Grupo ARCOS

uc3m | Universidad Carlos III de Madrid

Lesson 3 (I)
Fundamentals of assembler programming

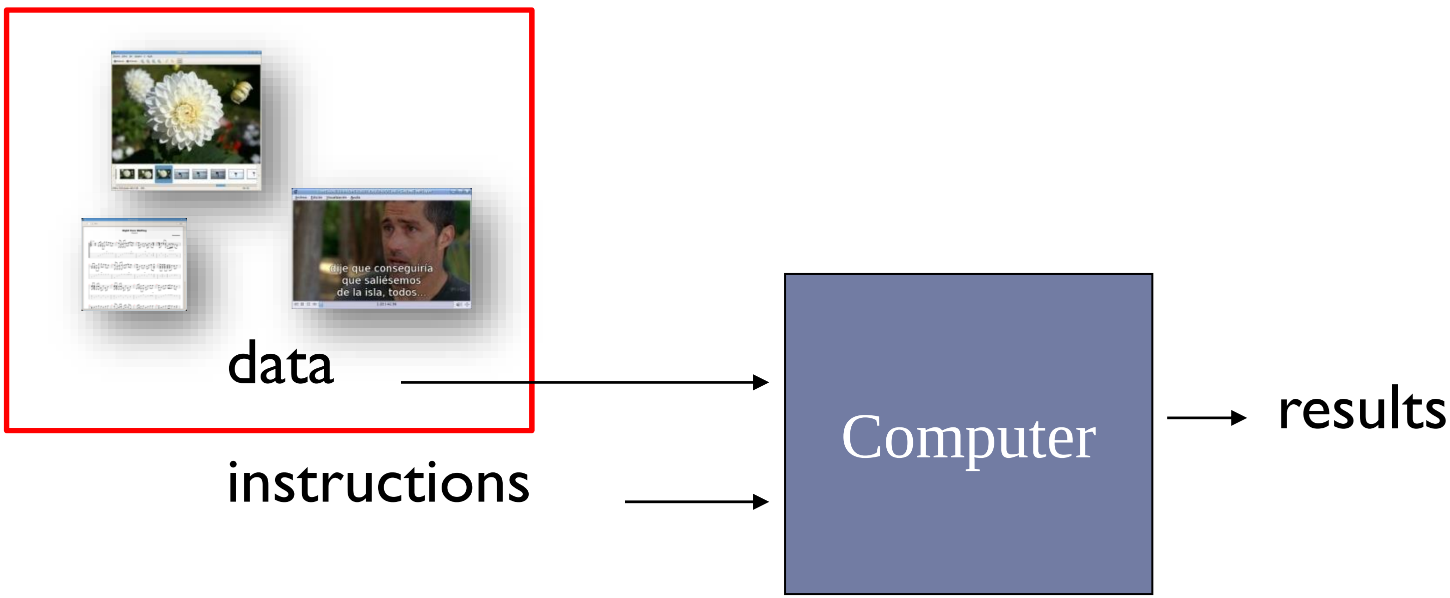
Computer Structure

Bachelor in Computer Science and Engineering

Contents

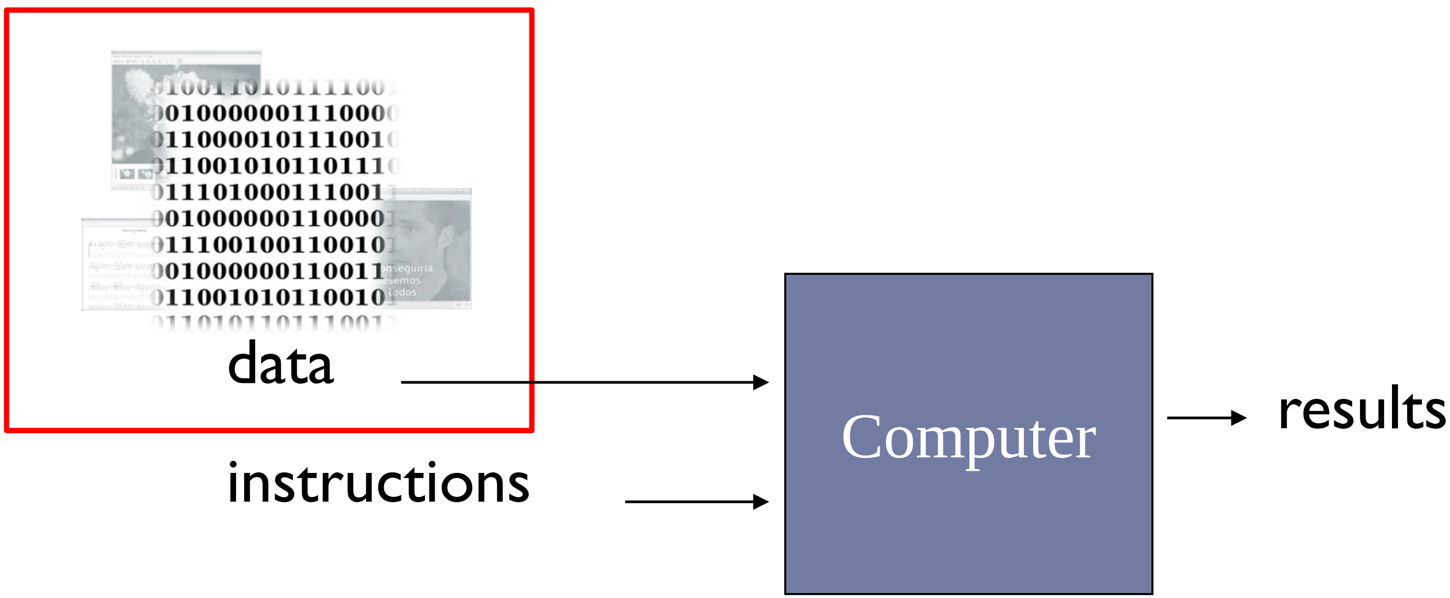
Types of information: instructions and data

