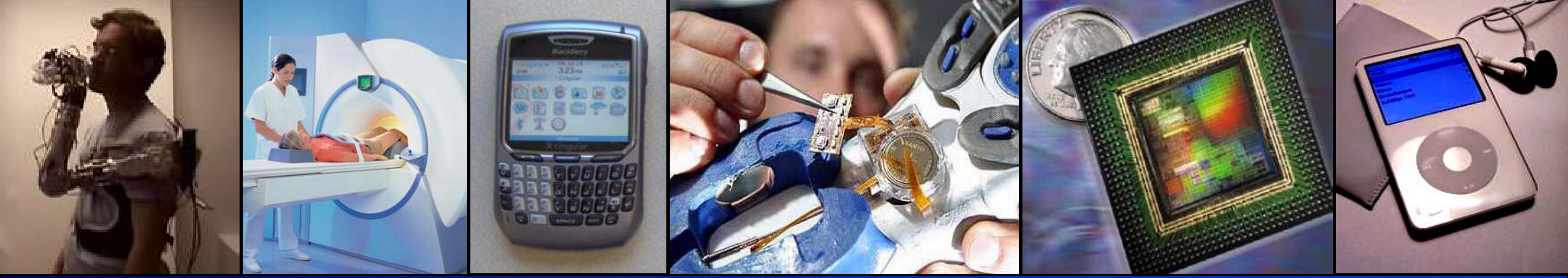


Microprocessor Fundamentals

Topic 3

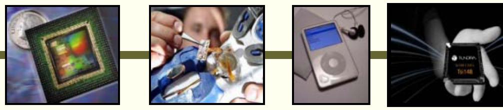
Addressing Modes

Objectives

- Examine the addressing modes of the AVR
  - Register Direct
    - Single Register
    - Two Registers
  - I/O Direct
  - Immediate
  - Data Direct
  - Data Indirect
  - Indirect Program Addressing
  - Relative Program Addressing
- Examine some simple instructions of the AVR

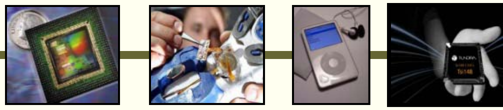
Addressing Modes

- There are 7 basic addressing modes for the AVR
  - Register Direct
    - Single Register
    - Two Registers
  - I/O Direct
  - Immediate
  - Data Direct
  - Data Indirect
  - Indirect Program Addressing
  - Relative Program Addressing



Addressing Modes

- The instructions for a microcontroller can be categorized in several ways:
  - How the instructions access the data
  - How the instructions operate on the data
  - What is the intention, or purpose, of the instruction:
    - Examples:
      - Adding two numbers
      - Controlling the flow of the program



Source Files/Instruction Format

- Programmers write assembly (or C) language program
  - Use a text editor
  - Use the IDE
- In assembly, the source file has four fields to enter information:
  - Label (optional)
  - Instruction (or Mnemonic)
  - Operand
  - Comment (optional)
- A space (or tab) character delineates the fields

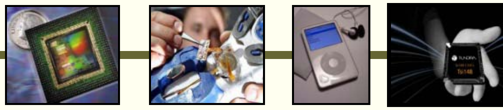
Source Files/Instruction Format

- Format of the source file:

```
label:   mnemonic   operand(s)    comment
```

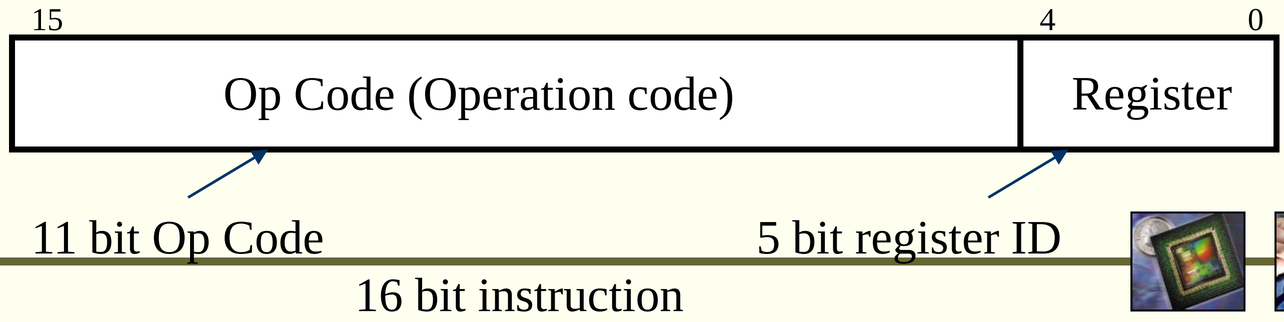
- Example:

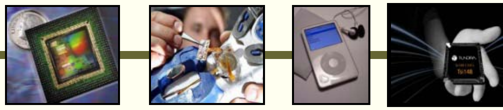
```
here:   add   r1,r2    ;add the two numbers and store in r1
```



Register Direct

- Register Direct (single operand):
  - Instructions can operate on any of the 32 registers
    - The group of 32 registers are referred to as the Register File
  - The microcontroller:
    - Reads the data in the register
    - Operates on the data in the register
    - Stores the results back in the register
  - Format:





Register Direct

- Examples:

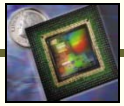
```
result:   com r4    ; compliment the contents of r4
```

```
          inc r15   ; increment the contents of r15 by 1
```

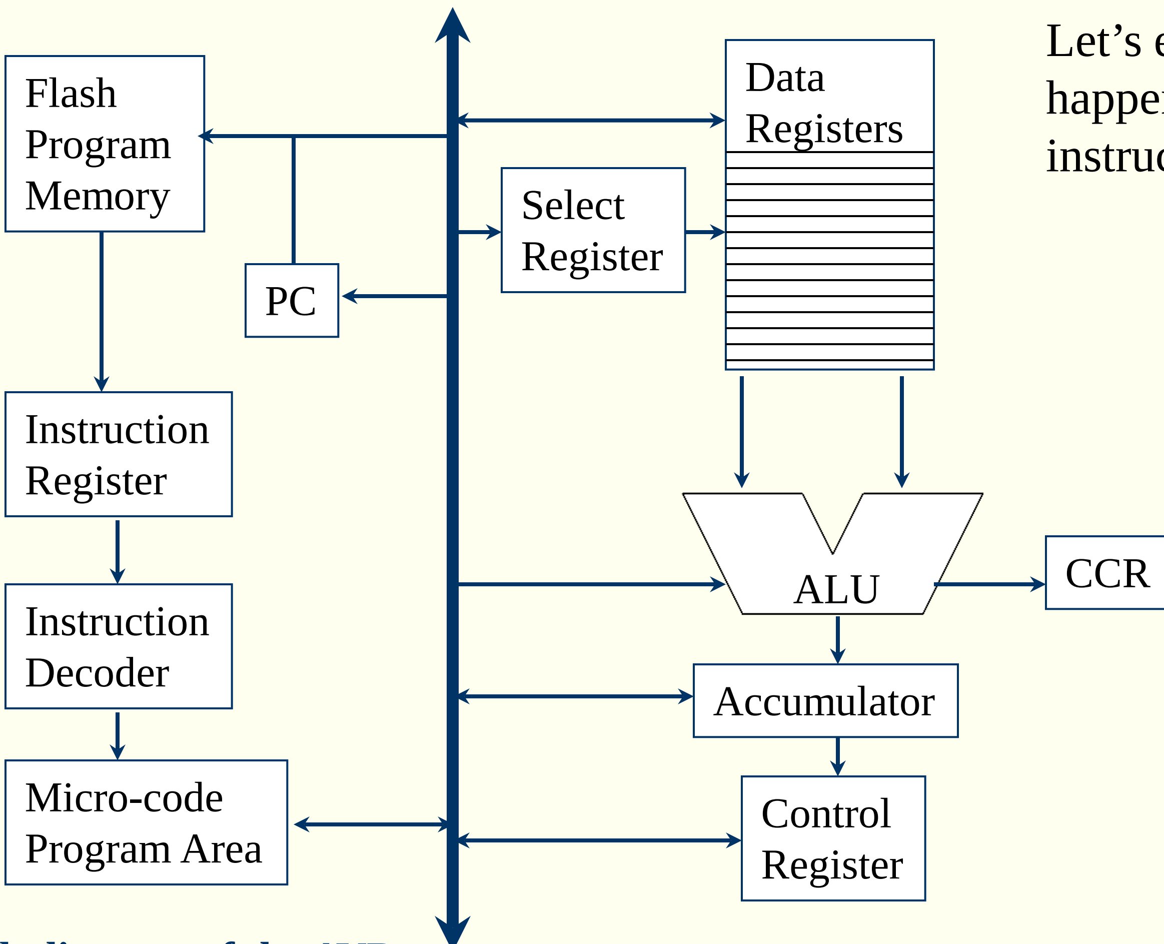
```
          clr r2     ; clear the contents of r2- all 0s in r2
```

```
poodu:  lsl r9      ; shift the contents of r9 1 position to the left
```

Note that some examples have labels, some do not (labels are optional); and there is only one operand in each of these instructions




