MIPS：指令类型

R类型：

←6→	←5→	←5→	←5→	←5→	←6→
000000	rs	rt	rd	shamt	func

ADD $rd, $rs, $rt ：$rd = $rs + $rt

还有：SUB, SLT, AND, OR, SLL, SLR, JR,

I类型：

←6→	←5→	←5→	←16→
op	rs	rd	立即数

LW $rd, Data($rs) ：$rd = M[$rs + Data]

SW $rd, Data($rs) ：M[$rs+Data] = $rd

BEQ $rs, $rd, add ：PC+=4; if($rs==$rd) PC+=add*4;

BNE ———— 不等 ————

J类型：

←6→	←26→
op	address

J add：PC = ((PC+4)&0xF0000000)|(add*4))

R：SLT $rd, $rs, $rt ：if($rs<$rt) $rd=1 else $rd=0

JR $rs ：PC=$rs

J：JAL add ：$rs=PC+4; J add; } 过程序调用与返回

lui rt, data

寻址模式

立即数寻址：操作数位于指令中的常数

寄存器寻址：操作数是寄存器

基址寻址/偏移寻址，操作数在内存中，其地址是指令中基址寄存器与常数的和

PC相对寻址：地址是PC和指令中常数的和

伪直接寻址：跳转地址由指令中26位字段和PC高位连成

ASCII码：

0~31和127 → 控制字符或通信专用字符

48~57 ⟶ 0~9　65~90 ⟶ A~Z　97~122 → a~z

汉字：GB2312

区号位号 + 0X20 ⟶ 国标码

国标码 + 0X80 ⟶ 汉字内码

浮点：

Exponent	significand	
0	0	±0
0	nonzero	Denorms：非规格化数
255	0	±∞
255	nonzero	NaN

E的范围：-127~127

双精度：1~11~52　$(-1)^S \times 1.M \times 2^{E-2013}$

M[R[bx]+R[di]-6] ← R[cl]

OF—CF

$OF = C_n \oplus C_{n-1}$　$CF = C_{out} \oplus C_{in}$　($C_n = C_{out}$)

无符号加溢出：CF=1；带符号加溢出：OF=1

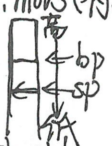
X86汇编　传送指令

mov[bx+di-6], cl / movb %cl, -6(%bx, %di)

mov, movs(符号扩展), movz(零扩展), XCHG(交换), Push, pop

入栈　pushw %ax　R[sp] ← R[sp]-2　M[R[sp]] = R[ax]

出栈：popw %ax　R[ax] ← M[R[sp]]　R[sp] ← R[sp]+2



push ebp

mov ebp, esp　R[ebp] ← R[esp]

定点算术指令

ADD, SUB, INC, DEC, NEG, CMP, MUL/IMUL, DIV/IDIV

IMUL, IDIV：带符号　cmp ax, bx ；ax-bx　ax比bx(大小)

乘法指令：给出1个操作数，另1个在AL/AX/EAX中　n*n = 2n位

结果：AX(16位)　DX-AX(32位)　EDX-EAX(64位)
　　　　　　　　高　低

除法指令：只明显指出除数，用EDX-EAX中的内容除以除数

8位：16位被除数AX，商AL，余数AH；16位：32位被除数DX-AX，商AX，余数DX；32位：64位被除数EDX-EAX，商EAX，余数EDX

按位运算指令：TEST，做"与"，仅影响flag

SHL/SHR：逻辑左/右移　SAL/SAR：算术左右移，左移判溢出，右移高位补符　ROL/ROR：循环左右移　RCL/RCR 带进位循环，将CF作为操作数　[CF ← ☐]　移出位送CF

