CC3301 Programación de software de sistemas

Assembler Risc-V

3era clase

Temario: registros resguardados de Risc-V, compilación optimizada, registro de activación, frame pointer, secciones del archivo assembler, especificación de Risc-V, codificación de instrucciones, punto flotante

Registros resguardados: *s0-s11*

Convención usada por los compiladores:

- Al programar una función puede usar *a0-a7* y *t0-t6* sin preocuparse de restaurarlos al retornar
- Puede modificar los registros *s0-s11*, pero debe restaurarlos al retornar
- Si llama a otra función perderá cualquier valor que haya dejado en *a0-a7* y *t0-t6*
- Si llama a otra función, se garantiza que cualquier valor que haya dejado en s0-s11 sigue estando ahí
- Ejemplo: ver uso de registro *s0* en *g.c*

Compilación optimizada

- El optimizador mantiene en lo posible las variables en los registros del procesador
- Evalúa una sola vez expresiones comunes
- Evalúa funciones simples durante la compilación
- Ejemplo: ver assembler para *optim.c*

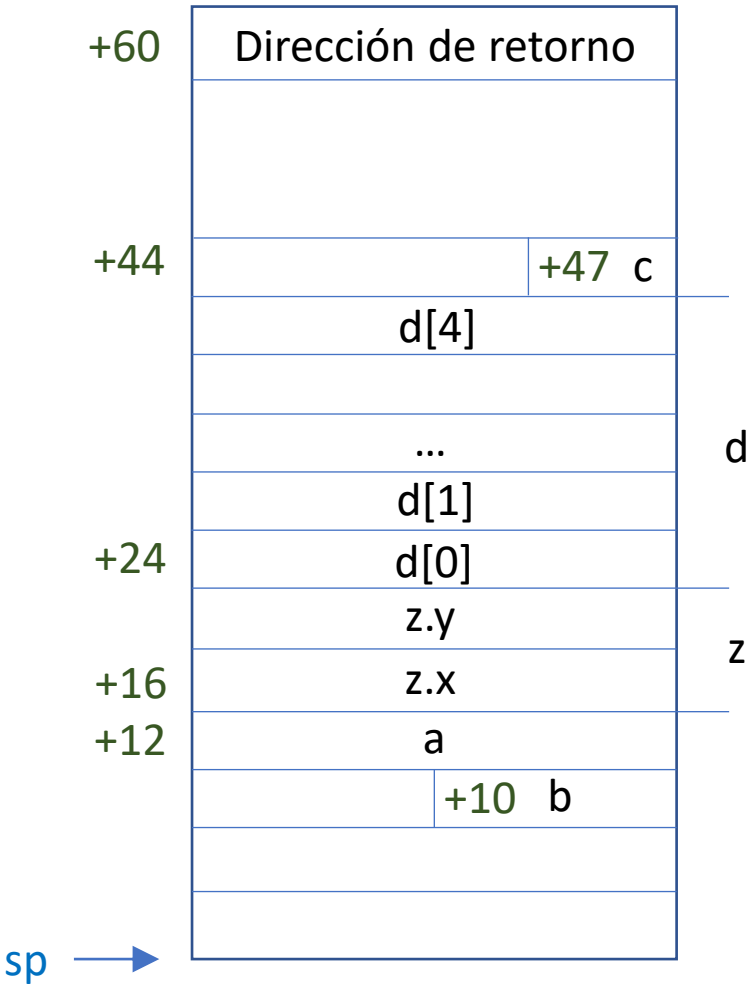
```
int f(int x, int y) {
  return x+y;
}

int g(int a) {
  return f(a+1, a+1);
}
```

```
f:
        add     a0,a0,a1
        ret
g:
        addi    a0,a0,1
        slli    a0,a0,1
        ret
```

El registro de activación

- Almacena las variables locales de una función
- Se apilan al ingresar a una función
- Se desapilan al retornar
- Ejemplo: ver assembler para *frame.c*




```c
typedef struct {
  int x, y;
} Complex;

int fun(int *pa, short *pb, char *pc,
        int *d, int *px, int *py,
        Complex *pz);

int frame(int a, short b) {
  char c;
  int d[5];
  Complex z;
  fun(&a, &b, &c, d, &z.x, &z.y, &z);
  return a+b+c+d[2]+z.x+z.y;
}
```

Arreglos y estructuras deben estar contiguos en memoria

```
frame:
        addi    sp,sp,-64
        sw      ra,60(sp)
        sw      a0,12(sp)
        sh      a1,10(sp)
        addi    a6,sp,16
        addi    a5,sp,20
        mv      a4,a6
        addi    a3,sp,24
        addi    a2,sp,47
        addi    a1,sp,10
        addi    a0,sp,12
        call    fun
        lh      a5,10(sp)
        lw      a4,12(sp)
        add     a5,a5,a4
        lbu     a4,47(sp)
        add     a5,a5,a4
        lw      a4,32(sp)
        add     a5,a5,a4
        lw      a4,16(sp)
        add     a5,a5,a4
        lw      a0,20(sp)
        add     a0,a5,a0
        lw      ra,60(sp)
        addi    sp,sp,64
        jr      ra
```

