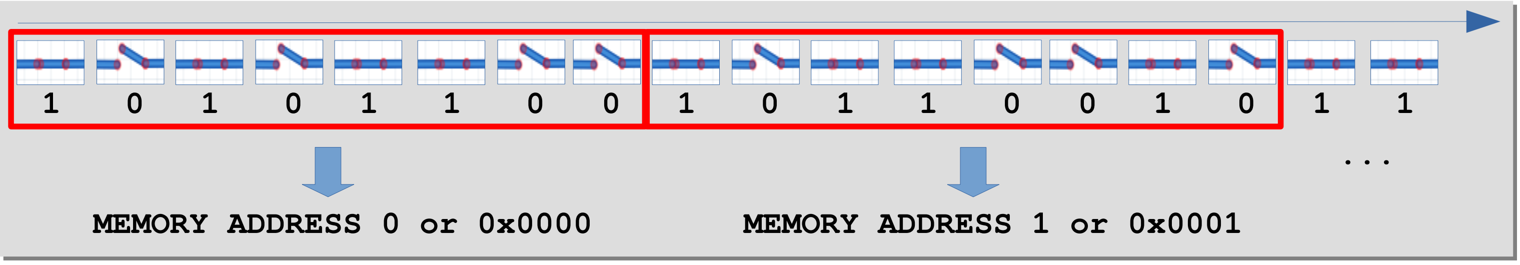
ATMEGA328/Arduino Uno – MEMORY ADDRESS
https://github.com/teaksoon/lmaewapm

MEMORY – A Long Sequence of ON/OFF “switches” or BITs represented by 1 or 0
MEMORY ADDRESS – LOCATION of a SINGLE MEMORY BLOCK of 8-BITS



One MEMORY ADDRESS = One BYTE (8-BIT)

MEMORY ADDRESS Starts from 0, increase by one for next MEMORY ADDRESS
(MEMORY ADDRESS = 0, MEMORY ADDRESS = 1, ...)

Very often we see MEMORY ADDRESS coded in Hexadecimal number format

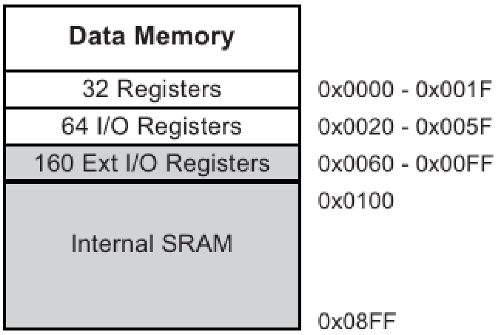
Example:

```
MEMORY ADDRESS 0     is often coded as 0x0000 in Hexadecimal format
MEMORY ADDRESS 1     is often coded as 0x0001 in Hexadecimal format
MEMORY ADDRESS 2298  is often coded as 0x08FA in Hexadecimal format
MEMORY ADDRESS 65535 is often coded as 0xFFFF in Hexadecimal format
```

The Hexadecimal number format contains letters like ABCDEF, which sometimes can scare off beginners. Do not worry, they are just numbers. IF you do not like the Hexadecimal numbers you can also use the Decimal number, they are the same.

There are many hex-decimal-binary conversion tools available online. We can use them to do the conversion for us (we can do manual conversion, but it takes time)

Figure 7-2. Data Memory Map



According to the **ATMEGA328P micro-controller, datasheet** above, Our MEMORY ADDRESS does not start from the WORKING MEMORY(SRAM) alone, the MEMORY ADDRESS also includes those from the CPU MEMORY(REGISTERS)

Our normal Variable will most likely use the SRAM MEMORY. So our normal Variable MEMORY ADDRESS will normally range from 0x0100(256) to 0x08FF(2303) – which has a total of 2048 MEMORY ADDRESS in SRAM (that is the total SRAM MEMORY we have in the ATMEGA328P micro-controller)

Each MEMORY ADDRESS for ATMEGA328P micro-controller is a 16-BIT number

One MEMORY ADDRESS = One BYTE (8-BIT)

When we **Declare a regular Variable**, each datatype will **"Reserve" different number of BITs from the MEMORY** for our Variable

Each "char" Variable will Reserve 8-BIT (1-BYTE)
- Means, each "char" Variable will occupy One(1) MEMORY ADDRESS

