Arquitectura y Organización

Para utilizar la física (partículas, electrones), para una aplicación útil se necesitan varias capas de abstracción ya que la brecha entre estas es muy grande. La arquitectura del computado es el estudio de estas capas y de las abstracciones para convertirlas en una aplicación.

El set de instrucciones (ISA) proporciona un modelo abstracto, le da al programador un estado visible (memoria y registros), define las operaciones que la computadora puede realizar (las instrucciones y cómo funcionan), define semánticas más complicadas (E/S, excepciones), y define el tipo de data y el tipo de data sobre el que se opera (enteros, decimales, tamaño de palabra).

Por otro lado la microarquitectua u organización decide como van a implementarse las definiciones de la ISA, dependiendo de las prioridades de diseño: Potencia, tamaño de tarjeta, orden de ejecución, velocidad...

El Modelo Van Neuman

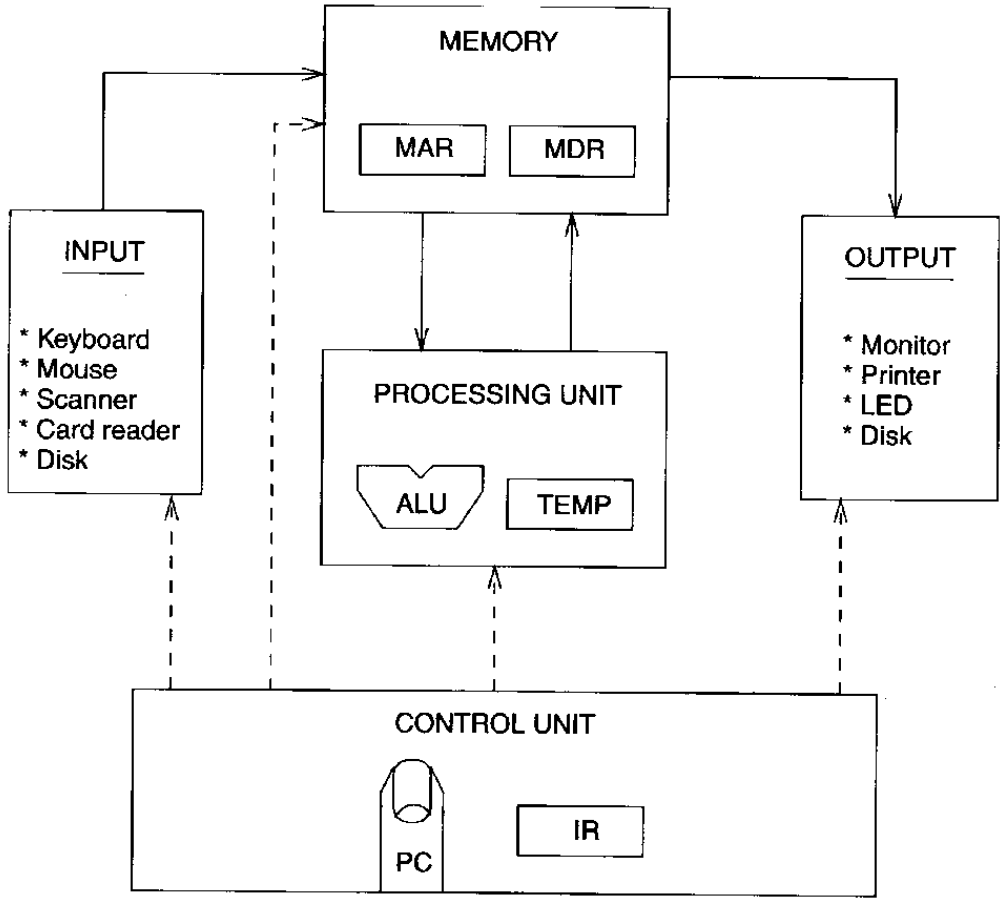


Figure 4.1 The von Neumann model, overall block diagram

Modelos de Máquinas

Existen varios modelos de cómo operar sobre los registros y la memoria: Pila, Acumulador, Registro/Memoria, Registro/Registro.

Para la arquitectura RISC-V el modelo es Registro/Registro (o “load/store”) y significa que las operaciones sólo pueden realizarse sobre registros.

Clases de Instrucciones (algunas)

- Transferencia de datos
- Operaciones ALU
- Flujo de Control

- Data Transfer
  - LD, ST, MFC1, MTC1, MFC0, MTC0
- ALU
  - ADD, SUB, AND, OR, XOR, MUL, DIV, SLT, LUI
- Control Flow
  - BEQZ, JR, JAL, TRAP, ERET
- Floating Point
  - ADD.D, SUB.S, MUL.D, C.LT.D, CVT.S.W,
- Multimedia (SIMD)
  - ADD.PS, SUB.PS, MUL.PS, C.LT.PS
- String
  - REP MOVSB (x86)

Como acceder a la memoria

Addressing Mode	Instruction	Function
Register	Add R4, R3, R2	Regs[R4] <- Regs[R3] + Regs[R2] **
Immediate	Add R4, R3, #5	Regs[R4] <- Regs[R3] + 5 **
Displacement	Add R4, R3, 100(R1)	Regs[R4] <- Regs[R3] + Mem[100 + Regs[R1]]
Register Indirect	Add R4, R3, (R1)	Regs[R4] <- Regs[R3] + Mem[Regs[R1]]
Absolute	Add R4, R3, (0x475)	Regs[R4] <- Regs[R3] + Mem[0x475]
Memory Indirect	Add R4, R3, @(R1)	Regs[R4] <- Regs[R3] + Mem[Mem[R1]]
PC relative	Add R4, R3, 100(PC)	Regs[R4] <- Regs[R3] + Mem[100 + PC]
Scaled	Add R4, R3, 100(R1)[R5]	Regs[R4] <- Regs[R3] + Mem[100 + Regs[R1] + Regs[R5] * 4]

