ARQUITECTURA DEL SET DE INSTRUCCIONES

ISA: Instruction Set Arquitecture

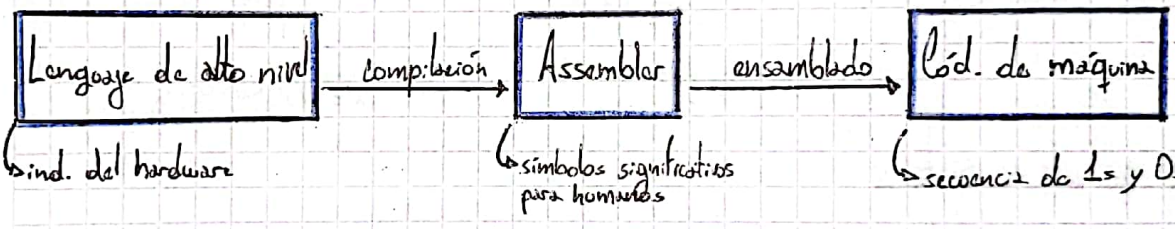
Código de máquina: Secuencia de 1s y 0s que entiende la máquina.

Assembler: • traducción del código de máquina con mensajes más claros.

• traducción del lenguaje en alto nivel a algo más simbólico del cód. de máquina

→ lenguaje intermedio entre el cód. de máquina y el lenguaje de alto nivel.

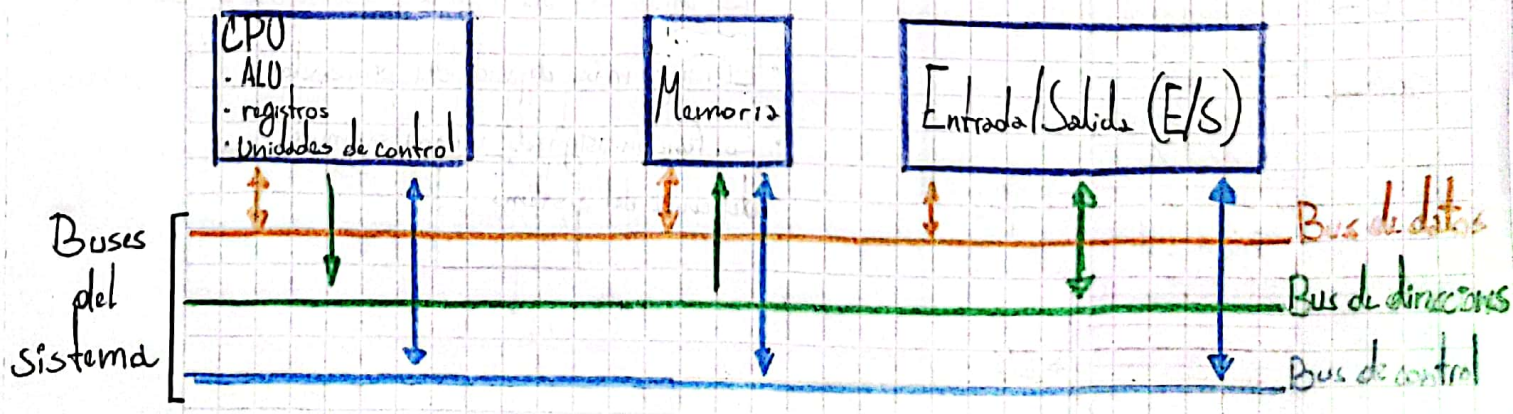
Lenguaje de alto: Lenguaje de programación con mensajes super claros para el humano



Componentes circuitales de la arquitectura de programación

La CPU se interconecta con la memoria y con los sist. de entrada y salida a través del bus del sistema

↳ en sist. complejos puede haber buses separados



Accesos a memoria RAM

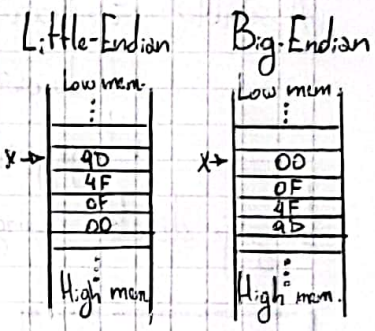
- Forma estructura de datos organizada tipo tabla.
- Cada renglón de la tabla es identificado por su dirección.
- Cada dato es agrupado físicamente de a 8 bits = 1 byte.
- Los procesadores en gral. tienen instrucciones para acceder simultáneamente 1, 2, 4 o más bytes (palabras)
- Palabras de más de 8 bits son guardadas como una serie de bytes

La información se guarda en bits:
1 nibble = 4 bits; 1 byte = 8 bits;
media palabra = 16 bits;
palabra = 32 bits.
doble palabra = 64 bits;
Cuádruple palabra = 128 bits.

