LAB02-A

Q1

Now we give the declaration of the variable:

```
int x;

float f;

double d;
```

Assume that neither `f` nor `d` equals $+\infty$, $-\infty$, or NaN.

If the value of the following expression must be true, you answer 1, otherwise 0:

```
A. 4/3 == 4/3.0

B. (d >= 0.0) || ((d*2) < 0.0)

C. f == (float)(double) f

D. d == (float) d

E. f == -(-f)

F. x == (int)(float) x

G. x == (int)(double) x

H. (d+f)-d == f

I. x == -(-x)

J. 4.0/3 == 4.0/3.0
```

Q2

Consider the following stupid C code:

```
#include <stdio.h>

int max(int a, int b){
    if(a > b)
        return a;
    else
        return b;
}

int main(){
    printf("%d\n", max(4, 5));
    return 0;
}
```

- Generate an executable object file using GCC and give the command of GCC.
- Use objdump to disassemble the executable file, give the command you use and the screenshot of the " `max` function" segment of the result.
- Find a jump instruction in the " `max` function" segment. How long of the the byte encodings of this jump instruction? Can you explain what the second byte means?

Q3

A function fun_a has the following overall structure:

```
int fun_a(unsigned x) {
    int val = 0;
    while ( ______ ) {
        ______
    }
    return ______;
}
```

The gcc C compiler generates the following assembly code:

```
;   x at %ebp+8

    movl 8(%ebp), %edx
    movl $0, %eax
    testl %edx, %edx
    je .L7
  .L10:
    xorl %edx, %eax
    shrl %edx
    jne .L10
  .L7:
    andl $1, %eax
```

Your task is to fill in the missing parts of the C code to get a program equivalent to the generated assembly code, then explain the function of `func_a` .

Q4

Consider the following assembly code:

```
; int loop(int x, int n);
; x in %edi, n in %esi

  loop:
  movl %esi,%ecx
  movl $1,%edx
  movl $0,%eax
  jmp .L2
  .L3:
  movl %edi,%ebx
  andl %edx,%ebx
  orl %ebx,%eax
  sall %cl,%edx
  .L2:
  testl %edx,%edx
  jne .L3
  ret
```

The preceding code was generated by compiling C code that had the following overall form:

```
int loop(int x, int n)
{
    int result = ____;
    int mask;
    for (mask = ____; mask ____; mask = ____) {
        result |= ____;
    }
    return result;
}
```

Your task is to fill in the missing parts of the C code to get a program equivalent to the generated assembly code.

Q5

- The following is a trial implementation using a conditional move instruction but it is not valid. Please explain why.

  ```
  int cread(int *xp) {
      return (xp ? *xp : 0);
  }
  ```

- Write a C function `cread_alt` that has the same behavior as `cread`, except that it can be compiled to use conditional data transfer. When compiled, the generated code should use a conditional move instruction rather than one of the jump instructions.

Q6

GNU C provides several language features not found in ISO standard C. These extensions are available in C and Objective-C. Most of them are also available in C++.

"Inline Assembly" provides instructions and extensions for interfacing C with assembler so that programmers can use assembly language in C code. GCC provides two forms of inline `asm` statements.

- **Basic Asm**: inline assembler without operands.

```
asm asm-qualifiers ( AssemblerInstructions )
```

```
/* An example of basic asm for i386 */
#define DebugBreak() asm("int $3")
```

- **Extended Asm**: inline assembler with operands.

```
asm asm-qualifiers ( AssemblerTemplate
                 : OutputOperands
                 [ : InputOperands
                 [ : Clobbers ] ])

asm asm-qualifiers ( AssemblerTemplate
                      :
                      : InputOperands
                      : Clobbers
                      : GotoLabels)
```

> 1. asm: GNU extentsion keyword.
> 2. AssemblerTemplate: assembler code with `asmSymbolicName` ; seperated by "\n\t".
>   - special format strings: '%%' for '%''.
> 3. Format of operands: [[asmSymbolicName]] constraint (cvariablename)
>
> - asmSymbolicName: when not specified, use the position of operands in the list, '%0' for the first, '%1' for the second.
> - Format of constraint: [modifier] constraint
>   - modifier: (1) '=': just written by the instruction; (2) '+': both read and written by the instruction. (Output constraints must begin with '=' or '+')
>   - constraint: (1) 'r': register; (2) 'm': memory; (3) 'rm': the compiler chooses the most efficient one between register and memory.
> - cvariablename: C lvalue expression.
>
> 4. Clobbers: the inline asm code may modify more than just the outputs and require additional registers such as '%eax'. These changes will be listed in the Clobbers and separated by commas.

```
/* An example of extended asm for i386 */
/* c = a + b */
int a = 1, b = 2, c;
asm (
    "mov %1, %0\n\t"  // AssemblerTemplate
    "add %2, %0"
    : "=r" (c)      // OutputOperands
    : "r" (a), "r" (b)  // InputOperands
);
printf("%d\n", c);
```

Write a C program to calculate the **multiplication** of the last 2 numbers of your student ID by `extended asm`. The following is the template. Submit your code and screenshot of the result.

```
// ID: 17373422
// last 2 numbers: a=2, b=2
#include <stdio.h>
int main()
{
    int a = 2, b = 2, prod;
    asm (
      // your code
    );
    printf("%d\n", prod);
    return 0;
}
```

HINT

If you have difficulty reading English, try **this**.

Reference

Using-Assembly-Language-with-C