**复旦大学计算机科学技术学院**

**《计算机原理》期中考试试卷**

**共 11 页**

**课程代码：COMP130007.0\_ 考试形式：□√开卷 □闭卷** 2014年5月

（本试卷答卷时间为120分钟，答案必须写在试卷上，做在草稿纸上无效）

专业 学号 姓名 成绩

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 题号 | 一 | 二 | 三 | 四 | 五 | 六 | 七 | 八 | 总分 |
| 得分 |  |  |  |  |  |  |  |  |  |

# Problem 1: (10 points)

We would like to write C function in 32-bit machine to set the penult(倒数第二个) significant byte of x to 0 and set the least significant byte to 0xFF. Please fill the blank and make the function portable(可移植) to 64-bit machine.

int bis ( int x )

{

int m = \_\_\_\_\_\_\_\_\_\_\_\_; /\* m is the mask word \*/

x = \_\_\_\_\_\_\_\_\_\_\_\_;

x = x | \_\_\_\_\_\_\_\_\_\_\_;

return x

}

# Problem 2: (15 points)

Consider a 9-bit floating-point representation based on the IEEE floating point format, with one sign bit, 3 exponent bits (k=3), and 5 fraction bits (n=5). The exponent bias is 2k-1-1 = 3 and V = (-1)sM2E, where M is the significand and E is the biased exponent..

Fill the blank in the table below. (You need not fill in entries marked with ”X”.)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | **Binary** | **M** | **E** | **Value** |
| X | 010000001 |  |  |  |
| Largest normalized (positive) |  |  |  |  |
| Smallest denormalized (negative) |  |  |  |  |
| Infinity |  | X | X |  |
| X |  |  |  | 7.25 |

**Problem 3: (10pts)**

In the C function that follows, we have omitted the body of the switch statement. In the C code, the case labels did not span a contiguous range, and some cases had multiple labels.

|  |
| --- |
| int switch2(int x) {  int result = 0;  switch (x) {  /\* Body of switch statement omitted \*/  }  return result;  } |

In compiling the function, GCC generates the assembly code that follows for the initial part of the procedure and for the jump table. Variable x is initially at offset 8 relative to register %ebp.

|  |  |
| --- | --- |
| *Setting up jump table access*  1 movl 8(%ebp),%eax *Retrieve x*  2 addl $4,%eax  3 cmpl $8,%eax  4 ja .L5  5 jmp \*.L11(,%eax,4) | *Jump table for switch2*  . L11 :  .long .L4  .long .L10  .long .L5  .long .L6  .long .L8  .long .L5  .long .L9  .long .L8  .long .L10 |

Use the foregoing information to answer the following questions:

A. What were the values of the case labels in the switch statement body?

B. What cases had multiple labels in the C code?

**Problem 4: (12pts)**

The following C code sets the diagonal elements of a ﬁxed-size array to val:

|  |
| --- |
| #define N \_\_ *%when in your program, you must fill in the blank to point out the value of N*  typedef int fix\_matrix[N] [N];  /\* Set all diagonal elements to val \*/  void fiX\_set\_diag(fix\_matriX A, int val)  {  int i;  for (i = 0; i < N; i++)  A[i][i] = val;  } |

When compiled, GCC generates the following assembly code:

|  |
| --- |
| movl 12(%ebp),%edx  movl 8(%ebp),%eax  movl $31,%ecx  addl $4092,%eax  .p2a1ign 4,,7 *%Added to optimize cache performance*  .L50:  movl %edx,(%eax)  addl $-132,%eax  decl %ecx  jns .L50 |

Create a C code program fiX\_Set\_diag\_Opt that uses optimizations similar to those in the assembly code, in the same style as the code in the Figure below. Notice that in your program, you must point out the value of N by “#define N \_\_” (fill in the blank).

|  |
| --- |
| /\* Compute i,k of fixed matrix product \*/  int fix\_prod\_ele\_opt(fix\_matrix A, fix\_matriX B, int i, int k)  {  int \*Aptr = &A[i][0];  int \*Bptr = &B[0][k];  int cnt = N - 1;  int result = 0;  do {  result +=(\*Aptr) \* (\*Bptr);  Aptr += 1;  Bptr += N;  cnt --;  } while (cnt >= 0);  return result;  } |

**Problem 5：(10pts)**

Complete the following blanks according to what you learned from Lab2 (Bomb Lab).

