**A.3**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **指令** | **lucas指令频率%** | | **swim指令频率%** | **指令比例%【lucas指令频率和swim指令频率的均值】** | |
| Load | 10.6 | | 9.1 | 9.85 | |
| Store | 3.4 | | 1.3 | 2.35 | |
| Add | 11.1 | | 24.4 | 17.75 | |
| Sub | 2.1 | | 3.8 | 2.95 | |
| Mul | 1.2 | | 0 | 0.6 | |
| Compare | 0 | | 0 | 0 | |
| Load imm | 1.8 | | 9.4 | 5.6 | |
| Cond branch | 0.6 | | 1.3 | 0.95 | |
| Cond mov | 0 | | 0 | 0 | |
| Jump | 0 | | 0 | 0 | |
| Call | 0 | | 0 | 0 | |
| Return | 0 | | 0 | 0 | |
| shift | 1.9 | | 0 | 0.95 | |
| And | 1.8 | | 0 | 0.9 | |
| Or | 1.0 | | 7.2 | 4.1 | |
| Xor | 0 | | 0 | 0 | |
| Other logic | 0 | | 0 | 0 | |
| Load FP | 16.2 | | 16.8 | 16.5 | |
| Store FP | 18.2 | | 5.0 | 11.6 | |
| Add FP | 8.2 | | 9.0 | 8.6 | |
| Sub FP | 7.6 | | 4.7 | 6.15 | |
| Mul FP | 9.4 | | 6.9 | 8.15 | |
| Div FP | 0 | | 0.3 | 0.15 | |
| Mov reg-reg FP | 1.8 | | 0.9 | 1.35 | |
| Compare FP | 0.8 | | 0 | 0.4 | |
| Cond mov FP | 0.8 | | 0 | 0.4 | |
| Other FP | 1.6 | | 0 | 0.8 | |
| **指令** | | **指令比例** | | | **时钟周期** | |
| All ALU instructions | | (17.75% + 2.95% + 0.6% + 5.6% + 0.95% + 0.9% + 4.1%) = 32.85% | | | 1 | |
| Loads-stores | | (9.85% + 2.35%) = 12.2% | | | 1.4 | |
| Conditional branches: | | 0.95% | | |  | |
| Taken [ 60% ] | | 0.95% \* 0.6 | | | 2.0 | |
| Not taken [ 40% ] | | 0.95% \* 0.4 | | | 1.5 | |
| Jumps | | 0% | | | 1.2 | |
| FP multiply | | 8.15% | | | 6 | |
| FP add | | (8.6% + 6.15%) = 14.75% | | | 4 | |
| FP divide | | 0.15% | | | 20 | |
| Load-store FP | | (16.5% + 11.6%) = 28.1% | | | 1.5 | |
| Other FP | | 1.35% + 0.4% + 0.4% + 0.8% = 2.95% | | | 2.0 | |

**CPI** = 32.85% × 1 + 12.2% × 1.4 + 0.95% × 2 × 0.6 + 0.95 × 1.5 × 0.4 + 8.15% × 6 + 14.75% × 4 + 0.15% × 20 + 28.1% × 1.5 + 2.95% × 2.0 = 2.1059 .

**A.7.a**

实现代码：

DADD R1,R0,R0 ; 初始化 i = 0

SD 7000(R0),R1 ; 把i存入地址 7000处

Loop: LD R1,7000(R0) ; 取出i存入 R1

DSLL R2,R1,#3 ; R2为B数组偏移量【64位操作数8个字节】

DADDI R3,R2,#3000； 得到B[i]的地址

LD R4,0(R3) ; 获取 B[i]并存入R4

LD R5,5000(R0) ; 获取 C并存入R5

DADD R6,R4,R5 ; B[i] + C的值存入R6

LD R1,7000(R0) ; 获取i的值

DSLL R2,R1,#3 ; R2为A数组偏移量

DADDI R7,R2,#1000 ; 得到A[i]的地址

SD 0(R7),R6 ; 把 B[i] + C的值存入A[i]