Each "int" Variable will Reserve 16-BIT (2-BYTE)
- Means, each "int" Variable will occupy Two(2) MEMORY ADDRESS

Each "long" Variable will Reserve 32-BIT (4-BYTE)
- Means, each "long" Variable will occupy Four(4) MEMORY ADDRESS

The **MEMORY ADDRESS** is the LOCATION where the DATA will be stored. If the Declared Variable **occupy multiple MEMORY ADDRESS**, the **MEMORY ADDRESS** will be the **first MEMORY ADDRESS** from the multiple MEMORY ADDRESS

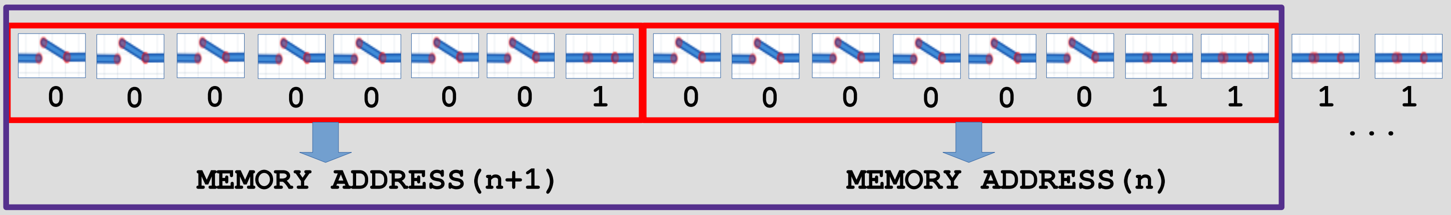
Arduino IDE|Save PROGRAM as: **c_variable_memory_address**
Enter codes below and upload. Use the Serial Monitor to see results

```
void setup() {
  Serial.begin(9600);Serial.print("\n\nSerial Monitor(9600)...");

  int var_name;
  Serial.print("\n\nNumber of BITs reserved for int Variable = ");
  Serial.print(sizeof(var_name)*8);
  Serial.print("\nThe Starting MEMORY ADDRESS of int Variable = ");
  Serial.print((unsigned int) &var_name); // get MEMORY ADDRESS

  var_name = 259; // change DATA in MEMORY
  Serial.print("\n\nRetrieved DATA from MEMORY = ");
  Serial.print(var_name); // read data from MEMORY

  Serial.print("\nThe 16-BITs in the MEMORY = ");
  for (int i=15; i>=0; i--) {
    Serial.print((var_name >> i) & 1);Serial.print(" ");
  }
  Serial.print(" ( int occupies Two(2) MEMORY ADDRESS )");
}
void loop(){}
```

int var_name;	- an int Variable "var_name" is declared
&var_name	- To Get the MEMORY address from var_name Variable
var_name = 259	- To change DATA in var_name Variable MEMORY to 259
var_name	- To Retrieve DATA from var_name Variable MEMORY

int var_name;
- var_name will occupy Two(2) MEMORY ADDRESS(n) and MEMORY ADDRESS(n+1)

var_name = 259;
- 259 in decimal will be stored in Two(2) MEMORY ADDRESS as 0000000100000011
MEMORY ADDRESS(n)   - store 00000011
MEMORY ADDRESS(n+1) - store 00000001

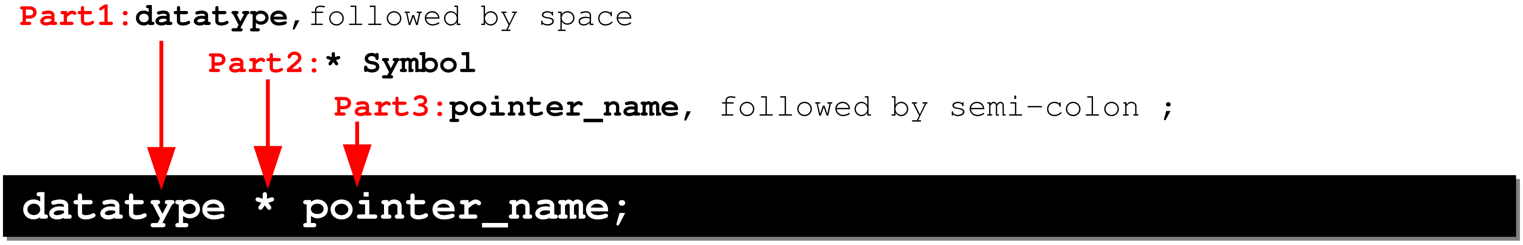


In order to manipulate the DATA stored in the MEMORY, we normally use Variables. After we declare our Variable, we only need to specify the name of the Variable, we can retrieve and store data in the MEMORY

