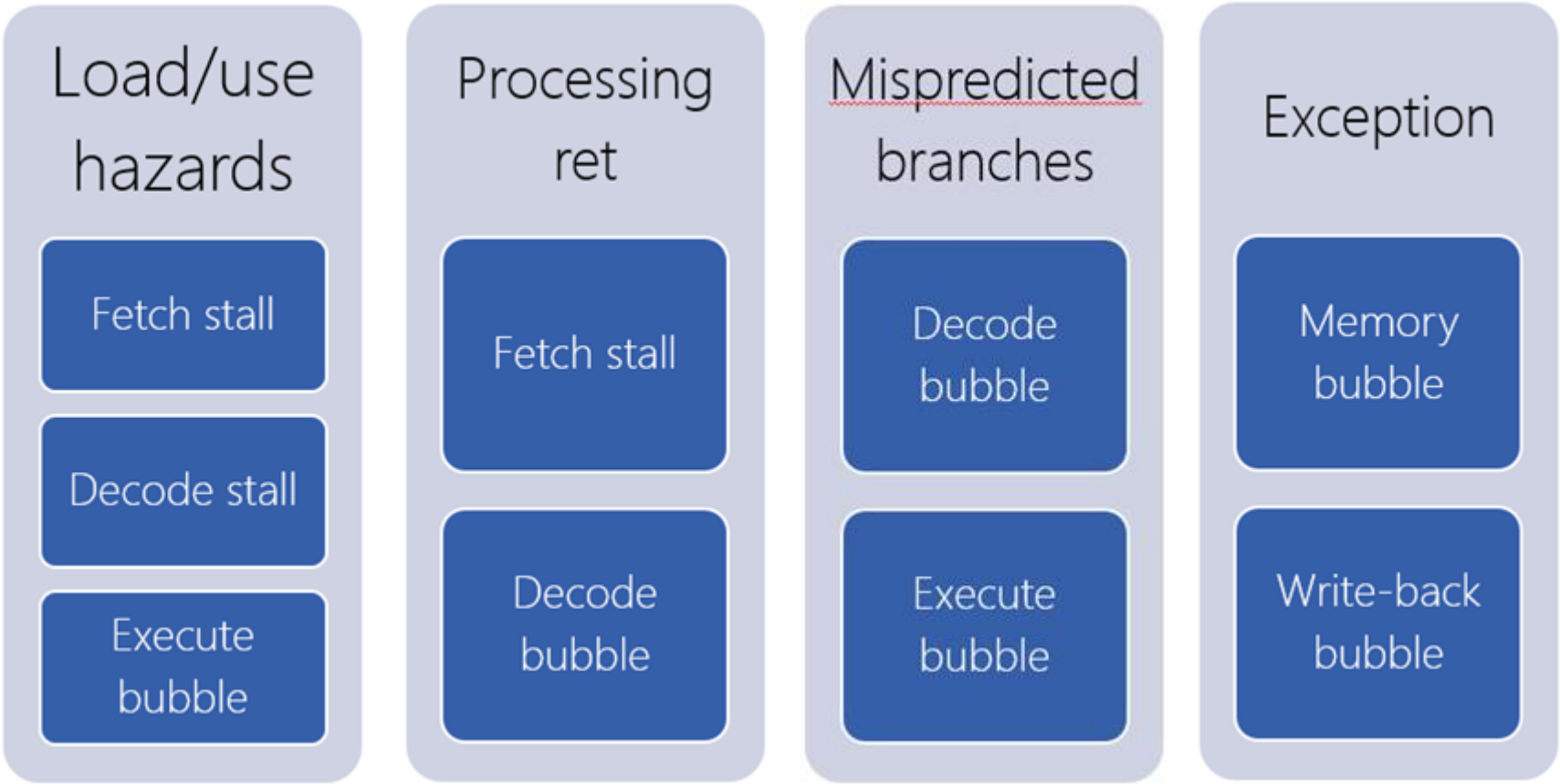
有两个错误的指令进入流水线

- 分别位于Fetch和Decode
- 没有对寄存器、条件码、内存的修改
- 将错误指令抹消即可

Execute：bubble

- 将传入Execute的输入信号抹除
- Decode：bubble 同理、
- Fetch正常执行

流水线冒险


Load/use hazards
Fetch stall
Decode stall
Execute bubble
Processing ret
Fetch stall
Decode bubble
Mispredicted branches
Decode bubble
Execute bubble
Exception
Memory bubble
Write-back bubble