Executing Instructions

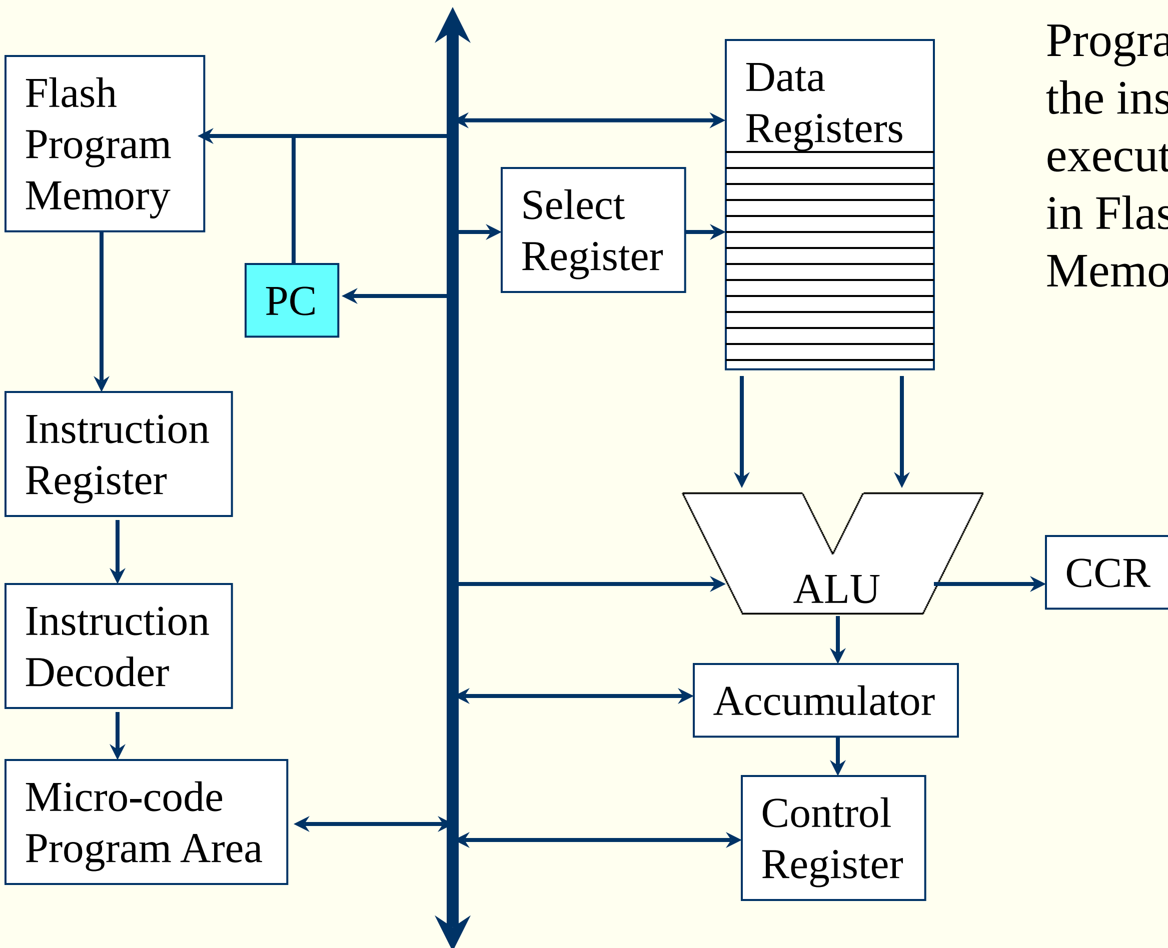


Let's examine what happens as the **COM r2** instruction is executed.

A functional block diagram of the AVR

Instruction: COM r2

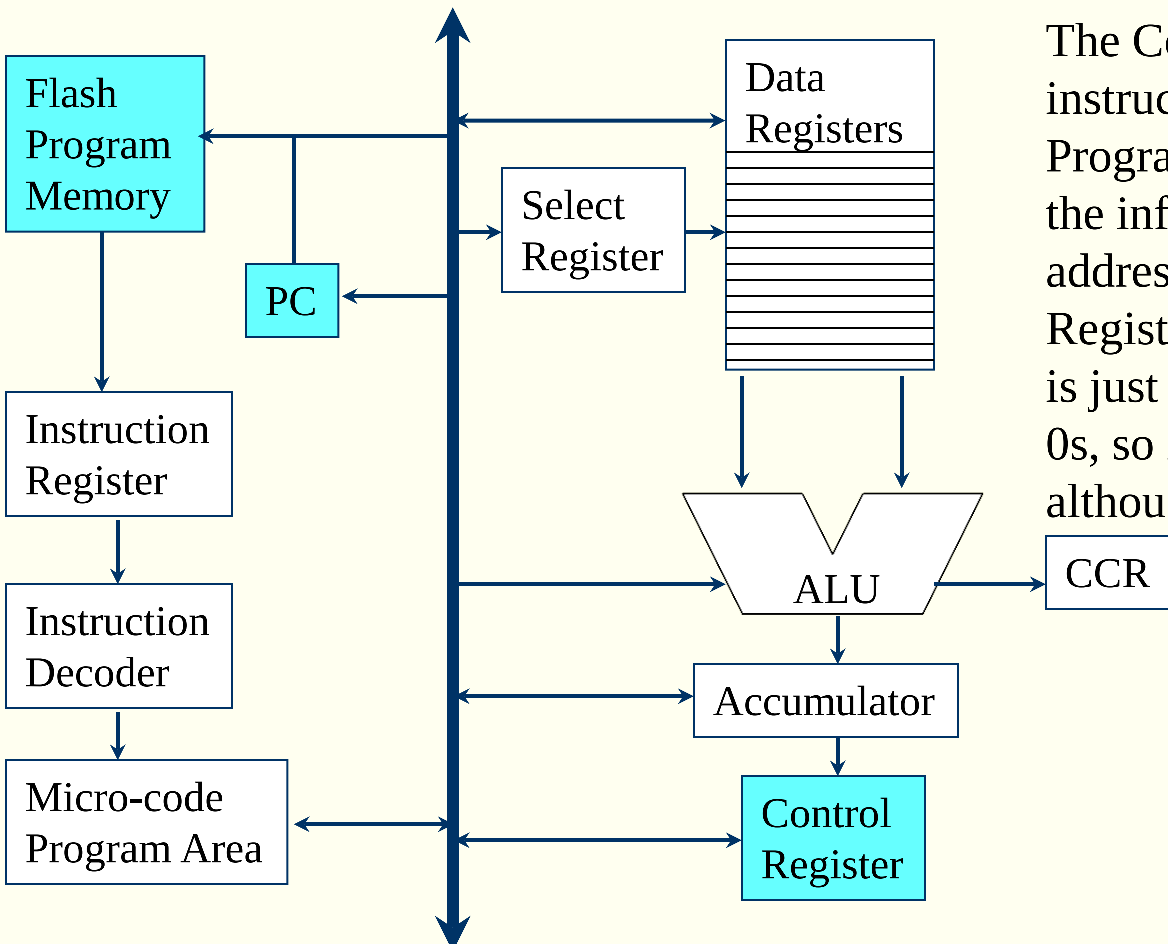
Executing Instructions



Program Counter points to the instruction to be executed (the instruction is in Flash Program Memory).

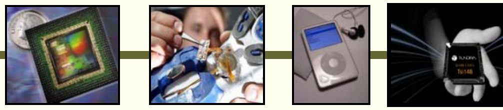
Instruction: COM r2

Executing Instructions

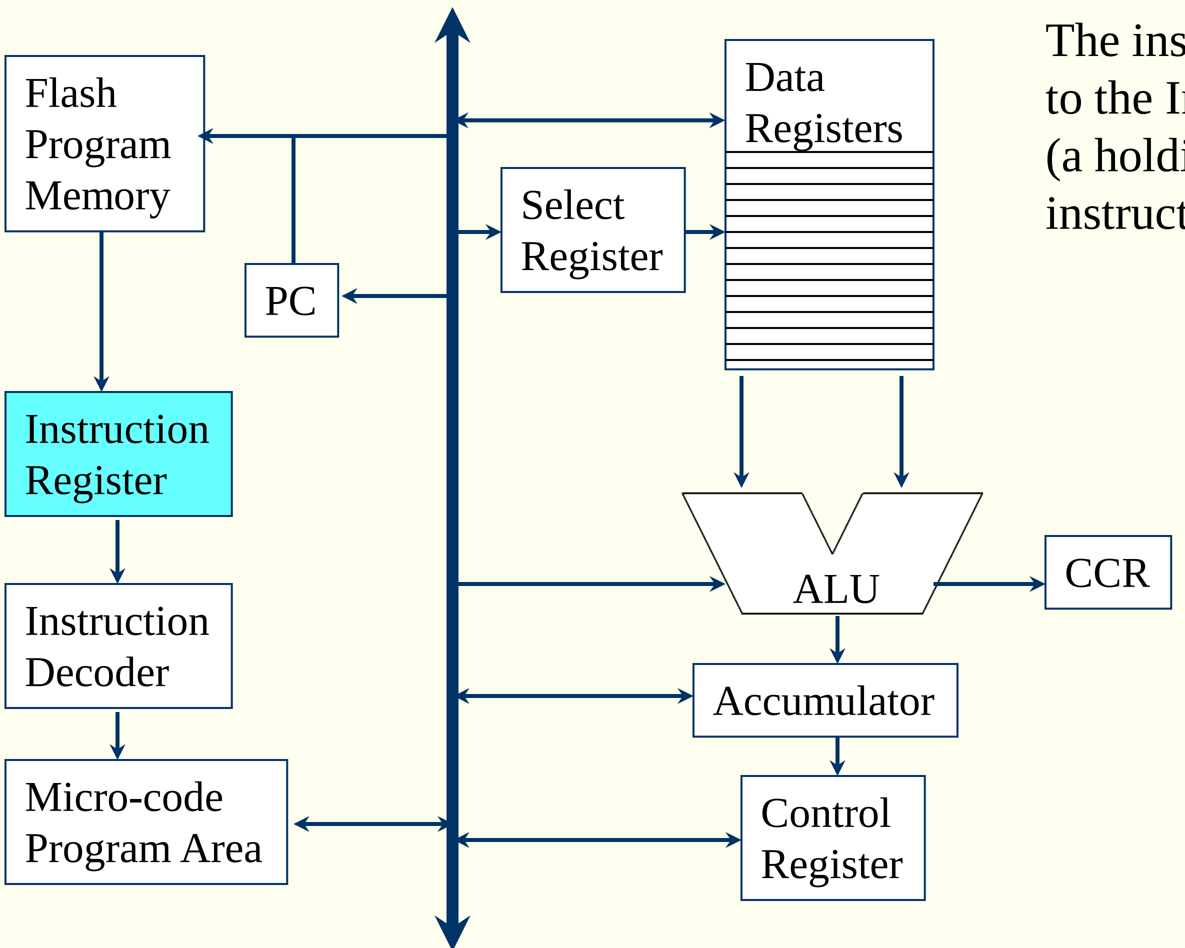


The Control Register instructs the Flash Program Memory to pass the information in that address to the Instruction Register (Remember, this is just a group of 1s and 0s, so it can be called data, although in this case the "data" is an instruction)