- Data representation…



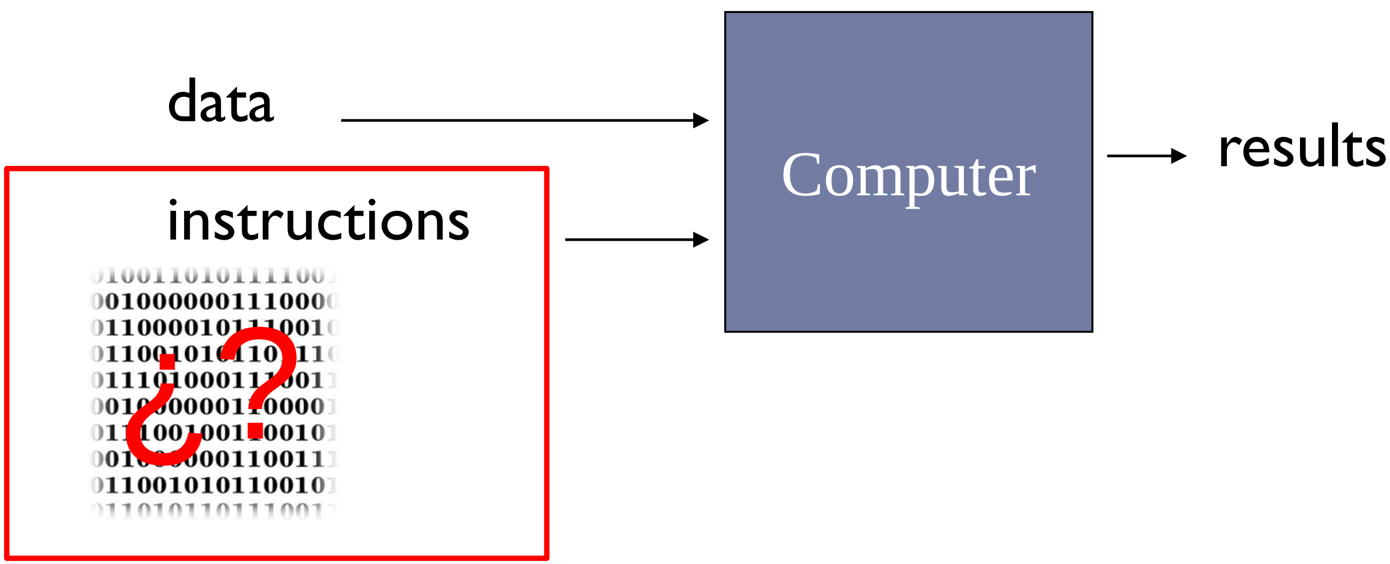
Types of information: instructions and data

- **Binary data** representation.



Types of information: instructions and data

- What about the instructions?



Machine instruction

- Machine instruction: elementary operation that can be executed directly by the processor.
- Example of instruction in MIPS:
  - Sum of a register (s) with an immediate value (i) and the result of the sum is stored in register (t).

001000	sssss	ttttt	iiii iiii iiii iiii

Operation code
addi

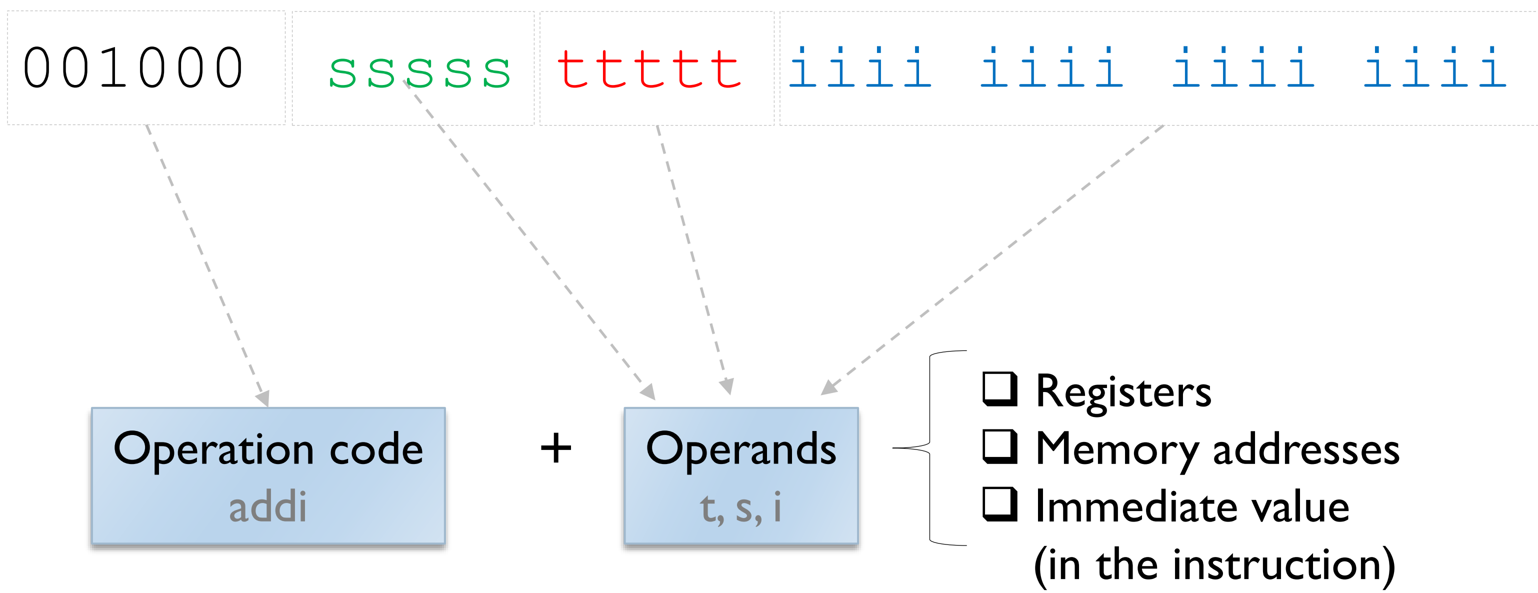
+

Operands
t, s, i

- ❑ Registers
- ❑ Memory addresses
- ❑ Immediate value (in the instruction)

Properties of machine instructions

- Perform a single, simple task
- Operate on a fixed number of operands
- Include all the information necessary for its execution



Information contained in a machine instruction

- The operation to be performed.
- Where the operands are located:
  - In registers
  - In memory
  - In the instruction itself (immediate)
- Where to leave the results (as operand)
- A reference to the next instruction to be executed
  - Implicitly: the following instruction
    - A program is a consecutive sequence of machine instructions.
  - Explicitly in branching instructions (as operand)

Operation code
addi

+

Operands
t, s, i

- ❑ Registers
- ❑ Memory addresses
- ❑ Immediate value (in the instruction)

Programming model of a computer

- A computer offers a programming model that consists of:
  - Instruction set (assembly language)
    - ISA: *Instruction Set Architecture*
    - An instruction includes:
      - Operation code
      - Other elements: registers, memory address, numbers
  - Storing elements
    - Registers
    - Memory
    - Registers of I/O controllers
  - Execution modes

Instruction sets

- Instruction Set Architecture (ISA)
  - Instruction set of a processor
  - Boundary between hardware and software
- Examples:
  - 80x86
  - ARM
  - MIPS
  - RISC-V
  - PowerPC
  - Etc.

