nesdoug

7. Input

Next we will cover button presses. Not too complicated… Joypad 1 is at $4016 and Joypad 2 is at $4017. I read them once a frame, right after all the Ppu updates and Sprite update and setting the scroll.

I always have 2 variables for each, the current buttons, and last frame's button presses. First, write 1 to $4016, then write 0 to $4016. Now you can read button presses (one at a time = 8 reads per Joypad) and shift them into the new button variables. The order of the bits read are A, B, Select, Start, Up, Down, Left, Right.

We will use the metasprite from the last time and let the user move him around. I thought long and hard, and decided to write this bit in ASM. You don't need to understand it. It works. You can move on and worry about other things. 🙂

You call an ASM function from the C code, you just need a prototype, the linker will put them together when you compile.

void Get_Input(void);

Of course you don't have to use (void), as I tend to do. If you pass a variable, I highly recommend you use __fastcall__ which will store 1 passed variable in the A and X registers, rather than the C Stack.

> *If a function is declared as* `__fastcall__` *(or* `fastcall` *), the last (rightmost) parameter is not passed on the stack, but passed in the primary register to the called function. This is* `A` *in case of an eight bit value,* `A` / `X` *in case of a 16 bit value, and* `A` / `X` / `sreg` *in case of a 32 bit value.*

http://wiki.cc65.org/doku.php?id=cc65:parameter_passing_and_calling_conventions (http://wiki.cc65.org/doku.php?id=cc65:parameter_passing_and_calling_conventions)

Ok, back to Get_Input();

Just call this function once per frame, and it will store the button presses for you.

Here's the link to the source code. Now you can move the little guy with a controller. You'll see I added a file called asm4c.s which has our asm function. Also, the compile.bat file has been changed to include/link asm4c into our final output file. Also, I replaced the auto-move code with this function…

```
void move_logic(void) {
 if ((joypad1 & RIGHT) != 0){
  state = Going_Right;
  ++X1;
 }
 if ((joypad1 & LEFT) != 0){
  state = Going_Left;
  --X1;
 }
 if ((joypad1 & DOWN) != 0){
  state = Going_Down;
  ++Y1;
 }
 if ((joypad1 & UP) != 0){
  state = Going_Up;
  --Y1;
 }
}
```

I had to define “RIGHT” “LEFT” “UP” “DOWN” to how they would appear in the joypad1 variable. Each bit is a button. It will either be a 0 or a 1. We are masking just that bit with the ‘&’ — which will be != 0 if that specific button was pressed.

```
#define RIGHT  0x01
#define LEFT  0x02
#define DOWN  0x04
#define UP   0x08
#define START  0x10
#define SELECT  0x20
#define B_BUTTON 0x40
#define A_BUTTON 0x80
```

Here’s the link to the source code…

http://dl.dropboxusercontent.com/s/zeubcy1ojyxbrgb/lesson5.zip (http://dl.dropboxusercontent.com/s/zeubcy1ojyxbrgb/lesson5.zip)

November 22, 2015April 16, 2017 dougfraker

One thought on “7. Input”

1. **Prk1001** says:
November 20, 2016 at 2:25 pm Edit
I had this error :
ld65: Error: Missing memory area assignment for segment `ONCE’

To correct this, in nes.cfg :
put in MEMORY :
ONCE: start = $0000, size = $0000;
put in SEGMENTS :
ONCE: load=PRG, type=ro, optional=yes;

Hope it can serve 🙂

Reply