Guardar palabras de varios bytes

Formas → Little-Endian: Byte menos significativo en la dirección más baja
↳ Big-Endian: Byte más significativo en la dirección más baja

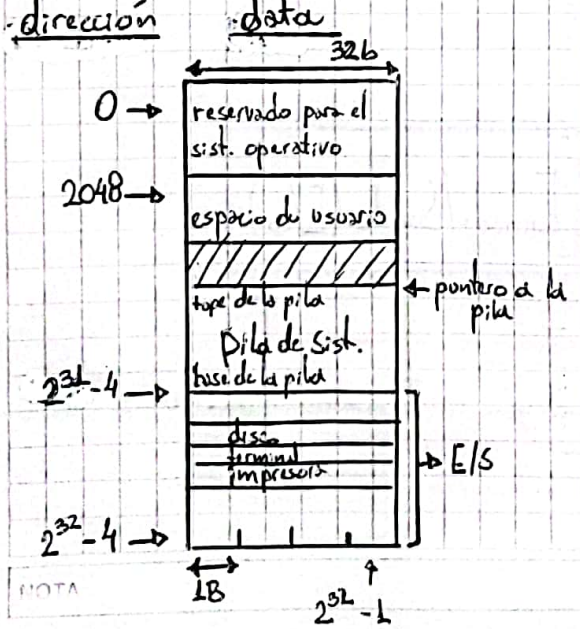
→ Que sea de una u otra forma depende del procesador



x = 000F4F9Dh

Mapa de memoria de un sistema

- Representa la asignación dada al espacio de direcciones
- El tamaño del espacio direccionable es específico del procesador
- El mapa de memoria es específico de un sistema (tipo de computadora)
- Dos sist. basados en el mismo procesador no tienen necesariamente el mismo mapa de memoria



- El rango máx. depende del procesador.
- La función asignada a cada segmento depende del sist.

NOTA

Arquitectura ARC

ARC → A risk computer : computadora con un conj reducido de instrucciones

- Memoria
  - Datos de 32 bits direccionables por byte
  - Espacio de direcciones: 2^{32}
  - Big - Endian
  - Solo puedo leer/escribir registros
- Set de Instrucciones
  - Subconjunto de SPARC
  - Todas las instrucciones ocupan 32 bits = 4B
  - 32 registros de 32 bits → El program status register (psr) guarda los flags de ALU
  - Solo dos instrucciones acceden a memoria principal:
    - i- Leer memoria a registro (load: ld)
    - ii- Escribir desde registro a memoria (store: st)

Instrucciones ARC → específicas para det. máquina / procesador

	Mnemónico	Acción
memoria	ld: load	Leer de memoria y cargar en registro
	st: store	Escribir desde registro a memoria
lógica	sethi	Cargar los 22 bits ⊕ significativos de un registro
	andcc	Producto lógico AND
	orcc	lógico OR
	orncc	Suma lógica negado NOR
	srl	Desplazamiento lógico a derecha
aritmética	addcc	Suma
control	call	Salto (llamado) a subrutina
	jmpl	Retorno de subrutina
	be	bifurcación (por igual)
	bneg	bifurcación (por n=1)
	bcs	bifurcación (por c=1)
	bvs	bifurcación (por v=1)
	bcd	bifurcación incondicional

↳ hay más

Flags
- z=1 → resultado=0
- n=1 → resultado<0
- c=1 → carry (arrastre)
- v=1 → overflow (desborde)

