



计算机系统基础
Programming Assignment

PA 1-1 – 数据的表示和存取

2020年9月17日 / 2020年9月18日
南京大学《计算机系统基础》课程组

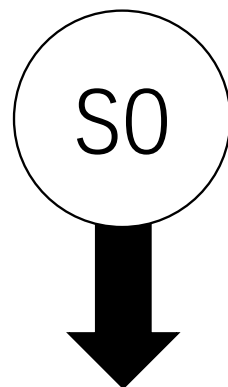
前言

Turing machine (1936)

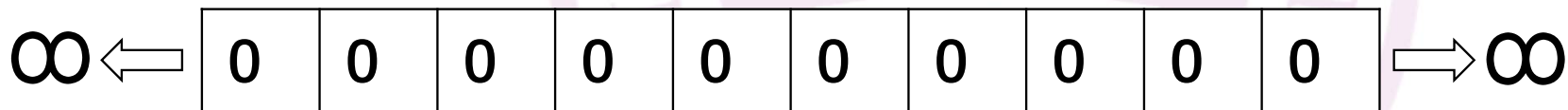
一个包含有限条指令的表格（程序）

一个状态寄存器，可以表示在一个有限的状态集合中，当前机器所处的状态。存在一个初始状态 S_0

机器所处状态	读到的单元值	符号操作	读头移动方向	下一个状态
S_0	0	-	-	停机
	1	-	右移	S_1
S_1	0	写1	右移	S_2
	1	-	右移	S_1
S_2	0	-	左移	S_3
	1	-	右移	S_2
S_3	0或1	写0	-	停机



一个读头，可以读写指向单元的符号，也可以向左或向右移动一个单元



一条两端无限长的，分成单元的纸带，每个单元中可以填入一个有限符号集合（必包含 空符号）中的某一个符号

$\{0, 1\}$ - 其中0表示空符号



Alan Mathison Turing

1912 - 1954

English mathematician, computer scientist, logician, cryptanalyst, philosopher, and theoretical biologist.

2020年9月17日星期四

南京大学-计算机系统基础-PA

前言

Turing machine (1936)

一个包含有限条指令的表格（程序）

一个状态寄存器，可以表示在一个有限的状态集合中，当前机器所处

机器所处状态	读到的单元值	符号操作	读头移动方向	下一个状态
	0			停机
1				1
2				2
1				3
3				2
2				机

Church–Turing thesis:

A function on the natural numbers is computable by a human being following an algorithm, ignoring resource limitations, if and only if it is computable by a Turing machine.

-- Wikipedia

图灵机的计算能力和人脑等价

{0, 1} - 其中0表示空符



Alan Mathison Turing
1912 - 1954

English mathematician, computer scientist, logician, cryptanalyst, philosopher, and theoretical biologist.

2020年9月17日星期四

南京大学-计算机系统基础-PA

前言

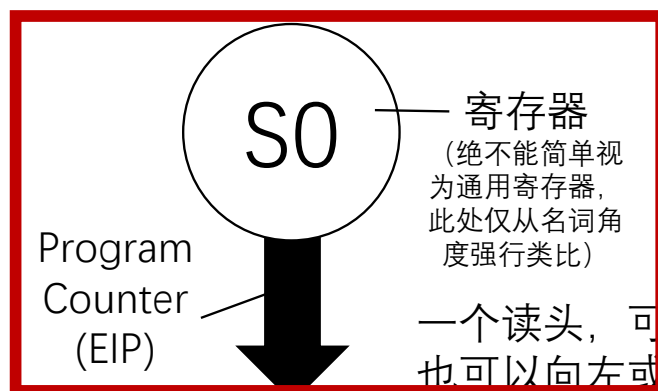
一个状态寄存器，可以表示在一个有限的状态集合中，当前机器所处的状态。存在一个初始状态 S0

一个包含有限条指令的表格（程序）

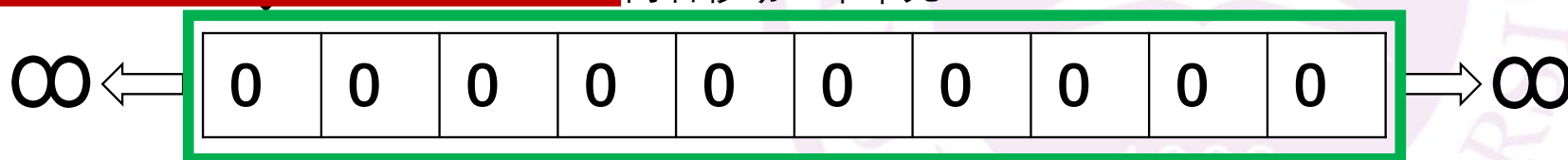
机器所处状态	读到的单元值	符号操作	读头移动方向	下一个状态
S0	0	-	-	停机
	1	-	右移	S1
S1	0	写1	右移	S2
	1	-	右移	S1
S2	0	-	左移	S3
	1	-	右移	S2
S3	0或1	写0	-	停机

程序
(算法)

CPU



一个读头，可以读写指向单元的符号，也可以向左或向右移动一个单元



主存

一条两端无限长的，分成单元的纸带，每个单元中可以填入一个有限符号集合（必包含 空符号）中的某一个符号

$\{0, 1\}$ - 其中0表示空符号

前言



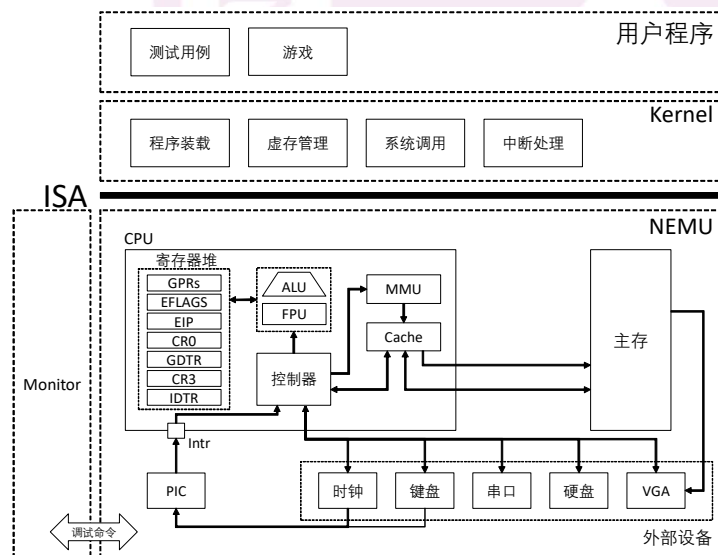
John von Neumann

1903 - 1957

Hungarian-American mathematician, physicist, computer scientist, and polymath.

