Universidad de los Andes

Ingeniería de Sistemas y Computación
ISIS1304 – Fundamentos de Infraestructura Tecnológica
Banco – Ensamblador procedimientos

Capacidades evaluadas:

- *Creación y destrucción de ambientes*
- *Direccionamiento en la pila*
- *Llamado de procedimientos*

En la siguiente tabla, escriba en las columnas de la derecha el código ensamblador que el compilador debe generar para implementar las sentencias de la columna izquierda. La columna de la izquierda es un solo programa.

Para referenciar las variables, debe usar desplazamientos en la pila con respecto al `ebp`; no puede usar los nombres simbólicos. Todas las modificaciones de las variables se deben reflejar en memoria.

En cada fila, escriba las instrucciones correspondientes en la columna 2 y si no le alcanza, siga en la columna 3

Código	**Columna 2**	**Columna 3**

`int f ( int *a,`
`  int *b,`
`  int n  ) {`
 	
`  int t, i;`
	`push ebp`
	`mov ebp, esp`
	`sub esp, 8`
	`push esi`
	`push edi`
	`push ebx`	

`  if (n == 0)`
`  {`
`  return 0;`
`  }`
	`mov eax, [ebp+16]`
	`cmp eax, 0`
	`je finf`	

`  else`
`  {`
 	
`  i = n - 1;`
	`dec eax`
	`mov [ebp-4], eax`	

`  t = a[i] * *(b + i);`	`mov esi, [ebp+8]`
	`mov ebx, (esi+4*eax]`
	`mov edi, [ebp+12]`
	`imul ebx, (edi+4*eax]`
	`mov [ebp-8], ebx`	

`  t += f(a, b, i);`	`push eax`
	`push edi`
	`push esi`
	`call f`
	`add esp, 12`
	`add ebx, eax`
	`mov [ebp-8], ebx`	

`  return t;`
 	
`  }`
	`mov eax, ebx`
	`finf:`
	`pop ebx`	

} 	pop edi pop esi add esp, 8 pop ebp ret	

Código	Columna 2	Columna3

int f (int *t, int n, int b){
  int x;
	push ebp
	mov ebp, esp
	sub esp, 4
	push ebx	

if (n == 0)
  {
  return -1;
  }
	mov ebx, [ebp+12]
	cmp ebx, 0
	jne else1
	  mov eax, -1
	  jmp finf	

else
  {
  x = n - 1;
	else1:
	  dec ebx
	  mov [ebp-4], ebx	

if (*(t + x) == b)
  {
  return x;
  }
	mov eax, [ebp+8]
	imul ebx, 4
	add ebx, eax
	mov ebx, [ebx]
	cmp ebx, [ebp+16]
	jne else2
	  mov eax, [ebp-4]
	  jmp finf	

else
  {
  return f(t, x, b);
  }
  }
}
	else2:
	  push [ebp+16]
	  push [ebp-4]
	  push [ebp+8]
	  call f
	  add esp, 12
	finf:
	pop ebx
	add esp, 4
	pop ebp
	ret	

Código	Columna 2	Columna3

int f (int *t, int n){
  int x;
	push ebp
	mov ebp, esp
	sub esp, 4
	push ebx	

if (n == 1)
  {
  return *t;
  }
	mov eax, [ebp+12]
	mov ebx, [ebp+8]
	cmp eax, 1
	jne else	

	mov eax, [ebx] jmp finf	

else
{
x = f(t+1, n-1);
	else:
	dec eax
	push eax
	add ebx, 4
	push ebx
	call f
	add esp, 8
	mov [ebp-4], eax	

if (t[0] <= x)
{
return x;
}
	mov ebx, [ebp+8]
	cmp [ebx], eax
	jle finf	

else
{
return *t;
}
}
}
	mov eax, [ebx]
	finf:
	pop ebx
	add esp, 4
	pop ebp
	ret	