Registros Accesibles al Programador

00:	%r0 → siempre vale 0	11:	%r11	22:	%r22
01:	%r1	12:	%r12	23:	%r23
02:	%r2	13:	%r13	24:	%r24
03:	%r3	14:	%r14 → stack pointer	25:	%r25
04:	%r4	15:	%r15 → link	26:	%r26
05:	%r5	16:	%r16	27:	%r27
06:	%r6	17:	%r17	28:	%r28
07:	%r7	18:	%r18	29:	%r29
08:	%r8	19:	%r19	30:	%r30
09:	%r9	20:	%r20	31:	%r31
10:	%r10	21:	%r21		

%psr → program status register

%pc → program counter

Registros de uso predefinido → %r14: stack pointer → apunta al último valor de la pila.
↳ %r15: link → dir. desde donde hago call.

Registros intocables → %pc: program counter → apunta a la instrucción de assembler
↳ %psr: program status register → flags nzvc

31 30 29 28 27 26 25 24 **23 22 21 20** 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

| . . . . n z v c . . . . |

Sintaxis

lab_1: addcc %r1 , %r2 , %r3 ! comentario

lab_1 = label; addcc = mnemonic; %r1, %r2 = origen; %r3 = destino; ! = inicia comentario

- Se distingen minúsculas de mayúsculas
- Números: default → base 10 (decimal)
  → si empieza con "0x" o termina con "h" → hexadecimal

Cargar ctes. en registros

- Hasta 13 bits
  add %r0, cte, %r1
- Hasta 22 bits
  sethi cte, %r1
  sra %r1, 10, %r1
- Más de 22 bits
  sethi 22baltos, %r1
  add 10bbajos, %r1
- Grandes (poco eficiente)
  ld [variable], %r1
  variable: const.

Ejemplos

- addcc %r1, 12, %r3
- ld [x], %r1
  ld [x], %r0, %r1
  ld %r0+x , %r1
- st %r1, [x]
- sethi 0x304F15, %r1
- andcc %r1, %r2, %r3
- orcc %r1, 1, %r1
- orncc %r1, %r0, %r1
- srl %r1, 3, %r2
- call sub_r
- jmpl %r15 + 4, %r0 (%r15 = pos. actual; 4 = sig)
- be label
- bneg label
- bcs label
- bvs label
- ba label.

NOTA

Directivas al Ensamblador (o pseudo instrucciones)

- Indican al ensamblador cómo procesar una sección de programa
- Específicas para un programa ensamblador,
- Algunas generan información en memoria.

directiva	forma de uso	función o significado
.equ	X .equ #LD	iguala símbolo X con valor (LD)$_{16}$
.begin	.begin	comienza a ensamblar
.end	.end	termina de ensamblar
.org	.org 2048	cambiar cont. de pos. a 2048
.dwb	.dwb 25	reserva bloque de 25 palabras = 100B = 800bits
.global	.global Y	la variable Y se usa en otro módulo
.extern	.extern Z	la variable Z se define en otro módulo
.macro	.macro M a,b	definir macroinstrucción M. parámetros formales: a, b.
.endmacro	.endmacro	fin de definición de macroinstrucción.
.if	.if <cond>	ensamblar solo si <cond> es cierta.
.endif	.endif	fin de estructura condicional

Subrutinas

→ Facilita la estructuración del programa, resolviendo problemas muy específicos

- Pasaje de parámetros → por registros
  → por pila
  → por área reservada de memoria

SUMAR

!por registro

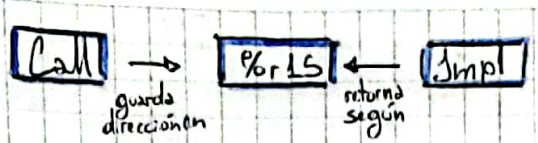
```
        .begin
        .org 2048
        ld [x],%r1
        ld [y],%r2
        addcc %r15,%r0,%r16 !guardo dir.
        call sbr_add
        st %r3,[z]
        jmpl %r16+4,%r0

sbr_add: addcc %r1,%r2,%r3
        jmpl %r16+4,%r0

x:      15
y:      9
z:      0
        .end
```