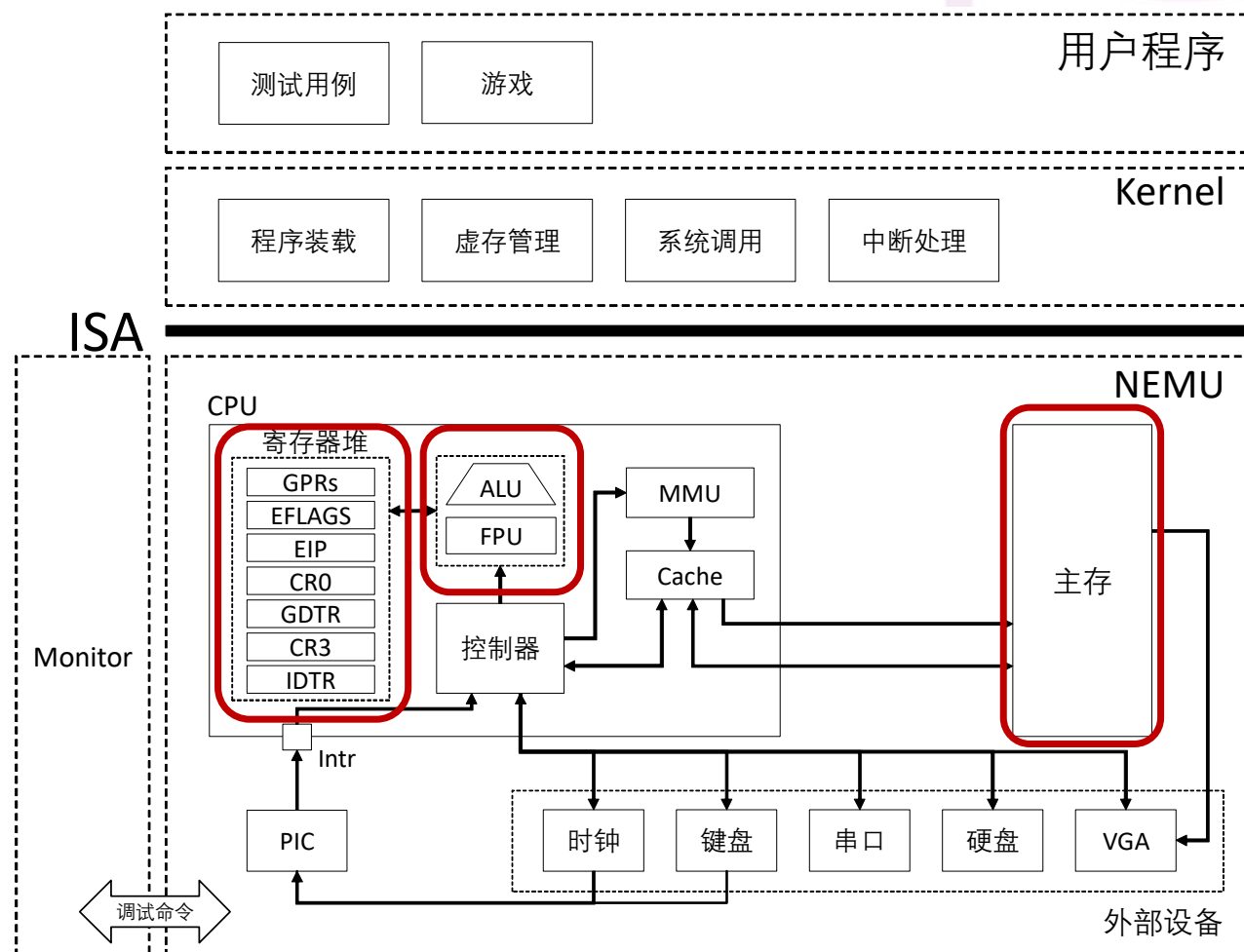
The **von Neumann architecture**:

1. A processing unit that contains an arithmetic logic unit and processor registers
2. A control unit that contains an instruction register and program counter
3. Memory that stores data and instructions
4. External mass storage
5. Input and output mechanisms



NEMU模拟i386
体系结构计算机,
符合冯·诺伊曼体
系结构

前言



PA 1涉及以下器件/功能模拟:

1. 主存 (PA 1-1中了解)
2. 寄存器 (PA 1-1任务)
3. ALU整数运算 (PA 1-2任务)
4. FPU浮点数运算 (PA 1-3任务)

