Homework Problems

Chapter 4

## 4.43

本题考察pushl指令的细节。

1. 根据Problem 4.6，我们知道pushl %esp将%esp的初始值置入stack。若是按照题干的实现方式，便是将修改后的将%esp修改后的值置入stack。因此不符合。
2. 以上方式的问题在于造成了dependency hazard。我们可以用额外的register保存%esp，来避免这一点，实现如下：

movl REG, %esi

subl $4, %esp

movl %esi, (%esp)

这种方式的不好之处是，需要使用额外的register，若该register已在使用，就要先将其中的值保存到stack。于是陷入循环。

若不用额外的register，也可以用displacement实现：

movl REG, -4(%esp)

subl $4, %esp

这里我们将两个instructions独立开，消除dependency hazard。

## 4.44

本题考察popl指令的细节。

1. 若是按照以上方式实现，输出结果将是0xabd1，不符合Problem 4.7的例子。如下所示：

movl (%esp), %esp # %esp = 0xabcd

addl $4, %esp # %esp = 0xabd1

1. 与4.44的分析相同，我们采用displacement来避免dependency hazard，实现如下：

addl $4, %esp

movl -4(%esp), RET

## 4.45

本题考察Y86指令的使用。题目给出用C表示的计算模型（bubble sort），然后要求改写该模型。

1. 用pointer表示bubble sort。这里用到第三章的知识。代码见bubble\_ptr.c。
2. 用Y86表示bubble sort。在将IA32转换成Y86过程中遇到很多麻烦，包括simulator的运行、Y86指令的编写等。虽然我尝试写出Y86版本的bubble sort，但是没能成功运行。

考虑后续题目的重点是考察Y86指令的实现，而非编程，因此依赖程度低。我选择先跳过该题。

我可以利用IA32，在必要的地方做修改，得到Y86表示。

## 4.46

本题考察Y86指令的使用，建立在4.45的基础上。。。

## 4.47

本题考察Y86指令的实现。该题要求参照popl指令的描述，给出leave的执行过程的描述。

popl指令的计算可以由mrmovl和addl组合得到。popl %ebp等价于

|  |
| --- |
| mrmovl (%esp), %ebp |
| addl 4, %esp |

因此，题中的两行指令等价于以下

|  |
| --- |
| rrmovl %ebp, %esp |
| mrmovl (%esp), %ebp |
| addl 4, %esp |

从上面可以看出，%esp的值可以消除而不影响执行效果。因此，在popl指令计算过程中，我们用%ebp替代%esp，得到leave的计算过程，如下所示：

|  |  |
| --- | --- |
| Stage |  |
| Fetch |  |
|  |  |
|  |  |
|  |  |
| Decode |  |
|  |  |
| Execute |  |
|  |  |
| Memory |  |
| Write back |  |
|  |  |
|  |  |
| PC update |  |

## 4.48

本题考察Y86指令的实现。该题要求参照irmovl和OPL指令，给出iaddl的执行过程的描述。

|  |  |
| --- | --- |
| Stage |  |
| Fetch |  |
|  |  |
|  |  |
|  |  |
| Decode |  |
|  |  |
| Execute |  |
|  |  |
| Memory |  |
| Write back |  |
|  |  |
| PC update |  |

## 4.49

本题考察Y86指令的细节，要求写出HCL描述。方法是根据4.47的设计，更改seq-full.hcl各stages的实现。要注意HCL解析器的工作原理。结果见seq-full.hcl。测试通过。

？？？

## 4.50

本题考察Y86指令的细节，要求写出HCL描述。方法是根据4.48的设计，更改seq-full.hcl。结果见seq-full.hcl。测试通过。

？？？

## 4.51

本题考察PIPE-的实现细节。Stage stalling，指执行完当前stage后时候继续执行下一个stage。

* Conditions
* Actions
* Combinations
* Implementations

首先要分析出需要stalling的conditions，再明确相应的actions。然后分析actions的组合，最后是implement。

根据4.5.11，PIPE control logic要处理4种类型的conditions。由于没有使用forwarding，因此比起PIPE，PIPE-要处理更多conditions。

分析PIPE用forwarding策略处理的conditions，我们可以知道PIPE-。。。

Conditions

* Data hazards
  + Processing RET
  + Load/use hazard
* Control hazards
  + Mispredicted branch
  + Register dependency
    - 见4.5.7
* Exceptions

Pipeline hazard分为data和control两类。在PIPE中，用stalling和forwarding处理hazards；在PIPE-中，仅用hazards。这便是差异。

### Register dependency的分析

~~根据figure 4.18-4.20，将指令分别在decode和write back阶段涉及的指令分类如下~~

~~Write back阶段写入到相应register的指令~~

* ~~dstE~~
  + ~~mrmovl, popl, pushl, ret, call~~
* ~~dstM~~
  + ~~OPL, rrmovl, irmovl, popl~~

~~Decode阶段从相应register读取数据的指令~~

* ~~srcA~~
  + ~~OPL, rrmovl, rmmovl, pushl, ret~~
* ~~srcB~~
  + ~~OPL, rmmovl, mrmovl, ret, call~~