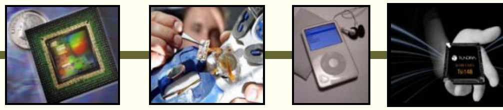
Instruction: COM r2



Executing Instructions

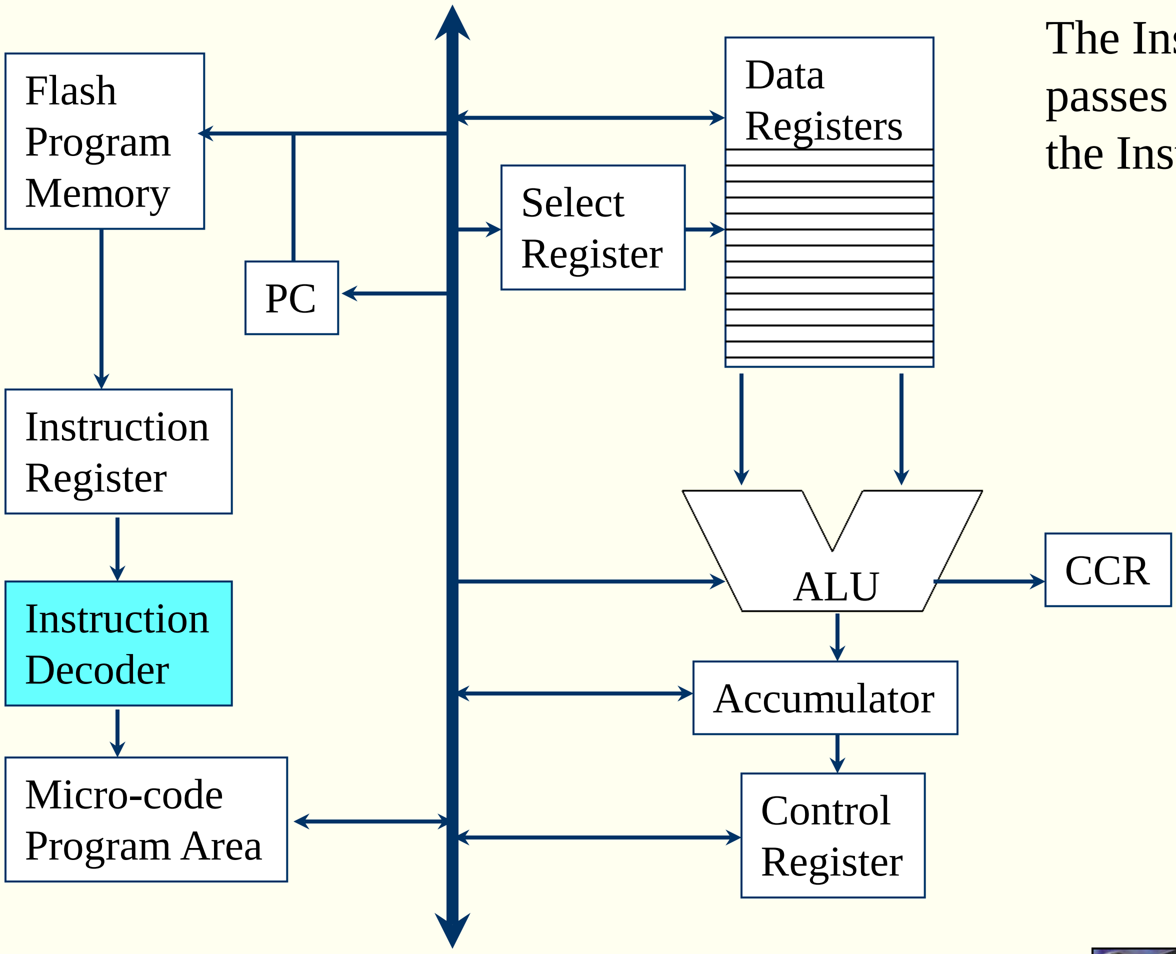


The instruction is passed to the Instruction Register (a holding area for instructions)

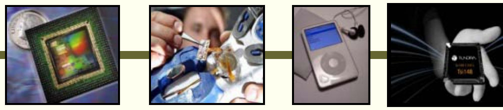


Instruction: COM r2

Executing Instructions

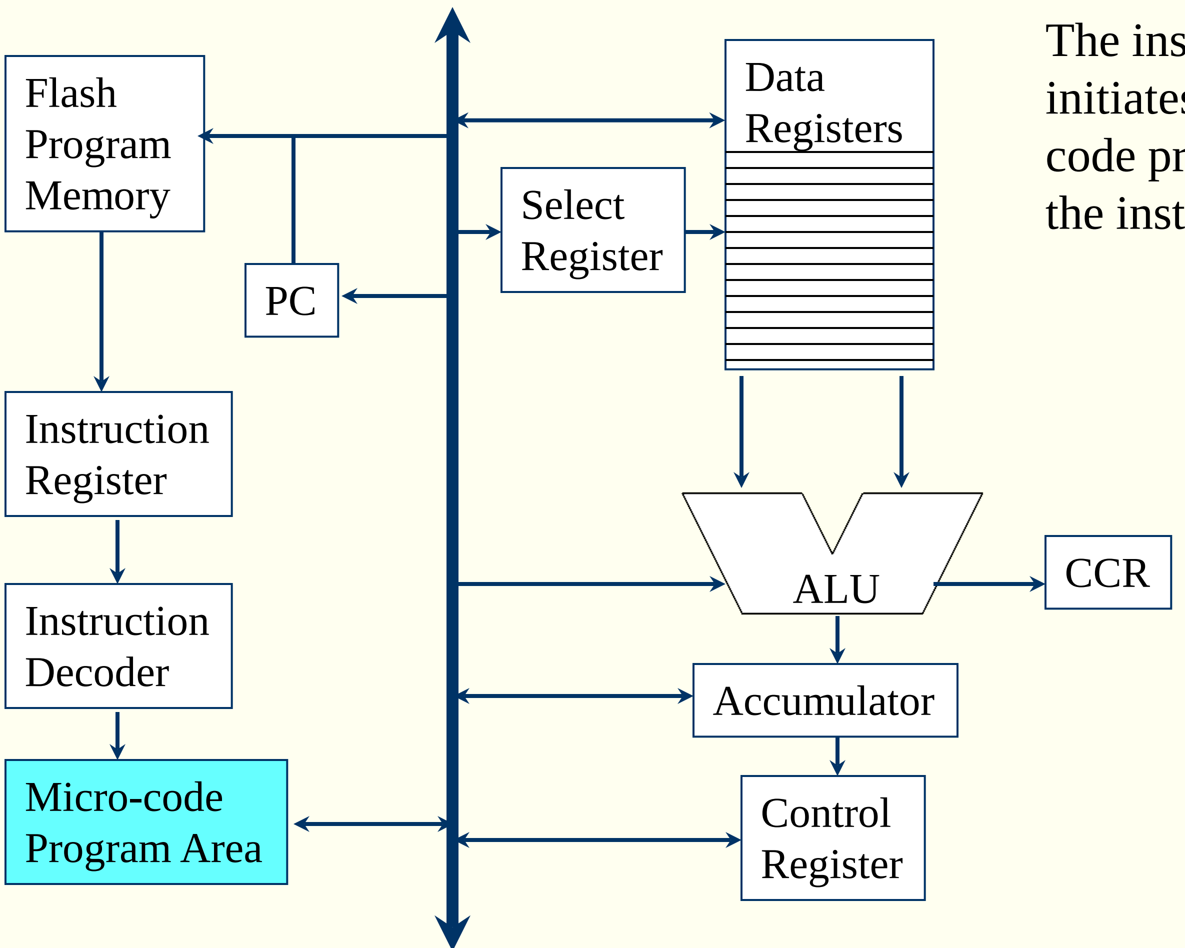


The Instruction Register passes the instruction to the Instruction Decoder



Instruction: COM r2

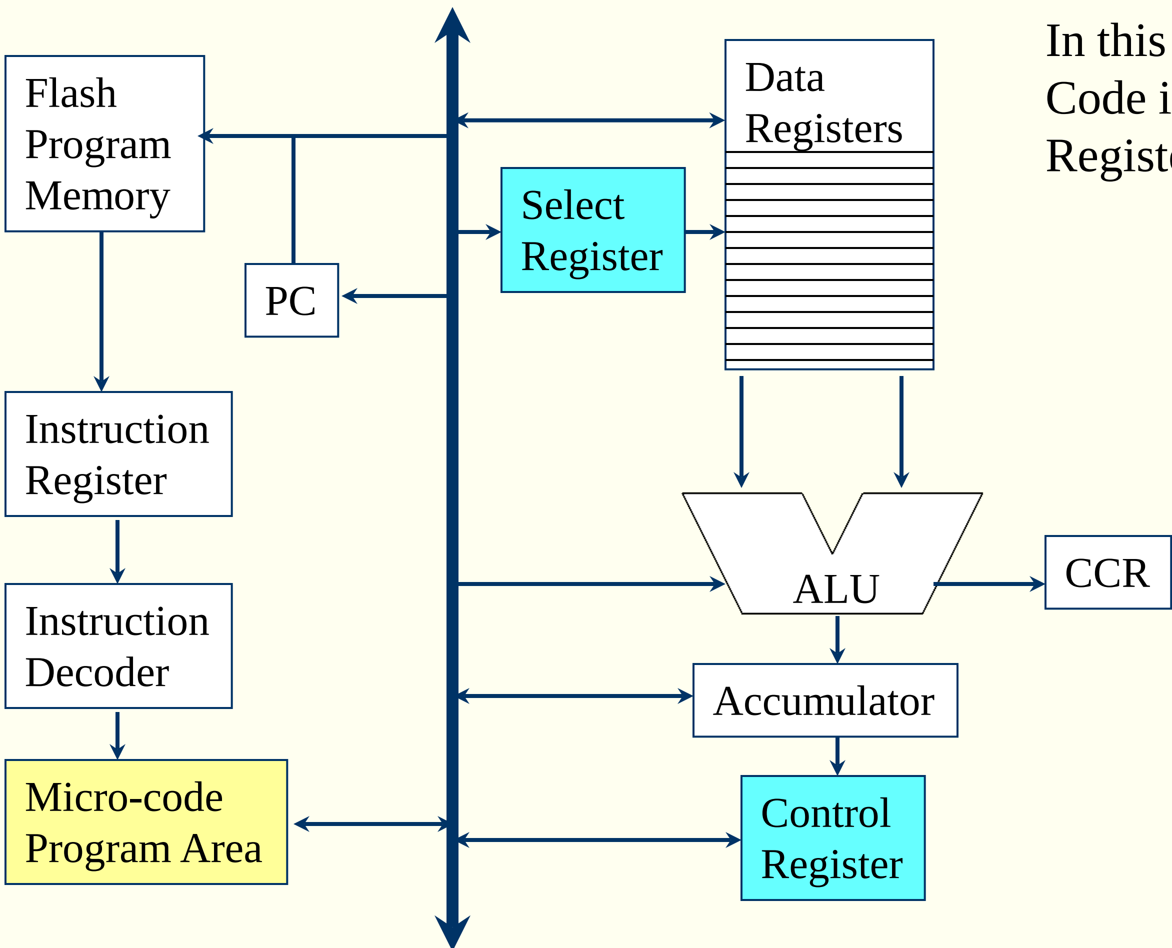
Executing Instructions



The instruction decoder initiates the correct micro-code program to execute the instruction.

