Simulator

1. Big picture

The program is used to simulate computer execute the program. The whole program is divided to three part.

The first part is about the memory simulation and register simulation. For memory simulation, the program firstly dynamically allocate 6MB memory space to store three segment which are text segment, data segment and stack segment. For register simulation, the program dynamically allocate 35*4 B memory space to store 35 registers which are not in 6MB talked above.

The second part is about data initialize. In this part, the program will firstly read from .asm file and put data in the .data segment to static data part. And then, the program will read from a .txt file and put the instructions which are already converted to binary code into text segment.

The third part is about simulation cycle. In this part, the program operate a three instructions cycle. The first step is get the instruction from PC register which point to the text segment. The second step is PC plus 4 which point to the next instruction. The third step is to execute the instruction using the functions, data in the 6MB memory space and 35 register. After that repeat three steps until the last instruction is executed.

The more details is as follows.

2. High-level Implement

This program can be divided to three part.

Memory simulation & Register simulation

For memory simulation, dynamically allocate 6MB memory space to store three segment which are text segment, data segment and stack segment.

For register simulation, dynamically allocate 4*35 B register memory space to store the value of 35 registers.

Data initialize

Data initialize can be divided to text data initialize and static data initialize.

For text data initialize, I convert the instruction number to integers and store them in the text segment by using text_data_ptr.

For static data initialize, I define a union called data_Block to store the data.

Simulation cycle

For the last part, there three steps.

The first step is to get the instruction in the text segment which is pointed by PC register.

The second step is PC plus four in order to point to the next instruction.

The third step is to execute the instruction by using the functions.
We cycle the above three steps until all the instructions are executed.

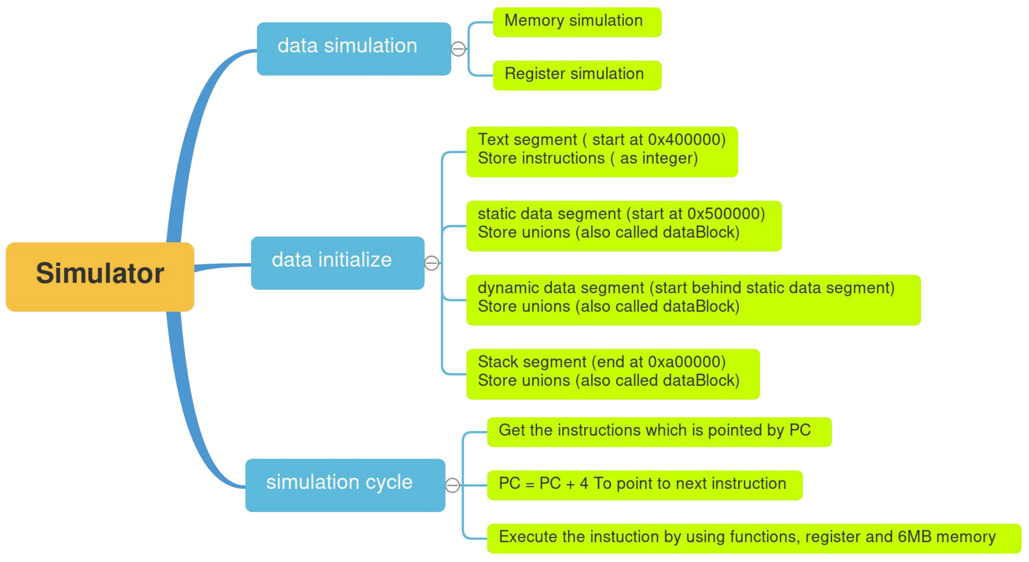
Simulator
data simulation
Memory simulation
Register simulation
data initialize
Text segment (start at 0x400000)
Store instructions (as integer)
static data segment (start at 0x500000)
Store unions (also called dataBlock)
dynamic data segment (start behind static data segment)
Store unions (also called dataBlock)
Stack segment (end at 0xa00000)
Store unions (also called dataBlock)
simulation cycle
Get the instructions which is pointed by PC
PC = PC + 4 To point to next instruction
Execute the instuction by using functions, register and 6MB memory

Figure 1: High level Implement

3. Implement Details

Memory simulation & Register simulation

In this part, I use the malloc methods to allocate the memory. The text segment start at 0x400000 (virtual address). The data segment start at 0x500000 (virtual address). This segment contains two part. The first is static data part which store the static data and start at 0x500000 (virtual address); The second is dynamic data data which store the dynamic data and start at right behind the static data part. The stack segment end at 0xa00000 (virtual address). And define 4 pointers. Memory_ptr point to the start of the 6MB. Static_data_ptr point to the end of the static data part. Dynamic_data_ptr point to the end of the dynamic data part. Stack_ptr point to end of the 6MB.