Registro s0: frame pointer (fp)

- Se necesita cuando se declaran arreglos locales de tamaño desconocido en tiempo de compilación
- Apunta hacia el final del registro de activación
- Se usa para acceder a las variables locales en vez de sp. Ejemplo: ver assembler para *frame-pointer.c*

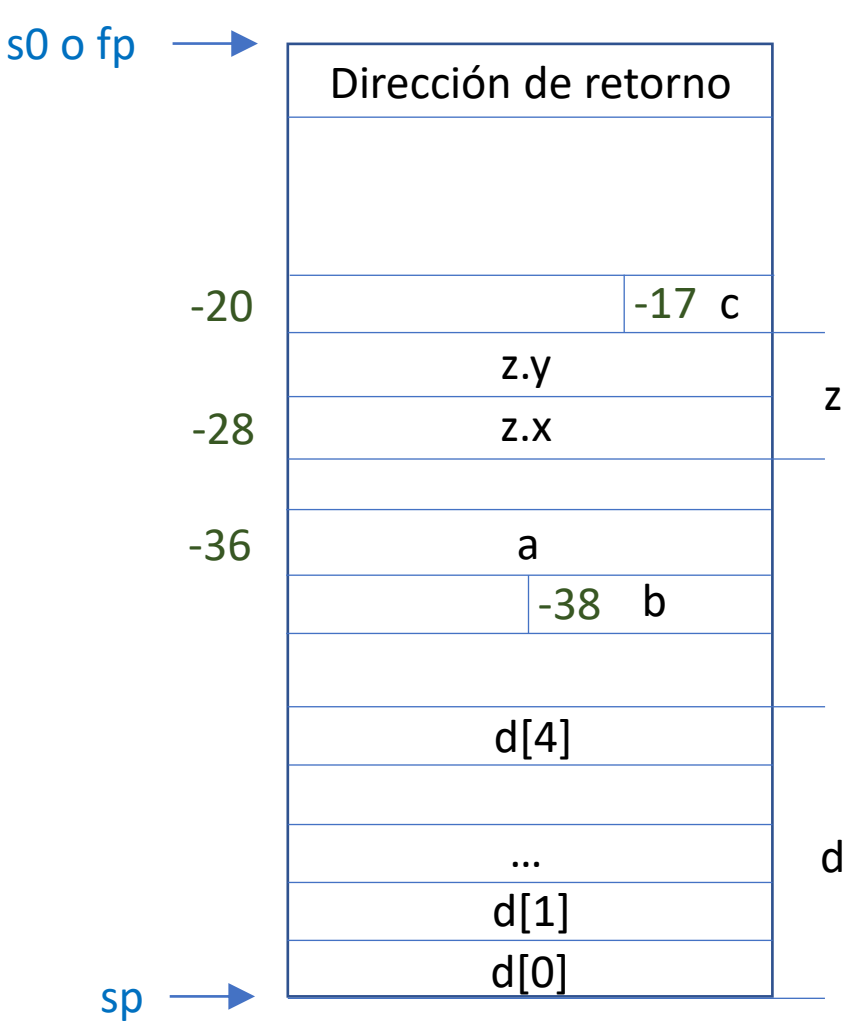
```
typedef struct {
  int x, y;
} Complex;

int fun(int *pa, short *pb, char *pc,
        int *d, int *px, int *py,
        Complex *pz);

int frame(int a, short b) {
  char c;
  int d[a];
  Complex z;
  fun(&a, &b, &c, d, &z.x, &z.y, &z);
  return a+b+c+d[b]+z.x+z.y;
}
```

```
frame:
        addi    sp,sp,-48
        sw      ra,44(sp)
        sw      s0,40(sp)
        sw      s1,36(sp)
        addi    s0,sp,48
        sw      a0,-36(s0)
        sh      a1,-38(s0)
        slli    a0,a0,2
        addi    a0,a0,15
        andi    a0,a0,-16
        sub     sp,sp,a0
        mv      s1,sp
        addi    a6,s0,-28
        addi    a5,s0,-24
        mv      a4,a6
        mv      a3,s1
        addi    a2,s0,-17
        addi    a1,s0,-38
        addi    a0,s0,-36
        call    fun
        lh      a4,-38(s0)
        lw      a5,-36(s0)
        add     a5,a4,a5
        lbu     a3,-17(s0)
        add     a5,a5,a3
        slli    a4,a4,2
        add     s1,s1,a4
        lw      a4,0(s1)
        add     a5,a5,a4
        lw      a4,-28(s0)
        add     a5,a5,a4
        lw      a0,-24(s0)
        add     a0,a5,a0
        addi    sp,s0,-48
        lw      ra,44(sp)
        lw      s0,40(sp)
        lw      s1,36(sp)
        addi    sp,sp,48
        jr      ra
```

Registro s0: frame pointer (fp)




```c
typedef struct {
  int x, y;
} Complex;

int fun(int *pa, short *pb, char *pc,
        int *d, int *px, int *py,
        Complex *pz);

int frame(int a, short b) {
  char c;
  int d[a];
  Complex z;
  fun(&a, &b, &c, d, &z.x, &z.y, &z);
  return a+b+c+d[b]+z.x+z.y;
}
```

Los arreglos de tamaño desconocido en compilación quedan abajo en el registro de activación.