LD R1,7000(R0) ; 获取i的值

DADDI R1,R1,#1； 执行 i ++ 操作

SD 7000(R0),R1 ; 把i存入地址 7000处

LD R1,7000(R0) ; 获取i的值

LD R8,#101 ; 将101存入R8

BNE R8,R1,Loop ; 如果R8不等于R1，跳转到Loop，否之不跳转

动态指令条数：2 + 101 x 16 = 1618

寄存器引用次数： 9 x 101 = 909

代码大小：4 x 18 = 72 byte

**A.11**

struct foo { 32为机器 [字节] 64位机器[字节]

double d; 8 8

double g; 8 8

char \* cptr; 4 8

float \* fptr; 4 8

int c; 4 4

int x; 4 4

float f; 4 4

short e; 2 2

char a; 1 1

bool b; 1 1

} 合计： 40 48

**32位机器【对象宽度为4】**：根据对象对齐规则，double、指针、int、float都刚好可以对齐，而short e为两个字节，需要补充两个字节，char a 和 bool b 都为一个字节，各自需要补充一个字节，刚好可以四个字节对齐，所以最小结构需要44个字节，具体如下图。

|  |  |  |  |
| --- | --- | --- | --- |
| short e | |  |  |
| char a |  | bool b |  |

**64位机器【对象宽度为8】**：根据对象对齐规则，double、指针、int刚好可以对齐，short e为两个字节，char a 和 bool b 都为一个字节，那么排列的时候char a和 bool b 各自需要补充一个字节，此时为6个字节那么还需要2个字节填充,并且float f 为4个字节，需要4个字节填充，，所以最小结构需要56个字节，具体如下图。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| short e | char b |  | bool b |  |  |  |
| float f | | |  |  |  |  |

**A.18**

**a.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Accumulator** | **Memory-memory** | **Stack** | **Load-store** |
| Load B;  Add C;  Store A ;  Add C ;  Store B ;  Load A;  Add -B ;  Store D ; | Add A, B, C ;  Add B, A, C ;  Sub D, A, B ; | Push B ;  Push C ;  Add ;  Pop A ;  Push A ;  Push C;  Pop B;  Push B;  Push A;  Sub;  Pop D; | Load R1,B;  Load R2,C;  Add R3,R1,R2;  Store R3,A;  Add R4,R3,R2;  Store R4,B;  Sub R5,R3,R4;  Store R5,D; |

**b.**

多次重复载入一个值：Stack

一条指令的结果作为操作数给另一条指令：Accumulator

涉及处理器内部存储：Accumulator、Memory-memory

涉及存储器内部存储：Stack、Load-store

|  |  |
| --- | --- |
| **c.** 操作码【8位 】寄存器【16位】立即操作数【16位】 | **d.** 操作码【8位 】寄存器【16位】立即操作数【16位】 |
| **Accumulator：**  instruction bytes = 3 x 8 = 24 .  data bytes = 2 x 3 = 6  **Memory-memory :**  instruction bytes = 7 x 3 = 21.  data bytes = 2 x 3= 6  **Stack:**  instruction bytes = 3 x 9 +2 = 29 .  data bytes = 2 x 9= 18  **Load-store**  instruction bytes = 5 x 5 + 3 x 7 = 46 .  data bytes = 2 x 8= 16  Memory-memory效率更高 | **Accumulator：**  instruction bytes = 9 x 8 = 72 .  data bytes = 8 x 3 = 24  **Memory-memory :**  instruction bytes = 25 x 3 = 75 .  data bytes = 8 x 3= 24  **Stack:**  instruction bytes = 9 x 9 + 2 = 83 .  data bytes = 8 x 9= 72  **Load-store**  instruction bytes = 5 x 17 + 25x3 = 160 .  data bytes = 8 x 8= 64  Accumulator效率更高 |