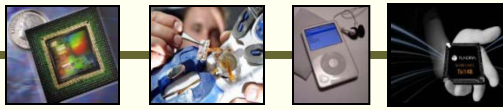
Instruction: COM r2

Executing Instructions

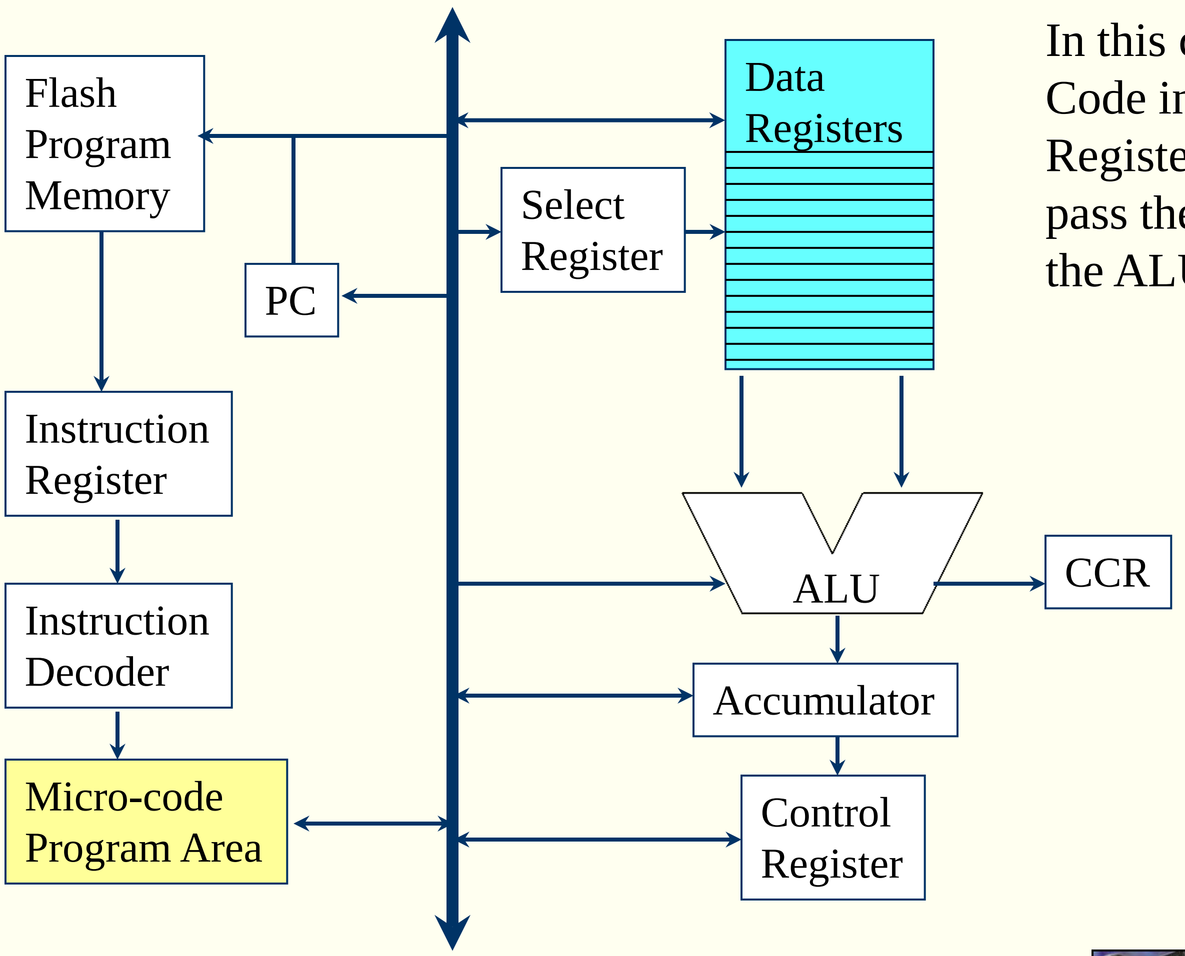


In this case, the Micro-Code instructs the Control Register to select r2 …..

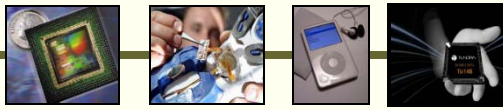
Instruction: COM r2



Executing Instructions

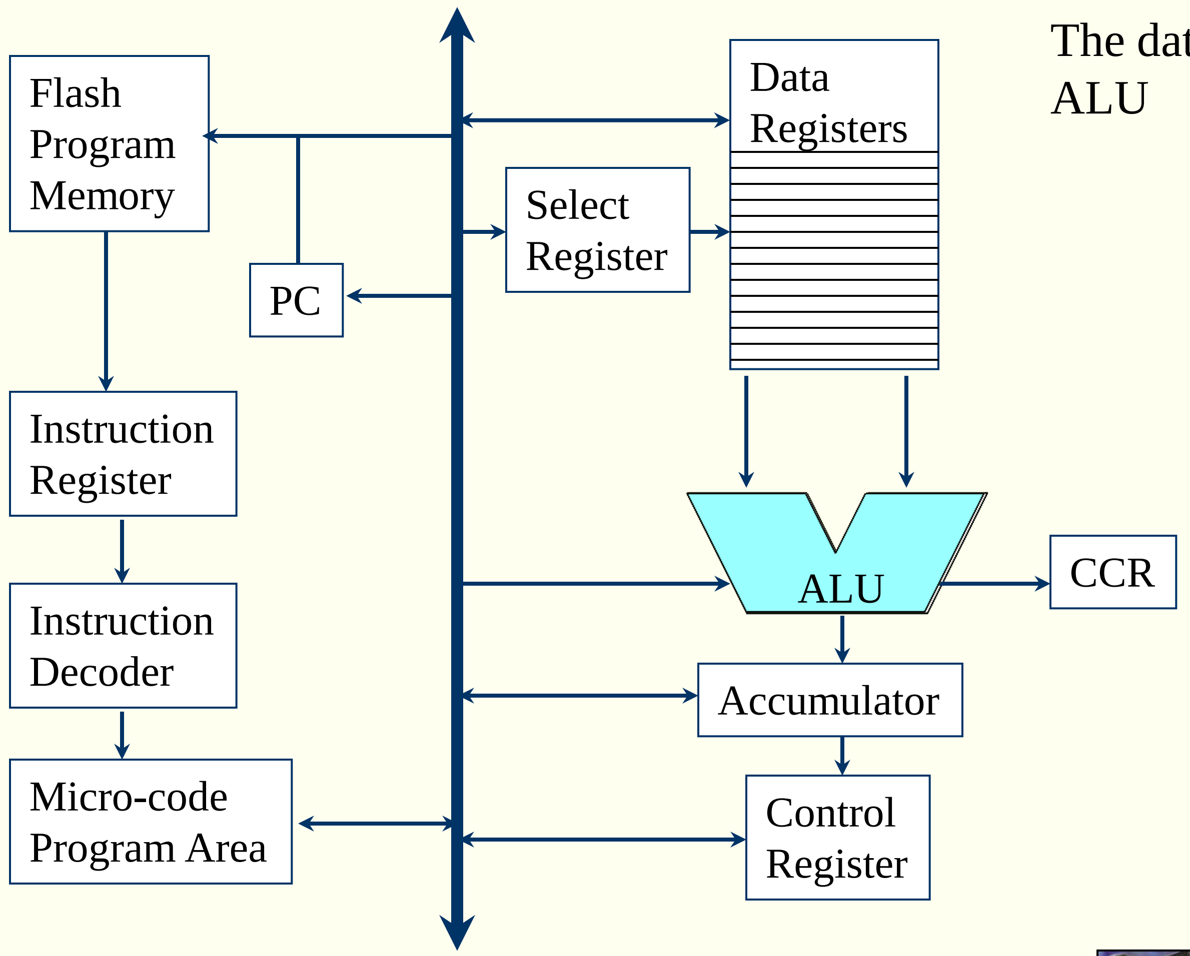


In this case, the Micro-Code instructs the Control Register to select r2 and pass the ***contents*** of r2 to the ALU