分类

流水线冒险

Exception

Condition：

m_stat == SADR || m_stat == SINS || m_stat == SHLT

W_stat类似

监测到之前执行的指令发生错误

- 需要停止修改
- 类似的在Execute阶段的setConditionCode也有体现

坚持"顺序执行"的原则

- 即使某个指令在Fetch阶段发生错误，也要把之前进入流水线的指令执行完
- （预测错误）

NO.3

优先顺序

If & else

优先顺序

```
void SelFwdA(){
    if (D_icode == ICALL || D_icode == IJXX) d_valA =
    D_valP;
    else if (d_srcA == e_dstE) d_valA = e_valE;
    else if (d_srcA == M_dstM) d_valA = m_valM;
    else if (d_srcA == M_dstE) d_valA = M_valE;
    else if (d_srcA == W_dstM) d_valA = W_valM;
    else if (d_srcA == W_dstE) d_valA = W_valE;
    else d_valA = d_rvalA;
    D_op = D_op + "d_valA <- " + int2str(d_valA) + '\n';
}
```

条件同时满足？？

优先顺序

```
void SelFwdA(){
    if (D_icode == ICALL || D_icode == IJXX) d_valA =
    D_valP;
    else if (d_srcA == e_dstE) d_valA = e_valE;
    else if (d_srcA == M_dstM) d_valA = m_valM;
    else if (d_srcA == M_dstE) d_valA = M_valE;
    else if (d_srcA == W_dstM) d_valA = W_valM;
    else if (d_srcA == W_dstE) d_valA = W_valE;
    else d_valA = d_rvalA;
    D_op = D_op + "d_valA <- " + int2str(d_valA) + '\n';
}
```

顺序执行！

SelFwdA

优先顺序

```
void SelFwdA(){
    if (D_icode == ICALL || D_icode == IJXX) d_valA =
    D_valP;
    else if (d_srcA == e_dstE) d_valA = e_valE;
    else if (d_srcA == M_dstM) d_valA = m_valM;
    else if (d_srcA == M_dstE) d_valA = M_valE;
    else if (d_srcA == W_dstM) d_valA = W_valM;
    else if (d_srcA == W_dstE) d_valA = W_valE;
    else d_valA = d_rvalA;
    D_op = D_op + "d_valA <- " + int2str(d_valA) + '\n';
}
```

还是同时满足？？

优先顺序

```
void SelFwdA(){
    if (D_icode == ICALL || D_icode == IJXX) d_valA =
    D_valP;
    else if (d_srcA == e_dstE) d_valA = e_valE;
    else if (d_srcA == M_dstM) d_valA = m_valM;
    else if (d_srcA == M_dstE) d_valA = M_valE;
    else if (d_srcA == W_dstM) d_valA = W_valM;
    else if (d_srcA == W_dstE) d_valA = W_valE;
    else d_valA = d_rvalA;
    D_op = D_op + "d_valA <- " + int2str(d_valA) + '\n';
}
```

popl %esp !

