Problem 1: HCL (10 points)

```
1. bool IMPLIES = !(a && !b); or bool IMPLIES = !a || b;
2. word MINMAX = [
            s && A <= B && A <= C : A;
            s && B <= C           : B;
            s                     : C;
            !s && A >= B && A >= C: A;
            !s && B >= C          : B;
            1                     : C;
   ];
```

Problem 2: Y86 (18 points)

1. [1] rrmovq %rsp, %rbp　　　[2] irmovq $-1, %rbx

   [3] 6222　　　[4] addq %rbx, %rdx

   [5] 743501000000000000　　　[6] irmovq $3, %rax

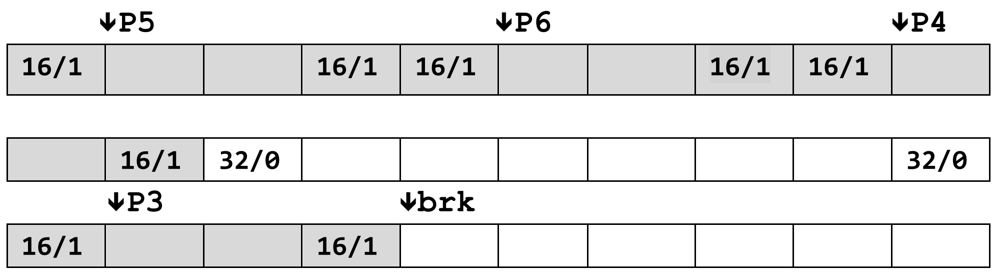
   [7] 800401000000000000

2. It wants to protect that %rdx is larger than 0. (2')

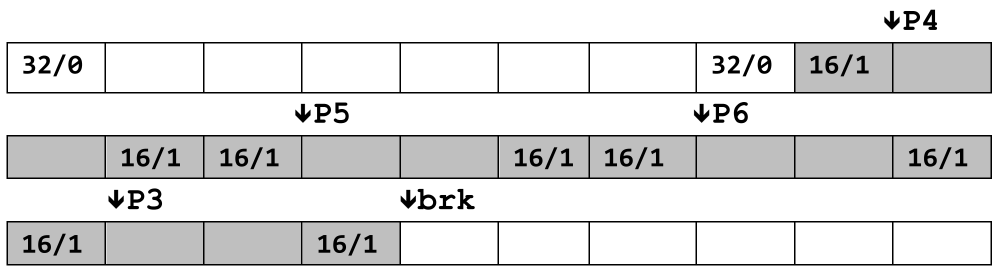
   If may cause dead loop. (1')

Problem 3: Memory Allocation (16 points)

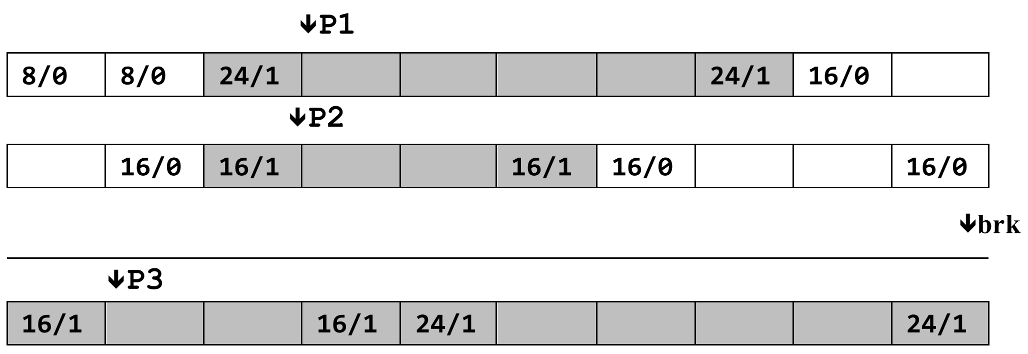
1.



2.



3. malloc will call sbrk() to increase the heap (1')



4. 48-5-6-7 = 30. (3') Optimization: we can store the allocate/free bit of pervious block in one of the excess low-order bits of current block. Thus we don't need a footer in allocated blocks. [Book 9.9.11](2')

Problem 4: Optimization (22 points)