Characteristics of an instruction set

- Operands:
  - Registers, memory, the instruction itself
- Memory addressing:
  - Most of them use byte addressing
  - They provide instructions for accessing multi-byte elements from a given position
- Addressing modes:
  - They specify where and how to access operands (register, memory or the instruction itself)
- Type and size of operands:
  - bytes: 8 bits
  - integers: 16, 32, 64 bits
  - floating-point numbers: single precision, double precision, etc.

Characteristics of an instruction set

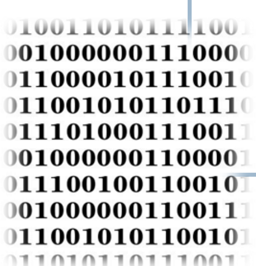
- Operations:
  - Arithmetic, logic, transfer, control, control, etc.
- Flow control instructions:
  - Unconditional jumps
  - Conditional jumps
  - Procedure calls
- Format and coding of the instruction set:
  - Fixed or variable length instructions
    - 80x86: variable (from 1 up to 18 bytes)
    - MIPS, ARM: fixed

Definition of program

- **Program**: Ordered sequence of machine instructions that are executed by default in order.

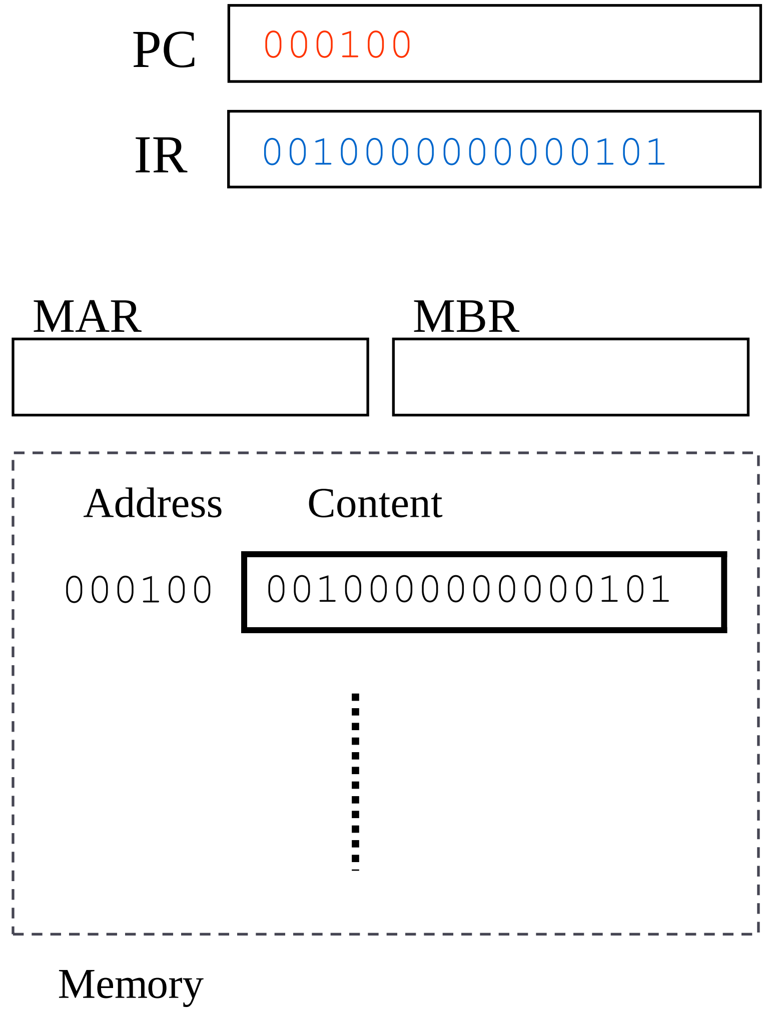
```
00001001110001101010111101011000
10101111010110000000100111000110
11000110101011110101100000001001
01011000000010011100011010101111
```

```
add   t k 1
mul   k * 2
move  k t
```



