Problem 8. (9 points):

Consider the following C declarations:

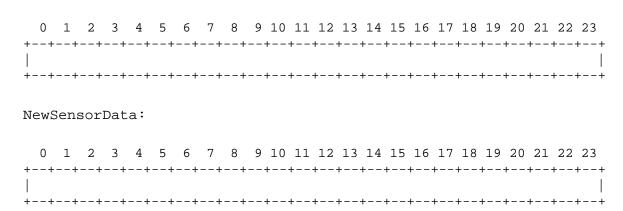
```
typedef struct {
    short code;
    long start;
    char raw[3];
    double data;
} OldSensorData;

typedef struct {
    short code;
    short start;
    char raw[5];
    short sense;
    short ext;
    double data;
} NewSensorData;
```

A. Using the templates below (allowing a maximum of 24 bytes), indicate the allocation of data for structs of type OldSensorDataNewSensorData. Mark off and label the areas for each individual element (arrays may be labeled as a single element). Cross hatch the parts that are allocated, but not used (to satisfy alignment).

Assume the Linux alignment rules discussed in class. Clearly indicate the right hand boundary of the data structure with a vertical line.

OldSensorData:



B. Now consider the following C code fragment:

```
void foo(OldSensorData *oldData)
{
    NewSensorData *newData;

    /* this zeros out all the space allocated for oldData */
    bzero((void *)oldData, sizeof(oldData));

    oldData->code = 0x104f;
    oldData->start = 0x80501ab8;
    oldData->raw[0] = 0xe1;
    oldData->raw[1] = 0xe2;
    oldData->raw[2] = 0x8f;
    oldData->raw[-5] = 0xff;
    oldData->raw[-5] = 0xff;
    oldData->data = 1.5;

    newData = (NewSensorData *) oldData;
    ...
```

Once this code has run, we begin to access the elements of newData. Below, give the value of each element of newData that is listed. Assume that this code is run on a Little-Endian machine such as a Linux/x86 machine. You must give your answer in hexadecimal format. **Be careful about byte ordering!**.