Registros

32 bits	16 bits	8 bits
%eax	%ax	%ah, %al
%ebx	%bx	%bh, %bl
%ecx	%cx	%ch, %cl
%edx	%dx	%dh, %dl
%edi	%di	
%esi	%esi	

- Regs que se guardan: %ebx, %esi, %edi
- Regs que no hace falta: %eax, %ecx, %edx
- Resultados siempre en %eax

Instrs. movimiento datos

```
MOVx op1,op2        op2 ← op1
MOVSxy op1,op2      op2 ← ext(op1)
MOVZxy op1,op2      op2 ← extzero(op1)
PUSHL op1
POPL op1
LEAL op1,op2        op2 ← &op1
```

Instrs. aritméticas

```
ADDx op1,op2        op2 ← op2+op1
SUBx op1,op2        op2 ← op2-op1
ADCx op1,op2        op2 ← op2+op1+CF
SBBx op1,op2        op2 ← op2-op1-CF
INCx op1            op1 ← op1+1
DECx op1            op1 ← op1-1
NEGx op1            op1 ← -op1
IMUL op1,op2        op2 ← op2*op1
IMUL imm,op1,op2    op2 ← imm*op1
IMULL op1           %EDX:%eax ← op1*%eax (enteros)
MULL op1            %edx:%eax ← op1*%eax (naturales)
CLTD o CDQ?..       %edx:%eax ← extsign(%eax)
IDIVL op1 (enteros) %edx:%eax ← %edx:%eax / op1
                    %edx:%eax ← %edx:%eax % op1
DIVL op1 (naturales) %edx:%eax ← %edx:%eax / op1
                     %edx:%eax ← %edx:%eax % op1
```

Compilar

```
gcc -m32 -o nombre nombre.c nombre.s
```

Debug

Añadir flag -g a gcc → sudo ddd ./ejecutable

Medir taskset -c 1 valgrind --tool=lackey ./prog

Acceso matrices

M[i][j] = @M + T(i*NC + j)

M[i][j][k] = @M + T(i*NC + j*NF + k)

Preámbulo función

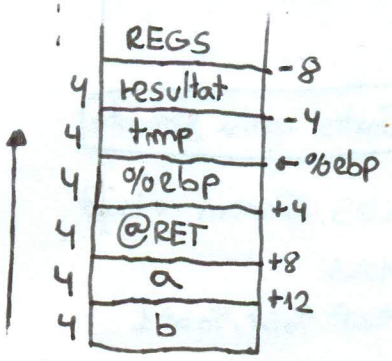
```
.section .text
.globl ejemplo
.type ejemplo, @function
ejemplo:
  pushl %ebp
  movl %esp, %ebp
  subl □, %esp → □   ← # bytes variables locales

  call blabla
  addl Δ, %esp → Δ   ← # bytes parámetros para el call

  movl %ebp, %esp
  popl %ebp
  ret
```

Ejemplo pila

int func(int a, int b)



Instrs. lógicas

```
ANDX op1,op2        op2 ← op2 & op1
ORX op1,op2         op2 ← op2 | op1
XORx op1,op2        op2 ← op2 ∧ op1
NOTx op1            op1 ← ~op1
SALx k,op1          op1 ← op1 <<K (aritmético)
SHLx K,op1          op1 ← op1 <<k (lógico)
SAR K,op1           op1 ← op1 >>K (aritmético)
SHR K,op1           op1 ← op1 >>K (lógico)
CMPx op1,op2
TESTx op1,op2       op2&op1
```

Instrs. secuenciamiento

JMP etiq	JE JNE	equal / not equal
CALL etiq	JG JGE	greater / greater equal
RET	JL JLE	less / less equal

Instrs. SIMD

```
PSUBB → substract packed byte integers
PCMPGTB → compare packed signed byte integers
MOVDQA → move aligned double quadword
MOVDQU → move unaligned double quadword
EMMS → empty mmx state
```

Registros SIMD (128 bits) %xmm0 ..... %xmm7

• Structs siempre se alinean a un múltiplo de su tipo de dato más grande

char (1 byte) short (2 bytes) int (4 bytes) puntero (4 bytes) double (8 bytes) long double (12 bytes)

↓ ! se alinea a 4 ↓ ! se alinea a 4

Instrs. coma flotante

FLDS @mem → load
FMULS
FMULP %st, %st1
FADDS
FADDP %st, %st1
FSTPS @mem → store

↓

La P es de POP, generalmente hacen pop de cada operando y pushean resultado a la pila (en %st ≡ %st(0))

Registros c.f

%st(0) ≡ %st ← top pila
%st(1)
%st(2)
⋮
%st(7)

Carga por variable

```
numero: db 1.372  → byte
        dw ...    → word
        dd ...    → dword
        :
fld numero
```

Fórmulas

$$MIPS = \frac{\#instrs}{t_{exec} \cdot 10^6} \qquad MFLOPS = \frac{\#instrs\ c.f}{t_{exec} \cdot 10^6} \qquad t_{exec} = \#instrs \cdot CPI \cdot \frac{1}{f_{clock}}$$

llei d'Amdahl $S_T = \frac{1}{(1-F_m) + \frac{F_m}{S_m}}$

S_T → speedup total
F_m → fracción tiempo función
S_m → speedup función (∞ si t=0)