目录

- PA 1-1 数据的类型和存取
- PA 1-2 整数的表示和运算
- PA 1-3 浮点数的表示和运算



目录

- PA 1-1 数据的类型和存取
- PA 1-2 整数的表示和运算
- PA 1-3 浮点数的表示和运算



数据的类型及其机器级表示

数据（真值）的类型

无符号整数

123, 0x8048000

带符号整数

-123, 123

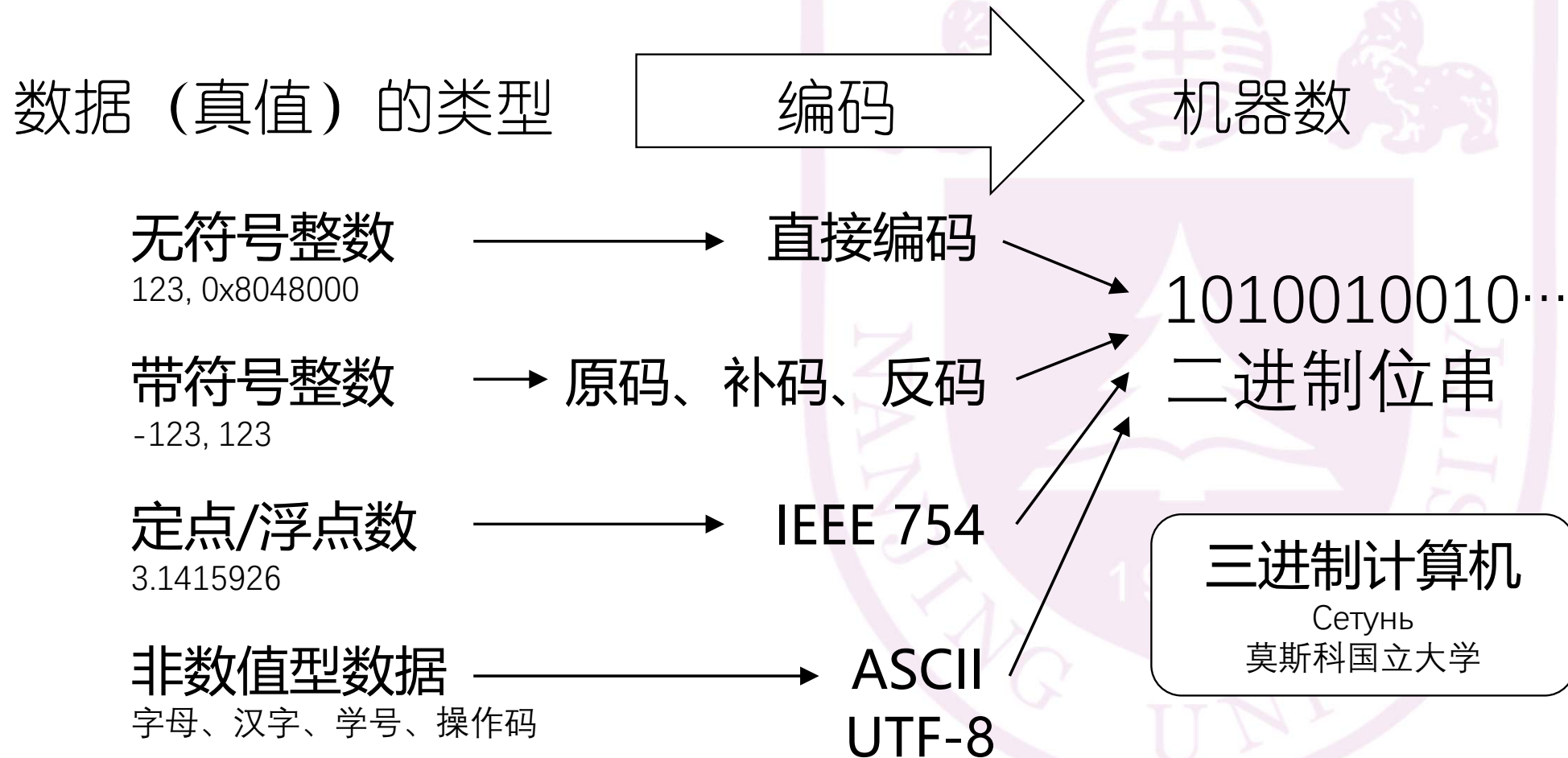
定点/浮点数

3.1415926

非数值型数据

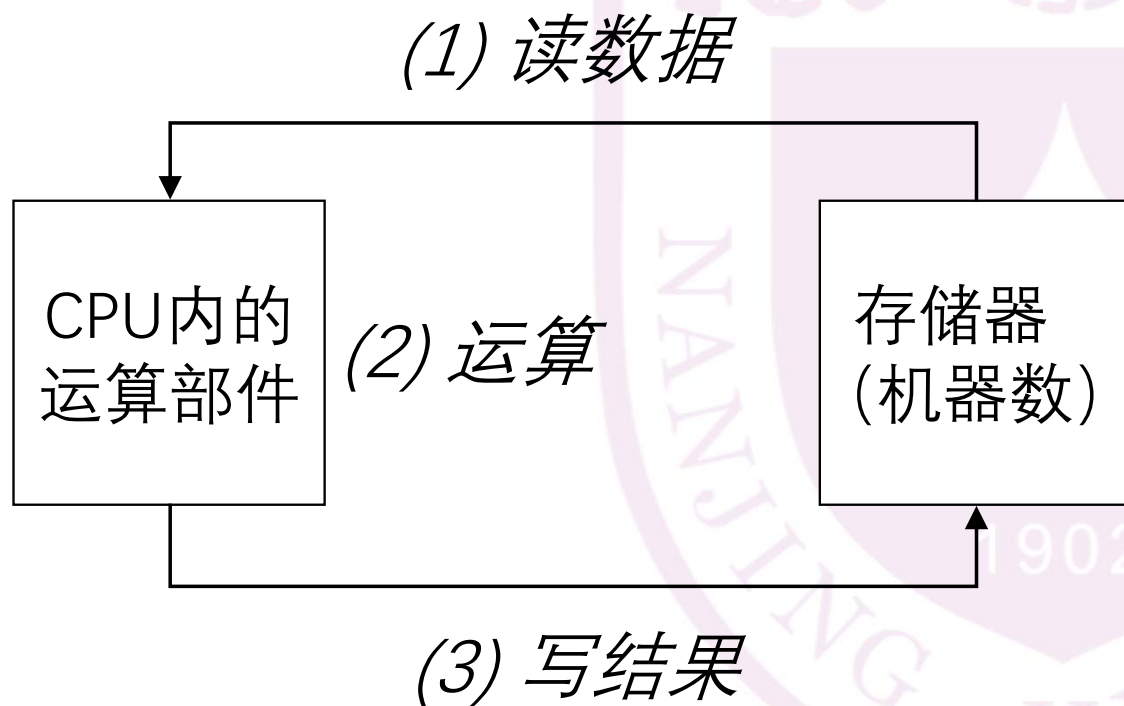
字母、汉字、学号、操作码

数据的类型及其机器级表示



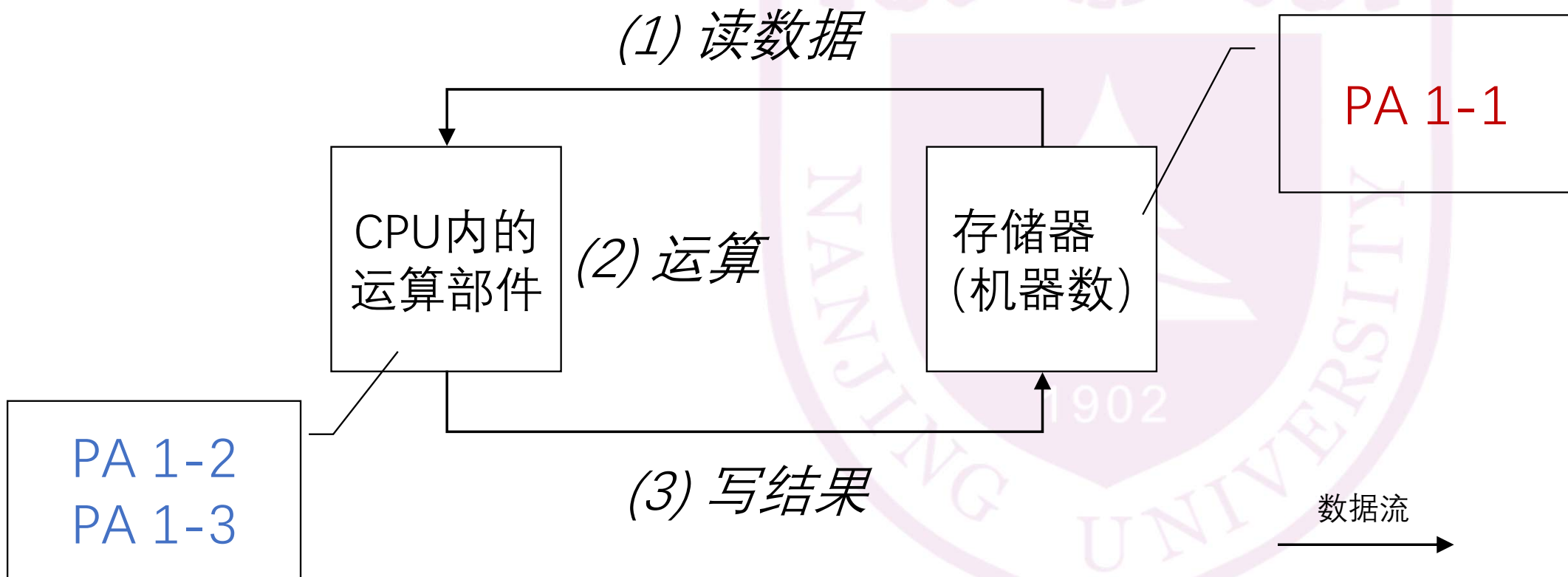
机器数的存取

- CPU对数据的处理

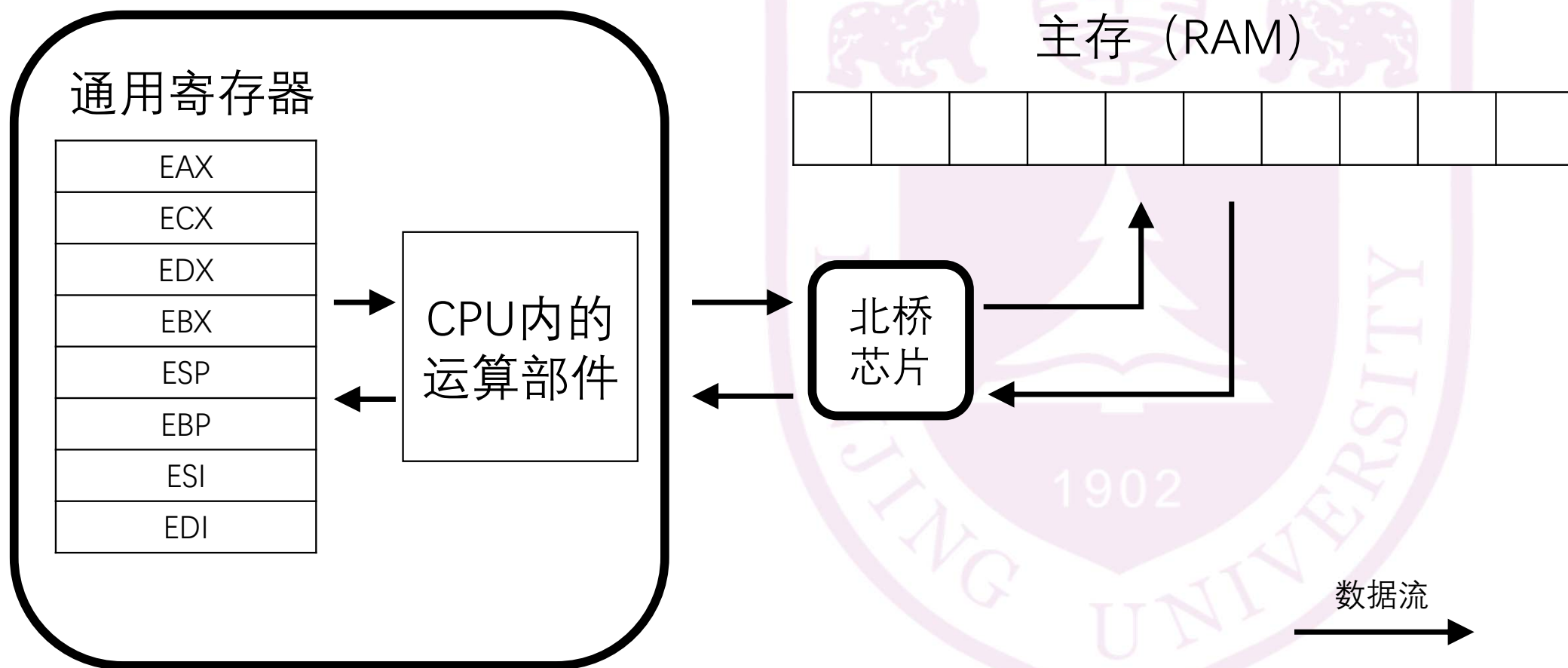


机器数的存取

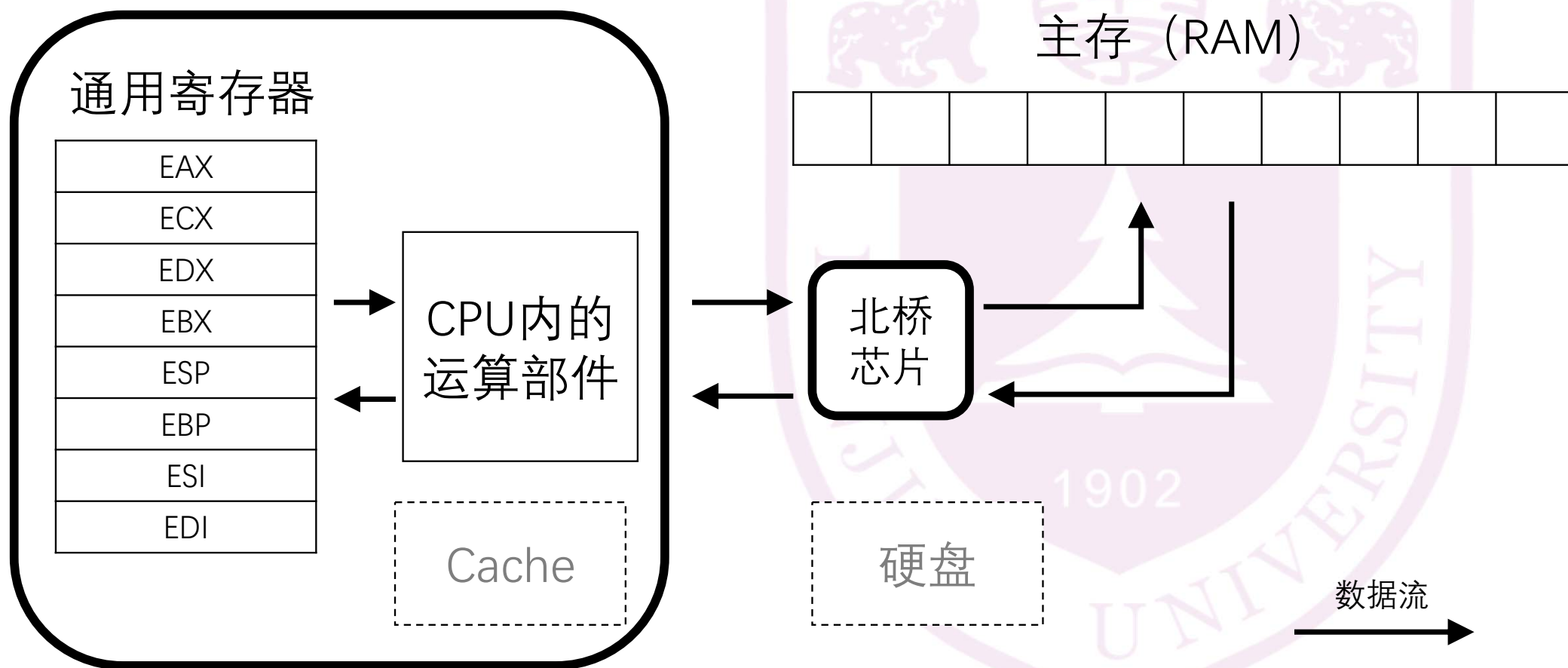
- CPU对数据的处理



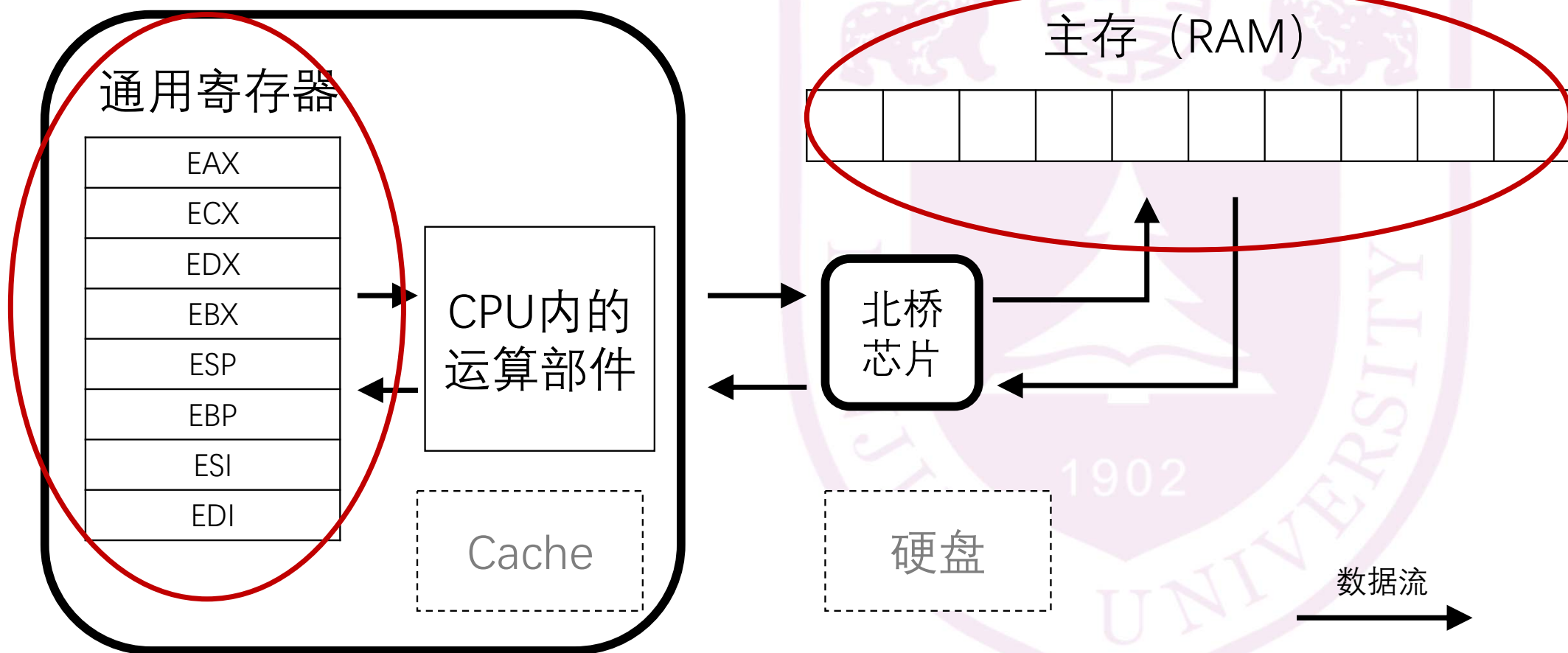
