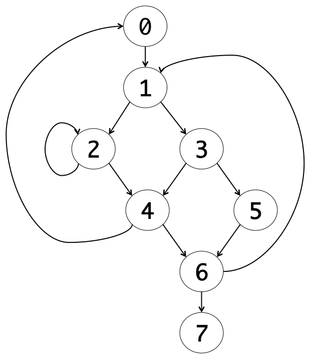

(a) A directed graph

```
(0,1) (2,4) (6,1)
(1,2) (3,4) (6,7)
(1,3) (3,5) (4,6)
(2,2) (4,0) (5,6)
```

(b) Edge list

To Node

From Node	0	1	2	3	4	5	6	7
0	0	1	0	0	0	0	0	0
1	0	0	1	1	0	0	0	0
2	0	0	1	0	1	0	0	0
3	0	0	0	0	1	1	0	0
4	1	0	0	0	0	0	1	0
5	0	0	0	0	0	0	1	0
6	0	1	0	0	0	0	0	1
7	0	0	0	0	0	0	0	0

(c) Adjacency matrix

Figure 1: An example of a directed graph and the corresponding edge list and adjacency matrix.

Question 4 (30 points):

Representation of a Directed Graph: A directed graph $G(V, E)$ is formed by a set of vertices V and a set of edges E. A vertex is often also called a "node" in the graph. Two common data representations for a directed graph are a list of edges and an adjacency matrix. In this question, we assume that the vertices are labeled with integer values: 0, 1, 2, $\cdots$. Figure **??** shows an example of a directed graph with the corresponding edge list and adjacency matrix to illustrate these two data representation. The graph shown in Figure **??** is only an example. This question asks that you write code that works for **ANY** directed graph.

Representation of an edge list in memory: In this question, the edge list is represented as a list of 32-bit words. Two words are used to represent each edge. The first word contains the number of the node that is the source of the edge, also called the `fromNode`, the second word contains the number of the node that is the target of the edge, also called the `toNode`. The end of the list is signalled by a sentinel value, which is a 32-bit word containing the value -1 (`0xFFFF FFFF`).

Representation of the Adjacency Matrix in memory: Consider a directed graph that has k nodes. The adjacency matrix will be represented by k bit vectors, each vector containing k bits. The bit vectors will be stored sequentially in memory. Let w be the number of words occupied by each bit vector. k is determined by the number of nodes in the graph. If $k \leq 32$, then $w = 1$. If $33 \leq k \leq 64$ then $w = 2$, *etc.* In general:

$$w = \left\lceil \frac{k}{32} \right\rceil \tag{1}$$

In this question you will write MIPS assembly to convert an edge-list representation of a directed graph into an adjacency-matrix representation of the same graph. Your code will be divided into two functions as specified below. You must write the two functions to work independently and you must follow all the MIPS calling conventions in both of them.

1. (**15 points**) Write the MIPS assembly code for a function called `SetBit` that receives the address in memory where a bit vector is stored and the position of a bit to be set to 1. This function must work for any bit-vector length (the bit position may be higher than 32). Therefore a bit vector may be formed by multiple words. Within each of the words of a bit vector, the lowest bit position will correspond to the Least-Significant Bit of the word (bit 0). The only change that this function does is to write the bit 1 is the specified bit position.

   Input:

   `$a0`: address of first word of bit vector `$a1`: bit to be set

Space to write code for `SetBit` function

2. (**15 points**) Write MIPS assembly code for a function called `ConvertGraph` that converts the representation of a directed graph from a list of edges to an adjacency-matrix representation. The adjacency matrix is formed by k bit vectors that are stored contiguously in memory, where k is the number of nodes in the graph. The interface for this routine is as follows:

   Input: A directed graph specified as an edge list stored in memory.

   Output: An adjacenty-matrix representation of the same directed graph, stored in memory.

   Library: A collection of bit-vector routines that include a `SetBit` routine.

   Precondition: The adjacency matrix is pre-allocated and zeroed.

   Parameters: These are the input parameters for the function:

   - **`$a0`:** address of first word of first edge in edge list
   - **`$a1`:** number of nodes in directed graph
   - **`$a2`:** address of first bit vector in adjacency matrix

Space to write code for `ConvertGraph` function