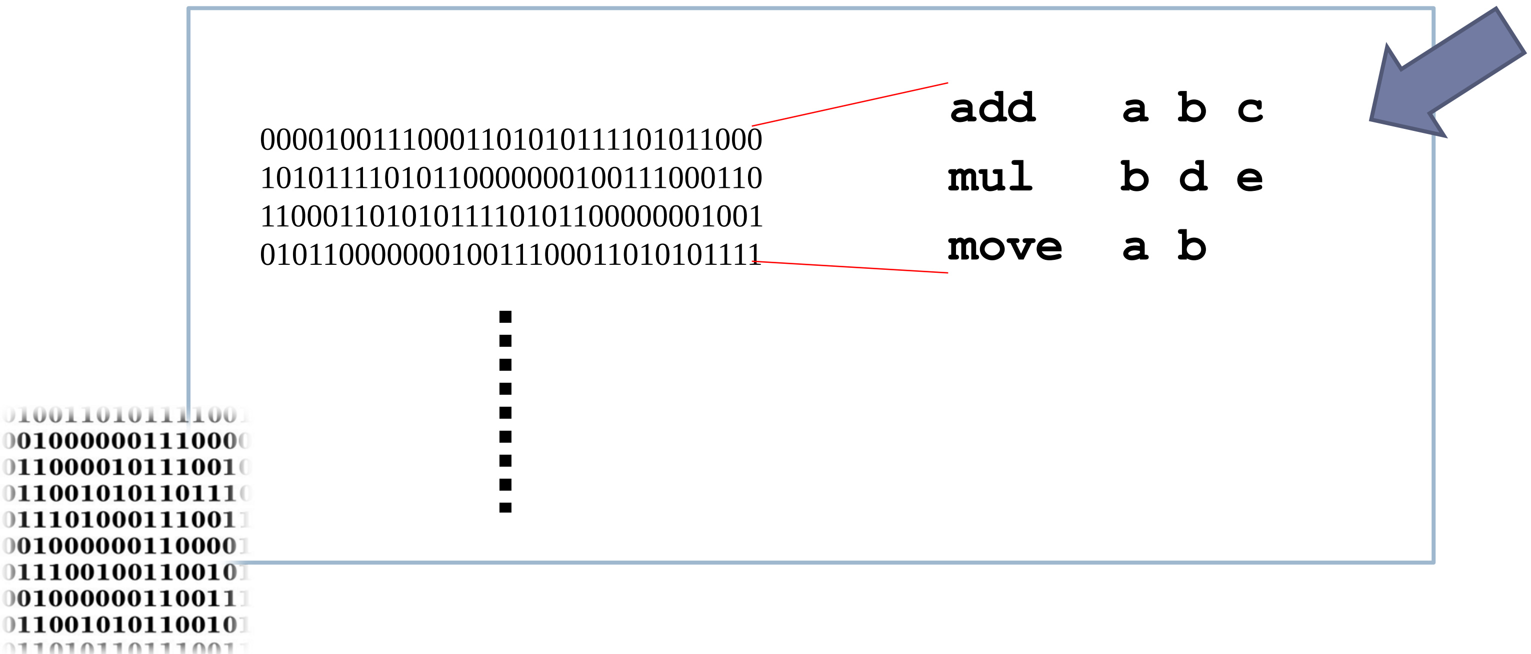
Steps to execute an instruction

- Fetch
  - MAR ← PC
  - Read
  - MBR ← Memory
  - PC ← PC + "1"
  - RI ← MBR
- Decoding
- Execution
- Jump to fetch



Assembly language definition

- **Assembly language**: programmer-readable language that is the most direct representation of architecture-specific machine code.



Félix García-Carballeira, Alejandro Calderón Mateos

Assembly language definition

- **Assembly language**: programmer-readable language that is the most direct representation of architecture-specific machine code.
  - Uses symbolic codes to represent instructions
    - `add` – addition
    - `lw` – Load a memory data
  - Uses symbolic codes for data and references
    - `$t0` – register
  - There is an assembly instruction per machine instruction
    - `add $t1, $t2, $t3`

Félix García-Carballeira, Alejandro Calderón Mateos

Languages levels

High level language (Eg: C, C++)

Compiler

Assembly language (Eg: MIPS)

Assembler

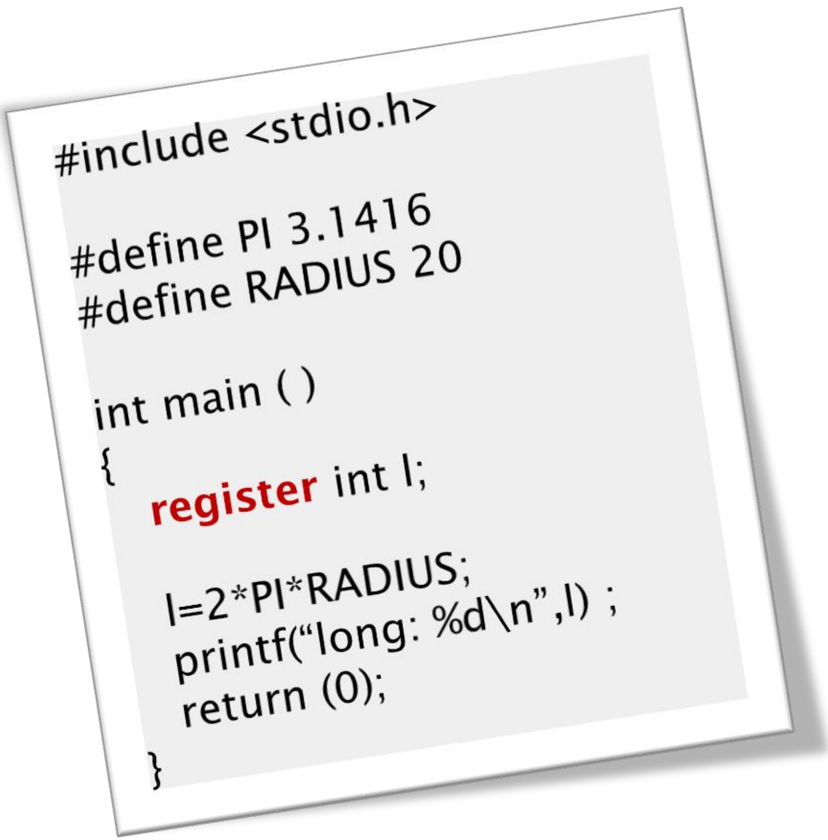
Machine language (MIPS)

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
lw    $t0, 0($2)
lw    $t1, 4($2)
sw    $t1, 0($2)
sw    $t0, 4($2)
```

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

Motivation to learn assembly

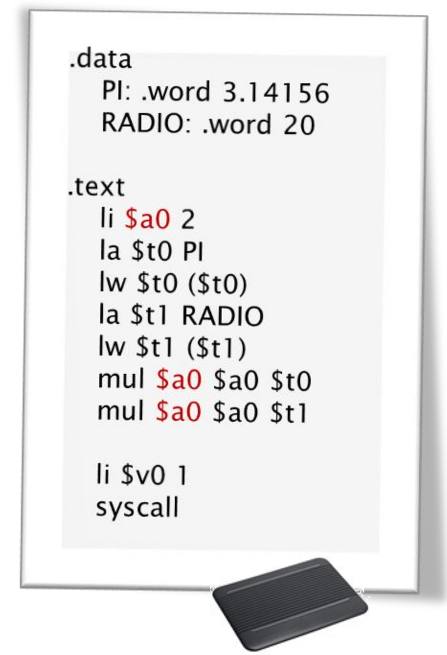


