2. 指令集

课后答案

A.2

根据 A.1 题给出的指令集的类型，我们将图 A.27 MIPS 的指令进行分类：

指令集种类		指令	时钟频率	gzip 和 perlbmk 平均频率
All ALU Instructions		add, sub, mul, compare, load imm, cond move, shift, and, or, xor, other logical	1.0	51.1%
Loads-stores		load, store	1.4	35%
Conditional branches	Taken	cond branch	2.0	60% * 11%
	Not taken		1.5	40% * 11%
	Jumps	jump, call, return	1.2	2.8%

那么，Effective CPI = 1.0 * 51.1% + 1.4 * 35% + 2.0 * 60% * 11 % + 1.5 * 40% * 11% + 1.2 + 2.8% = 1.23

A.7

对于 C 语言代码：

```
for (i=0; i<=100; i++)
 A[i] = B[i] + C;
```

A 和 B 均为 64 位整型数组，C 是 64 位的整数，所有数据都保存在内存中，A、B、C 和 i 的地址分别为 1000、3000、5000、7000。

要分别写出 MIPS 和 x86 体系架构下汇编代码，代码写法有多种，下面仅给出其中的一种可能情况：

（a）MIPS：

```
ex_a_7:   DADD      R1,R0,R0       ;R0 = 0, initialize i = 0
   SW       7000(R0),R1    ;store i
loop:     LD        R1,7000(R0)    ;get value of i
   DSLL     R2,R1,#3       ;R2 = word offset of B[i]
   DADDI    R3,R2,#3000    ;add base address of B to R2
   LD       R4,0(R3)       ;load B[i]
   LD       R5,5000(R0)    ;load C
   DADD     R6,R4,R5       ;B[i] + C
   LD       R1,7000(R0)    ;get value of i
   DSLL     R2,R1,#3       ;R2 = word offset of A[i]
   DADDI    R7,R2,#1000    ;add base address of A to R2
   SD       0(R7),R6       ;A[i] ← B[i] + C
   LD       R1,7000(R0)    ;get value of i
   DADDI    R1,R1,#1       ;increment i
   SD       7000(R0),R1    ;store i
   LD       R1,7000(R0)    ;get value of i
   DADDI    R8,R1,#-101    ;is counter at 101?
   BNEZ     R8,loop        ;if not 101, repeat
```

- 动态运行需要的指令条数：初始化 2 条 +101 次循环16 条，共 2 + 101*16 = 1618

- 访存类指令的条数：初始化 1 条SW + 101 次循环 8 条 LD/SW， 共 1 + 8*101 = 809
- 代码大小，对于 MIPS 每条指令的大小都是相等的（4个字节），因此总的代码大小为 4*18 = 72 B

（b）x86：

```
=> 0x0804854d <+9>:     movl   $0x0,-0x4(%ebp)
   0x08048554 <+16>:    jmp    0x8048571 <main()+45>
   0x08048556 <+18>:    mov    -0x4(%ebp),%eax
   0x08048559 <+21>:    mov    -0x4(%ebp),%edx
   0x0804855c <+24>:    mov    -0x330(%ebp,%edx,4),%edx
   0x08048563 <+31>:    add    -0x8(%ebp),%edx
   0x08048566 <+34>:    mov    %edx,-0x19c(%ebp,%eax,4)
   0x0804856d <+41>:    addl   $0x1,-0x4(%ebp)
   0x08048571 <+45>:    cmpl   $0x64,-0x4(%ebp)
   0x08048575 <+49>:    setle  %al
   0x08048578 <+52>:    test   %al,%al
   0x0804857a <+54>:    jne    0x8048556 <main()+18>
```

A.10

在设计芯片时，寄存器是不是越多越好呢？显示不是，寄存器的个数增加既有好处也有坏处，在设计时主要是做个trade-off ：

带来的好处：

- 给需要寄存器的编译技术带来更大的灵活性，例如循环展开、公共子表达式消除以及避免名字依赖等。
- 在函数传参时有更多的位置。
- 减少了需要保存和重加载内存的次数。