SelFwdA

NO.4

优先顺序

Processing

优先顺序

- Write-back
- Memory
- Execute
- Decode
- Fetch

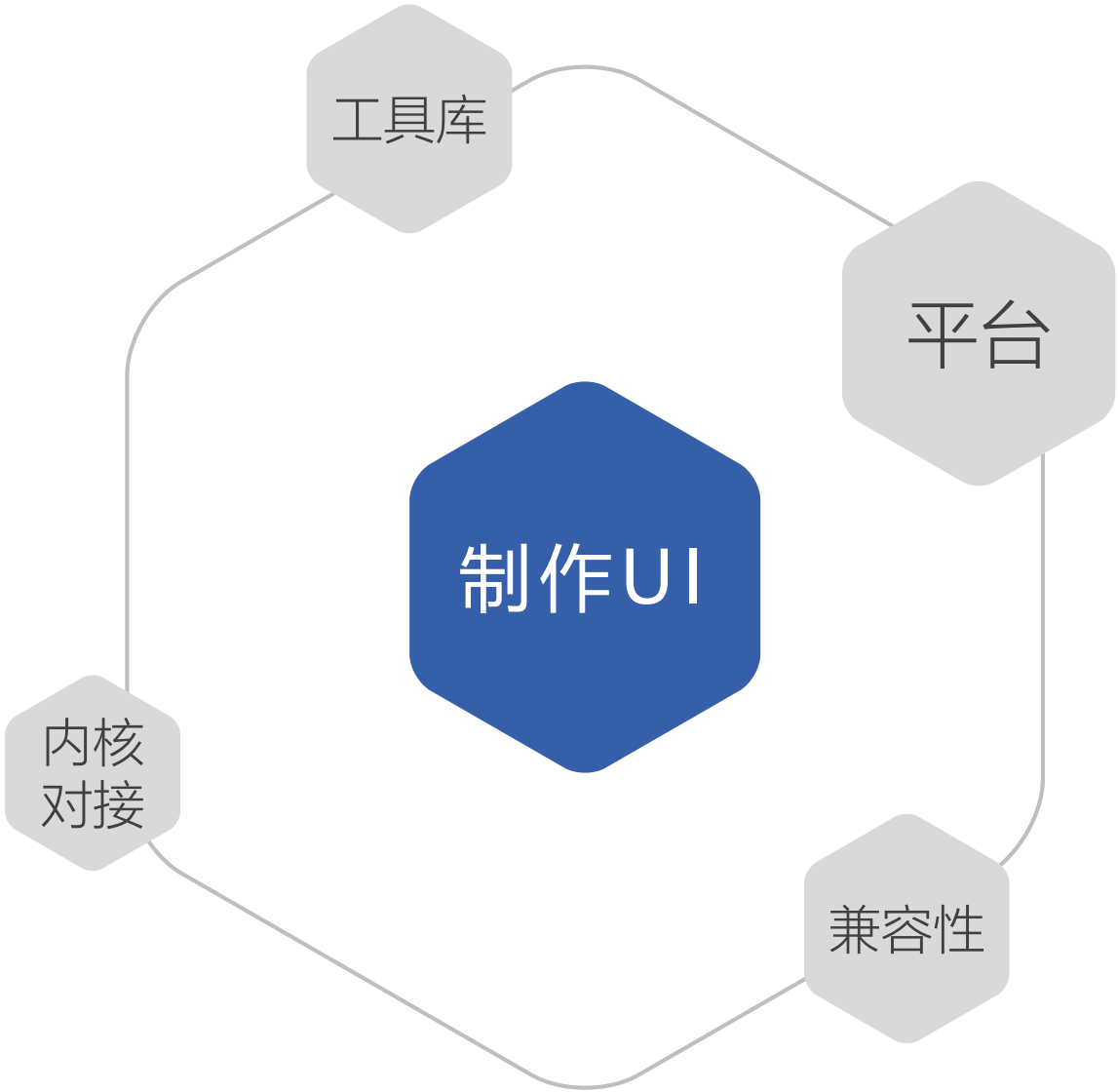
优先顺序

- Write-back
- Memory
- Execute
- Decode
- Fetch
- 照顾转发

NO.5

制作UI

absolute beginner


工具库
平台
制作UI
内核
对接
兼容性

平台和工具库

Why this ?

1、使用Qt Creator和其自带的GUI工具库来实现simulator的UI
2、Qt完全兼容C++，使得内核和UI的对接十分方便。
3、Qt具有良好的跨平台性能，在Windows和Ubuntu下均展现出了良好的兼容性和稳定性
4、Qt的GUI功能十分强大，基础功能就可以满足需要的GUI控件

UI与内核代码的对接

UI与内核代码的对接

Qt本身使用C++实现，同样支持Standard C++的所有功能，可以将编写好的内核作为头文件，轻易调用已经提供的接口，一个编译器可以解决内核和UI的编译

将实现好的内核作为一个类提供给UI部分的代码调用，在适当的时候调用适当的函数即可

Reference

1. https://lug.ustc.edu.cn/sites/qtguide/
2. Help documents in Qt Creator
3. Search Engine and some blogs

NO.6

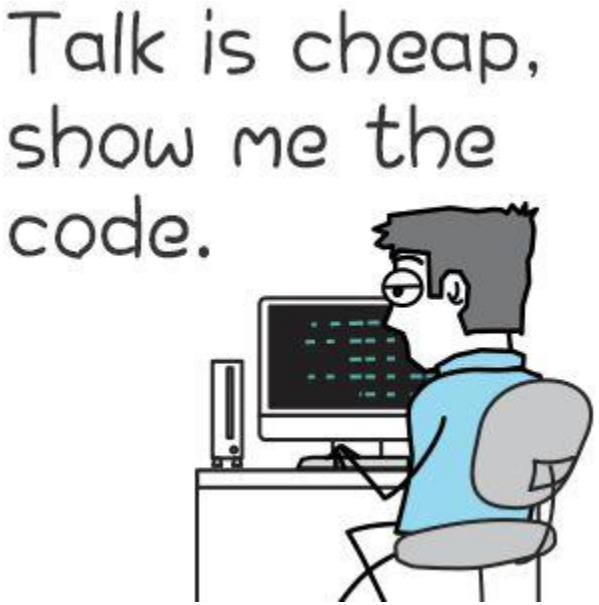
DEMO

UI show

Q & A

Acknowledgement

感谢我们的指导老师金博→



Thanks