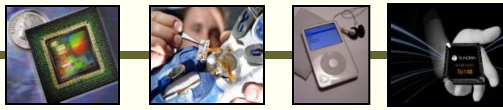
Instruction: COM r2

Executing Instructions

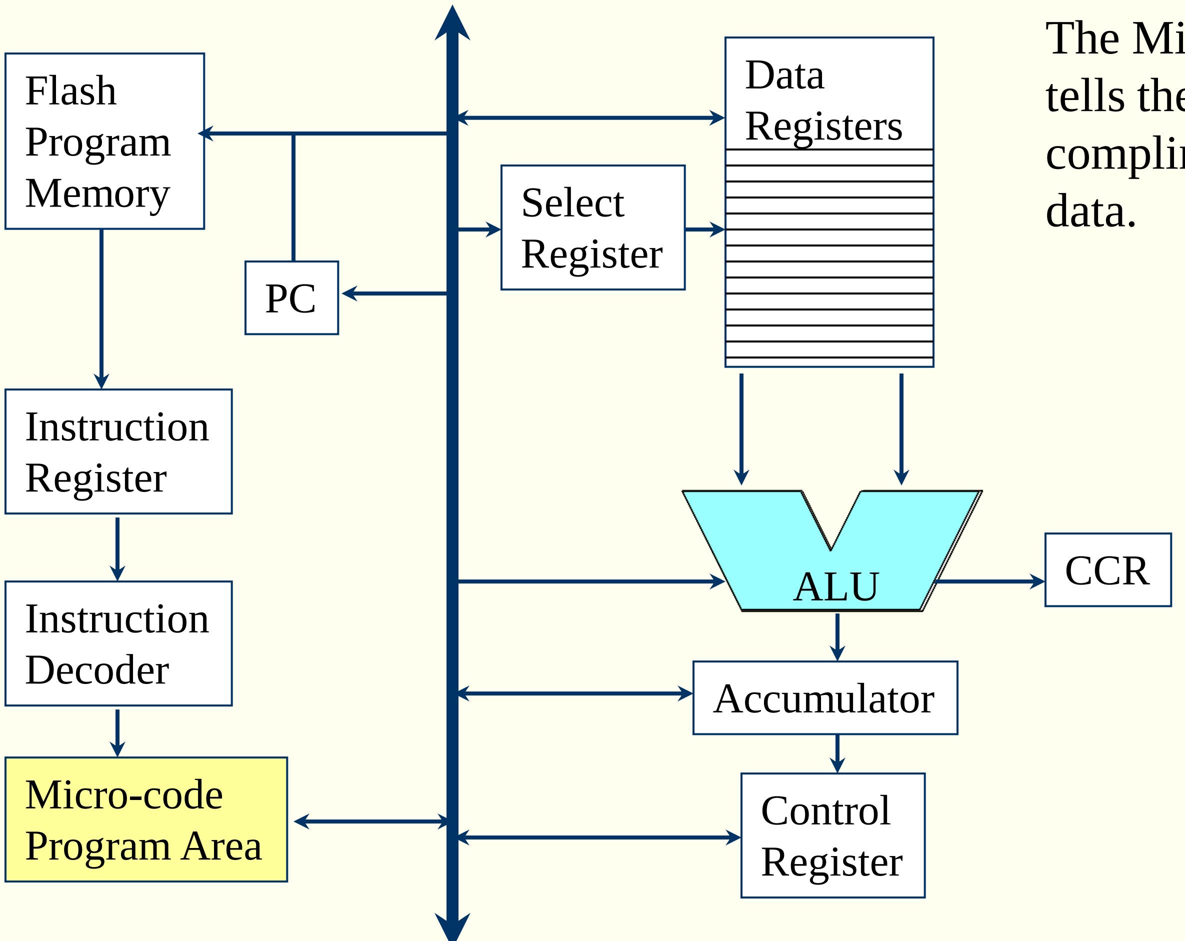


The data is passed to the ALU

Instruction: COM r2

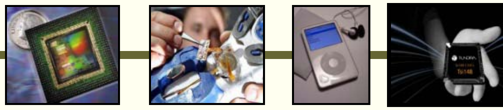


Executing Instructions

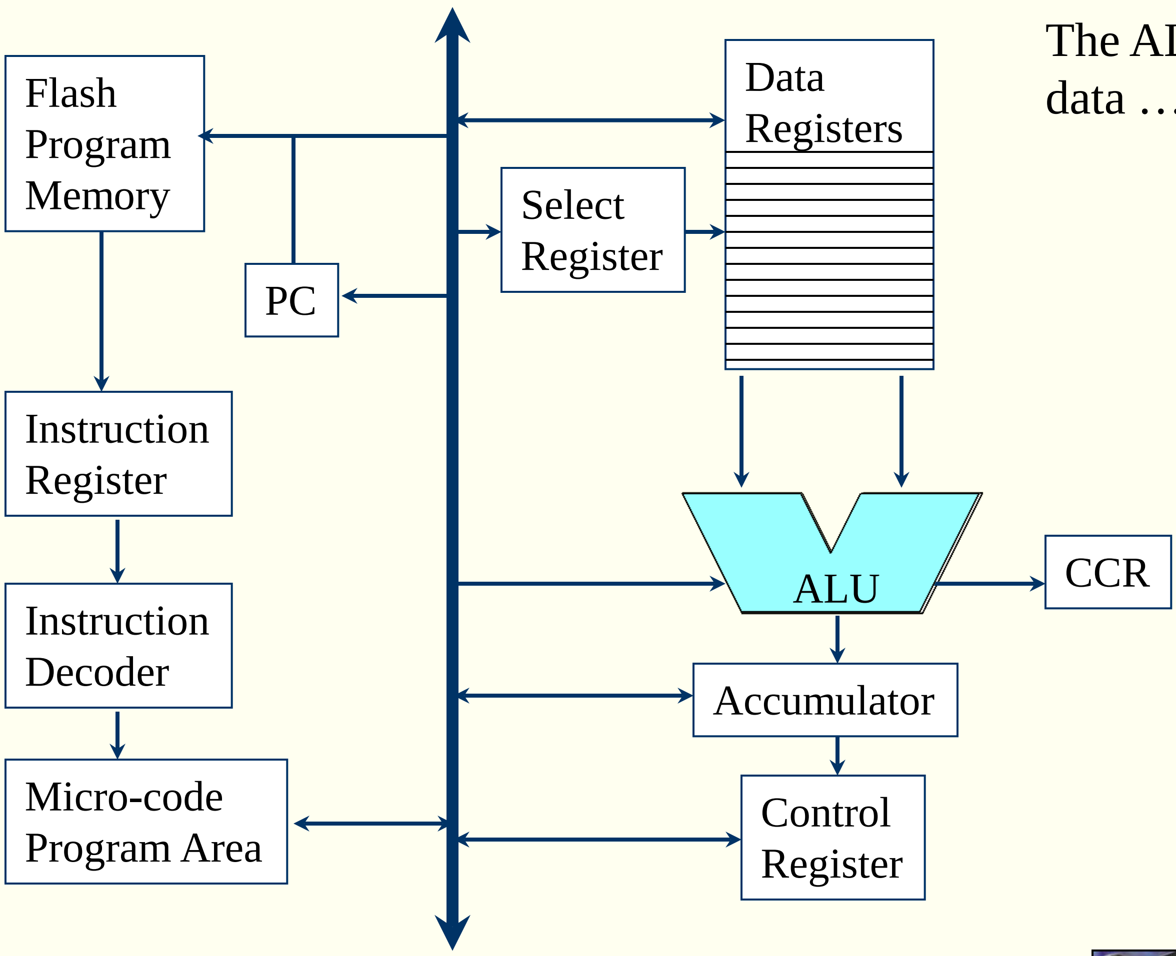


The Micro Code Program tells the ALU to compliment each bit of the data.

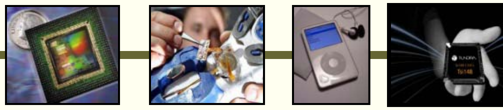
Instruction: COM r2



Executing Instructions

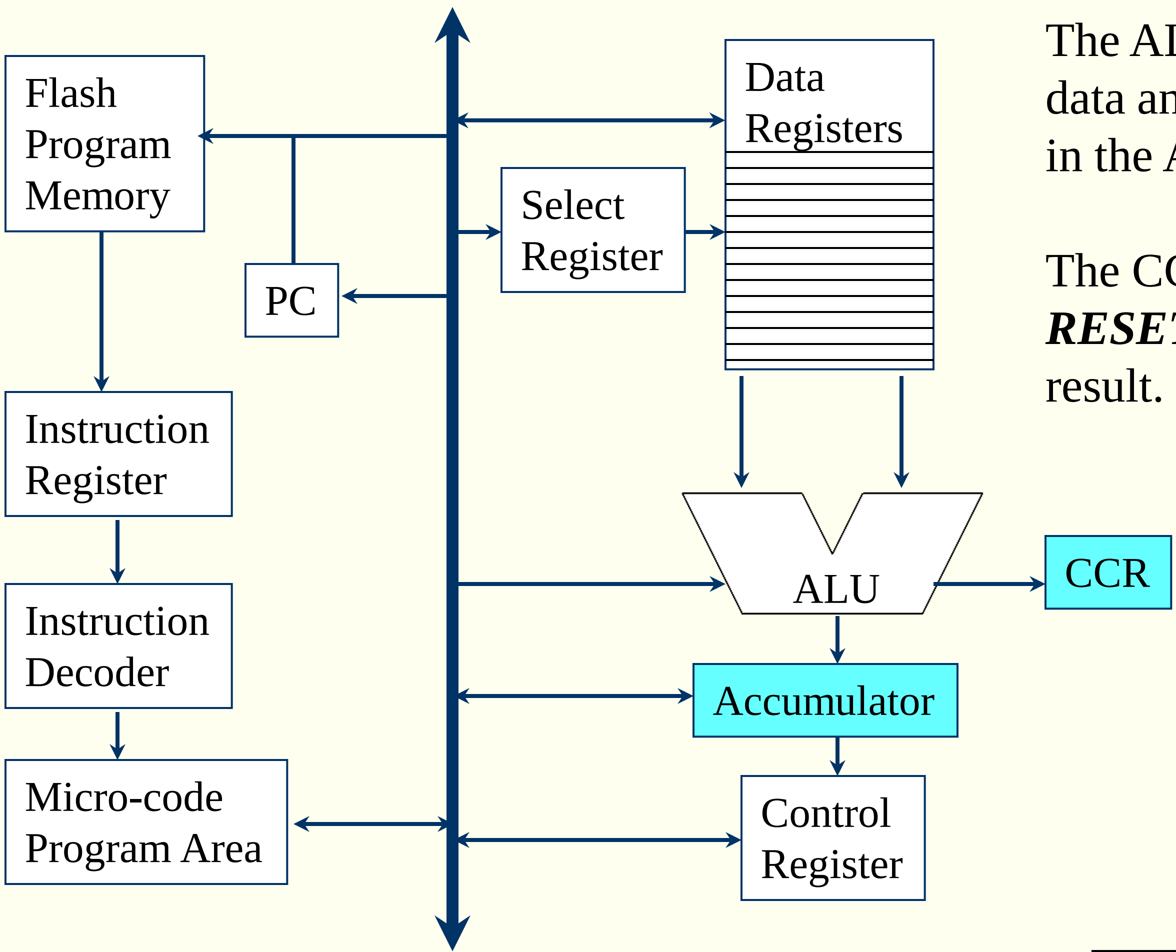


The ALU compliments the data ….



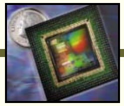
Instruction: COM r2

Executing Instructions



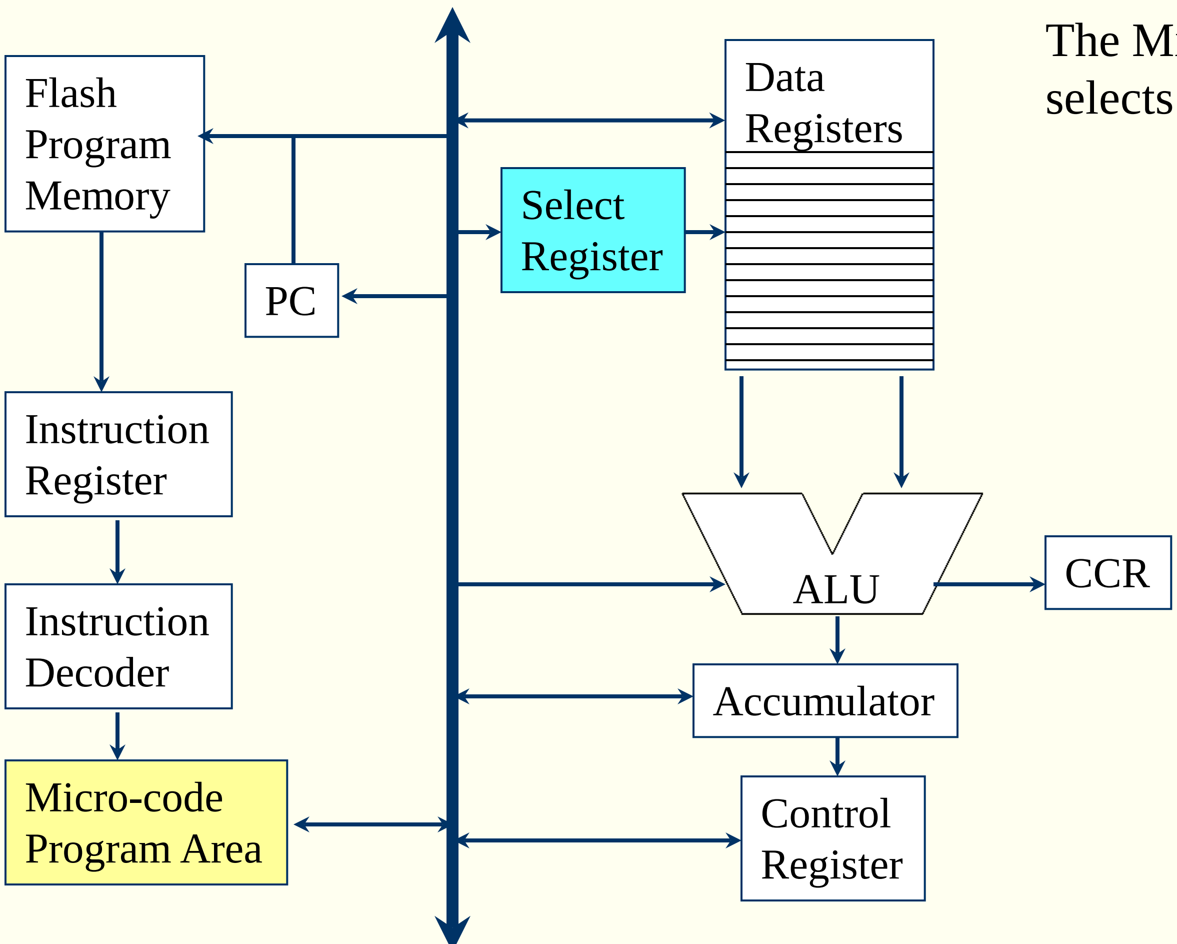
The ALU compliments the data and stores the results in the Accumulator.