El resto de las variables locales están a un distancia de s0 conocida en tiempo de compilación.

Secciones de un archivo assembler

- Código: .text
- Datos inicializados: .data (o .section .sdata)
- Datos no inicializados: .bss (o .section .sbss)
- Datos de solo lectura: .section .rodata
- Ejemplo: ver assembler para *glob.c*
- Otras directivas:
  - Declarar etiquetas globales: .globl
  - Alinear a un tamaño potencia de 2^n: .align n
  - Declarar espacio para enteros (int): .word
  - Declarar espacio para enteros (short): .half
  - Declarar espacio para enteros (char): .byte
  - Declarar espacio para strings constantes: .string
  - Declarar n bytes para espacio en cero: .zero n
  - Declarar etiqueta de una función f: .type f, @function
  - % de una variable global g: .type g, @object

```c
int a;
int b= 1;
char c= 1+2;
short h= -1;

int *p= &a;

int d[5];
int e[]= {1,2,3};
char *s= "hola";

int main() {
  return 0;
}
```

```
        .text
        .align  2
        .globl  main
        .type   main, @function
main:
        li      a0,0
        ret
        .size   main, .-main
        .globl  s
        .section        .rodata.str1.4,"aMS",@progbits,1
        .align  2
.LC0:
        .string "hola"
        .globl  e, d, p, h, c, b, a
        .data
        .align  2
        .type   e, @object
        .size   e, 12
e:
        .word   1
        .word   2
        .word   3
        .bss
        .align  2
        .type   d, @object
        .size   d, 20
d:
        .zero   20
        .section        .sbss,"aw",@nobits
        .align  2
        .type   a, @object
        .size   a, 4
a:
        .zero   4
        .section        .sdata,"aw"
        .align  2
        .type   s, @object
        .size   s, 4
s:
        .word   .LC0
        .type   p, @object
        .size   p, 4
p:
        .word   a
        .type   h, @object
        .size   h, 2
h:
        .half   -1
        .type   c, @object
        .size   c, 1
```

La especificación de Risc-V

- Sitio web: https://riscv.org/
- Especificaciones: https://riscv.org/technical/specifications/
- Instrucciones no privilegiadas: https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMAFDQC/riscv-spec-20191213.pdf
- Tutoriales:
  - googlear RiscV assembler tutorial
  - recomendar en el foro por favor

Codificación de instrucciones

RV32I Base Instruction Set

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

Punto flotante

- No es obligatorio implementarlo
- Extensión rv32f: operaciones con *float*
- Extensión rv32d: operaciones con *double*

RV32F Standard Extension

imm[11:0]		rs1	010	rd	0000111	FLW
imm[11:5]	rs2	rs1	010	imm[4:0]	0100111	FSW
rs3	00	rs2	rs1	rm	rd	1000011	FMADD.S
rs3	00	rs2	rs1	rm	rd	1000111	FMSUB.S
rs3	00	rs2	rs1	rm	rd	1001011	FNMSUB.S
rs3	00	rs2	rs1	rm	rd	1001111	FNMADD.S
0000000	rs2	rs1	rm	rd	1010011	FADD.S
0000100	rs2	rs1	rm	rd	1010011	FSUB.S
0001000	rs2	rs1	rm	rd	1010011	FMUL.S
0001100	rs2	rs1	rm	rd	1010011	FDIV.S
0101100	00000	rs1	rm	rd	1010011	FSQRT.S
0010000	rs2	rs1	000	rd	1010011	FSGNJ.S
0010000	rs2	rs1	001	rd	1010011	FSGNJN.S
0010000	rs2	rs1	010	rd	1010011	FSGNJX.S
0010100	rs2	rs1	000	rd	1010011	FMIN.S
0010100	rs2	rs1	001	rd	1010011	FMAX.S
1100000	00000	rs1	rm	rd	1010011	FCVT.W.S
1100000	00001	rs1	rm	rd	1010011	FCVT.WU.S
1110000	00000	rs1	000	rd	1010011	FMV.X.W
1010000	rs2	rs1	010	rd	1010011	FEQ.S
1010000	rs2	rs1	001	rd	1010011	FLT.S
1010000	rs2	rs1	000	rd	1010011	FLE.S
1110000	00000	rs1	001	rd	1010011	FCLASS.S
1101000	00000	rs1	rm	rd	1010011	FCVT.S.W
1101000	00001	rs1	rm	rd	1010011	FCVT.S.WU
1111000	00000	rs1	000	rd	1010011	FMV.W.X

Próxima clase

- La última arquitectura CISC: Intel/AMD x86
- Assembler Intel/Amd x86