```c
#include <stdio.h>

#define PI 3.1416
#define RADIUS 20

int main ()
{
    register int l;

    l=2*PI*RADIUS;
    printf("long: %d\n",l) ;
    return (0);
}
```

- Understand how high level languages are executed
  - C, C++, Java, …
- Analyze the execution time of high level instructions.
- Useful in specific domains:
  - Compilers
  - Operating Systems
  - Games
  - Embedded systems
  - Etc.

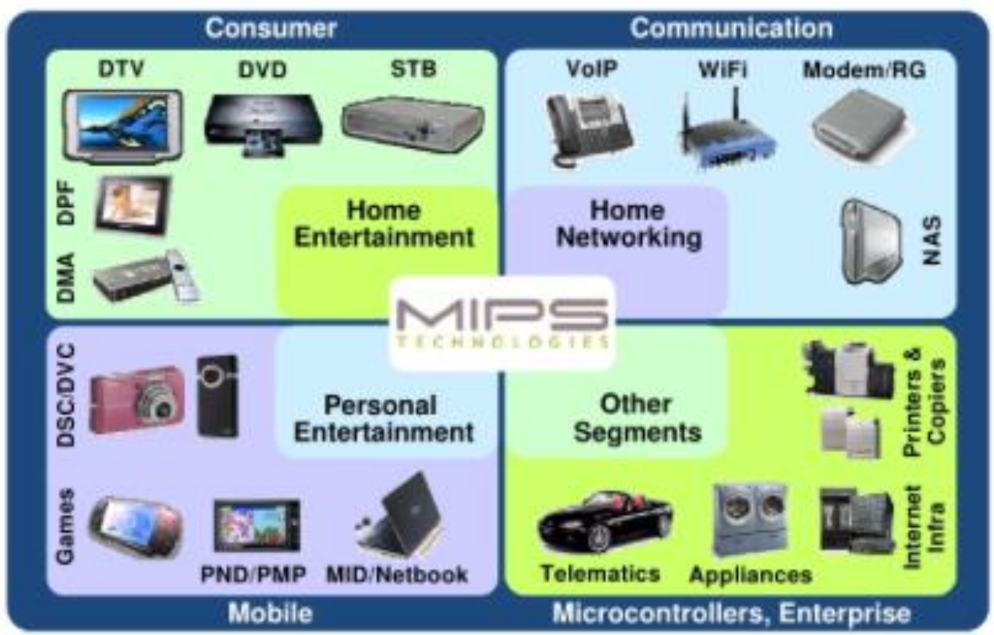
Goals

- Know how the elements of a high-level assembly language are represented.:
  - Data types (int, char, ...)
  - Control structures (if, while, ...)
- Be able to write small programs in assembler

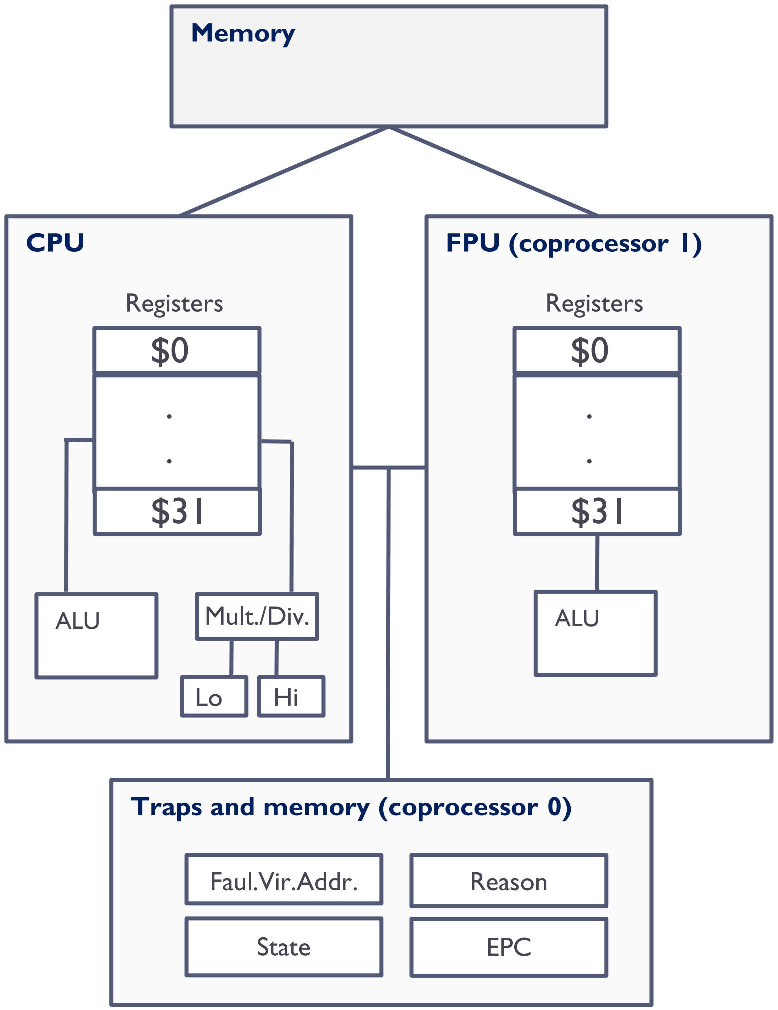


Example assembler: MIPS 32

- RISC (Reduced Instruction Set Computer) Processor
- Examples of RISC processors:
  - MIPS, ARM, RISC-V



MIPS architecture



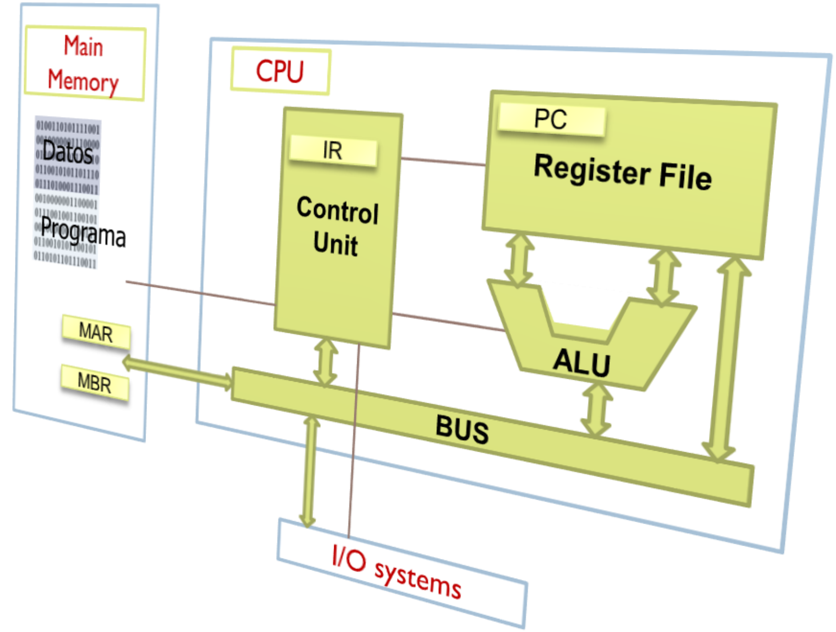
- MIPS 32
  - 32 bits processor
  - RISC type
  - CPU + auxiliary coprocessors
- Coprocessor 0
  - exceptions, interrupts and virtual memory system
- Coprocessor 1
  - FPU (floating point unit)

Register File (integers)

Symbolic name	Number	Usage
zero	0	Constant 0
at	1	Reserved for assembler
v0, v1	2, 3	Results of functions
a0, …, a3	4, …, 7	Function arguments
t0, …, t7	8, …, 15	Temporary (NO preserved across calls)
s0, …, s7	16, …, 23	Saved temporary (preserved across calls)
t8, t9	24, 25	Temporary (NO preserved across calls)
k0, k1	26, 27	Reserved for operating system
gp	28	Pointer to global area
sp	29	Stack pointer
fp	30	Frame pointer
ra	31	Return address (used by function calls)

- There are 32 registers
  - Size: 4 bytes (1 word)
  - Used a $ at the beginning
- Use convention
  - Reserved
  - Arguments
  - Results
  - Temporary
  - Pointers

Types of instructions



- Data transfer
- Arithmetic
- Logical
- Shifting
- Rotation
- Comparison
- Branches
- Conversion
- Input/output
- System calls

Data transfer

- Copy data:
  - Between registers
  - Between registers and memory (later)

- Examples:
  - Store a value in a register. Immediate load
    - li $t0 5 # $t0 ← 5
  - Register to register
    - move $a0 $t0 # $a0 ← $t0

Arithmetic instructions

- Integer operations (ALU) or floating point operations (FPU)
- Examples (ALU):
  - Addition

    add $t0, $t1, $t2 $t0 ← $t1 + $t2 Addition with overflow

    addi $t0, $t1, 5 $t0 ← $t1 + 5 Addition with overflow

    addu $t0, $t1, $t2 $t0 ← $t1 + $t2 Addition without overflow
  - Subtraction

    sub $t0 $t1 1
  - Multiplication

    mul $t0 $t1 $t2
  - Division

    div $t0, $t1, $t2 $t0 ← $t1 / $t2 Integer division

    rem $t0, $t1, $t2 $t0 ← $t1 % $t2 remainder

Example