The CCR bits are ***SET*** or ***RESET*** according to the result.

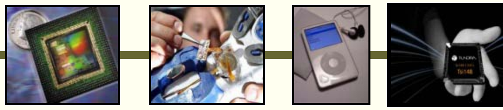



Instruction: COM r2

Executing Instructions

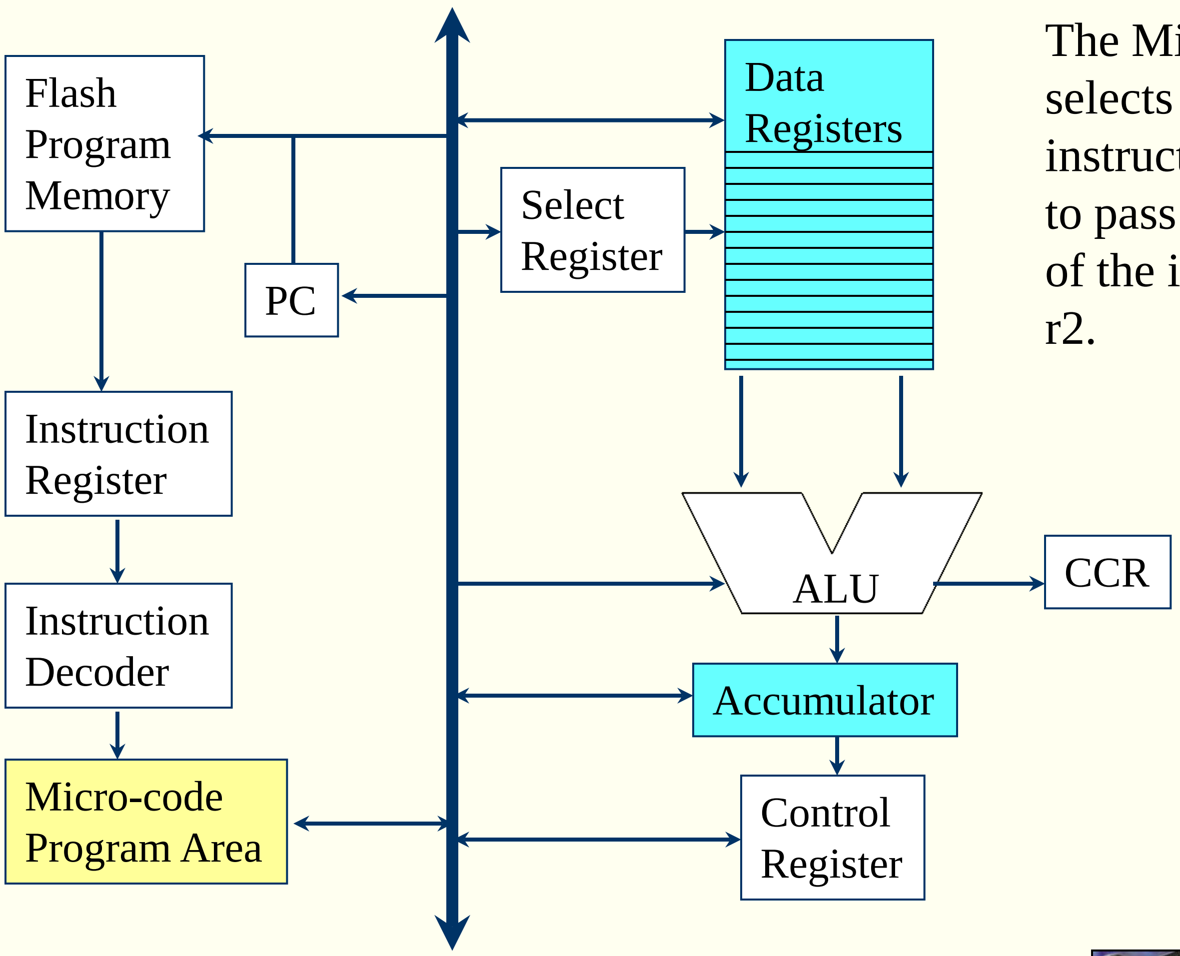


The Micro Code Program selects register r2 …..



Instruction: COM r2

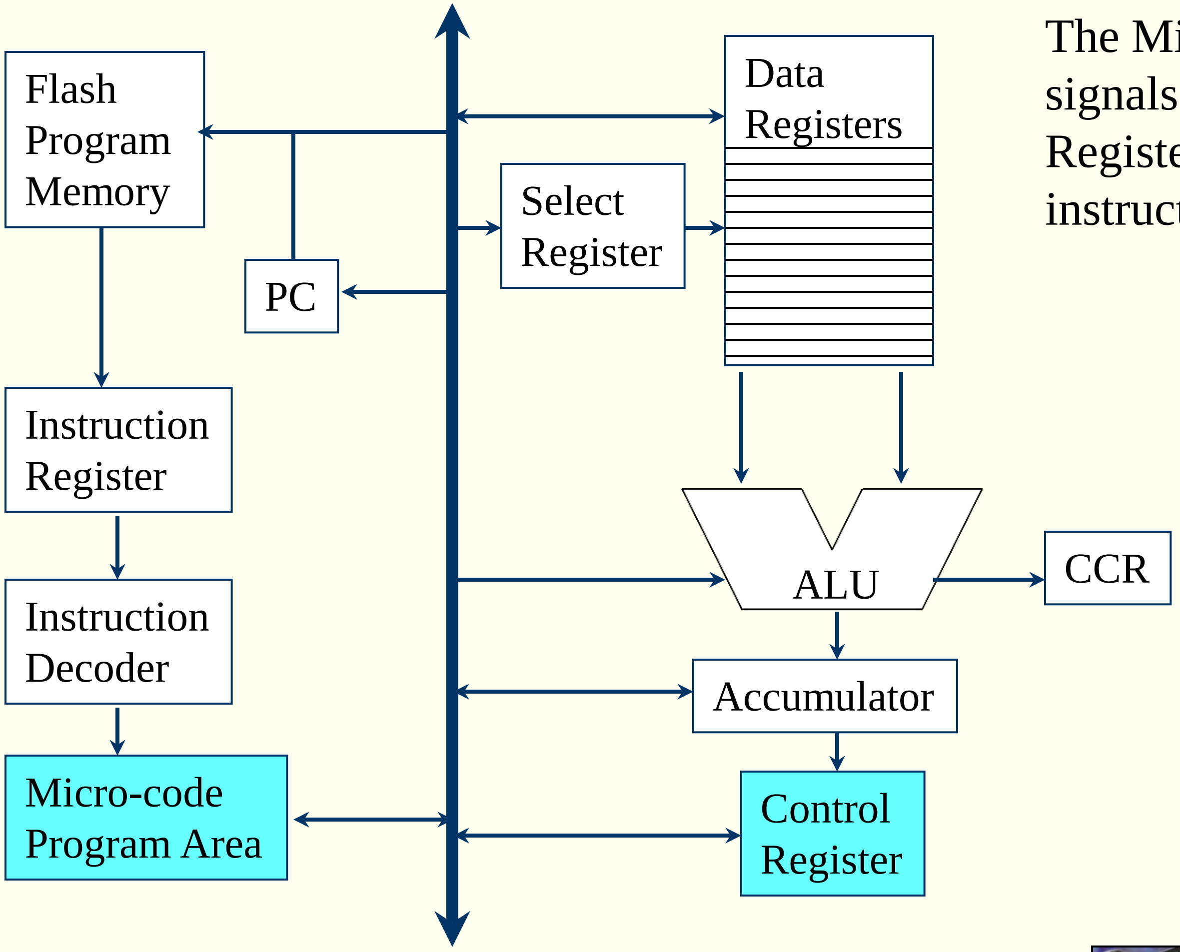
Executing Instructions



The Micro Code Program selects register r2 and instructs the Accumulator to pass the data (the results of the instruction) back to r2.