但也有坏处：

- 寄存器多了之后，在指令字中要表示一个寄存器的位数就要增多，这样会增加指令的长度或者减少其它部分的长度。
- 在发生异常时，意味着需要保存更多的上下文状态。
- 增加了芯片的面积，也增加了成本和功耗。

A.18

对于每种体系结构：

Accumlator 体系结构代码：

```
Load B          ;Acc ← B
Add C           ;Acc ← Acc + C
Store A         ;Mem[A] ← Acc
Add C           ;Acc ← "A" + C
Store B         ;Mem[B] ← Acc
Negate          ;Acc ←  Acc
Add A           ;Acc ← " B" + A
Store D         ;Mem[D] ← Acc
```

MM 体系结构代码：

```
Add A, B, C  ;Mem[A] ← Mem[B] + Mem[C]
Add B, A, C  ;Mem[B] ← Mem[A] + Mem[C]
Sub D, A, B  ;Mem[D] ← Mem[A]  Mem[B]
```

Stack 体系结构代码：

```
Push B          ;TOS ← Mem[B], NTTOS ← *
Push C          ;TOS ← Mem[C], NTTOS ← TOS
Add     ;TOS ← TOS + NTTOS, NTTOS ← *
Pop A           ;Mem[A] ← TOS, TOS ← *
Push A          ;TOS ← Mem[A], NTTOS ← *
Push C          ;TOS ← Mem[C], NTTOS ← TOS
Add     ;TOS ← TOS + NTTOS, NTTOS ← *
Pop B           ;Mem[B] ← TOS, TOS ← *
Push B          ;TOS ← Mem[B], NTTOS ← *
Push A          ;TOS ← Mem[A], NTTOS ← TOS
Sub     ;TOS ← TOS  NTTOS, NTTOS ← *
Pop D           ;Mem[D] ← TOS, TOS ← *
```

Load-Store 体系结构代码：

```
Load R1,B     ;R1 ← Mem[B]
Load R2,C     ;R2 ← Mem[C]
Add R3,R1,R2     ;R3 ← R1 + R2 = B + C
Add R1,R3,R2     ;R1 ← R3 + R2 = A + C
Sub R4,R3,R1     ;R4 ← R3  R1 = A  B
Store A,R3    ;Mem[A] ← R3
Store B,R1    ;Mem[B] ← R1
Store D,R4    ;Mem[D] ← R4
```

	16 位处理器				64 位处理器			
	Acc	MM	Stack	LS	Acc	MM	Stack	LS
How many instruction bytes are fetched?	8*2B=16B	3*2B=6B	12*2B=24B	8*2B=16B	8*8B=64B	3*8B=24B	12*8B=96B	8*8B=64B
How many bytes of data are transferred from/to memory?	7*2B=14B	9*2B=18B	9*2B=18B	5*2B=10B	7*8B=56B	9*8B=72B	9*8B=72B	5*8B=40B
Which architecture is most efficient as measured by total memory traffic?	30B	24B	42B	26B	120B	96B	168B	104B
	MM				MM			

A. 22

64 位十六进制表示的数：434F 4D50 5554 4552

（a）采用大端序列来存储：

地址	低 ----------> 高							
	0	1	2	3	4	5	6	7
数值	43	4F	4D	50	55	54	45	52

ASCII	C	O	M	P	U	T	E	R

（b）采用小端序列来存储：

地址	低 ----------> 高							
	0	1	2	3	4	5	6	7
数值	45	52	55	54	4D	50	43	4F
ASCII	E	R	U	T	M	P	C	O

（c）对于（a）中大端的存储该 64 位双字来说，所有没有 2 字节对齐的数有：4F4D、5055、5445，其它没有对齐的数超出了该 64 位地址。

（d）对于（b）中小端的存储该 64 位双字来说，所有没有 4 字节对齐的数有：5255544D、55544D50、544D5043，其它没有对齐的数超出了该 64 位地址。