```
int a = 5;
int b = 7;
int c = 8;
int d;

d = a * (b + c)
```

```
li $t0, 5
li $t1, 7
li $t2, 8


add $t1, $t1, $t2
mul $t3, $t1, $t0
```

Example

```
int a = 5;
int b = 7;
int c = 8;
int d;

d=-(a*(b-10)+c)
```

```
li $t0, 5
li $t1, 7
li $t2, 8
li $t3 10



sub $t4, $t1, $t3
mul $t4, $t4, $t0
add $t4, $t4, $t2
li  $t5, -1
mul $t4, $t4, $t5
```

Types of arithmetic operations

- Pure binary or two's complement arithmetic

- Examples:
  - Signed sum (ca2)
    add $t0 $t1 $t2
  - Immediate signed sum
    addi $t0 $t1 -5
  - Unsigned sum (binary)
    addu $t0 $t1 $t2
  - Immediate unsigned sum
    addiu $t0 $t1 2

- Without **overflow**:

```
li $t0 0x7FFFFFFF
li $t1 5
addu $t0 $t0 $t1
```

- With **overflow**:

```
li $t0 0x7FFFFFFF
li $t1 1
add $t0 $t0 $t1
```

Exercise

```
li $t1 5
li $t2 7
li $t3 8

li   $t0 10
sub  $t4 $t2 $t0
mul  $t4 $t4 $t1
add  $t4 $t4 $t3
li   $t0 -1
mul  $t4 $t4 $t0
```

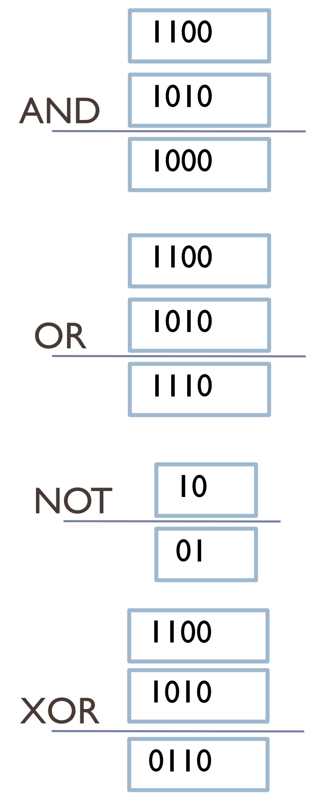
Exercise (solution)

```
li $t1 5
li $t2 7
li $t3 8

li   $t0 10
sub  $t4 $t2 $t0
mul  $t4 $t4 $t1
add  $t4 $t4 $t3
li   $t0 -1
mul  $t4 $t4 $t0
```

```
li $t1 5
li $t2 7
li $t3 8

addi  $t4 $t2 -10
mul   $t4 $t4 $t1
add   $t4 $t4 $t3
mul   $t4 $t4 -1
```

Logical instructions

- Boolean operations
- Examples:
  - AND
    and $t0 $t1 $t2   ($t0 = $t1 & $t2)
  - OR
    or $t0 $t1 $t2   ($t0 = $t1 | $t2)
    ori $t0 $t1 80   ($t0 = $t1 | 80)
  - NOT
    not $t0 $t1   ($t0 = ! $t1)
  - XOR
    xor $t0 $t1 $t2   ($t0 = $t1 ^ $t2)



Example