The C-Language have the ability to manipulate the MEMORY directly, without using Variables. All C-Language need is the MEMORY ADDRESS(which is just a 16-BIT number, a LOCATION in the MEMORY). Meaning, the entire MEMORY is now exposed to the C-Language and there is nothing we can do to stop the C-Language from doing whatever it wishes anywhere in the MEMORY. Just imagine, even when we are typing half-way, it involves MEMORY and what if C-Language knows the MEMORY ADDRESS and starts to meddle with it

The C-Language has something called "Pointers"

Declare a "Pointer Variable" (declaration is similar to a Normal Variable)
Part1:datatype
Part2:* Symbol
Part3:pointer_name



Part1:datatype, followed by space

Part2:* Symbol

Part3:pointer_name, followed by semi-colon ;

```
datatype * pointer_name;
```

Once "Pointer Variable" is Declared like the above, we can start using the Pointer Variable. We can do Two things with the Pointer Variable,

1. Store MEMORY ADRESSS (a 16-BIT number) into the Pointer Variable

2. Use Pointer Variable to Retrieve or Store the DATA in the PHYSICAL MEMORY based on the MEMORY ADDRESS in the Pointer Variable

Each Pointer Variable can only store ONE MEMORY ADDRESS (a 16-BIT number)

1. Store MEMORY ADRESSS (a 16-BIT number) into the Pointer Variable

```
int var_name;  // Normal Variable Declaration
int *ptr_name; // Pointer Variable Declaration

ptr_name = &var_name; // "var_name" Normal Variable MEMORY ADDRESS

ptr_name = 2298; // direct MEMORY ADDRESS assignment, ADDRESS LOCATION 2298
```

2. Use Pointer Variable to Retrieve or Store the DATA at the PHYSICAL MEMORY based on the MEMORY ADDRESS in the Pointer Variable

```
int *ptr_name;

ptr_name = 2298; // direct MEMORY ADDRESS assignment, ADDRESS LOCATION 2298

*ptr_name = 259; // Store int DATA Starting MEMORY ADDRESS 2298

*ptr_name; // Retrieve int DATA Starting MEMORY ADDRESS 2298
```

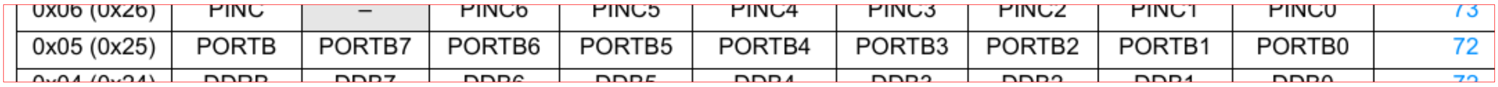
Arduino IDE|Save PROGRAM as: **c_variable_pointer**
Enter codes below and upload. Use the Serial Monitor to see results

```
void setup() {
  Serial.begin(9600);Serial.print("\n\nSerial Monitor(9600)...");

  int var_name = 259; // DATA stored in a Normal Variable

  int *ptr_int;
  ptr_int = &var_name; // get MEMORY ADDRESS from Variable

  Serial.print("\n\nMEMORY ADDRESS stored in Pointer Variable = ");
  Serial.print((unsigned int) ptr_int);
  Serial.print("\nDATA retrieved from MEMORY using Pointer = ");
  Serial.print(*ptr_int);
  Serial.print("\nThe 16-BITs in the MEMORY = ");
  for (int i=15; i>=0; i--) {
    Serial.print((*ptr_int >> i) & 1);Serial.print(" ");
  }
  Serial.print(" ( int occupies Two(2) MEMORY ADDRESS ) ");

  Serial.print("\n\nStore DATA into MEMORY using Pointer ");
  Serial.print("*ptr_int = 258;");
  *ptr_int = 258;

  Serial.print("\n\nMEMORY ADDRESS stored in Pointer Variable = ");
  Serial.print((unsigned int) ptr_int);
  Serial.print("\nDATA retrieved from MEMORY using Pointer = ");
  Serial.print(*ptr_int);
  Serial.print("\nThe 16-BITs in the MEMORY = ");
  for (int i=15; i>=0; i--) {
    Serial.print((*ptr_int >> i) & 1);Serial.print(" ");
  }
  Serial.print(" ( int occupies Two(2) MEMORY ADDRESS ) ");

  // 8-BIT Pointer Variable to access individual MEMORY ADDRESS
  // - each MEMORY ADDRESSS is 8-BIT
  char *ptr_char;
  ptr_char = (unsigned int) &var_name; // MEMORY ADDRESS(n) from Variable

  Serial.print("\n\nDATA in individual MEMORY ADDRESS( ");
  Serial.print((unsigned int) ptr_char); Serial.print(" ) = ");
  for (int i=8; i>=0; i--) {
    Serial.print((*ptr_char >> i) & 1);Serial.print(" ");
  }
  ptr_char = ptr_char+1; // Next MEMORY ADDRESS(n+1)
  Serial.print("\nDATA in individual MEMORY ADDRESS( ");
  Serial.print((unsigned int) ptr_char);Serial.print(" ) = ");
  for (int i=8; i>=0; i--) {
    Serial.print((*ptr_char >> i) & 1);Serial.print(" ");
  }
}
void loop(){}
```