机器数的存取



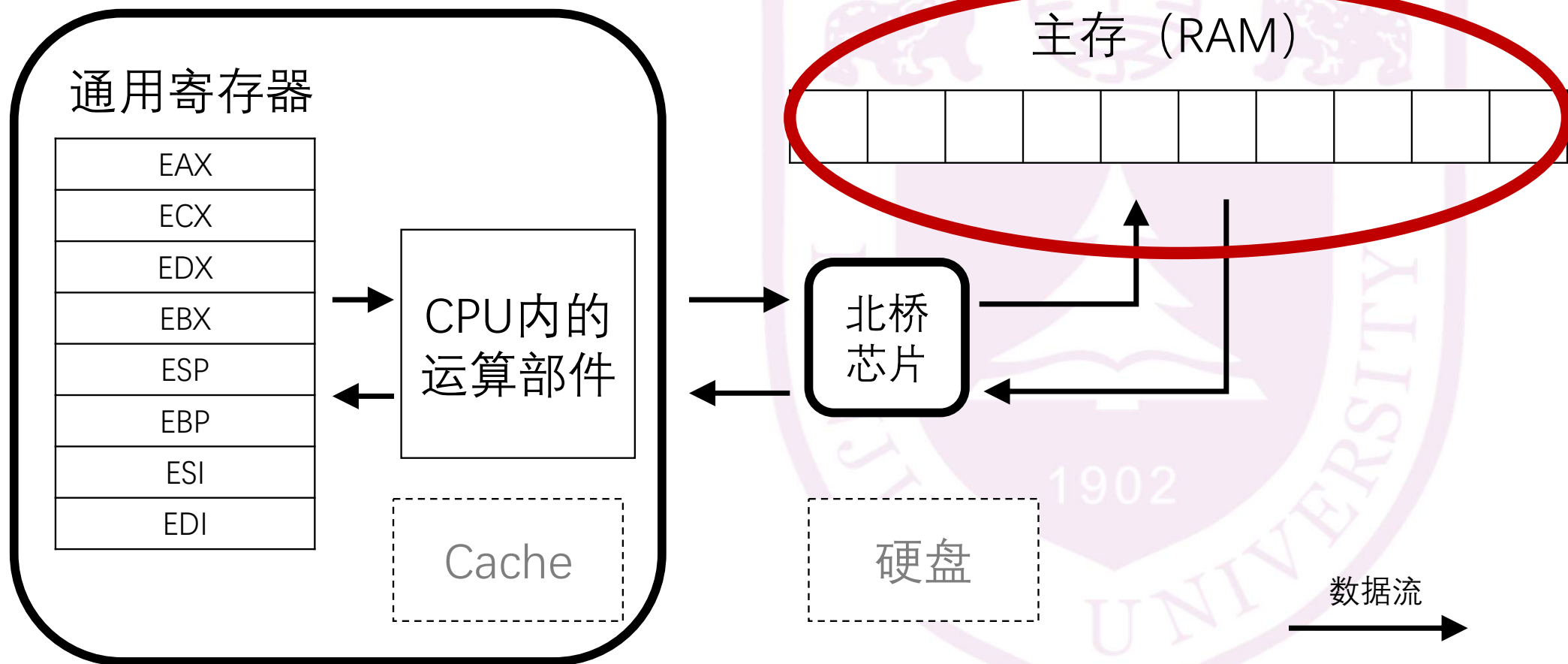
机器数的存取



机器数的存取



机器数的存取



数据的类型及其机器级表示

- 机器数存储的基本单位

比特 b: 0或1

字节 B: 8比特 (0xFF)

Intel i386

字 : 2个字节 (0x1234)

双字: 4个字节 (0x12345678)

低地址	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	高地址
	字节	字节	字节	字节	字节	字节	字节	字节	字节	字节	

主存 (RAM) 按字节编址

数据的类型及其机器级表示

- 机器数存储的基本单位

比特 b: 0或1

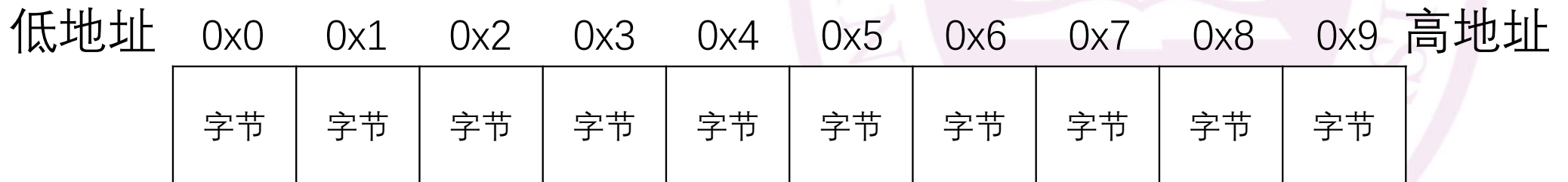
字节 B: 8比特 (0xFF)

Intel i386

字 : 2个字节 (0x1234)

双字: 4个字节 (0x12345678)

- 小端和大端方式: 0x 12 34 56 78



主存 (RAM) 按字节编址

数据的类型及其机器级表示