~~举个例子：如果，那么I1要在write back阶段将数据写入E\_dstE。如果E\_dstE正好是后续指令I2的d\_srcA或d\_srcB，那么需要等I1完成write back再执行I2的Excute阶段。这里我们要stalling延迟I2的执行过程。~~

~~所有组合如下：~~

|  |  |
| --- | --- |
| ~~Detection conditions for PIPE- control logic~~ | |
| ~~Condition~~ | ~~Trigger~~ |
| ~~#1~~ |  |
| ~~#2~~ |  |
| ~~#3~~ |  |
| ~~#4~~ |  |
| ~~ret~~ |  |
| ~~L/U~~ |  |
| ~~Mis~~ |  |
| ~~Exception~~ |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ~~Actions for pipeline control logic~~ | | | | | |
|  | ~~PIPE- pipeline register~~ | | | | |
| ~~Condition~~ | ~~F~~ | ~~D~~ | ~~E~~ | ~~M~~ | ~~W~~ |
| ~~#1~~ | ~~stall~~ | ~~stall~~ | ~~normal~~ | ~~normal~~ | ~~normal~~ |
| ~~#2~~ | ~~stall~~ | ~~stall~~ | ~~normal~~ | ~~normal~~ | ~~normal~~ |
| ~~#3~~ | ~~stall~~ | ~~stall~~ | ~~normal~~ | ~~normal~~ | ~~normal~~ |
| ~~#4~~ | ~~stall~~ | ~~stall~~ | ~~normal~~ | ~~normal~~ | ~~normal~~ |
| ~~ret~~ | ~~stall~~ | ~~bubble~~ | ~~normal~~ | ~~normal~~ | ~~normal~~ |
| ~~L/U~~ | ~~stall~~ | ~~stall~~ | ~~bubble~~ | ~~normal~~ | ~~normal~~ |
| ~~Mis~~ | ~~normal~~ | ~~bubble~~ | ~~bubble~~ | ~~normal~~ | ~~normal~~ |

~~Combinations for control logic~~

### 附注

刚刚看了参考答案。我的解法考虑到了register dependency。这一点是正确的，值得鼓励。然而，后续的分析过程完全偏离可行的方向。

d\_srcA != RNONE，表示D阶段的Instruction需要用到srcA。

如果d\_srcA in { E\_dstM, E\_dstE, M\_dstM, M\_dstE, W\_dstM, W\_dstE }，表示前一个instruction在W阶段将向srcA表示的register写入数据。

如果以上两个条件同时成立，则会造成generate use hazard。

d\_srcB的情况也是一样。

根据参考答案，利用stage registers和bubble可以得出直观的解法。

参考答案见CSAPP:Instructor’s Solution Manual第24页，Problem 4.36。

## 4.52

本题考察PIPE的实现细节，要给PIPE增加ILEAVE指令。在Probably 4.47中，我们分析了leave指令；在Problem 4.49中，为SEQ增加了该指令的实现。比起SEQ，在PIPE中实现该指令需要考虑data和control hazards。

根据4.47，我们知道leave指令需要的计算如下：

|  |  |
| --- | --- |
| Stage |  |
| Fetch |  |
|  |  |
|  |  |
|  |  |
| Decode |  |
|  |  |
| Execute |  |
|  |  |
| Memory |  |
| Write back |  |
|  |  |
|  |  |
| PC update |  |

在PIPE中实现ILEAVE，要改动三个部分

* 各阶段的计算
* Forward logic
* Control logic

通过全部测试！比较pipe-full和pipe-std，可以看出具体的改动内容。

## 4.53

本题考察PIPE的实现细节，要给PIPE增加ILEAVE指令。在Probably 4.48中，我们分析了iaddl指令；在Problem 4.50中，为SEQ增加了该指令的实现。比起SEQ，在PIPE中实现该指令需要考虑data和control hazards。

根据4.48，我们知道iaddl指令需要的计算如下：

|  |  |
| --- | --- |
| Stage |  |
| Fetch |  |
|  |  |
|  |  |
|  |  |
| Decode |  |
|  |  |
| Execute |  |
|  |  |
| Memory |  |
| Write back |  |
|  |  |
| PC update |  |

## 4.54

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Type | Behavior | Predict PC | Cnd | Should | Mis/Right | Actions |
| Coditional | Taken | valC | True | Taken | Right | - |
| Coditional | Taken | valC | False | Not Taken | Mis | valC-> valA |
| Uncoditional | Not Taken | valA(valP) | - | - | - | - |

## 4.55

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Direction | Behavior | Predict PC | Cnd | Should | Mis/Right | Actions |
| Backward | Taken | valC | True | Taken | Right | - |
| Backward | Taken | valC | False | Not Taken | Mis | valC-> valA |
| Forward | Not Taken | valA(valP) | True | Taken | Mis | valA-> valC |
| Forward | Not Taken | valA(valP) | False | Not Taken | Right | - |

根据以上分析，将逻辑关系用HCL描述。

一个关键点：通过ALU，将E\_valC放入M\_valE中，以便在确定PC时使用。

## 4.56

MR，POP要写入的reg是否要在下一instruction的excute阶段使用