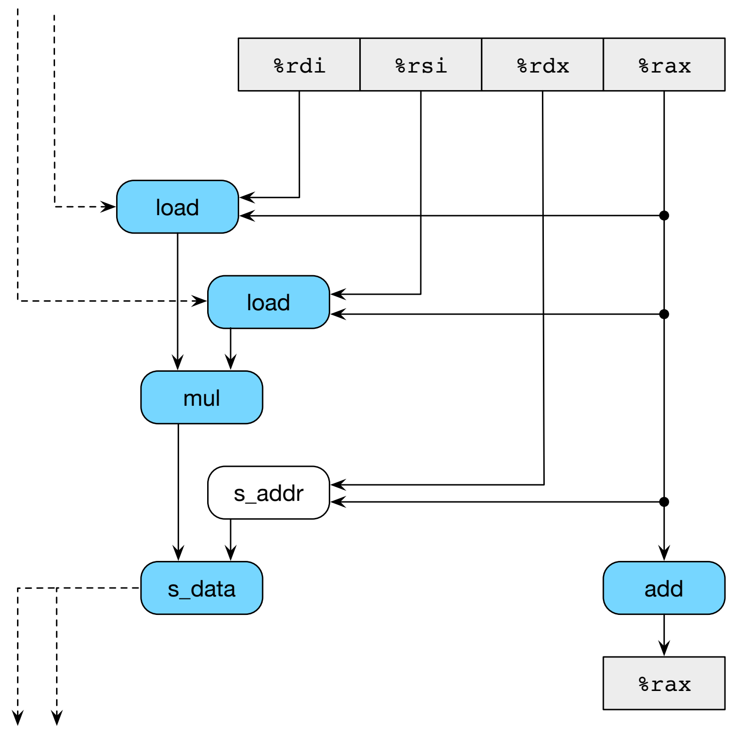
1

```
void do_arr(fp_arr arr1, fp_arr arr2, fp_arr arr3, double *res) {
    long i;
    double *p1 = arr1.data;  /* reduce procedure calls */
    double *p2 = arr2.data;  /* reduce procedure calls */
    double *p3 = arr3.data;  /* reduce procedure calls */
    long len = arr1.len;     /* eliminate loop inefficiencies */
    long limit = len - 1;
    double tmp1, tmp2;
    double acc = *res;      /* eliminate unneeded memory reference */

    /* loop unrolling and enhancing parallelism */
    for (i = 0; i < limit; i += 2) {
        tmp1 = p1[i] * p2[i];
        tmp2 = p1[i+1] * p2[i+1];
        p3[i] = tmp1;
        p3[i+1] = tmp2;
        acc += tmp1 + tmp2;
    }

    for (; i < len; i++) {
        tmp1 = p1[i] * p2[i];
        p3[i] = tmp1;
        acc += tmp1;
    }

    *res = acc;
}
```

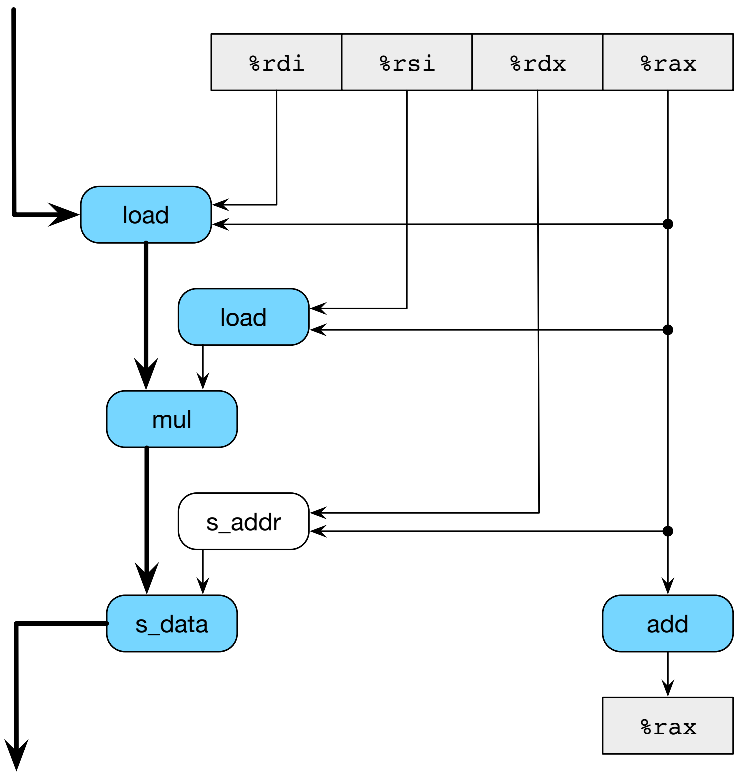
2.

$$Speedup = \frac{T_{old}}{T_{new}} = \frac{1}{(1-\alpha)+\alpha/k} = \frac{1}{(1-0.4)+0.4/5} \approx 1.47$$

3.



4. The CPE is expected to be 11. In this case, the load of one iteration depends on the result of the store from the previous iteration. The critical path is the “load-multiply-store” showed in the figure below.



Problem 5 : Processor (34 points)

1. a. Load/use hazard: 5
   misprediction of jle: 8
   misprediction of jg: 2

   b. 59/40 = 1.475

2. [1] !e_cnd [2] normal [3] normal [4] normal
   [5] e_cnd [6] stall [7] stall [8] normal

3.

```
   Loop:
mrmovq    (%rdi), %r10
andq      %r10, %r10
credoq
iaddq     $1, %rax
iaddq     $4, %rdi
iaddq     $-1, %rdx
jg        Loop
halt
```

59-51=8 cycles

4. a.

	taken
M	
E	credoxx
D	

	ret 1
M	
E	
D	ret

b. [1] stall [2] stall [3] normal [4] normal

5. bool F_stall = { … || E_icode == ICREDOXX && e_cnd };

   bool D_bubble = { … && !(E_icode == ICREDOXX && e_cnd)} ;

   bool D_stall = { … || E_icode == ICREDOXX && e_cnd };

6. Yes. Ret and jxx misprediction will pass W_valM or M_valA to f_pc. But these instructions will bubble the following instructions at D stage or E stage. So if E_icode == ICREDOXX, it means the instruction is not bubbled, so the previous instructions should not be ret or jxx(misprediction), which means the f_pc would not use value W_valM or M_valA.