- 机器数存储的基本单位

比特 b: 0或1

字节 B: 8比特 (0xFF)

Intel i386

字 : 2个字节 (0x1234)

双字: 4个字节 (0x12345678)

- 小端和大端方式: 0x 12 34 56 78

低地址	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	高地址
	0x78	0x56	0x34	0x12	字节	字节	字节	字节	字节	字节	

小端: 低有效字节在低地址

主存 (RAM) 按字节编址

数据的类型及其机器级表示

- 机器数存储的基本单位

比特 b: 0或1

字节 B: 8比特 (0xFF)

Intel i386

字 : 2个字节 (0x1234)

双字: 4个字节 (0x12345678)

- 小端和大端方式: 0x 12 34 56 78

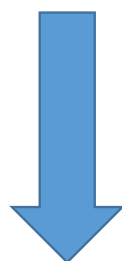
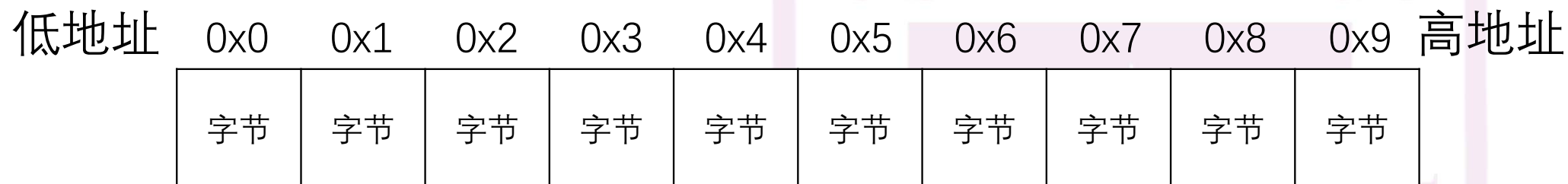
低地址	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	高地址