Código	Columna 2	Columna3

int f(int *p, int n){
int m;
	push ebp
	mov ebp, esp
	sub esp, 4	

if(n == 0)	cmp [ebp+12], 0 jne else	
return 0;	mov eax, 0 jmp fin	

else {
m = f(p + 1, n - 1);
	else:
	mov eax,[ebp+12]
	dec eax
	push eax
	mov eax,[ebp+8]
	add eax, 4
	push eax
	call f
	add esp, 8
	mov [eax-4], eax	

return (m + *p);
}
}
	mov ebx,[ebp+8]
	add eax, [ebx]
	fin:
	add esp, 4
	pop ebp
	ret	

int f (int * p, int n){ int s1, s2, m;	**push ebp** **mov ebp, esp** **sub esp, 12** **push ebx**	

`if( n == 1 )`	`cmp [ebp+12], 1` `jne else`	

`return *p;`	`mov eax, [ebp+8]`
	`mov eax, [eax]`
	`jmp finF`	

`else {`
`m = n /2 ;`
	`else:`
	`mov eax, [ebp+12]`
	`sar eax, 1`
	`mov [ebp-4], eax`	

`s1 = f( p, m );`	`push [ebp-4]`
	`push [ebp+8]`
	`call f`
	`add esp, 8`
	`mov [ebp-12], eax`	

`s2 = f(p + m, n - m);`	`mov ebx, [ebp-4]`
	`mov eax, [ebp+12]`
	`sub eax, ebx`
	`push eax`
	`mov eax, [ebp+8]`
		`add eax, ebx`
		`push eax`
		`call f`
		`add esp, 8`
		`mov [ebp-8], eax`

`return s1 + s2;`
`}`
`}`
	`add eax, [ebp-12]`
	`finF:`
	`pop ebx`
	`add esp, 12`
	`pop ebp`
	`ret`	

Código	Columna 2	Columna3

`int h (int *t, int n){`
`int x;`
`int m;`
	`push ebp`
	`mov ebp, esp`
	`sub esp, 8`	

`if (n == 1)`
`{`
`return *t;`
`}`
	`cmp [ebp + 12], 1`
	`jnz else`
	`mov eax, [ebp + 8]`
	`mov eax, [eax]`
	`jmp fin`	

`else`
`{ m = n / 2;`
`t = t + m;`
	`else:`
	`mov eax, [ebp + 12]`
	`sar eax, 1`
	`mov [ebp - 4], eax`
	`imul eax, 4`
	`add [ebp + 8], eax`	

`x = *t;`	`mov eax, [ebp + 8]`
	`mov eax, [eax]`
	`mov [ebp - 8], eax`	

`return x + h(t, n-m);`
`}`
`}`
	`mov eax, [ebp + 12]`
	`sub eax, [ebp - 4]`
	`push eax`
	`push [ebp + 8]`
	`call h`
	`add esp, 8`
	`add eax, [ebp - 8]`
	
	`fin: add esp, 8`	

	pop ebp ret	

int
f (char * p, char * q){
  int i = 0 ;
	f proc
	push ebp
	mov ebp, esp
	sub esp, 4
		push ebx
		push ecx
		push esi
		mov [ebp-4], 0

while(p[i] == q[i] &&
p[i]){
	mov eax, 0
	mov ebx, [ebp+8]
	mov ecx, [ebp+12]
	mov esi, [ebp-4]
	while:
		mov al, [ebx+esi]
		cmp al, [ecx+esi]
		jne finWhile
		cmp al, 0
		je finWhile

i++; }	inc esi mov [ebp-4], esi jmp while finWhile:	

return !p[i];
}
	cmp eax, 0
	je cierto
	mov eax, 0
	jmp finF
	cierto:
	mov eax, 1
	finF:
		pop esi
		pop ecx
		pop ebx
		add esp, 4
		pop ebp
		ret
		f endp

int
g (char * p, char * q
){
int t;
	g proc
	push ebp
	mov ebp, esp
	sub esp, 4
		push ebx

do{ t = f(p, q);	do: push [ebp+12] push [ebp+8] call f	add esp, 2*4 mov [ebp-4], eax
if (t) return t;	cmp eax, 0 jne finG	