控制转移指令

无条件JMP，调用 CALL DST，返回地址RA入栈，转到DST

返回 RET，从栈中取回地址RA，转到RA

jc	CF=1 有进位/借位	ja/jnbe	CF=0 AND ZF=0	A>B	无符号	
jnc	CF=0 无 ——	jae/jnb	CF=0 OR ZF=1	A≥B		
je/jz	ZF=1 相等/等于0	jb/jnae	CF=1 AND ZF=0	A<B		
jne/jnz	ZF=0 不 ——	jbe/jna	CF=1 OR ZF=1	A≤B		
js	SF=1 负数	jg/jnle	SF=OF AND ZF=0	A>B	带符号	
jns	SF=0 不负	jge/jnl	SF=OF OR ZF=1	A≥B		
jo	OF=1 溢出	jl/jnge	SF≠OF AND ZF=0	A<B		
jno	OF=0 无溢出	jle/jng	SF≠OF OR ZF=1	A≤B		

```
ax = cx*bx + dx        1000/3 = x...x        mov bx,80
mov ax,bx              mov ax,1000           mul bx    ax*80 = ax*(16+64)
push dx                xor dx,dx             mov bx,ax
mul cx                 mov bx,3              shl bx,4
pop dx                 div bx                shl ax,6
add ax,dx                                    mov ax,bx
```

```
显示12345
    mov bx,12345
    xor cx,cx
    mov di,10       ; 除数
    mov ax,bx
R1: xor dx,dx
    div di          ; ax...dx
    add dx,'0'      ; 数→字符
    push dx
    inc cx          ; 计数
    cmp ax,0
    jnz R1
R2: pop dx
    mov ah,2
    int 21h
    dec cx
    jnz R2
```

```
int sum(int*x, int n){
    int i,s=0;
    for(i=0,i<n,(i++)
      s+=x[i];
}

    push bp
    mov bp,sp
    sub sp,-4
    mov word ptr[bp-4],0   ;s
    mov word ptr[bp-2],0   ;i
    mov bx,[bp+6]          ;&x[0]
    mov es,[bp+8]          ;seg
    mov cx,[bp+10]         ;n
R0: cmp [bp-2],cx          ;i<n
    jnl Exit               int: 2 bytes
    mov si,[bp-2]
    add si,[bp-2]          i*size(int)
    mov ax,es:[bx+si]      x[i]
    add [bp-4],ax          s+=x[i]
    inc word ptr [bp-2]    i++
    jmp R0
Exit:
    mov ax,[bp-4]
    mov sp,bp
    pop bp
    ret
```

computer componets：input output memory control datapath five classic

sign-magnitude 原码　biased notation 移码　2'complement 补码

过程调用

EAX, EDX, ECX调用者保存；EBX, ESI, EDI被调用者保存

call add　R[esp] ← R[esp]-4　M[R[esp]] ← 返回地址　R[eip] ← add首地址

过程开始：push ebp　mov ebp esp

返回地址
EBP在main中值

EBP →

返回地址是call的下一条指令的地址

Cache：4GB，cache：64KB　Block：16B　32，16，4

direct：| 主存标记TAG | Cache索引 Index | 块内地址 ofs |

TAG = 32-16 = 16　Blocks = 64KB/16 = 4K　Total = (1+16)×4K/8+64K

full：| 主存标记TAG = 主存块号 | 块内地址 ofs |　TAG = 32-4 = 28

4-way Set：TAG = 32-16+2 = 18　Sets = 4K/4 = 1K　Set-index

写策略：write through，在改写存储器内容时，cache与主存一起改；write back：只写cache，在替换时，再将数据写到主存

替换策略：LRU(最近最少)　LFU(Least Frequently)　FIFO先进先出

cache失配：compulsory：第一次启动时，总是失配；capacity：cache容量小于主存，总是失配；conflict 相联失配，由于映射关系影响

cache存取时间：$T = T_c + (1-H_i)T_m$；T_c：time for cache；T_m：time for main memory；H_i：rate of hit for instruction，H_d：rate of hit for data；p = percent for LW/SW in the program。$T = T_i + T_d * p$

$= (T_c + (1-H_i)T_m) + (T_c + (1-H_d)T_m) * p$

FIFO命中率不随组数增加而提高，LRU随提高　Write Allocate：将主存块装入cache，然后更新　Not Write Allocate：直接写主存，不装入cache

虚拟存储 单页映射：整个物理存储器为一页 反向页表：在物理存储器端建页表 正向页表：在虚拟地址端建页表

TLB：把经常要查的页表项放到cache中。

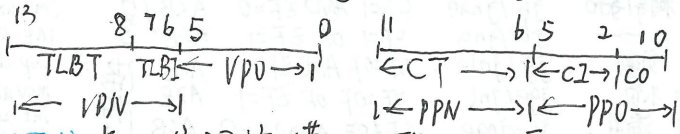
Virtual Address (tag)	Phisical Address (对应物理页框号)	...