	0x12	0x34	0x56	0x78	字节	字节	字节	字节	字节	字节	

大端: 低有效字节在高地址

主存 (RAM) 按字节编址

存储器的模拟

- 对主存的模拟

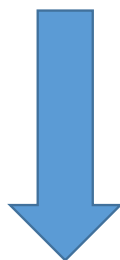
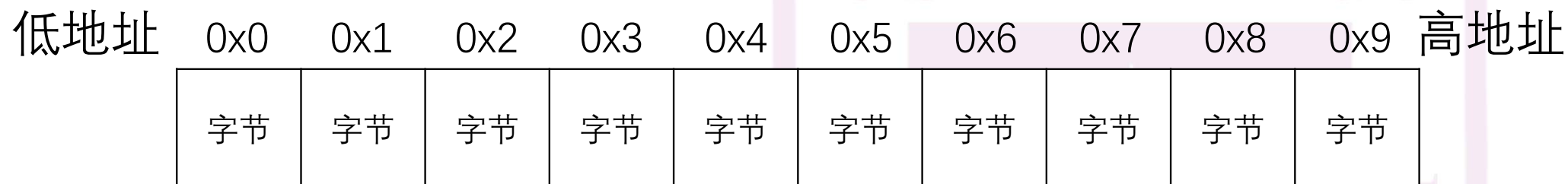


主存（RAM）按字节编址

如何用C语言进行模拟？

存储器的模拟

- 对主存的模拟



主存 (RAM) 按字节编址

如何用C语言进行模拟? => 数组

存储器的模拟

- 对主存的模拟

[nemu/src/memory/memory.c](#)

```
uint8_t hw_mem[MEM_SIZE_B] // 模拟主存, 128MB

// 顶层读接口
uint32_t vaddr_read(vaddr_t vaddr, uint8_t sreg, size_t len) { ... }

// 顶层写接口
void vaddr_write(vaddr_t vaddr, uint8_t sreg, size_t len, uint32_t data) { ... }
```