Tipos de datos

- Types
  - Binary Integer
  - Binary Coded Decimal (BCD)
  - Floating Point
    - IEEE 754
    - Cray Floating Point
    - Intel Extended Precision (80-bit)
  - Packed Vector Data
  - Addresses
- Width
  - Binary Integer (8-bit, 16-bit, 32-bit, 64-bit)
  - Floating Point (32-bit, 40-bit, 64-bit, 80-bit)
  - Addresses (16-bit, 24-bit, 32-bit, 48-bit, 64-bit)

Fixed Width: Every Instruction has same width

- Easy to decode

(RISC Architectures: MIPS, PowerPC, SPARC, ARM...)

Ex: MIPS, every instruction 4-bytes

Variable Length: Instructions can vary in width

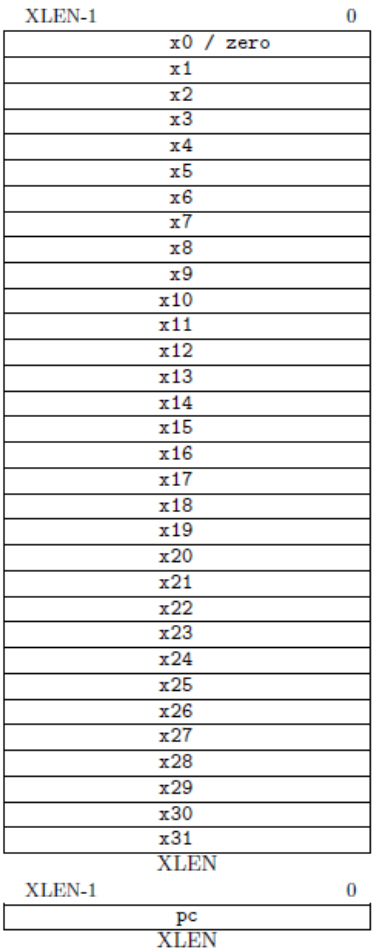
- Takes less space in memory and caches

(CISC Architectures: IBM 360, x86, Motorola 68k, VAX...)

Ex: x86, instructions 1-byte up to 17-bytes

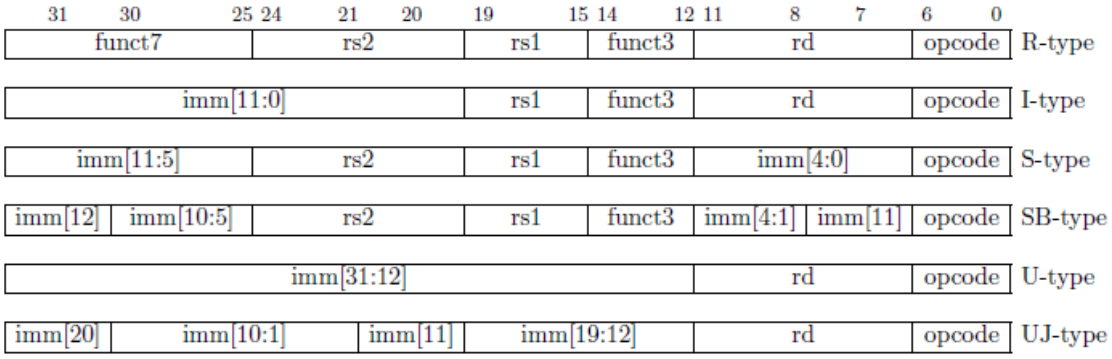
Aplicado a RISC-V

Registros



Existen 32 registros de propósito general, con x0 siendo constante a cero, además está visible el registro PC que tiene la dirección de la instrucción que se ejecuta.

Codificación



Las instrucciones son de ancho fijo. Y existen seis tipos de formato.

- Tipo R para operaciones aritméticas
- Tipo I para operaciones aritméticas con constantes y operaciones para cargar datos
- Tipo S para guardar datos
- Tipo U para cargar constantes a registros
- Tipo SB para saltos condicionales
- Tipo UJ para saltos no condicionales

inst[4:2] / inst[6:5]	000	001	010	011	100	101	110	111 ($> 32b$)
00	LOAD	LOAD-FP	*custom-0*	MISC-MEM	OP-IMM	AUIPC	OP-IMM-32	$48b$
01	STORE	STORE-FP	*custom-1*	AMO	OP	LUI	OP-32	$64b$
10	MADD	MSUB	NMSUB	NMADD	OP-FP	*reserved*	*custom-2/rv128*	$48b$
11	BRANCH	JALR	*reserved*	JAL	SYSTEM	*reserved*	*custom-3/rv128*	$\geq 80b$

Table 9.1: RISC-V base opcode map, inst[1:0]=11

RV32I Base Instruction Set

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

Con esta información se puede diseñar el datapath y el controlpath del procesador

Sabemos:
Cantidad de registros: 32
Tamaño de instrucciones: 32 bits
Tamaño de las direcciones de memoria: 32 bits
Operaciones de la ALU
Modelo Van Neuman

Ejemplo de implementación de un subconjunto de instrucciones MIPS