**Note**: the bomb is generated in an **AMD64** Linux machine.

**C code**

|  |
| --- |
| int func4(int a, int b, int c)  {  int d;  d = b + (c - b) **[1]** ;  if (d > a)  return func4(a, b, d-1) <<  **[2]** ;  else if ( **[3]** )  return (func4(a, d+1, c) <<  **[4]** ) + 1;  else  return 0;  }  void phase\_4(char \*input) {  int user\_val, user\_path, result, target\_path, numScanned;  numScanned = sscanf(input, "%d %d", &user\_val, &user\_path);  if ((numScanned != 2) **[5]** ) {  explode\_bomb(); % Program terminate  }  target\_path = 3;  result = func4( **[6]** );  if (result != target\_path || user\_path != target\_path) {  explode\_bomb();  }  } |

**Assembly Code**

|  |
| --- |
| 00000000004010f4 <func4>:  4010f4: \*\* \*\* \*\* \*\*  **[7] \_\_**  4010f8: 89 d0 mov %edx,%eax  4010fa: 29 f0 sub %esi,%eax  4010fc: 89 c1 mov %eax,%ecx  4010fe: c1 e9 1f shr $0x1f,%ecx  401101: 01 c8 add %ecx,%eax  401103: d1 f8 sar %eax  401105: 8d 0c 30 lea (%rax,%rsi,1),%ecx  401108: 39 f9 cmp %edi,%ecx  40110a: \*\* \*\*  **[8] \_\_**  40110c: 8d 51 ff lea -0x1(%rcx),%edx  40110f: e8 e0 ff ff ff callq 4010f4 <func4>  401114: 01 c0 add %eax,%eax  401116: eb 15 jmp 40112d <func4+0x39>  401118: b8 00 00 00 00 mov $0x0,%eax  40111d: 39 f9 cmp %edi,%ecx  40111f: \*\* 0c \_[9]\_\_ 40112d <func4+0x39>  401121: 8d 71 01 lea 0x1(%rcx),%esi  401124: e8 cb ff ff ff callq 4010f4 <func4>  401129: 8d 44 00 01 lea 0x1(%rax,%rax,1),%eax  40112d: 48 83 c4 08 add $0x8,%rsp  401131: c3 retq  0000000000401132 <phase\_4>:  401132: 48 83 ec 18 sub $0x18,%rsp  401136: 48 8d 4c 24 0c lea 0xc(%rsp),%rcx  40113b: 48 8d 54 24 08 lea 0x8(%rsp),%rdx  401140: be 31 2b 40 00 mov $0x402b31,%esi  401145: b8 00 00 00 00 mov $0x0,%eax  40114a: e8 31 fb ff ff callq 400c80 <\_\_isoc99\_sscanf@plt>  40114f: 83 f8 02 cmp $0x2,%eax  401152: 75 0d jne 401161 <phase\_4+0x2f>  401154: 8b 44 24 08 mov 0x8(%rsp),%eax  401158: \*\* \*\* \_\_\_\_\_\_**[10]\_\_\_\_\_\_\_\_\_\_\_**  40115a: 78 05 js 401161 <phase\_4+0x2f>  40115c: 83 f8 0e cmp $0xe,%eax  40115f: 7e 05 jle 401166 <phase\_4+0x34>  401161: e8 9e 05 00 00 callq 401704 <explode\_bomb>  401166: ba 0e 00 00 00 mov $0xe,%edx  40116b: be 00 00 00 00 mov $0x0,%esi  401170: 8b 7c 24 08 mov 0x8(%rsp),%edi  401174: e8 7b ff ff ff callq 4010f4 <func4>  401179: \*\* \*\* \*\* \_\_\_\_\_\_\_\_**[11] \_\_\_\_\_\_\_\_**  40117c: 75 07 jne 401185 <phase\_4+0x53>  40117e: 83 7c 24 0c 03 cmpl $0x3,0xc(%rsp)  401183: 74 05 je 40118a <phase\_4+0x58>  401185: e8 7a 05 00 00 callq 401704 <explode\_bomb>  40118a: 48 83 c4 18 add $0x18,%rsp  40118e: c3 retq |