} while (*q++ != '\0'
);
	mov ebx, [ebp+12]
	inc [ebp+12]
	cmp [ebx], 0
	jne do	

return 0;
}
	mov eax, 0
	finG:
	pop ebx
	add esp, 4
		pop ebp
		ret
		g endp

int g (int * p){	g proc push ebp mov ebp, esp	

if (*p == 0)
return 1;
	mov eax, [ebp+8]
	cmp [eax], 0
	jne else
	mov eax, 1
	jmp fing	

else return *p; }	else: mov eax, [eax]	fing: pop ebp ret g endp

int
f (int n, int * * p){
int t, i;
	f proc
	push ebp
	mov ebp, esp
	sub esp, 8	

t = 1;	mov [ebp-8], 1	

for (i=0; i<n; i++){	mov [ebp-4], 0
	for:
	mov eax, [ebp+8]
	cmp [ebp-4], eax
	je finf	

t *= g(*p++);
}
	mov eax, [ebp+12]
	push [eax]
	add eax, 4
	mov [ebp+12], eax
	call g
		add esp, 4
		imul eax, [ebp-8]
		mov [ebp-8], eax
		inc [ebp-4]
		jmp for

return t;
}
	finf:
	mov eax, [ebp-8]
	add esp, 8
	pop ebp
	ret
	f endp	

int g (char * p){
int t = 0;
	g proc
	push ebp
	mov ebp, esp
	sub esp, 4
	mov [ebp-4], 0	

while (*p){	mov eax, [ebp+8],	

	while:
	cmp [eax], 0
	je finWhile	

if('a'<=*p &&
*p<='z')
t++;
	cmp [eax], 'a'
	jb finIf
	cmp [eax], 'z'
	ja finIf
	inc [ebp-4]	

p++; }	finIf: inc eax jmp while	

return t;
}
	finWhile:
	move ax, [ebp-4]
	add esp, 4
	pop ebp
	ret
	g endp	

int
f (char * * p, int n){
int i, t = 0;
	f proc
	push ebp
	mov ebp, esp
	sub esp, 8
	mov [ebp-4], 0	

for (i=0; i<n; i++){	mov [ebp-8], 0
	for:
	mov eax, [ebp-8]
	cmp eax,
	[ebp+12]
	jge finFor	

t += g(*p++); }	mov eax, [ebp+8] push [eax] add eax, 4 mov [ebp+8], eax call g	add esp, 4 add [ebp-4], eax inc [ebp-8] jmp for finFor:

return t;
}
	mov eax, [ebp-4]
	add esp, 8
	pop ebp
	ret
	f endp	

Bool
f(int * p, int n, int
x){
int i;
	push ebp
	mov ebp, esp
	sub esp, 4
	push esi	

if (n == 1)	cmp [ebp+12], 1 jne else1	

return (*p == x);	mov eax, 0
	sub esi, [ebp+8]
	mov esi, [esi]
	cmp esi, [ebp+16]
		jne finF
		mov eax, 1
		jmp finF

else{
i = n/2;
	else1:
	mov eax, [ebp+12]
	sar eax, 1
	mov [ebp-4], eax	

if (f(p, i, x))	push [ebp+16] push [ebp-4] push [ebp+8] call f add esp, 12	
return 1;	cmp eax, 1 je finF	

else
return f(p+i, n-i, x);
}
}
	push [ebp+16]
	mov eax, [ebp+12]
	sub eax, [ebp-4]
	push eax
	mov eax, [ebp+8]
	add eax, [ebp-4]
	push eax
	call f
	add esp, 12
		finF:
		pop esi
		jmp finF
		add esp, 4
		pop ebp
		ret

int f (char c, char * p){
int t = 0;
	push ebp
	mov ebp, esp
	sub esp, 4
	push esi
	push ecx
	mov word ptr[ebp-4], 0	

while(*p != '\0' && !t){	mov esi, [ebp+12]
	whileF:
	cmp [esi], 0
	je finWhileF
	cmp [ebp-4], 1
	je finWhileF	

t = (*p++ == c);
}
	mov cl, [ebp+8]
	cmp [esi], cl
	je cierto
	mov [ebp-4], 0
	jmp incrementar
	cierto:
	mov [ebp-4], 1
	incrementar:
	inc esi
	jmp whileF	