For register simulation, we initialize some registers. PC register is equal to 0x400000 (virtual address), $fp and $sp are equal to 0xa00000 (virtual address) and $gp is equal to 0x500080 (virtual address). I also define a map to store the register number and corresponding ptr. The map I used is <string, int*>. The string is the register number, int* is the pointer to an integer. The register is used to store integer. If the register store the address, they all are the virtual address. I define a method to convert the virtual address to real address.

And after the part 1, we can get a following memory figure.

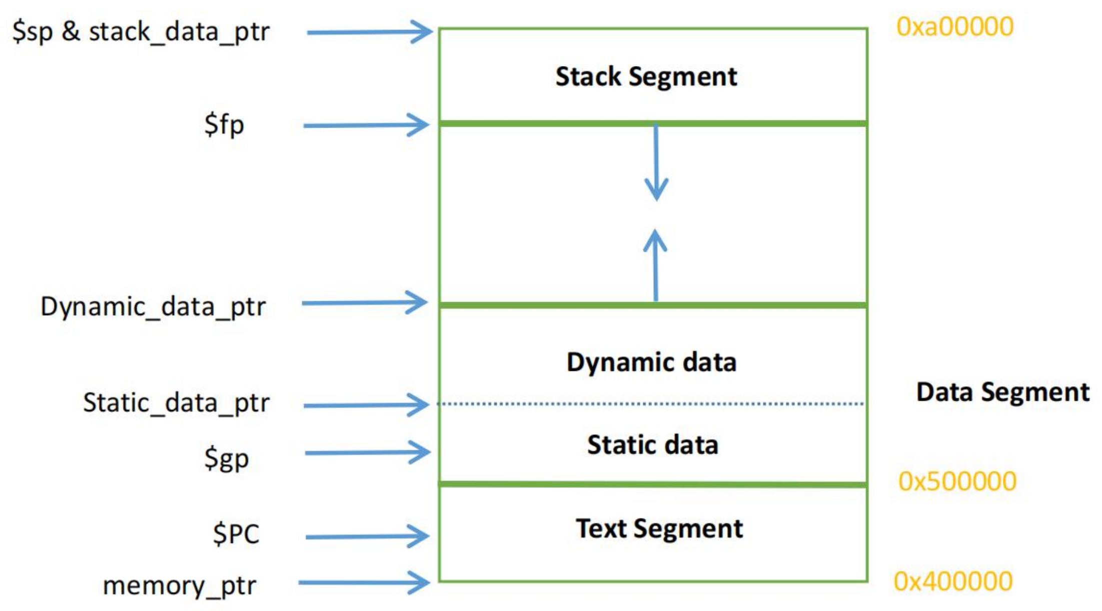

Figure 2: Memory simulation & Register simulation

Data initialize

For text segment, I store the instructions in form of integer. For data segment, I store the data in form of union. The size of the union is 4 B which can store one integer or two short numbers or four chars.

For static data initialize, I define a union called data_Block to store the data from .asm file. The size of the data_Block is 4 B which can store one integer or two short numbers or four chars. There are 5 type data in the static data. For asciiz, ascii and byte types, I store them as the char. For half type, I store them as short. For word type, I store them as integer.

Simulation cycle

For the last part, there three steps.

The first step is to get the instruction in the text segment which is pointed by PC register.

The second step is PC plus four in order to point to the next instruction.

After get the instruction, we first judge the type of the instruction. If the instruction type is R, then we store the six elements in six strings. They are opcode, rs, rt, rd, sa and function. If the instruction type is I, then we store the four elements in four strings. They are opcode, rs, rt and immediate. If the instruction type is J, then we store the two elements in two strings. They are opcode and target. After that, we execute the instruction by using the functions we defined. The function may use the registers, which can be get using map and the value of rs, rt or rd. By the way, the registers store the virtual address. We need to convert is to real memory.