[1] \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[2] \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[3] \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[4] \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[5] \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[6] \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[7] \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[8] \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[9] \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[10] \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[11]\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Problem 6: (9 points)**

1. Please make a comparison between fixed and variable length instructions. Discuss each’s advantages and disadvantages.(4pts)
2. Suppose we were to modify the Y86 PIPE implementation to include a small, hidden hardware stack for return address prediction. We’d push this stack on call instructions (in addition to doing what we normally due to the programmer-visible stack). We’d pop it in the fetch stage of ret instructions, and use it to predict the address of the next instruction. If the average subroutine is 50 cycles long, and if our prediction is right 90% of the time, what percentage improvement in performance can we expect?(5pts)

**Problem 7: (5 + 5 + 5 = 15points)**

Implement the following functions as you did in Lab1(Data.Lab).

**Part 1**

|  |
| --- |
| /\*  \* absVal - absolute value of x  \* Example: absVal(-1) = 1.  \* You may assume -TMax <= x <= TMax  \* Legal ops: ! ~ & ^ | + << >>  \* Max ops: 10  \*/  int absVal(int x) {  **/\* Please fill your code here\*/**  } |

**Part 2**

|  |
| --- |
| /\*  \* bitParity - returns 1 if x contains an odd number of 0's  \* Examples: bitParity(5) = 0, bitParity(7) = 1  \* Legal ops: ! ~ & ^ | + << >>  \* Max ops: 20  \*/  int bitParity(int x) {  **/\* Please fill your code here\*/**  } |

**Part 3**

|  |
| --- |
| /\*  \* float\_f2i - Return bit-level equivalent of expression (int) f  \* for floating point argument f.  \* Argument is passed as unsigned int, but  \* it is to be interpreted as the bit-level representation of a  \* single-precision floating point value.  \* Anything out of range (including NaN and infinity) should return  \* 0x80000000u.  \* Legal ops: Any integer/unsigned operations incl. ||, &&. also if, while  \* Max ops: 30  \*/  int float\_f2i(unsigned uf) {  /\* Please fill code in blanks. \*/  unsigned sign = uf >> 31;  unsigned exp = \_\_\_ [1]\_\_\_ ;  unsigned frac = uf & 0x7FFFFF;  /\* Create normalized value with leading one inserted,  and rest of significand in bits 8--30. \*/  unsigned val = 0x80000000u + (frac << 8);  if ( [2] ) { /\* Absolute value is < 1 \*/  return 0;  }  if (exp > 158) { /\* Overflow \*/  return 0x80000000u;  }  /\* Shift val right \*/  val = val >> [3] ;  if (sign) { /\* Negative \*/  /\* Check if out of range \*/  return [4] ;  } else { /\* Positive \*/  /\* Check if out of range \*/  return [5] ;  }  } |

Part3:

[1] \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[2] \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[3] \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[4] \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[5] \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Problem 8：(18pts)**

Suppose we want to add a new instruction **irsubl** with the following format:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Byte |  | 0 | | 1 | | 2 | | 3 | 4 | 5 |
| **irsubl** | **V, rA, rB** | C | 1 | rA | rB | | V | | | | |

This instruction substracts rA from the constant value V and save the result to register rB, i.e rB 🡨 V-rA. Describe the computations performed to implement this instruction. Please fill the blank of certain stage with “*Nothing to do*” if the stage has no work to do to accomplish this instruction.

|  |  |
| --- | --- |
| **Stage** | **irsubl V, rB** |
| Fetch |  |
| Decode |  |
| Execute |  |
| Memory |  |
| Write Back |  |
| PC Update |  |