NOTE: In this example PROGRAM:
*ptr_int - manipulating the DATA stored in the PHYSICAL MEMORY
ptr_int - manipulating the MEMORY ADDRESS stored in the Pointer Variable

Lets do something “nasty”. Here, we bypass everything and go straight to the MEMORY ADDRESS 0x25 with our Pointer Variable (dont worry, the MEMORY ADDRESS 0x25 is safe to play with, some are not). We can do something worse by accessing or putting something into a MEMORY LOCATION that we should not

Below is from the ATMEGA328P Datasheet, it says 0x25 is the MEMORY ADDRESS for PORTB (confirmed harmless). We are going to change the BIT at position PORTB5 with our Pointer Variable(that BIT controls the Arduino Uno Pin-13)



0x06 (0x26)	PINC	–	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	73
0x05 (0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	72
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	72

Arduino IDE|Save PROGRAM as: **c_variable_pointer_direct**
Enter codes below and upload. Watch the LED on the Arduino Uno board

```
char *ptr_char;

void setup() {
  pinMode(13,OUTPUT);
  ptr_char = (unsigned int) 0x25; // MEMORY ADDRESS 0x25
}
void loop(){
  *ptr_char = *ptr_char | (1<<5);
  delay(250);
  *ptr_char = *ptr_char & ~(1<<5);
  delay(250);
}
```

We can also go “stealth”. In the example above we can see 0x25 in our code. What if, we do not want people to know that we are accessing 0x25. Lets make another PROGRAM(below). I am not encouraging “hacking” with Pointers. For micro-controllers, this is not an issue because we only have one PROGRAM running and we can also make the micro-controller chip unable to load new PROGRAM. However for the regular Computers, this kind of thing can become a big problem. Anyway, lets get back on-track. The purpose of this example is to show a **simple Pointer Variable math**

Arduino IDE|Save PROGRAM as: **c_variable_pointer_math**
Enter codes below and upload. Watch the LED on the Arduino Uno board

```
char *ptr_char;

void setup() {
  ptr_char  = (unsigned int) 0x00; // Start from 0x00
  ptr_char  = ptr_char+36; // Add Decimal 36 to Pointer Variable (Hex 0x24)
  *ptr_char = *ptr_char | (1<<5);
  ptr_char  = ptr_char+1;  // Add Decimal  1 to Pointer Variable (Hex 0x25)
}
void loop(){
  *ptr_char = *ptr_char | (1<<5);
  delay(250);
  *ptr_char = *ptr_char & ~(1<<5);
  delay(250);
}
```

- If the **datatype for Pointer Variable is 1-BYTE**, MEMORY **ADDRESS** will **increase by 1** when we add 1 to the Pointer Variable name

