1.

B[g]=A[f]+A[f+1];

2.

i.

without:

Instruction	I Mem	Reg Rd	ALU	D Mem	Reg Wr	Total
R-Type	400	200	120		200	920
load	400	200	120	350	200	1270
store	400	200	120	350		1070
branch	400	200	120			720
jump	400					400

with:

Instruction	I Mem	Reg Rd	ALU	D Mem	Reg Wr	Total
R-Type	400	200	420		200	1020
load	400	200	420	350	200	1370
store	400	200	420	350		1170
branch	400	200	420			820
jump	400					400

ii.

1370*0.95=1235

Speedup = 1270/1235=1.028

3. In the original ISA, the funct field is only used when the op field is all 0's(000000). Therefore, in the original ISA, there are 127 total instructions (2^6 - 1 + 2^6). Therefore, the new ISA would have 254 instructions. One straightforward solution is to make the op field 8 bits and remove the funct field (2^8 = 256). That would decrease our shift amount, so the implication is that we would need more than 1 shift instruction if we want to shift by more than a 3-bit value.

Another solution is to make a new instruction type for shift instructions; We can call that type S. Our op field can be increased to 7 bits, and then we can have more op codes utilize the funct field (as opposed to only all 0's). More precisely: 2^7 - x + (2^4) x = 254, where x is the number of op codes that utilize the 4 funct bits. That means that 8 op codes will need to use the funct field so that we can have at least 254 different instructions. Also, in this solution the shift amount would be decreased to 4 bits.

R type

op 7bits	rs 7bits	rt 7bits	rd 7bits	Funct 4bits

I type

op 7bits	rs 7bits	rt 7bits	Immediate 11 bits

J type

op 7bits	Address 25 bits

Or you can have

R type

op 8bits	rs 7bits	rt 7bits	rd 7bits	Shamt 3bits

I type

op 8bits	rs 7bits	rt 7bits	Immediate 10 bits

J type

op 8bits	Address 24 bits

As a result, I-type immediate decrease from 16 bits to 11 bits (or 10 bits), which is -2^10 to 2^10-1 (-2^9 to 2^9 -1).

Branch is an I-type so now can only reach 2047(1023) instructions away. Jump address is divided in half.

4.

```
1   for(i=1;i<=n;i++){
2     for(j=1;j<=n;j++){
3       if(a[i]==b[j])
4         opt[i][j]=opt[i-1][j-1];
          else{
5         if(opt[i-1][j]<opt[i][j-1])
6           opt[i][j]=opt[i-1][j]+1;
            else
7           opt[i][j]=opt[i][j-1]+1;
          }
       }
    }
8   return opt;
```