Arquitecturas Computacionales

Formatos e instrucciones en Assembler

Facultad de Ingeniería / Escuela de Informática
Universidad Andrés Bello, Viña del Mar.

Assembler

```
int simple(int *xp, int y)
{
  int t = *xp + y;
  *xp = t;
  return t;
}
```

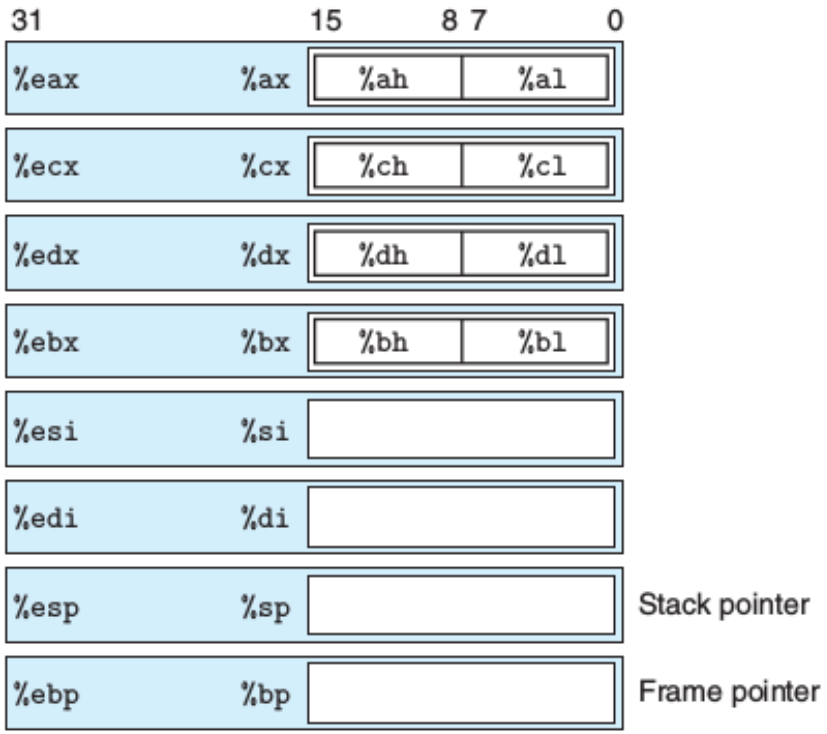
```
simple:
  pushl   %ebp              Save frame pointer
  movl    %esp, %ebp        Create new frame pointer
  movl    8(%ebp), %edx     Retrieve xp
  movl    12(%ebp), %eax    Retrieve y
  addl    (%edx), %eax      Add *xp to get t
  movl    %eax, (%edx)      Store t at xp
  popl    %ebp              Restore frame pointer
  ret                       Return
```

Tipos de datos en IA32

C declaration	Intel data type	Assembly code suffix	Size (bytes)
`char`	Byte	`b`	1
`short`	Word	`w`	2
`int`	Double word	`l`	4
`long int`	Double word	`l`	4
`long long int`	—	—	4
`char *`	Double word	`l`	4
`float`	Single precision	`s`	4
`double`	Double precision	`l`	8
`long double`	Extended precision	`t`	10/12

