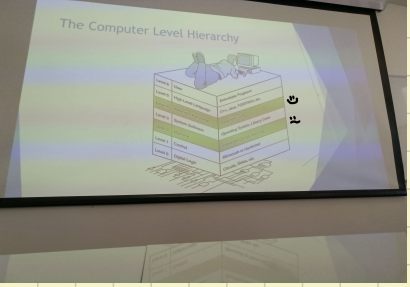
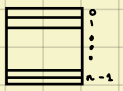
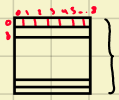


* Modelo de von Neumann
* Lenguajes ensambladores -> Abstracción, no existe un lenguaje ensamblador
* Lenguaje maquina -> Son distintos para cada maquina

- Marie
  - Machine Architecture that is Really Intuitive an Easy
    - Memory: To storage programs and data
    - CPU: ALU and several register
    - I/O: Input/Output
  - Binary, two's complement -> Solo enteros
  - Stored program, fixed word length
  - Word (but not byte) addressable
    - Marie es direccionado por palabras.

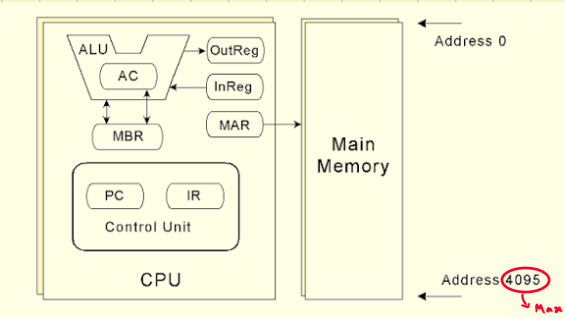


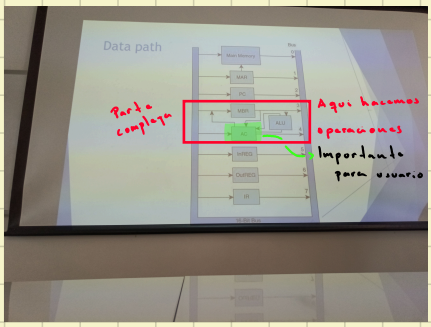


Direccionamiento de Bytes -> Actualmente la mayoria de PC's

  - 4Kib words (0-4095) of main memory (this implies 12 bits per address)
  - 16-bit data (words have 16 bits)
  - 16-bit instructions: 4 for the opcode and 12 for the address
    - Homogenea
  - 16-bit registers:
    - Accumulator (AC), instruction register (IR), memory buffer register (MBR)
  - 12-bit registers
    - program counter (PC), memory address register (MAR)
  - 8-bit registers
    - Input register, output register

- Esquema

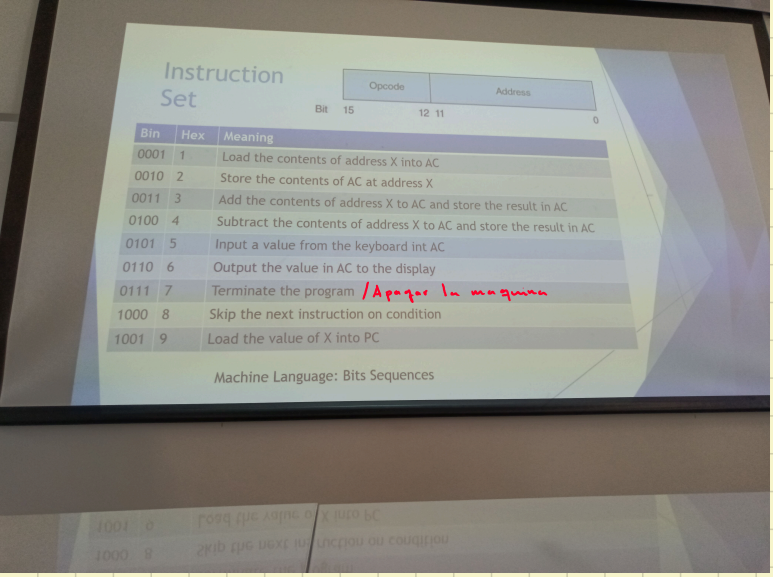




* Sacar datos de memoria -> sigue el ciclo
* Al poner algo en MAR, algo cambia en Main Memory

- Instruction set architecture
  - Nivel 2 -> Lo que puede hacer el computador
  - The instruction set architecture (ISA) of a machine specifies the instructions that the computer can perform and the format for each instruction.
  - The ISA is essentially an interface between the software and the hardware } Aqui nos juntamos sistemas y electronica

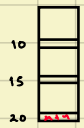
Mitad arriba — Mitad abajo



* Esto es al computador
* 16 operaciones potenciales
* En computadores de hoy en día, al apagar es bajar la potencia

* Esto es lenguaje maquina

Ejemplo



Instrucciones:

Posición

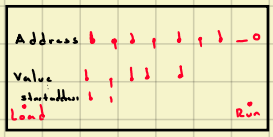
0: 0001 00000000 1010
Cargue en AC — Cargamos 10 en AC

