Practice Problems

*CSAPP 2ed Chapter 4*

### 4.1

本题考察Y86 instructions的byte encoding。

|  |  |
| --- | --- |
| 0x100 | 30F3E0000000 |
| 0x101 | 2031 |
| 0x102 | 4013DFFFFFFF |
| 0x103 | 6031 |
| 0x104 | 7020100000 |

要点：

1. 地址。内存是byte array，地址表示byte的在内存中的位置。Instruction的地址为第一个byte的地址。相邻instructions的地址，与instruction的length有关。
2. 数字的编码。最小单位是byte，虽说按照litte endian，bytes要逆序排列，但byte之中的顺序不改变。
3. 十六进制。一个byte表示为两个十六进制字符。

|  |  |
| --- | --- |
| 0x100 | 30F30E000000 |
| 0x106 | 2031 |
| 0x108 | 4013FDFFFFFF |
| 0x10e | 6031 |
| 0x110 | 7020100000 |

### 4.2

本题考察Y86 instructions的byte encoding。要求根据binary notations写出ASM。

|  |  |  |
| --- | --- | --- |
| 0x100 | 30f3fcffffff | irmovl $-4, %ebx |
| 0x106 | 406300080000 | rmmovl %esi, 2048(%ebx) |
| 0x10c | 00 | halt |

|  |  |  |
| --- | --- | --- |
| 0x200 | a06f | pushl %esi |
| 0x202 | 8008020000 | call 0x208 |
| 0x207 | 00 | halt |
| 0x208 | 30f30a000000 | irmovl $10, %ebx |
| 0x20e | 90 | ret |



|  |  |  |
| --- | --- | --- |
| 0x300 | 505407000000 | mrmovl 7(%esp), %ebp |
| 0x306 | 10 | nop |
| 0x307 | f0 | -invalid- |
| 0x308 | b01f | popl %ecx |



|  |  |  |
| --- | --- | --- |
| 0x400 | 6113 | subl %ecx, %ebx |
| 0x402 | 7300040000 | je 0x400 |
| 0x408 | 00 | halt |



|  |  |  |
| --- | --- | --- |
| 0x500 | 6362 | xorl %esi, %edx |
| 0x502 | a0f0 | -invalid- |

附注：有两处不符合ASM的标准

### 4.3

本题考察Y86 instructions的使用，要求根据C代码，写出相应的Y86代码。

本题建立在Y86 instructions规范的基础上。作为例子，我们可以参考Sum函数、IA32 ASM以及Y86代码的对应关系。

具体代码见rSum\_y86\_gcc.s。

### 4.4

本题考察Y86 instructions的使用，要求在Sum函数的基础上做修改，实现AbsSum函数。这里涉及三个部分：C、IA32、Y86。

C部分的实现最容易。

### 4.5

本题考察特殊指令。

%eax中的值是%esp的原始值，%edx中为进栈的%esp值。返回值为0，表示进栈的%esp值等于%esp的原始值。因此，IA32采用第一种方式。

### 4.6

本题考察pushl指令的细节。

显然是从内存中读取到的值存进了%esp。我猜测，在Y86指令集中，XXmovl和cmovXX针对%esp也是同样的效果。

### 4.7

本题考察popl指令的细节。

Popl指令的执行过程包含两个步骤：修改目标register、修改%esp。按照题意，其过程先修改%esp，再修改目标register。

### 4.11

本题考察Y86指令的执行，要求根据pattern写出指令的执行过程。参考p.401 Aside.

|  |  |  |
| --- | --- | --- |
|  | Generic | Specific |
| Stage | irmovl V, rB | irmovl $128, %esp |
| Fetch |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| Decode |  |  |
| Excute |  |  |
| Memory |  |  |
|  |  |  |
| Write back |  |  |
|  |  |  |
| PC update |  |  |

附注：与Aside和参考答案相比，以上结果缺少了分析和结论。

### 4.12

本题考察Y86指令的执行，要求根据pattern写出指令的执行过程。参考p.404 Aside.