```
li $t0, 5
li $t1, 8



and $t2, $t1, $t0
```

What is the value of $t2?

Solution

```
li $t0, 5
li $t1, 8



and $t2, $t1, $t0
```

What is the value of $t2?

	000	….	0101	$t0
	000	…..	1000	$t1
and	000	…..	0000	$t2

Exercise (solution)

```
li $t0, 5
li $t1, 0x007FFFFF

and $t2, $t1, $t0
```

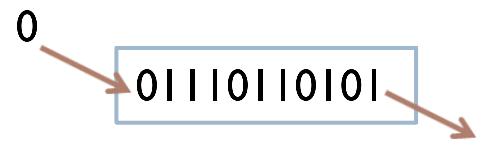
What does an "and" with 0x007FFFFF allow to do?

Obtain the 23 least significant bits

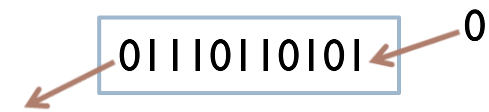
The constant used for bit selection is called a mask.

Shift instructions

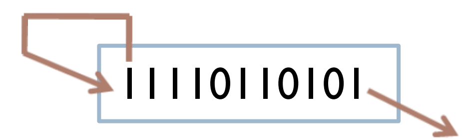
- Bits movement
- Examples:
  - Shift right logical
    srl $t0 $t0 4 ($t0 = $t0 >> 4 bits)



  - Shift left logical
    sll $t0 $t0 5 ($t0 = $t0 << 5 bits)



  - Shift right arithmetic
    sra $t0 $t0 2 ($t0 = $t0 >> 2 bits)



Example

```
li $t0, 5
li $t1, 6



sra $t0, $t1, 1
```

What is the value of $t0?

Example

```
li $t0, 5
li $t1, 6


sra  $t0, $t1, 1
```

Waht is the value of $t0?

000 …. 0110 $t1
shift one bit to right
000 ….. 0011 $t0

Example

```
li $t0, 5
li $t1, 6



srl $t0, $t1, 1
```

What is the value of $t0?

Example

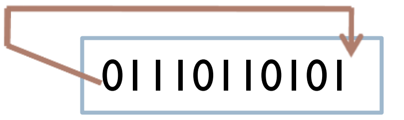
```
li $t0, 5
li $t1, 6


srl $t0, $t1, 1
```

What is the value of $t0?

000 …. 0110 $t1
Shit one bit to left
000 ….. 1100 $t0

Rotations

- Bits movement
- Example:
  - Rotate left
    rol $t0 $t0 4   rotate 4 bits



  - Rotate right
    ror $t0 $t0 5   rotate 5 bits

Exercise (solution)

Make a program that detects the sign of a stored number $t0 and leaves in $t1 a 1 if it is negative and a 0 if it is positive.

```
li   $t0 -3

move $t1 $t0
rol  $t1 $t1 1
and  $t1 $t1 0x0000 0001
```


Félix García-Carballeira, Alejandro Calderón Mateos

Comparison instructions

- seq $t0, $t1, $t2

  if ($t1 == $t2) $t0 = 1; else $t0 = 0
- sneq $t0, $t1, $t2

  if ($t1 !=$t2) $t0 = 1; else $t0 = 0
- sge $t0, $t1, $t2

  if ($t1 >= $t2) $t0 = 1; else $t0 = 0
- sgt $t0, $t1, $t2

  if ($t1 > $t2) $t0 = 1; else $t0 = 0
- sle $t0, $t1, $t2

  if ($t1 <= $t2) $t0 = 1; else $t0 = 0
- slt $t0, $t1, $t2

  if ($t1 < $t2) $t0 = 1; else $t0 = 0

Comparison instructions

- seq $t0, $t1, $t2 Set if equal
- sneq $t0, $t1, $t2 Set if no equal
- sge $t0, $t1, $t2 Set if greater or equal
- sgt $t0, $t1, $t2 Set if greater than
- sle $t0, $t1, $t2 Set if less or equal
- slt $t0, $t1, $t2 Set if less than

Branch instructions

- Alter the flow of control and the sequence of instructions
- Types:
  - Conditional branches:
    - beq $t0 $t1 0xE00012
      - Branch to address 0xE00012, if $t0 == $t1
    - beqz $t1 address
      - Branch to instruction labeled with address if $t1 == 0
  - Unconditional branches:
    - Always branch
      j 0x10002E
      b address
  - Function calls:
    - jal 0x20001E …… jr $ra

Félix García-Carballeira, Alejandro Calderón Mateos

Branch instructions

- beqz $t0, address
  Branch if $t0 == 0
- beq $t0, $t1, address
  Branch if equal (t0 == t1)
- bneq $t0, $t1, address
  Branch if not equal (t0 ≠ t1)
- bge $t0, $t1, address
  Branch if greater or equal (t0 ≥ t1)
- bgt $t0, $t1, address
  Branch if greater than (t0 > t1)
- ble $t0, $t1, address
  Branch if less or equal (t0 ≤ t1)
- blt $t0, $t1, address
  Branch if less than (t0 <t1)

Control flow structures while

```
int i;

 i=0;
 while (i < 10)
 {
   /* action*/
   i = i + 1 ;
 }
}
```

```
           li   $t0 0
           li   $t1 10
while:     bge $t0 t1 end
           # action
           addi $t0 $t0 1
           b while
end:       ...
```

Example

- Calculate 1 + 2 + 3 + ... + 10

```
  i=0;
  s=0;
  while (i < 10)
  {
     s = s + i;
     i = i + 1;
  }
}
```

Result in $t1

Solution

- Calculate 1 + 2 + 3 + ... + 10

```
  i=0;
  s=0;
  while (i < 10)
  {
     s = s + i;
     i = i + 1;
  }
}
```

```
          li    $t0 0
          li    $t1 0
          li    $t2 10
while:    bge   $t0 t2 end
          add   $t1 $t1 $t0
          addi  $t0 $t0 1
          b     while
end:      ...
```

Result in $t1

Example

- Calculate the number of 1's of a register ($t0). Result in $t3.