1: 0011 0000 0000 1111
Cargamos 15

2: 0010 0000 0001 0100

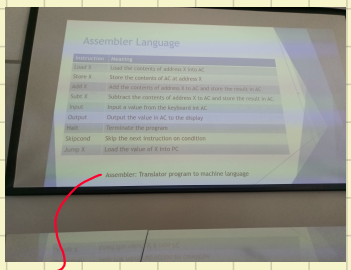
3: 0111 0000 0000 0000

• En 1948



} Definitente

• Lenguaje ensamblador:
  ▸ Usamos nemonicos y un número decimal (Originalmente Octal - Hexa)



* Empezamos a usar etiquetas, más sencillo:

Load a
Add b
Store c
Halt

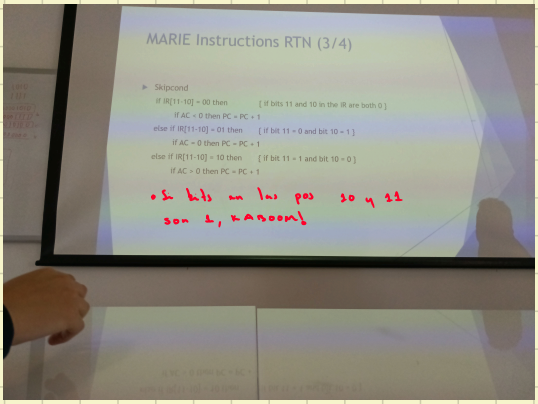
} Más sentido que los libros

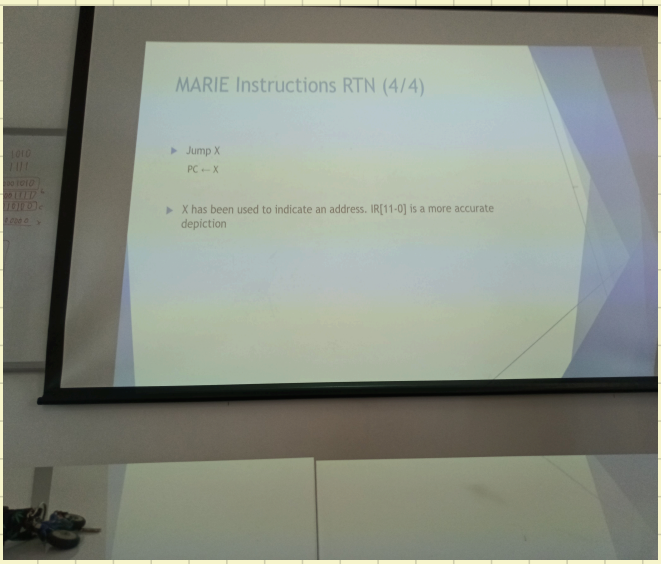
→ Por cada lenguaje maquina hay un ensamblador → No significa que se deban usar siempre
  • GNU → Ensamblador de libre uso

  • Mayoria del ensamblador solo en ensamblador
  • Cross Assembly → Generamos código de plataforma a plataforma
  • Muy ambiguo
    → Espacio para la ambiguedad
  • Descripción no ambigua
    • Se crearon un lenguaje artisanal:
      • Register transfer notation
      • Register transfer language
        - Multiples "mini-instrucciones"
        - "mini-instrucciones" son microoperaciones
        - A microoperation specifies the elementary operations that can be performed.

- Marie instructions
  - Load x
    - MAR ← x
    - MBR ← M[MAR]
    - AC ← MBR

    Micro-operaciones
  - Store x
    - MAR ← x
    - MBR ← AC
    - M[MAR] ← MBR
  - Add x
    - MAR ← x
    - MBR ← M[MAR]
    - AC ← AC + MBR
  - Subt x
    - MAR ← x
    - MBR ← M[MAR]
    - AC ← AC − MBR





Primero que funcione luego se optimiza

En assembler no hay constantes ni variables, hay posiciones de memoria. (Etiquetas) El uso declara si es variable o constante.

Ciclos en Marie:

Ejemplo: Entrada n → Salida 1, ..., n-1

En C

```
for (i=0; i<n; i++)
```

En menor nivel

```
i = 0          // Inicialización
while (i < n){ // Condición
  i++;         // Incremento
}
```

A bajo nivel

→ con <

```
/ Escribir n - 1 enteros
/ Autor: Gerardo Ospina
/ read n
      Input
      Store n
/ Inicializar i
      Load one
      Store
while, Load i
      Subt n
      Skipcond 800    / i > n?
      Jump whilebody  / no
      Jump endwhile   / yes
whilebody Load i
      Output
      Load i  / print i
      Add one
      Store i  / i = i + 1
      Jump while
Endwhile Halt
One,     Dec 1
i,       Dec 0   / o cualquier nom
n,       Dec 0
```