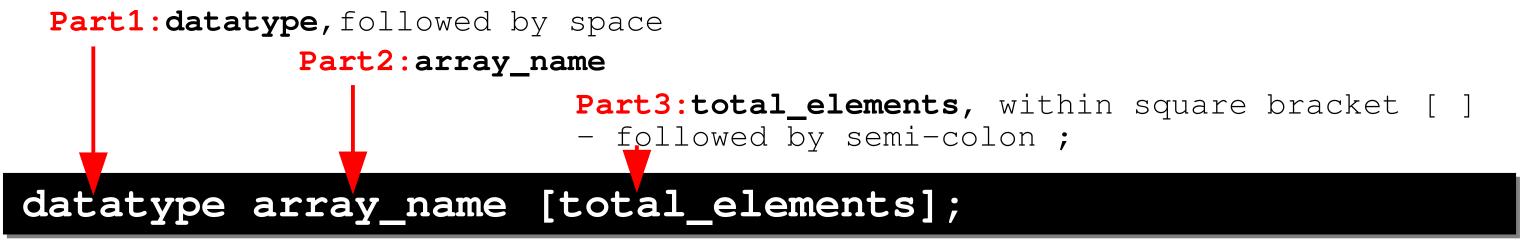
- If the **datatype for Pointer Variable is 2-BYTE**, MEMORY **ADDRESS** will **increase by 2** when we add 1 to the Pointer Variable name

In this example PROGRAM: Pointer Variable datatype is char(1-BYTE), so we code **“ptr_char = ptr_char+1;”** MEMORY ADDRESS will be increased by 1

Array Variable is very similar to Pointer Variable. Both Pointer and Array Variable stores MEMORY ADDRESS

Difference between them: Apart from holding a MEMORY ADDRESS, the Array Variable will also have MEMORY Reserved for all its elements, while the Pointer Variable just hold a single MEMORY ADDRESS

Declare a "Array Variable" (like a Normal Variable)
Part1:datatype
Part2:array_name
Part3:total_elements (within a square bracket [] pair)



Part1:datatype,followed by space

Part2:array_name

Part3:total_elements, within square bracket [ ]
– followed by semi-colon ;

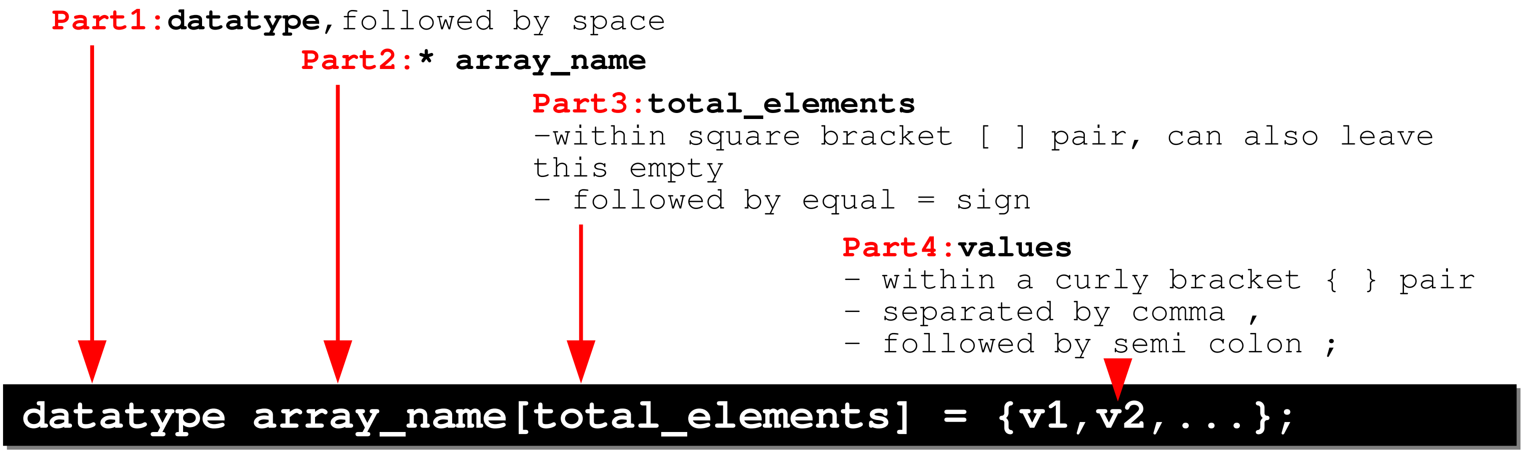
```
datatype array_name [total_elements];
```