存储器的模拟

- 对主存的模拟


[nemu/src/memory/memory.c](#)

```
uint8_t hw_mem[MEM_SIZE_B] // 模拟主存, 128MB

// 顶层读接口
uint32_t vaddr_read(vaddr_t vaddr, uint8_t sreg, size_t len) { ... }

// 顶层写接口
void vaddr_write(vaddr_t vaddr, uint8_t sreg, size_t len, uint32_t data) { ... }
```

地址 (32位)



存储器的模拟

- 对主存的模拟

[nemu/src/memory/memory.c](#)

```
uint8_t hw_mem[MEM_SIZE_B] // 模拟主存, 128MB

// 顶层读接口
uint32_t vaddr_read(vaddr_t vaddr, uint8_t sreg, size_t len) { ... }

// 顶层写接口
void vaddr_write(vaddr_t vaddr, uint8_t sreg, size_t len, uint32_t data) { ... }
```

地址 (32位) 段寄存器 (PA 3-2)

存储器的模拟

- 对主存的模拟

[nemu/src/memory/memory.c](#)

```
uint8_t hw_mem[MEM_SIZE_B] // 模拟主存, 128MB

// 顶层读接口
uint32_t vaddr_read(vaddr_t vaddr, uint8_t sreg, size_t len) { ... }

// 顶层写接口
void vaddr_write(vaddr_t vaddr, uint8_t sreg, size_t len, uint32_t data) { ... }
```

地址 (32位) 段寄存器 (PA 3-2) 数据长度 (1、2、4字节)

存储器的模拟

- 对主存的模拟

[nemu/src/memory/memory.c](#)

```
uint8_t hw_mem[MEM_SIZE_B] // 模拟主存, 128MB

// 顶层读接口
uint32_t vaddr_read(vaddr_t vaddr, uint8_t sreg, size_t len) { ... }

// 顶层写接口
void vaddr_write(vaddr_t vaddr, uint8_t sreg, size_t len, uint32_t data) { ... }
```

读取的结果

要写的数据

存储器的模拟

- 对主存的模拟

[nemu/src/memory/memory.c](#)

```
uint8_t hw_mem[MEM_SIZE_B] // 模拟主存, 128MB

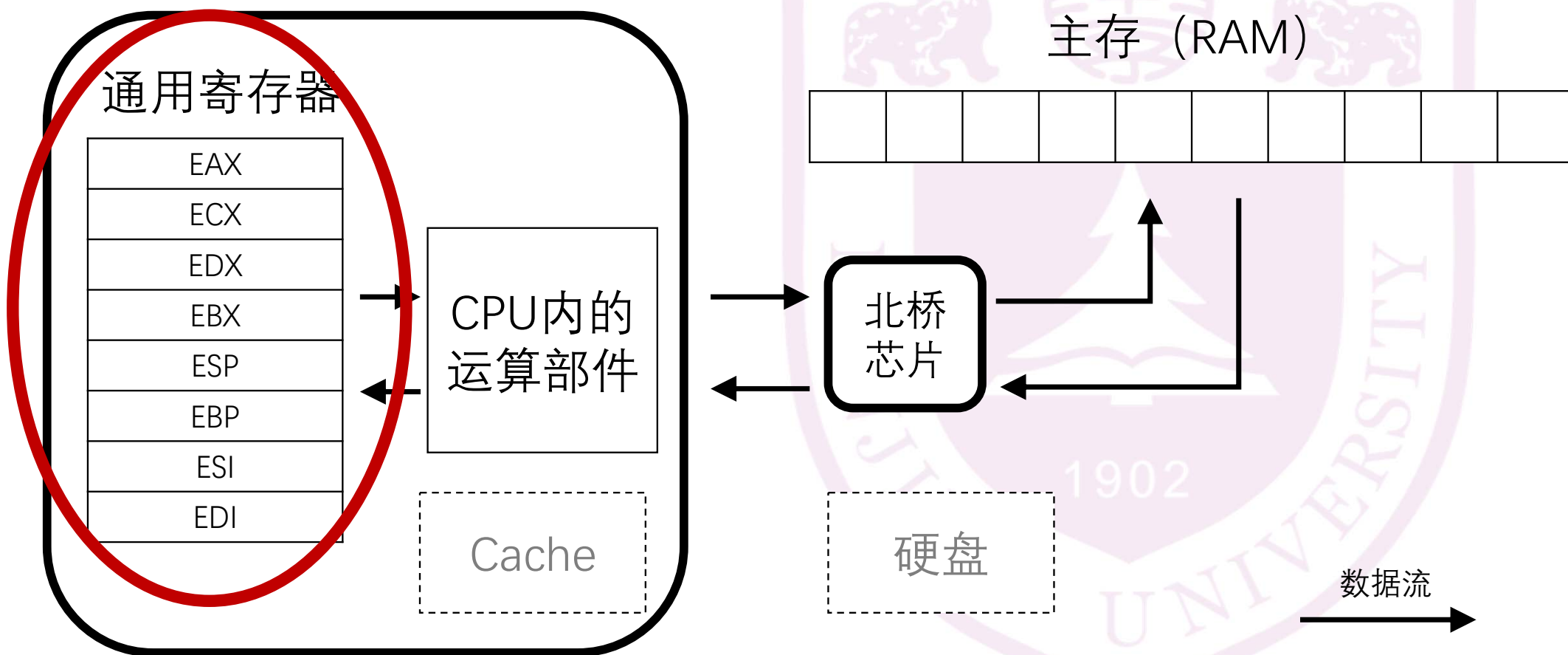
// 顶层读接口
uint32_t vaddr_read(vaddr_t vaddr, uint8_t sreg, size_t len) { ... }

// 顶层写接口
void vaddr_write(vaddr_t vaddr, uint8_t sreg, size_t len, uint32_t data) { ... }
```

`vaddr_write(0, 0, 1, 0x12);` \longrightarrow `hw_mem[0] = 0x12`

`printf("0x%x", vaddr_read(0, 0, 1));` \longrightarrow `0x12`

机器数的存取



存储器的模拟

- 对通用寄存器的模拟

31	23	15	7	0
		EAX	AH AX	AL
		EDX	DH DX	DL
		ECX	CH CX	CL
		EBX	BH BX	BL
		EBP	BP	
		ESI	SI	
		EDI	DI	
		ESP	SP	

General Purpose Registers:

```
movl $0x12345678, %eax
movl $0x87654321, %ecx
movb $0xFF, %al
movb $0x00, %ah
movw %ax, %cx
```