Instruction: COM r2

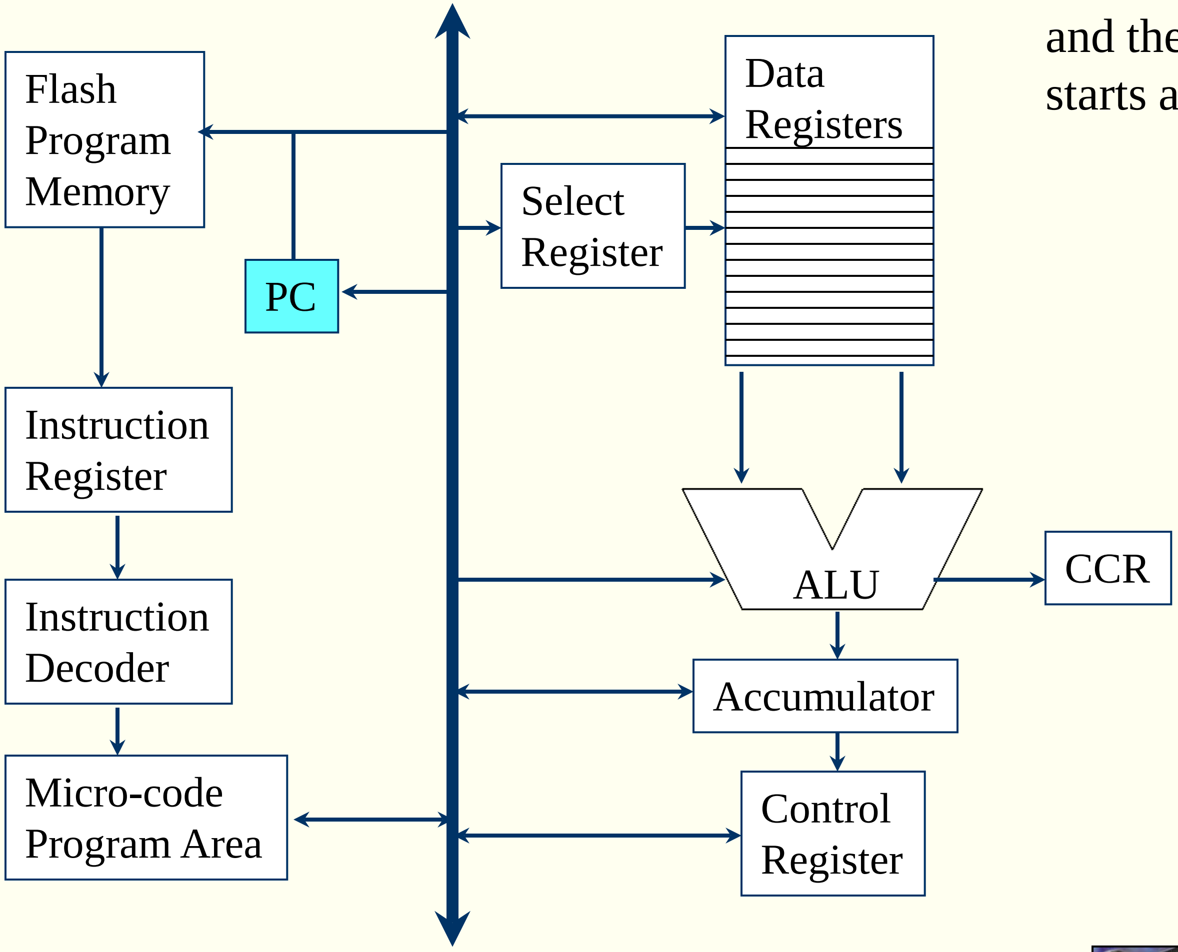
Executing Instructions



The Micro Code Program signals the Control Register that the instruction is complete

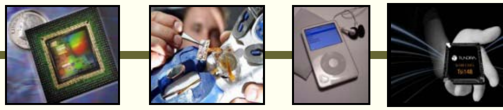
Instruction: COM r2

Executing Instructions



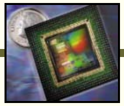
and the whole process starts again ….

Instruction: COM r2



Executing Instructions

- This sequence of events is essentially the same for every instruction
  - Sometimes more than 1 register is selected
  - Sometimes data is retrieved from “data memory” instead of a register and then passed to the ALU
  - Sometimes the results are stored in memory or an I/O register




Register Direct

- Register Direct (two operands):
  - Instructions can operate on any of the 32 registers
    - One of these registers is the ***source*** register (Rs) and one is the ***destination*** register (Rd)
      - » Relative to the data
  - The microcontroller:
    - Reads the data in the registers
    - Operates on the data in the registers
    - Stores the results in the destination register

Register Direct

- Format:

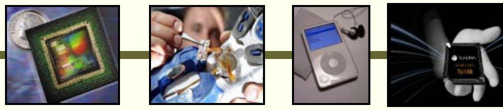
```
label:  mnemonic  destination_reg, source_reg    comment
```

- Examples

```
        add  r2,r5      ; add the contents of the 2 registers
```

```
        and  r6,r1      ; logically AND the contents of the 2 registers
```

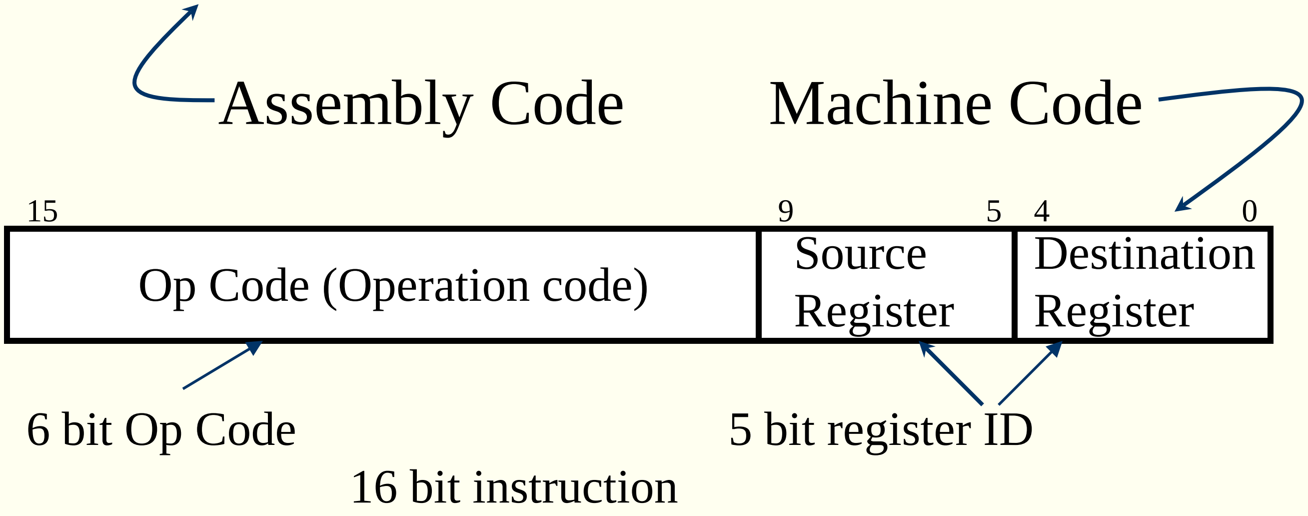
```
        mov  r14,r15    ; move the contents of r15 to r14
```

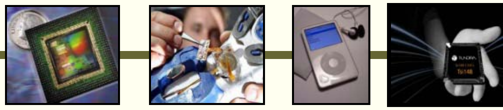


Register Direct

- Format:

label: mnemonic destination_reg, source_reg comment





Register Direct

- Assume:
  - (r2) = 10 (contents of r2 is 10)
  - (r5) = 99
- What is in r2 and r5 after the following instruction?

```
add  r2,r5    ; add the contents of the 2 registers
```

In-Class Exercise

Register Direct

- Assume:
  - (r2) = 10 (contents of r2 is 10)
  - (r5) = 99
- What is in r2 and r5 after the following instruction?

```
add  r2,r5    ; add the contents of the 2 registers
```

r2 is the destination

(r2) = A9
(r5) = 99

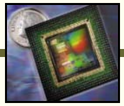
Both are hex numbers (by convention)

Register Direct

- Assume:
  - (r6) = 0F
  - (r1) = F0
- What is in r6 and r1 after the following instruction?

```
and  r6,r1    ;logically AND the contents of the 2 registers
```

In-Class Exercise




Register Direct

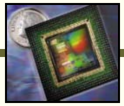
- Assume:
    - (r6) = 0F
    - (r1) = F0
- What is in r6 and r1 after the following instruction?

```
and  r6,r1    ;logically AND the contents of the 2 registers
```

(r6) = 00
(r1) = F0

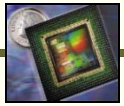
Each bit is ANDed 00001111
<u>11110000</u>
00000000

r6 is the destination




I/O Direct

- I/O Direct:
  - Used to access I/O space (I/O registers and ports)
  - I/O registers may only be accessed with two instructions:
    - IN: for reading data from an input port: PINx
    - OUT: for sending data out the output port: PORTx



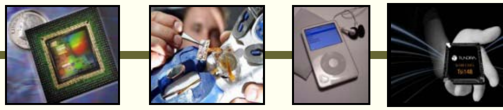

I/O Direct

- Format:

```
label:  IN rd,Port_Address    comment
```

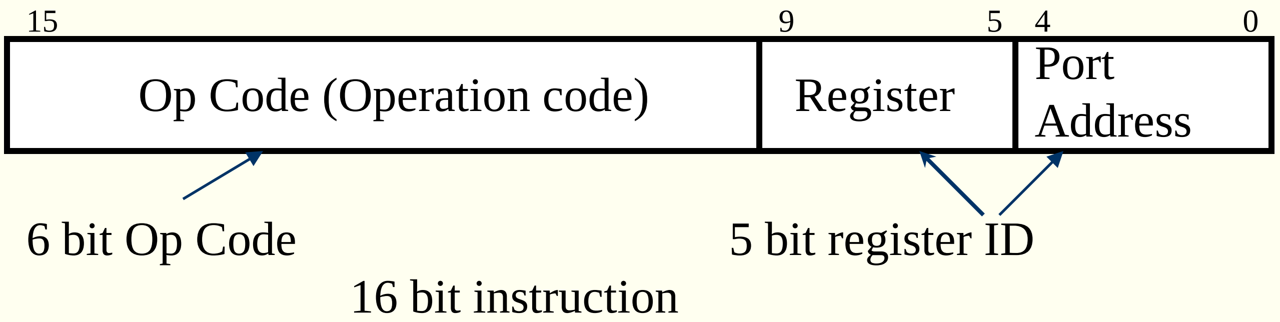
```
label:  OUT Port_Address,rs    comment
```

- Registers:
  - rd and rs can be any of the 32 registers
  - rd is a destination register when data is read from a port
    - Input ports are referred to as PIN (e.g.; PinA, PinB, PinC, etc)
  - rs is the source register when data is being sent out
- I/O Registers
  - Can be any of the I/O registers



I/O Direct

- Format:

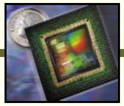


I/O Direct

- Assume:
  - (r2) = 1E
  - (PortB) has all input pins tied to a high voltage (3.5 – 5.5 v)
- What is in r2 after the following instruction?

```
Read:      in  r2,PinB     ; read the contents of PortB
```

In-Class Exercise

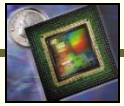



I/O Direct

- Assume:
  - (r2) = 1E
  - (PortB) has all input pins tied to a high voltage (3.5 – 5.5 v)
- What is in r2 after the following instruction?

```
Read:     in  r2,PinB     ; read the contents of PortB
```

(r2) = FF   (all inputs have a logic 1 on them)




I/O Direct

- Assume:
  - (PortC) = C5
  - (r1) = F0
- What data on the output port? What voltage values are on the output pins after the following instruction?

```
out   PortC,r1    ;send the contents of r1 out Port C
```