CPU访存时，地址中虚页号被分成 tag+index，tag用于和TLB页表项中tag比较，index用于定位需比较的表项

VA中：TLBI（TLB索引） TLBT（TLB标记） VPO（页内偏移地址） VPN（虚拟页号） PA中：PPO（页内偏移地址） PPN（物理页号） CO（块内偏移地址） CI（cache索引） CT（cache标记）

例：14-bit V；12-bit P；页64B 共 $2^{14-6}=256$ 页

TLB：16项，4路组相连 cache：16行，block4B，直接映射

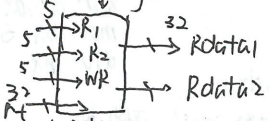


段页式：虚拟空间按模块分段，段内再分页

CPU 逻辑门：与： 或： 非：

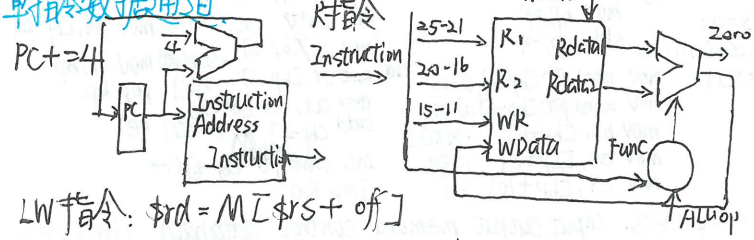
异或：

寄存器组：

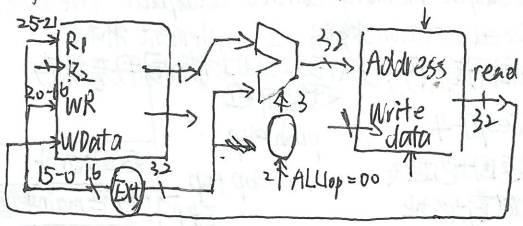


$rd=$rs Func $rt

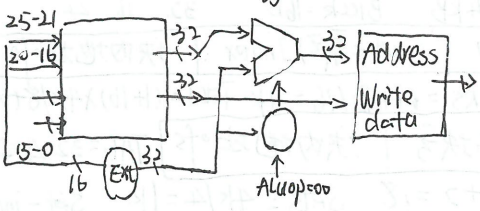
单指令数据通道



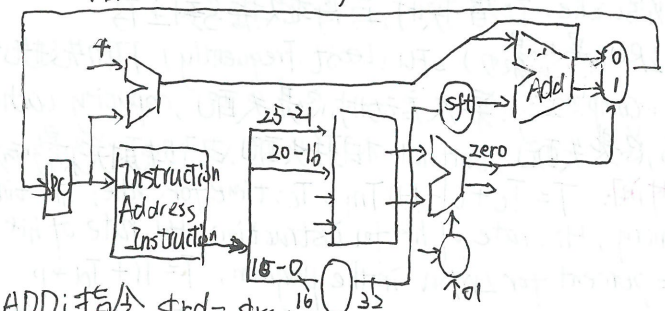
LW指令：$rd=M[$rs+off]



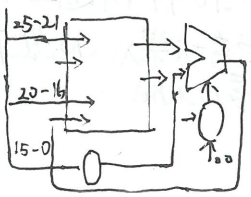
SW指令 M[$rs+off]=$rt



BEQ指令 PC+=4; if($r1==$r2) PC+=address*4



ADDi指令 $rd=$rs+Data



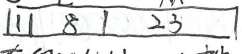
IO：Polling CPU主动查询 I/O Interrupt DMA：数据传送无需CPU参与

pipeline：Structural Hazard：inadequate hardware to simultaneously support all instructions in the pipeline in the same clock cycle. Control Hazard：need to make a decision based on... ①stall the ~~piplin~~ pipeline ②Predict branch outcome ③delayed branch Data Hazard：instruct needs data from the result of a previous instruction

4M*1 adr:22 dat:1；8M*4 adr:23 dat:4

整数：除基取余，上右下左；小数：乘基取整，上右下左

IEEE 754



$(-1)^S(1+M)2^{E-127}$

大端：高位在低地址 小端：低位在高地址

CBW AL→AX；CWDE AX→EAX；CWD：AX→DX:AX

ADC DI,0；DI=DI+0+CF