我们要分析Figure 4.17的第7行指令的执行过程，此时PC=0x01c。根据Aside (p.404)，%esp=124，M[%esp]=9。

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

该指令将M[%esp]的值放置到%esp，并使%esp的值增加4，等于128，从而改变了stack。

### 4.13

本题考察Y86指令的执行。

在pushl执行的decode部分，用两个值分别保存rA和%esp。在memory阶段，使用未改变的rA。当rA为%esp时，意味着将未改变的%esp值存入到memory中。因此，仍然遵从了Problem 4.6中的convention。

### 4.14

本题考察Y86指令的执行。

在pushl执行的decode部分，用两个值分别保存%esp。在memory阶段，使用未改变的valA获取目标值。在write back阶段，先改变%esp的值，再改变rA。那么，当rA=%esp时，意味着将未最初的M[%esp]值存入到%esp中。

因此，仍然遵从了Problem 4.7中的convention。

### 4.15

本题考察Y86指令的执行，要求根据rrmovl和jXX两类指令的实现，设计cmovXX指令的实现方式。

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

附注：根据p.497的参考答案，是可以使用if来判断情况的，这完全是意料之外。

### 4.16

本题考察Y86指令的执行，要求根据call指令的通用形式，写出特殊形式。

根据Figure 4.17，在第9行时，PC=0x023。由Problem 4.12，知道%esp=128。

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

保存了return address；将PC设置为Dest，转移到目标指令，实现函数调用。

### 4.17

本题考察Fetch阶段的实现。通过查阅4.3.1中的Figures，知道哪些instructions需要用到valC。再根据Figure 4.26，知道对应的Name。最后参考need\_regids，写出HCL代码，如下：

bool need\_valC = icode in { IIRMOVL, IRMMOVL, MRMOVL, IJXX, ICALL };

### 4.18

本题考察Decode阶段的实现。

int srcB = [

icode in { IOPL, IRMMOVL, IMRMOVL, } : rB;

icode in { IPUSHL, IPOPL, ICALL, IRET } : RESP;

1 : RNONE;

];

### 4.19

本题考察Write-Back阶段的实现。

int dstM = [

icode in { IMRMOVL, IPOPL } : rA;

1 : RNONE;

];

### 4.20

本题考察popl指令的实现。根据figure 4.20以及Problem 4.14，port M的优先级高于port E。

### 4.21

本题考察Execute阶段的实现。

int aluB = [

icode in { IOPL, IRMMOVL, IMRMOVL, IPUSHL, IPOPL, ICALL, IRET } : valB;

icode in { IRRMOVL, IIRMOVL } : 0;

# other instructions don’t need ALU

];

### 4.22

本题考察Execute阶段的实现。要求修改dstE实现cmovXX。

int dstE = [

icode in { IRRMOVL } && Cnd : rB;

icode in { IOPL, IIRMOVL } : rB;

icode in { IPUSHL, IPOPL, ICALL, IRET } : RESP;

1 : RNONE; # Don’t write any register

];

### 4.23

本题考察Memory阶段的实现。

int mem\_data = [

icode in { IRMMOVL, IPUSHL, } : valA;

icode in { ICALL } : valP;

1 : RNONE; # Don’t write any register

];

### 4.24

本题考察Memory阶段的实现。

bool mem\_write = icode in { IRMMOVL, IPUSHL, ICALL };

### 4.25

本题考察Memory阶段的实现。确实是利用icode, imem\_error, instr\_valid和dmem\_error，在Stat和SAOK, SADR, SINS及SHLT之间建立联系。问题在于，我不清楚其中的联系。参考答案：

int Stat = [

imem\_error || dmem\_error : SADR;

!instr\_valid : SINS;

icode == IHALT : SHLT;

1 : SAOK;

];

### 4.26

本题考察Partitioning的策略。对于该题目，我的解决方法是，先平均划分，再做调整。调整的时候，比较左右移动后与不移动时的throughout，取最优解。持续该策略，直至找到最优位置。

1. 新插入的register应位于C、D之间。
2. 新插入的registers应分别位于（B、C）和（D、E）之间。
3. 插入3个registers 后，分组如下：((8),(3,6),(5),(7,1))。
4. 再插入4个registers呢？会怎样？

### 4.27

……

### 4.28

本题考察PIPE Fetch阶段的实现，与Problem 4.26类似。

|  |  |  |
| --- | --- | --- |
| int f\_stat = [ | |  |
|  | imem\_error : SADR; | |
|  | !instr\_valid : SINS; | |
|  | f\_icode == IHALT : SHLT; | |
|  | 1 : SAOK; | |
| ] | |  |

### 4.29

本题考察PIPE Decode阶段的实现，参考p.421。此处特别注意符号的前缀（“d\_”或“D\_”）。

|  |  |  |
| --- | --- | --- |
| int d\_dstE = [ | |  |
|  | D\_icode in { IRRMOVL, IIRMOVL, IOPL } : D\_rB; | |
|  | D\_icode in { IPUSHL, IPOPL, ICALL, IRET } : RESP; | |
|  | 1 : RNONE; | |
| ] |  | |

### 4.30

本题考察PIPE Decode阶段的实现中forwarding order对程序行为的影响。

一般情况下（case 3和case 4的顺序不变），popl %esp的执行效果符合Section 4.1.6描述的第一种情况。在调换case 3和case 4的顺序后，可参考第二种情况。