Once Array Variable is Declared

The Array Variable store the MEMORY ADDRESS of its first element (a 16-BIT number). It also reserve PHYSICAL MEMORY for each of its elements

We can Retrieve and Store data into the Array elements by using an "ADDRESS" which is an index number. First element has an index of 0, second element has index of 1 and so on...

Declare a "Array Variable" (like a Normal Variable) with **Initial Values**
Part1:datatype
Part2:array_name
Part3:total_elements (within a square bracket [] pair)
Part4:values (within a curly bracket { } pair, separated by comma,)



Part1:datatype,followed by space

Part2:* array_name

Part3:total_elements
–within square bracket [ ] pair, can also leave this empty
– followed by equal = sign

Part4:values
– within a curly bracket { } pair
– separated by comma ,
– followed by semi colon ;

```
datatype array_name[total_elements] = {v1,v2,...};
```

```
datatype array_name[] = {v1,v2,...};
```

Using Array Variable

```
array_name[0];      // Retrieve DATA from the Array element at index 0
array_name[0] = 10; // Store 10 into the Array element at index 0
```

Arduino IDE|Save PROGRAM as: **c_variable_array**
Enter codes below and upload. Use the Serial Monitor to see results

```
void setup() {
  Serial.begin(9600);Serial.print("\n\nSerial Monitor(9600)...");

  int array_name[3]; // Declare Array with 3 elements
  Serial.print("\n\nArray Variable MEMORY ADDRESS ( ");
  Serial.print((unsigned int) array_name);
  Serial.print(" )\nNumber of BITs reserved for Array Variable = ");
  Serial.print(sizeof(array_name)*8);
  Serial.print("\nNumber of elements in Array Variable = ");
  Serial.print((sizeof(array_name)*8)/(sizeof(array_name[0])*8) );

  array_name[0] = 1;
  array_name[1] = 2;
  array_name[2] = 3;
  Serial.print("\n\nData in Array[0]=");Serial.print(array_name[0]);
  Serial.print("\nData in Array[1]=");  Serial.print(array_name[1]);
  Serial.print("\nData in Array[2]=");  Serial.print(array_name[2]);

  Serial.print("\n\nRetrieve the BITs from int Array using int Pointer\n");
  int *ptr;

  ptr = array_name; // Starting ADDRESS
  // ptr = &array_name[0]; // this is the same as above

  Serial.print("\nStarting from ADDRESS ( ");
  Serial.print((unsigned int) ptr);Serial.print(" ) = ");
  for (int i=16; i>=0; i--) {
    Serial.print((*ptr >> i) & 1);Serial.print(" ");
  }
  ptr = ptr+1; // Move to next int ADDRESS
  Serial.print("\nStarting from ADDRESS ( ");
  Serial.print((unsigned int) ptr);Serial.print(" ) = ");
  for (int i=16; i>=0; i--) {
    Serial.print((*ptr >> i) & 1);Serial.print(" ");
  }
  ptr = ptr+1; // Move to next int ADDRESS
  Serial.print("\nStarting from ADDRESS ( ");
  Serial.print((unsigned int) ptr);Serial.print(" ) = ");
  for (int i=16; i>=0; i--) {
    Serial.print((*ptr >> i) & 1);Serial.print(" ");
  }
}
void loop(){}
```

In this example PROGRAM:

This Array Variable has 3 elements. Since we specify “int” datatype for the Array Variable, 16-BIT MEMORY will be “reserved” for each of the elements. So this Array Variable will “reserve” a total of 48-BITS of MEMORY in a sequence.

Each Element will be referenced by an index number,
array_name[0] for first element,
array_name[1] for second element
array_name[2] for third element
Each element is similar to a normal int Variable

The Array Variable Name, “array_name” stores the MEMORY ADDRESS of the first element in the Array Variable. The Code **array_name** and **&array_name[0]** will give us the same value (a 16-BIT number, MEMORY ADDRESS)