!por pila

```
        .begin
        .org 2048
        ld [x],%r1
        ld [y],%r2
        addcc %r14,-4,%r14 | push %r1
        st %r1,%r14
        addcc %r14,-4,%r14 | push %r2
        st %r2,%r14
        addcc %r15,%r0,%r16 !guardo dir.
        call sbr_add
        ld %r14,%r3
        addcc %r14,4,%r14 | pop en %r3
        st %r3,[z]
        jmpl %r16+4,%r0

sbr_add: ld %r14,%r8
        addcc %r14,4,%r14 | pop en %r8
        ld %r14,%r9
        addcc %r8,%r9,%r10 | push %r10 en %r14
        st %r10,%r14
        jmpl %r15+4,%r0

x:15
y:9
z:0     .end
```

!por área de memoria reservada

```
        .begin
        .org 2048
        st %r1,[x]
        st %r2,[x+4]
        addcc %r0,x,%r5 |
        addcc %r15,%r0,%r16 !guardo
        call sbr_add
        st [x+8],%r3
        jmpl %r16+4,%r0

sbr_add: ld %r5,%r8
        ld %r5+4,%r9
        addcc %r8,%r9,%r10
        st %r10,%r5+8
        jmpl %r15+4,%r0
x: .dwv 3 ! reservo 12Bytes

        .end
```

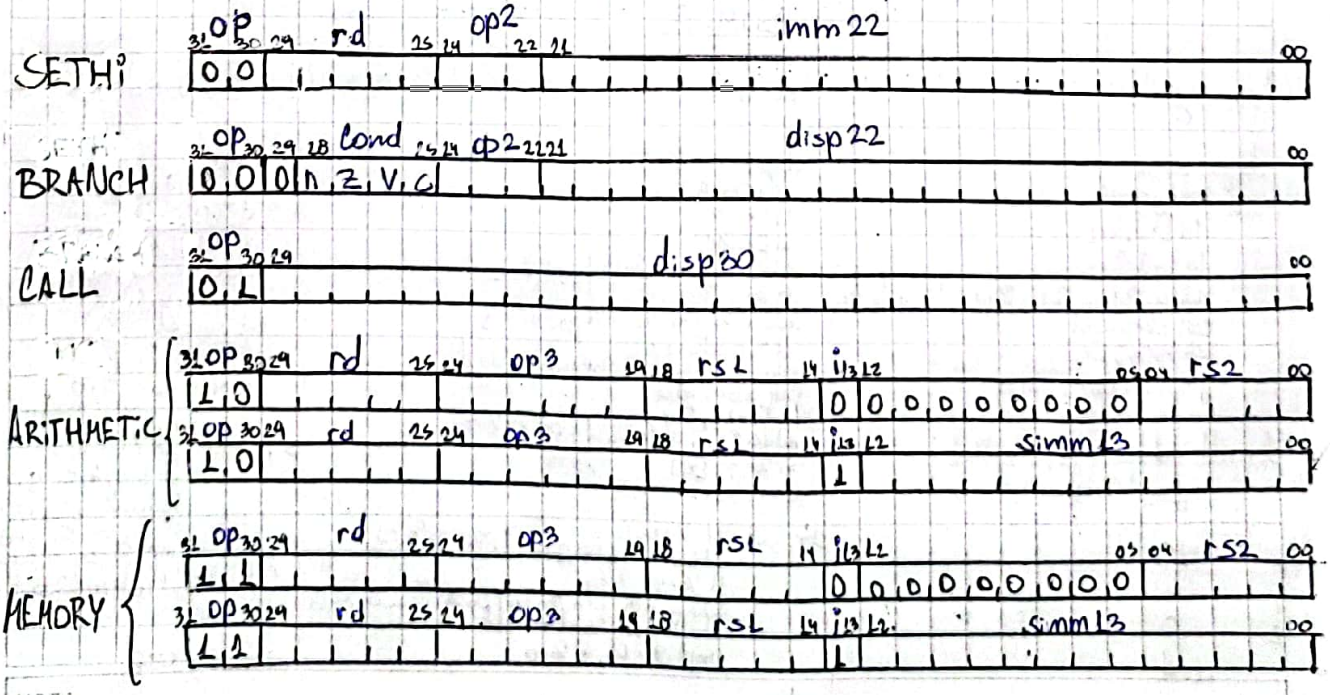


Macros vs. Subrutinas

Macro	Subrutina
• Se accede en tiempo de ensamblado	• Se accede por un call en tiempo de ejecución
• Sus parámetros son interpretados por el ensamblador y reemplazados por lo que corresponda	• Sus parámetros le llegan en tiempo de ejecución
• Su cód. de máquina está repetido tantas veces como la macro fue invocada	• Su cód. de máquina está localizado en memoria