`%ecx = ?`

存储器的模拟

- 对通用寄存器的模拟

31	23	15	7	0
		EAX	AH AX	AL
		EDX	DH DX	DL
		ECX	CH CX	CL
		EBX	BH BX	BL
		EBP	BP	
		ESI	SI	
		EDI	DI	
		ESP	SP	

General Purpose Registers:

```
movl $0x12345678, %eax
movl $0x87654321, %ecx
movb $0xFF, %al
movb $0x00, %ah
movw %ax, %cx
```

`%ecx = $0x876500FF`

存储器的模拟

- 对通用寄存器的模拟

nemu/include/cpu/cpu.h

```
#include "cpu/reg.h"

extern CPU_STATE cpu;
```

nemu/src/cpu/cpu.c

```
#include "cpu/cpu.h"

CPU_STATE cpu;
```

```
typedef struct
{
    struct
    {
        struct
        {
            uint32_t _32;
            uint16_t _16;
            uint8_t _8[2];
        };
        uint32_t val;
    } gpr[8];
    struct
    { // do not change the order of the registers
        uint32_t eax, ecx, edx, ebx, esp, ebp, esi, edi;
    };
};

...

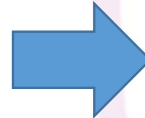
} CPU_STATE;
```

nemu/include/cpu/reg.h

存储器的模拟

- 对通用寄存器的模拟

```
movl $0x12345678, %eax
movl $0x87654321, %ecx
movb $0xFF, %al
movb $0x00, %ah
movw %ax, %cx
```



```
#include "cpu.h"
```

```
cpu.eax = 0x12345678
cpu.ecx = 0x87654321
cpu.eax._8[0] = $0xFF
cpu.eax._8[1] = $0x00
cpu.ecx._16 = cpu.eax._16
```



```
cpu.ecx = 0x876500FF
```

存储器的模拟

- 针对通用寄存器的测试用例：[nemu/src/cpu/reg.c](#)
- 执行测试用例

terminal

```
pa2020_fall$ make test_pa-1
```

顺序执行PA 1-1, 1-2,
1-3的测试用例

nemu: src/cpu/reg.c:91: reg_test: Assertion `r.val == (sample[REG_EAX] & 0xff)' failed.

存储器的模拟

- 对通用寄存器的模拟

```
typedef struct
{
    union
    {
        union
        {
            union
            {
                uint32_t _32;
                uint16_t _16;
                uint8_t _8[2];
            };
            uint32_t val;
        } gpr[8];
        struct
        { // do not change the order of the registers
            uint32_t eax, ecx, edx, ebx, esp, ebp, esi, edi;
        };
    };
} CPU_STATE;
```

nemu/include/cpu/reg.h

存储器的模拟

- 对通用寄存器的模拟

PA 1-1 pass

PA 1-2 fail

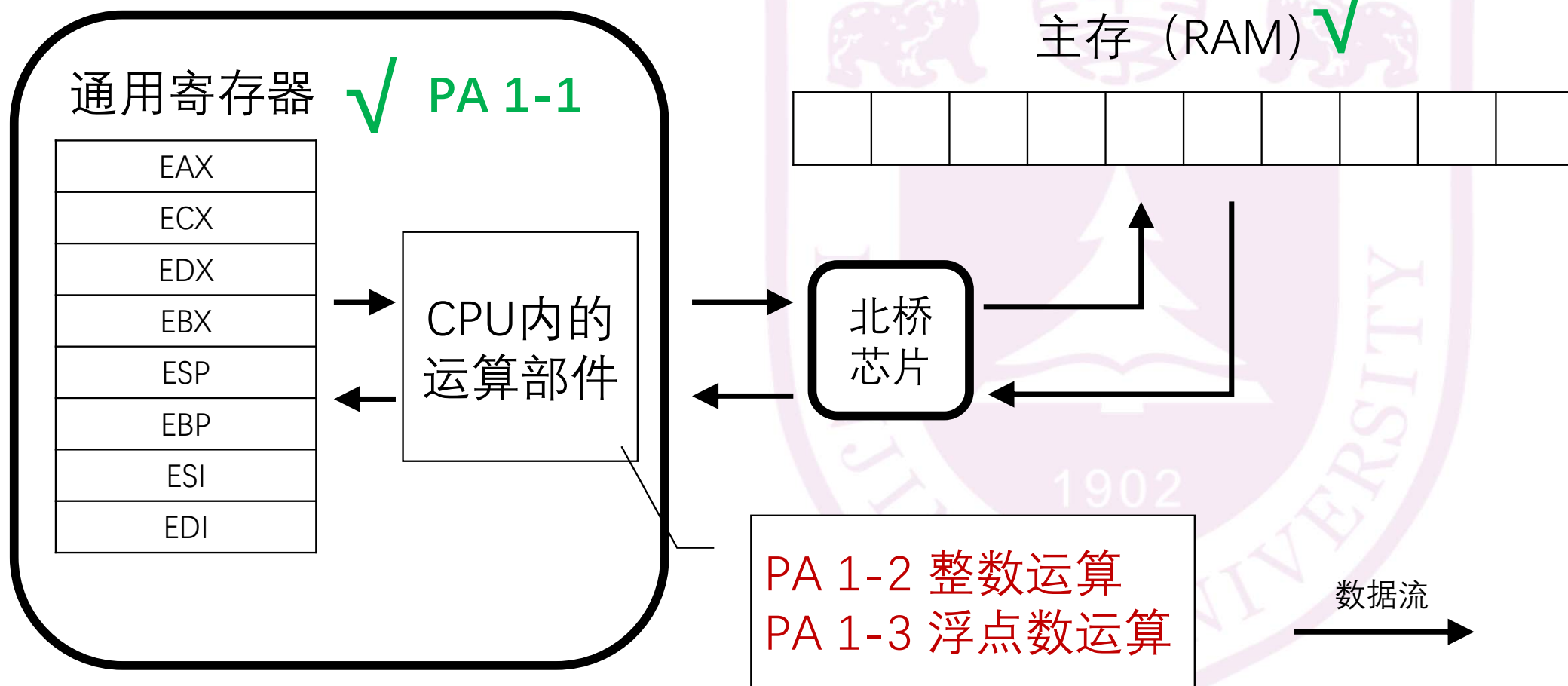
terminal

```
pa2020_fall$ make clean
pa2020_fall$ make test_pa-1

./nemu/nemu --test-reg
NEMU execute built-in tests
reg_test()          pass

./nemu/nemu --test-alu add
NEMU execute built-in tests
Please implement me at alu.c
```

机器数的存取





PA 1-1 结束