UNIVERSIDAD ANDRES BELLO

Registros en IA32

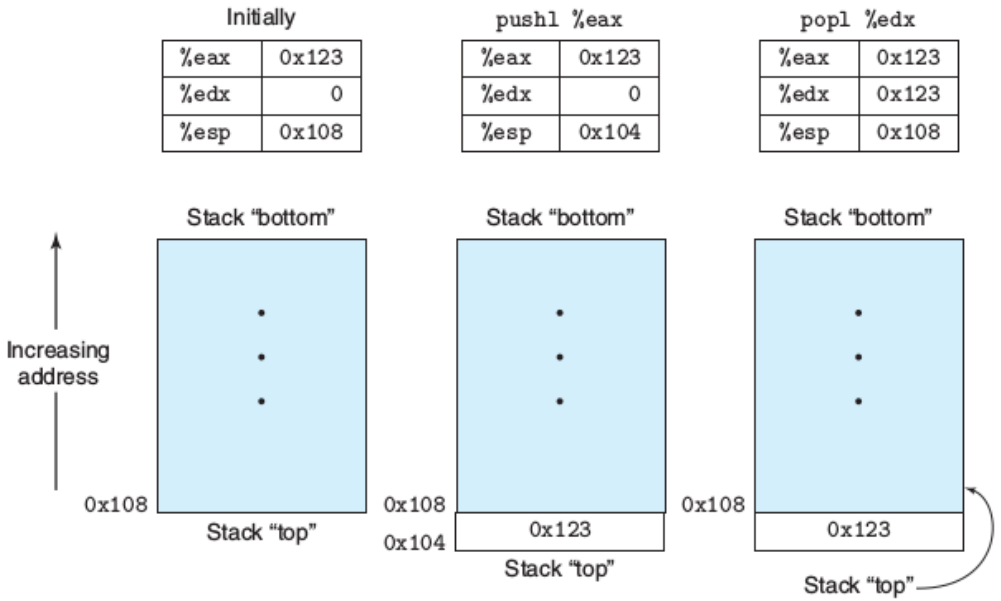


Instrucciones de movimiento de datos

Instruction		Effect
MOV	S, D	$D \leftarrow S$
`movb`		Move byte
`movw`		Move word
`movl`		Move double word
MOVS	S, D	$D \leftarrow \text{SignExtend}(S)$
`movsbw`		Move sign-extended byte to word
`movsbl`		Move sign-extended byte to double word
`movswl`		Move sign-extended word to double word
MOVZ	S, D	$D \leftarrow \text{ZeroExtend}(S)$
`movzbw`		Move zero-extended byte to word
`movzbl`		Move zero-extended byte to double word
`movzwl`		Move zero-extended word to double word
`pushl`	S	R[%esp] $\leftarrow$ R[%esp] $-$ 4; M[R[%esp]] $\leftarrow S$
`popl`	D	$D \leftarrow$ M[R[%esp]]; R[%esp] $\leftarrow$ R[%esp] $+$ 4

UNIVERSIDAD ANDRES BELLO

Operación del stack



- Se ilustra desde abajo hacia arriba
- push: guarda en memoria y decrementa el puntero a stack (registro %esp)
- pop: lee de memoria e incrementa el puntero a stack

Equivalencia de instrucciones

- pushl %ebp
  - subl $4, %esp
  - movl %ebp,(%esp)
- popl %eax
  - movl (%esp), %eax
  - addl $4, %esp

Operaciones aritméticas

Instruction		Effect	Description
`leal`	S, D	$D \leftarrow \&S$	Load effective address
INC	D	$D \leftarrow D + 1$	Increment
DEC	D	$D \leftarrow D - 1$	Decrement
NEG	D	$D \leftarrow -D$	Negate
NOT	D	$D \leftarrow \sim D$	Complement
ADD	S, D	$D \leftarrow D + S$	Add
SUB	S, D	$D \leftarrow D - S$	Subtract
IMUL	S, D	$D \leftarrow D * S$	Multiply
XOR	S, D	$D \leftarrow D \hat{} S$	Exclusive-or
OR	S, D	$D \leftarrow D \mid S$	Or
AND	S, D	$D \leftarrow D \,\&\, S$	And
SAL	k, D	$D \leftarrow D << k$	Left shift
SHL	k, D	$D \leftarrow D << k$	Left shift (same as SAL)
SAR	k, D	$D \leftarrow D >>_A k$	Arithmetic right shift
SHR	k, D	$D \leftarrow D >>_L k$	Logical right shift

Operaciones unarias y binarias

- x++
  - incl (%esp): elemento de 4 bytes en el tope del stack se incrementa
  - subl %eax, %edx : valor en registro %edx se decrementa por el valor en %eax

Ejemplo

```
1   int arith(int x,
2             int y,
3             int z)
4   {
5       int t1 = x+y;
6       int t2 = z*48;
7       int t3 = t1 & 0xFFFF;
8       int t4 = t2 * t3;
9       return t4;
10  }
```

```
    x at %ebp+8, y at %ebp+12, z at %ebp+16
1       movl    16(%ebp), %eax          z
2       leal    (%eax,%eax,2), %eax     z*3
3       sall    $4, %eax                t2 = z*48
4       movl    12(%ebp), %edx          y
5       addl    8(%ebp), %edx           t1 = x+y
6       andl    $65535, %edx            t3 = t1&0xFFFF
7       imull   %edx, %eax              Return t4 = t2*t3
```

Ejercicio

Defina el código correspondiente en C, para el siguiente programa en assembler (IA32).

```
funcion2:
    pushl %ebp
    movl %esp, %ebp
    addl %8, %esp
    movl 4( %ebp), %edx
    movl 8( %ebp), %eax
    addl %eax, %1
    addl %eax, %edx
    movl ( %edx),( %eax)
    movl %ebp, %esp
    popl %ebp
    ret
```

UNIVERSIDAD ANDRES BELLO

Solución

```
int funcion2(int x, int y){
    y += 1;
    y += x;
    y = x;
    return (y);
}
```

Solución

Se destaca que al optimizar la función queda como: int

```
funcion2(int x, int y){
    return (x);
}
```