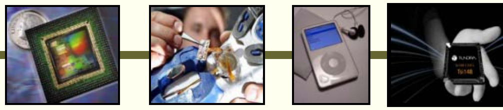
In-Class Exercise

I/O Direct

- Assume:
  - (PortC) = C5
  - (r1) = F0
- What data is on the output port? What voltage values are on the output pins after the following instruction?

```
out   PortC,r1    ;send the contents of r1 out Port C
```

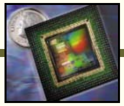
(PortC) = F0 which means bits 4, 5, 6, & 7 are high (5v) and bits 0, 1, 2, & 3 are low (0v)



Immediate

- Immediate:
  - The destination operand is one of the 32 registers
  - The source operand is immediate data
    - The actual data that will be used in the instruction
    - Immediate mode is denoted by an “i” in the mnemonic
      - » Example   ldi r2,0x62   ; load hex 62 into register r2

label: mnemonic destination_reg,data   comment

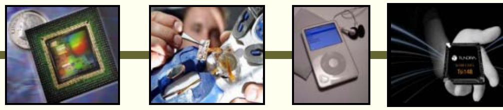



Immediate

- Assume:
  - (r17) = 1E
- What is in r17 after the following instruction?

```
Read:     ldi  r17,10    ; put 10 in r 17
```

In-Class Exercise



Immediate

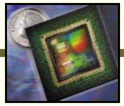
- Assume:
  - (r17) = 1E
- What is in r17 after the following instruction?

```
Read:      ldi  r17,10    ; put 10 in r 17
```

(r17) = 0A

Assemblers usually read numbers as decimal numbers unless the programmer tells it otherwise. In this case the instruction would have been:
  ldi r17,0x0A to specify a hex number

The contents of a register is a **binary** number. Hex may be thought of as shorthand notation for binary. Therefore, I will also specify the contents of a register, port or memory location as a hex or binary number.




Data Direct

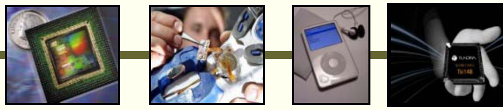
- Data Direct:
  - Instructions are two word (16-bit)
  - One of the operands is the address of the data (address of where the data is stored)
  - The other operand is a register

```
label:  mnemonic  destination_reg, address_of_data    comment
```

Moves data from memory to a register

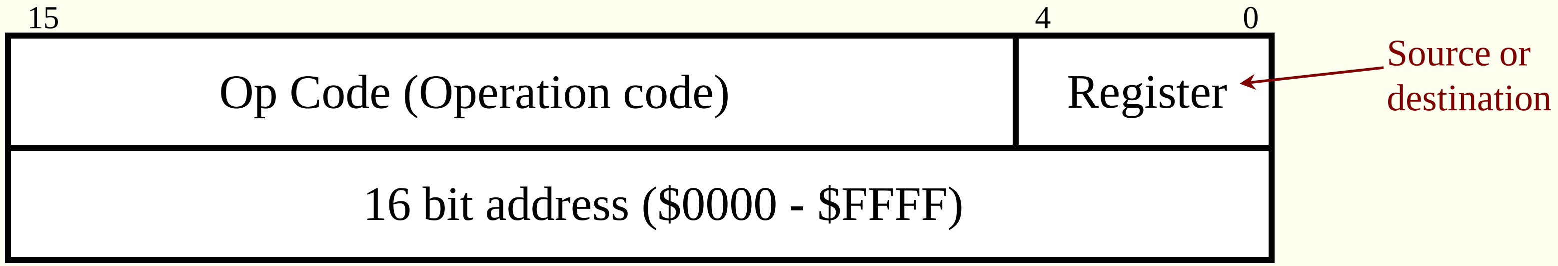
```
label:  mnemonic address_of_data, source_reg    comment
```

Moves data from a register to memory



Data Direct

- Data Direct:
  - Instructions are two word (16-bit)
  - One of the operands is the address of the data (address of where the data is stored)
  - The other operand is a register



Data Direct

- Assume:
  - (r15) = C5
  - (r1) = F0

Memory

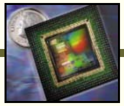
Addresses — Data stored at those addresses

3FFE	F3
3FFF	1A
4000	32
4001	BE

- What is the data in r1 and r15 after the following instruction?

```
lds r1,0x4000
sts 0x3ffe,r15
```

In-Class Exercise




Data Direct

- Assume:
  - (r15) = C5
  - (r1) = F0

Memory

Addresses

3FFE	C5
3FFF	1A
4000	32
4001	BE

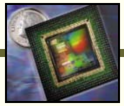
Data stored at those addresses
These are also binary or hex numbers

- What is the data in r1 and r15 after the following instruction?

```
lds r1,0x4000
sts 0x3ffe,r15
```

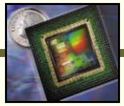
(r1) = 32

(r15) = C5, but the value in address $3FFE changed to $C5




Data Indirect

- Data Indirect:
  - Very similar to Data Direct
    - In Data Direct, one of the operands is an explicitly specified address (to store or retrieve data)
    - In Data Indirect, the address is specified as the contents of the X, Y, or Z register
      - X is the combination of r26 & r27
      - Y is the combination of r28 & r29
      - Z is the combination of r30 & r31
      - X, Y, or Z are referred to as the “pointer register”




Data Indirect

- Format:

label: mnemonic destination_reg, X comment

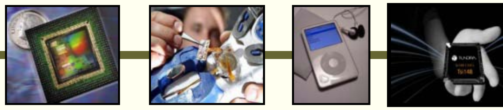
Moves data from memory to a register

label: mnemonic X, source_reg comment

Moves data from a register to memory

In the formats shown above, X could be register X, Y, or Z

X =	r27: High Byte of address	r26: Low Byte of address
Y =	r29: High Byte of address	r28: Low Byte of address
Z =	r31: High Byte of address	r30: Low Byte of address



Data Indirect

- Example:
  - Assume (r27) = 40 and (r26) = 00

```
ld   r0,X          ; get data from $4000
```

  - If (X) = $4000, then the instruction is equivalent to:

```
ld   r0,$4000    ; get data from $4000
```

  - and ($4000) = 32
  - So, after this instruction is executed (r0) = 32

Memory

Address	Data
3FFE	F3
3FFF	1A
4000	32
4001	BE

Data Indirect

- Assume:
  - (r0) = 41 (hex number)
  - (r26) = FE
  - (r27) = 3F
- What is the data in r0 after the following instruction?

Memory

Address	Data
3FFE	F3
3FFF	1A
4000	32
4001	BE

```
lds r0,X
```

In-Class Exercise

Data Indirect

- Assume:
  - (r0) = 41 (hex number)
  - (r26) = FE
  - (r27) = 3F

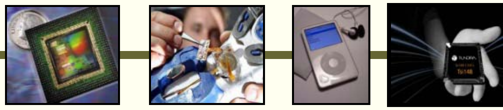
Memory

Address	Data
3FFE	F3
3FFF	1A
4000	32
4001	BE

- What is the data in r0 after the following instruction?

```
lds r0,X
```

(X) = 3FFE (r27 is the HB, r26 is the LB)
(3FFE) = F3
So, (r0) = F3



Data Indirect

- Assume:
  - (r0) = 41 (hex number)
  - (r30) = FF
  - (r31) = 3F
- Which register (X, Y, or Z) is specified by r30 & r31? What is the data in r0 after the following instruction?

```
lds r0,?
```

Memory

Address	Data
3FFE	F3
3FFF	1A
4000	32
4001	BE

In-Class Exercise

Data Indirect

- Assume:
  - (r0) = 41 (hex number)
  - (r30) = FF
  - (r31) = 3F

Memory

Address	Data
3FFE	F3
3FFF	1A
4000	32
4001	BE

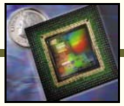
- Which register (X, Y, or Z) is specified by r30 & r31? What is the data in r0 after the following instruction?

```
lds r0,Z
```

Z is specified by r30 & r31

(Z) = 3FFF and (3FFF) = 1A

So, (r0) = 1A



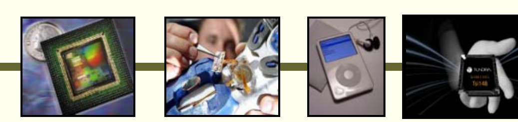

Data Indirect

- Assume:
  - (r0) = 41 (hex number)
  - (r28) = 01
  - (r29) = 40
- Write the instruction that would store the contents of r0 into $4001 using Data Indirect

Memory

Address	Data
3FFE	F3
3FFF	1A
4000	32
4001	BE

In-Class Exercise



Data Indirect

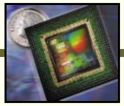
- Assume:
  - (r0) = 41 (hex number)
  - (r28) = 01
  - (r29) = 40

Memory

Address	Data
3FFE	F3
3FFF	1A
4000	32
4001	BE

- Write the instruction that would store the contents of r0 into $4001 using Data Indirect

```
st  Y,r0
```




Data Indirect

- The pointer register (X, Y, or Z) may also:
  - Have a post-increment
  - Have a pre-decrement
  - Y and Z can have an offset added to them
- Post-Increment Example

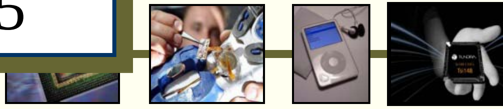
```
ld r0,X+
```
or
```
st X+,r15
```

- Pre-Decrement Example

```
ld r0,-X
```
or
```
st -X,r15
```

- Offset Example

```
ldd r0,Z+0x10
```
or
```
std Y+20,r15
```



Indirect Program Addressing

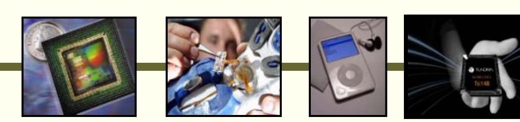
- The Z register is used as a pointer
  - To Program Memory
  - Up to 64k (16 bit register)
  - Used for Indirect Jumps or Subroutine Calls
- Example

```
ijmp
```

or

```
icall
```

- Z has to be loaded with the correct target address before the instruction is executed



Relative Program Addressing

- The current PC is used as a pointer
  - Can have a + or – 2k offset from the current PC
  - Used for Relative Jumps or Subroutine Calls
- Example

rjmp or rcall

Summary

- In this topic we:
  - Examined the addressing modes of the AVR
    - Register Direct
      - Single Register
      - Two Registers
    - I/O Direct
    - Immediate
    - Data Direct
    - Data Indirect
    - Indirect Program Addressing
    - Relative Program Addressing
  - Examined some simple instructions of the AVR