Código de Máquina

- Todas las instrucciones ocupan 4 Bytes = 32 bits.
- No todas las instrucciones siguen el mismo formato
- Hay 5 formatos (respecto a su cód. de máquina)
  - sethi
  - branch
  - call
  - aritméticas
  - acceso a memoria



NOTA

- Branch format → pc = pc + 4 × disp22 (puede ser negativo)
- Arithmetic format → Si i=0 ⇒ no inmediato: dos registros de origen (rs1 y rs2)
  i=1 ⇒ inmediato: solo un registro de origen (rs1) y una cte (simm13)

Ejemplo
```
addcc %r1, %r2, %r1  ! i=0
addcc %r1, 25, %r1   ! i=1.
```

- Memory format → si i=0 ⇒ no inmediato: dos registros de origen (rs1 y rs2).
  i=1 ⇒ inmediato: un registro de origen (rs1) y una cte (simm13)

Ejemplo
```
ld %r8, %r9, %r1  ! i=0
ld %r8, [A], %r1  ! i=1
```

op	format
00	sethi/branch
01	call
10	arithmetic
11	memory

op2	inst.
010	branch
100	sethi

op3 (op=10)	
010000	addcc
010001	andcc
010010	orcc
010110	orncc
100110	srl
111000	jmpl

op3 (op=11)	
000000	ld
000100	st

cond	branch
0001	be
0101	bcs
0110	bneg
0111	bvs
1000	ba

Modos de direccionamiento

Modo	Significado
Inmediato	cte incluida en la instrucción → ej: addcc %r3, 9, %r3.
Por registro	el registro tiene el dato → ej: addcc %r3, %r4, %r8
Directo o absoluto	dirección de memoria incluida en la instrucción → ej: ld [A], %r1
Indirecto	dirección de memoria donde está el puntero al dato (poco usado, lento)
Indirecto por registro	el registro tiene el puntero al dato
Indexado por registro	un registro da la dirección inicial, el otro un incremento (arrays)

NOTA

Set de Instrucciones

Set de Instrucciones = Conjunto de instrucciones + Registros disponibles

- Contiene
  - Tamaño de las instrucciones (tamaño ocupado por el cód. de máquina)
  - Tipo de operaciones admitidas
  - Tipo de operandos (ubicación y tamaño)
  - Tipo de resultados (ubicación y tamaño)
  - Formas de indicar la indicación de los datos (modos de direccionamiento)
- Modelos de Arquitectura
  - CISC (Complex Instruction Set Computer)
  - RISC (Reduced Instruction Set Computer)

Arquitectura CISC

- Cierto grado de complejidad en sus instrucciones
- Cód. binario de largo variable (entre 1 y 15 Bytes)
- El nº de operandos y su tipo dependen de la instrucción

Conclusión
- Complica la lógica para encontrar instrucciones en memoria.
- Complica la decodificación de las instrucciones
- Complica la lógica de interconexiones dentro del procesador
- Facilita el hacer un compilador (las instrucciones son más poderosas)

Ejemplo: Intel x86, x86-32, x86-64 (aunque INTEL tiene interfase para usarse como RISC)

Arquitectura RISC

- Cierto grado de simplificación en sus instrucciones
- En todas ellas el cód. binario tiene la misma longitud.
- Todas están localizadas en direcciones de memoria múltiplos de 4.
- Las únicas instrucciones que acceden a memoria son "leer" y "escribir"
- Operaciones aritméticas y lógicas sólo entre registros (modos de direccionamiento)
- Dispositivos de E/S mapeados en memoria

Conclusión
- Simplifica lógica para encontrar instrucciones en memoria (se mueve de a 4B)
- Simplifica decodificación de instrucciones (mismo tamaño, formato similar)
- El hardware más simple permite optimizarlo para obtener mayor velocidad
- Genera mayor cant. de instrucciones en assembler ⇒ ⊕ memoria

NOTA