return t; }	finWhileF: mov eax, [ebp-4] pop ecx	

	pop esi add esp, 4 pop ebp ret	
char * g (char * p, char * q){	push ebp mov ebp, esp	

while (*p != '\0'){	mov esi, [ebp+8]
	whileG:
	cmp [esi], 0
	je finWhileG	

if (f(*p, q))
return p;
	push, [ebp+12]
	push word ptr [esi]
	call f
	add esp, 8
	cmp eax, 0
	je else
	mov eax, esi
	jmp finG	

else p++; }	else: inc esi jmp whileG	
return NULL; }	finWhileG: mov eax, 0 finG: pop ebp ret	

int f (int m, int n){
int t;
	push ebp
	mov ebp, esp
	sub esp, 4	

t = n - m;	mov eax, [ebp+12]
	sub eax, [ebp+8]
	mov [ebp-4], eax	

if (m == 0) return n;	cmp [ebp+8], 0 jne else1 mov eax , [ebp+12] jmp fin	

else
if (n > t)
return f(t, n);
	else1: mov eax, [ebp+12]
	cmp eax, [ebp-4]
	jle else2
	push [ebp+12]
		push [ebp-4]
		call f
		add esp, 8
		jmp fin

else return f(n, t);	else2: push [ebp-4] push [ebp+12]	call f add esp, 8
}	fin: add esp, 4 pop ebp ret	

void f (int n, char ** p){
  char d;
	push ebp
	mov ebp, esp ;ebp apunta a la pila (al ebp
	;salvado). Debajo está la
	;dirección de retorno y luego
	;vienen los parámetros.
	sub esp, 4 ;Espacio para la variable local
	;Es un char pero de todas
	;maneras ocupa 4 bytes en pila

if (n > 9){	cmp [ebp+8], 9 ;n está en ebp + 8 jle finIf ;si n > 9 entra en el if, ;luego si es <= sale.

f (n / 10, p);
}
	push [ebp+12] ;p está en ebp + 12
	mov eax, [ebp+8]
	mov edx, 0 ;edx:eax = n
	mov ecx, 10
	idiv ecx ;eax = n/10
	push eax ;Guarda parámetro n/10
	call f
	add esp, 8 ;sacar parámetros

**p = n % 10 + '0';	finIf:
	mov eax, [ebp+8]
	mov edx, 0 ;edx:eax = n
	mov ecx, 10
	idiv ecx ;edx = n%10
	add edx, '0' ; n%10 + '0'
	mov esi, [ebp+12] ;esi = p
	mov esi, [esi] ;esi = *p
	mov [esi], dl ;**p = n%10 + '0'
	;**p es un char

(*p)++;
}
	mov esi, [ebp+12] ;esi = p
	inc [esi] ;(*p)++
	;*p es un apuntador a char
	;luego se incrementa en 1
	add esp, 4 ;Retornar locales
	pop ebp
	ret

void g (int n){
  char c[100];
  char * q;
	push ebp
	mov ebp, esp ;ebp apunta a la pila (al ebp
	;salvado). Debajo está la
	;dirección de retorno y luego
	;vienen los parámetros.
	sub esp, 104 ;Espacio para locales:
	;Vector de 100 char (100 bytes)
	;y un apuntador (4 bytes)

q = c;	lea eax, [ebp-104] ;eax = dirección de c mov [ebp-4], eax ;q = dirección de c
f (n, &q);	lea eax, [ebp-4] ;eax = dirección de q push eax ;Parámetro &q push [ebp+8] ;Parámetro n call f add esp, 8 ;sacar parámetros

*q = 0;
}
	mov eax, [ebp-4] ;eax = q
	mov [eax], 0 ;*q = 0
	add esp, 104 ; Retornar locales
	pop ebp
	ret

Código	Columna 2	Columna 3

`int f(int *p, int *q){`
 	
`  int s;`		

`  if ( p >= q )`
`  {`
`  return 0;`
`  }`		

`  if ( *p < *q )`
`  {`
`  s = *p;`
`  }`		

`  else`
`  {`
`  s = *q;`
`  }`		

`  return s + f(p+1,q-1);`
 	
`}`		

Nota: para no atafagar el código se obvió poner “word ptr” y “byte ptr” en varios sitios.