附注：与参考答案接近，不足之处是表述不完整。

### 4.31

本题考察PIPE Decode阶段的实现中forwarding order对程序行为的影响。

在Problem 4.30的基础上，我们得出以下Y86代码：

|  |  |
| --- | --- |
| 1 | irmovl $5, %edx |
| 2 | irmovl $0x100, %esp |
| 3 | rmmovl %edx, 0(%esp) |
| 4 | popl %esp |
| 5 | nop |
| 6 | nop |
| 7 | rrmovl %esp, %eax |

插入第5、6两行，使得rrmovl在执行decode阶段时，pop正处于write-back。由于我们改变了d\_valA的实现（调换forwarding优先顺序），执行rrmovl时，会将原始的%esp值写入%eax。

附注：虽然通过阅读材料，我知道用插入nop的方式解该题，但是比起参考答案，我的解答仍是非常简陋。

### 4.32

本题考察PIPE Decode阶段的实现，要求写出d\_valB的实现。参考d\_valA(p.461)和Figure 4.56(p.460)，我们知道d\_valB的判断条件中不考虑D\_icode，其余条件与d\_valA一致。关键是其中的顺序。

|  |  |  |
| --- | --- | --- |
| int d\_valB = [ | |  |
|  | d\_srcB == e\_dstE : e\_valE; | |
|  | d\_srcB == M\_dstM : m\_valE; | |
|  | d\_srcB == M\_dstM : M\_valE; | |
|  | d\_srcB == W\_dstM : W\_valM; | |
|  | d\_srcB == W\_dstE : W\_valE;  1 : d\_rvalB; | |
| ] |  | |

### 4.33

本题考察PIPE Execute阶段的实现。

如果用E\_dstE替换e\_dstE，那么，就没有将Cnd信号传递forwarding到Execute阶段。因此，jXX指令将不会正确执行。

附注：参考答案里说，是将影响conditional move instructions。没有提到jXX指令。。。。

### 4.34

本题考察PIPE Memory阶段的实现。

|  |  |  |
| --- | --- | --- |
| int m\_stat = [ | |  |
|  | Dmem\_error : SADR; | |
|  | 1 : M\_stat; | |
| ] |  | |

### 4.35

本题考察CC的组合情况。根据Figure 4.67，可在p.466和p.468的基础上，得到本题的解答。

|  |  |  |
| --- | --- | --- |
| 0x000 | | xorl %eax, %eax |
| 0x002 | | jne target # Not taken |
| 0x007 | | irmovl $1, %eax # Fall through |
| 0x00d | | halt |
| 0x00e | target: | |
| 0x00e | | ret # Target |
| 0x014 | | irmovl $3, %eax # Target+1 |
| 0x01a | | halt |

附注：与参考答案只有部分接近。。。

### 4.36

本题考察CC的组合情况。根据Figure 4.67，以及p.467中对load/use hazard的描述。我们知道该题的关键是组合mrmovl/popl和ret指令。

|  |
| --- |
| popl %esp |
| ret |

或者

|  |
| --- |
| immovl $1, 0(%esp) |
| mrmovl 0(%esp), %esp |
| ret |

附注：解答不完整。。。

### 4.37

本题考察Control Logic的实现。根据Figure 4.66，并参考F\_stall的实现（p.475），我们知道：

|  |  |  |
| --- | --- | --- |
| int D\_stall = | |  |
|  | # Conditions for a load/use hazard | |
|  | E\_icode in { IMRMOVL, IPOPL } && | |
|  | E\_dstM in { d\_srcA, d\_srcB }; | |

### 4.38

本题考察Control Logic的实现。根据Figure 4.66，我们知道，在load/use hazard或mispredicted情况下，需要inject a bubble:

|  |  |  |
| --- | --- | --- |
| int E\_bubble = | |  |
|  | # Conditions for a load/use hazard | |
|  | E\_icode in { IMRMOVL, IPOPL } && | |
|  | E\_dstM in { d\_srcA, d\_srcB } || | |
|  | # Mispredicted branch | |
|  | (E\_icode = IJXX && !e\_Cnd); | |

### 4.39

本题考察Control Logic的实现。根据Figure 4.58，Set CC的inputs包括W\_stat、m\_stat和E\_icode。

|  |  |  |
| --- | --- | --- |
| int set\_cc = | |  |
|  | E\_icode == IOPL && | |
|  | !W\_stat in { SADR, SINS, SHLT } &&  !m\_stat in { SADR, SINS, SHLT }; | |

### 4.40

本题考察Control Logic的实现。参考p.469第二段的第2种情况，针对exceptions的处理过程。