$i \le n \equiv (i < n) \vee (i = n)$

$i \le n \equiv \neg(i > n)$

Discretas

← Ahora con <=

```
/ Escribir n - 1 enteros
/ Autor: Gerardo Ospina
/ read n
      Input
      Store n
/ Inicializar i
      Load one
      Store
while, Load i
      Subt n
      Skipcond 000    / i < n?
      Jump endwhile / no
      Load i        / yes
      Output
      Load i   / print i
      Add One
      Store i   / i = i + 1
      Jump while
endwhile Halt
One,     Dec 1
i,       Dec 0   / o cualquier nom
n,       Dec 0
```

Instrucciones

Clear /AC = 0

Jump and Store

JnS

JnS (In)

Jump I

25:	JnS	200

PC 26:

200	26 → Dirección de retorno
201	

Instrucciones

Clear /AC = 0

Jump I (Volver)

25:	JnS	200

PC 26:

200	26 → Dirección de retorno
201	≡
220	Jump I 200

Esto no existe en assembler, toca implementarlo

En C

```
int mul(int a, int b)
 {
    int m, i;
      m = 0;
      i = 0;
    while (i < a) {
      m = m+b;
      i++;
    }
    return m;
 }
```

Ejemplo: Dados a, b, c, x calcular el polinomio ax^2+bx+C

Toca implementar la multiplicación

Orden de escritura:

- Primero main, luego la función

Main

Etiqueta	Instrucción	Dirección_memoria
/Calcular ax^2+bx+c		
/Autor: Gerardo Ospina		
/read a,b,c,x		
	Input	
	Store	a /read a
	Input	
	Store	b /read b
	Input	c /read c
	Input	
	Store	x /read x
/$p = ax^2+bx+c$		
	Load	x
	Store	mul_a
	Store	mul_b
	Ins	mul /mul(x,x), respuesta en acumulador
	Store	p /$p = x^2$
	Load	a
	Store	mul_a
	Load	p
	Store	mul_b
	InS	mul /mul(a,x^2)
	Store	p /$p = ax^2$
	Load	b
	Store	mul_a
	Load	x
	Store	mul_b
	InS	mul /mul(b,x)
	Add	p
	Add	c
	Store	p /$p = ax^2+bx+C$
/imprimir	p	
	Load	p
	Output	/print p
	Halt	
a,	Dec	11
b,	Dec	22
c,	Dec	33
x,	Dec	44
p	Dec	55

Función (solo si $a \geq 0$) → Si $a < 0$ da loop

Etiqueta	Instrucción	Dirección_memoria
/multiplicar dos enteros		
/mul(a,b)		
/res: a*b		
mul,	Dec	0 /fundamental dejar este espacio
	Clear	
	Store	mul_m /m=0
	Store	mul_i /i=0
mul_while,	Load	mul_i
	Sobt	mul_a
	Skipcont	000 /i < a?
	Jump	mul_endw /no
	Load	mul_m /yes
	Add	mul_b
	Store	mul_m /m = m+b
	Load	mul_i
	Add	mul_one
	Store	mul_i /i = i+1
	Jump	mul_while
	Load	mul_m /return m
	Jump I	mul
mul_a	Dec	11
mul_b	Dec	22
mul_i	Dec	33
mul_m	Dec	44 /El valor no importa
mul_one	Dec	1 /Obligatorio que sea 1

Cuando se va a dejar algo, se deja en el acumulador

Extended Instruction Set

Bit	15 … 12	11 … 0
	Opcode	Address

Bin	Hex	Meaning
0000	0	Store the PC at address X and jump to X + 1
1010	A	Put all zeros in AC
1011	B	Add the contents of address MEM[X] to AC and store the result in AC
1100	C	Load the value of address MEM[X] into PC
1101	D	Load the contents of address MEM[X] into AC
1110	E	Store the contents of AC at address MEM[X]

En Assembly

Extended Assembler Language

Instruction	Meaning
JnS X	Store the PC at address X and jump to X + 1
Clear	Put all zeros in AC
AddI X	Add the contents of address MEM[X] to AC and store the result in AC
JumpI X	Load the value of address MEM[X] into PC
LoadI X	Load the contents of address MEM[X] into AC
StoreI X	Store the contents of AC at address MEM[X]

Microinstrucciones en Marie

Extended MARIE Instructions RTN (1/3)

- JnS X
  - MBR <- PC
  - MAR <- X
  - M[MAR] <- MBR
  - MBR <- X
  - AC <- 1
  - AC <- AC + MBR
  - PC <- AC
- Clear
  - AC <- 0

Extended MARIE Instructions RTN (2/3)

- AddI X
  - MAR <- X
  - MBR <- M[MAR]
  - MAR <- MBR
  - MBR <- M[MAR]
  - AC <- AC + MBR
- JumpI X
  - MAR <- X
  - MBR <- M[MAR]
  - PC <- MBR

Extended MARIE Instructions RTN (3/3)

- LoadI X
  - MAR <- X
  - MBR <- M[MAR]
  - MAR <- MBR
  - MBR <- M[MAR]
  - AC <- MBR
- StoreI X
  - MAR <- X
  - MBR <- M[MAR]
  - MAR <- MBR
  - MBR <- AC
  - M[MAR] <- MBR