```
 i = 0;
 n = 45;  #number
 s=0;
 while (i < 32)
 {
   b = last bit of n
   s = s + b;
   sift n one bit to
   right
   i = i + 1 ;
 }
}
```

Example

- Calculate the number of 1's of a register ($t0). Result in $t3.

```
 i = 0;
 n = 45;  #number
 s=0;
 while (i < 32)
 {
   b = last bit of n
   s = s + b;
   sift n one bit to
   right
   i = i + 1 ;
 }
}
```

```
i = 0;
n = 45;  #numero
s = 0;
while (i < 32)
{
  b = n & 1;
  s = s + b;
  n = n >> 1;
  i = i + 1 ;
}
```

Solution

- **Calculate the number of 1's of a register ($t0). Result in $t3**

```
i = 0;
n = 45;   #number
s=0;
while (i < 32)
{
   b = last bit of n
   s = s + b;
   sift n one bit to
   right
   i = i + 1 ;
}
}
```

```
         li    $t0 0     #i
         li    $t1 45    #n
         li    $t2 32
         li    $t3 0     #s
while:   bge   $t0 t2 end
         and   $t4 $t1 1
         add   $t3 $t3 $t4
         srl   $t1 $t1 1
         addi  $t0 $t0 1
         b     while
end:     ...
```

Example

- Calculate the number of 1's of a `int` in C/Java

  Another solution :

```
int count[256] = {0,1,1,2,1,2,2,3,1, . . .8};
int i;
int c = 0;

for (i = 0; i <4; i++) {
     c =  count[n & 0xFF];
     s = s + c;
     n = n >> 8;
}
printf(”There is  %d\n", c);
```

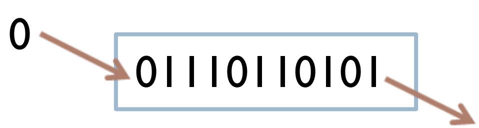
Example

- Obtain the 16 first bits of a register ($t0) and store them in the 16 last bits of other register ($t1)

Solution

- **Obtain the 16 first bits of a register ($t0) and store them in the 16 last bits of other register ($t1)**

```
srl  $t1,  $t0,  16
```



Shift 16 bits to right

Control flow structures
if

```
int b1 = 4;
int b2 = 2;


if (b2 == 8) {
   b1 = 0;
}
 ...
```

```
      li    $t0 4
      li    $t1 2
      li    $t2 8

      bneq $t0 $t2  end
      li    $t0 0
end:  ...
```

Control flow structures
if-else

```
int a=1;
int b=2;

if (a < b)
{
    // action 1
}
else
{
   // action 2
}
```

```
      li   $t1 1
      li   $t2 2

      blt $t1 $t2 then    # cond.
else: ...
      # action 2
      b    end           # uncond.

then: ...
      # action 1

end: ...
```

Exercise

```
int b1 = 4;
int b2 = 2;

 if (b2 == 8) {
     b1 = 1;
 }
 ...
```

Exercise (solution)

```
int b1 = 4;
int b2 = 2;

 if (b2 == 8) {
     b1 = 1;
 }
 ...
```

```
       li   $t0 4
       li   $t1 2
       li   $t2 8

       bneq $t0 $t2  fin1
       li   $t1 1
fin1:  ...
```

Typical faults

1) Poorly designed program
   - Does not do what is requested
   - Incorrectly does what is requested
2) Programming directly in assembler
   - Do not code in pseudo-code the algorithm to be implemented
3) Write unreadable code
   - Do not tabulate the code
   - Do not comment the assembly code or make reference to the algorithm initially proposed.

Compilation process

High level language

```
#include <stdio.h>

#define PI 3.1416
#define RADIO 20

int main ( )
{
   int l;

   l=2*PI*RADIO;
   printf("long: %d\n",l) ;
   return (0);
}
```

Assembly language

```
.data
    PI: .word 3.14156
    RADIO: .word 20

.text
    li $a0 2
    la $t0 PI
    lw $t0 ($t0)
    la $t1 RADIO
    lw $t1 ($t1)
    mul $a0 $a0 $t0
    mul $a0 $a0 $t1

    li $v0 1
    syscall
```

Binary language

```
0100110101111001
0010000001110000
01100000101110010
0110010101101110
0111010001110011
0010000001100001
0111001001100101
0010000001100111
0110010101100101
0110101101110011
```

Example

- Determine if the number stored in $t2 is even. If $t2 is even the program stores 1 in $t1, else stores 0 in $t1

Solution

- Determine if the number stored in $t2 is even. If $t2 is even the program stores 1 in $t1, else stores 0 in $t1

```
       li  $t2  9
       li  $t1  2
       rem $t1  $t2  $t1      # remainder
       beq $t1  $0  then      # cond.
else:  li $t1 0
       b end                   # uncond.
then:  li $t1 1
end:  ...
```

Example

- Determine if the number stored in $t2 is even. If $t2 is even the program stores 1 in $t1, else stores 0 in $t1. In this case, analyze the last bit

Solution

- Determine if the number stored in $t2 is even. If $t2 is even the program stores 1 in $t1, else stores 0 in $t1. In this case, analyze the last bit

```
       li  $t2 9
       li  $t1 1
       and $t1  $t2  $t1    # get the last bit
       beq $t1 $0  then     # cond.
else:  li $t1 0
       b end                 # uncond.
then:  li $t1 1
end:  ...
```

Example

- Calculate a^n
  - a in $t0
  - n in $t1
  - Result in $t2

```
  a=8
  n=4;
  i=0;
  p = 1;
  while (i < n)
  {
    p = p * a
    i = i + 1 ;
  }
}
```

Solution

- Calculate a^n
  - a in $t0
  - n in $t1
  - Result in $t2

```
  a=8
  n=4;
  i=0;
  p = 1;
  while (i < n)
  {
    p = p * a
    i = i + 1 ;
  }
}
```

```
            li    $t0 8
            li    $t1 4
            li    $t2 1
            li    $t4 0

while:      bge   $t4 $t1 end
            mul   $t2 $t2 $t0
            addi  $t4 $t4 1
            b     while
end:        move  $t2  $t4
```