Estiven Escobar Pulgarín

Cod: 1088339047

Taller parcial

**1. Describa la taxonomía de flynn**

SISD: single instruction-single data

SIMD: single instruction-multiple data

MIMD: multiple instruction-multiple data

MISD: multiple instruction-single data

**2. Diga los 4 principios de diseño**

*1.* la simplicidad favorece la regularidad

*2.* entre mas pequeño mas rápido

*3.* hacer el caso común mas rápido

*4.* buenos diseños implican grandes compromisos

**3. Explique los tres formatos de instrucciones principales existentes en la arquitectura SPARCV8**

***Formato (1)***

La instrucción call utiliza este formato

Op disp. 30

|  |  |
| --- | --- |
| 01 |  |

31 29 0

***Formato (2)***

Se utiliza para los sethi,branch,flolat point

Sethi

Op rd op2 imm22

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

31 29 24 21 0

Branch

Op a cond op2 disp22

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |

31 29 28 24 21 0

***Formato (3)***

Para instrucciones de memoria y instrucciones aritméticas lógicas

Con imm

Op rd op3 rs1 i=1 imm3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |

31 29 24 18 13 12 0

Sin imm

Op rd op3 rs1 i=1 unusead/zero rs2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |

31 29 24 18 13 12 4 0

shift

Op rd op3 rs1 i=1 unusead/zero shift

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |

31 29 24 18 13 12 4 0

**4. Explique cómo inicializar un valor grande que ocupe mas de 13 bits, en la arquitectura SPARCV8**

Se utiliza sethi y utilizamos el formato 2

Ejemplo:

D= -19600

00000000000000000100110010010000 => 19600

Utilizamos complemento a 2

00000000000000000100110010010000   
11111111111111111011001101101111   
 + 1  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
11111111111111111011001101110000

Cogemos los 22 bits mas significativos y dejamos aparte los otros 10bits

1111111111111111101100 ---------------------------1101110000

Volvemos a hacer complemento a 2 al numero de 22 bits y el numero de 10 bits lo pasamos a decimal

1111111111111111101100   
0000000000000000010011   
+ 1  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

0000000000000000010100 => 20 --------------- 1101110000=>880

Y inicializamos así:

D:%L4

Al sethi le colocamos el numero de 22 bits que hicimos complemento a 2 con un menos por que es negativo y en un or guardamos el numero de 10 bits.

SETHI -20,%L4  
Or %L4,880,%L4

**5. Como puedo reescribir la instrucción (Or y Subcc) cuando inicializo y comparó dos registros.**

-Ejemplo OR:

Or %L1,5,%L1   
el Or utiliza el registro %g0 entonces al sumar el 5 con el registro g0 que siempre es 0 queda el 5 y eso se guarda en el registro destino entonces con la instrucción MOV simplemente movemos el inmediato al registro es una instrucción sintética

Mov 5,%L1

-Ejemplo Subcc:

Subcc %L1,5,%L1   
el Subcc hace un resta y modifica el icc(integer conditional code)entonces si uno es mayor al otro modifica el campo negativo o carry dependiendo y si son iguales modifica el campo zero entonces podemos utilizar una instrucción sintética que se llama CMP que compara que si analizamos lo que hace el subcc es como una comparación

CMP 5,%L1

**6. Implemente el lenguaje de alto nivel y lenguaje de maquina el siguiente programa**

|  |
| --- |
|  |
| A. |
| int main(){ |
| int a = 8; |
| int b = -16800 |
| int c = 33; |
| if((a+b)<=b\*32){ |
| c=a+(b\*2); |
| } |
| else{ |
| return b;  } |
|  |
| return a+c;  } |

*DESARROLLO*

B= -16800

00000000000000000100000110100000 => 16800

Utilizamos complemento a 2

00000000000000000100000110100000  
11111111111111111011111001011111   
 + 1  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
11111111111111111011111001100000

Cogemos los 22 bits mas significativos y dejamos aparte los otros 10bits

1111111111111111101111 ---------------------------1001100000

Volvemos a hacer complemento a 2 al numero de 22 bits y el numero de 10 bits lo pasamos a decimal

1111111111111111101111  
0000000000000000010000   
+ 1  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

0000000000000000010001 => 17 --------------- 1001100000=>608

a:%L0   
b:%L1   
c:%L2

|  |  |
| --- | --- |
| 0X0000 | MOV 8,%L0 |
| 0X0004 | SETHI 17,%L1 |
| 0X0008 | OR %L1,608,%L1 |
| 0X000C | MOV 33, %L2 |
| 0X0010 | ADD %L0,%L1,%L3 |
| 0X0014 | SLL %L1,32,%L4 |
| 0X0018 | CMP %L3,%L4,%L5 |
| 0X001C | BG a,ELSE |
| 0X0020 | SLL %L1,2,%L6 |
| 0X0024 | ADD %L0,%L6,%L2 |
| 0X0028 | Ba a,EXIT |
| ELSE |  |
| 0X002C | MOV %L1,%Oo |
| EXIT |  |
| 0X0030 | ADD %L0,%L2,%O1 |

FORMATO 2: F 2 FORMATO 3: F 3

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 10 | 100000 | 000010 | 00000 | 1 | 0000000001000 | |
| 00 | 10001 | 100 | 1111111111111111101111 | | | |
| 10 | 10001 | 000010 | 10001 | 1 | 0001001100000 | |
| 10 | 10010 | 000010 | 00000 | 1 | 0000000100001 | |
| 10 | 10011 | 000000 | 10000 | 0 | 00000000 | 10001 |
| 10 | 10100 | 100101 | 10001 | 1 | 00000000 | 11111 |
| 10 | 10101 | 010100 | 10011 | 0 | 00000000 | 10100 |
| 00 | 1 | 1010 | 010 | 0000000000000000000100 | | |
| 10 | 10110 | 100101 | 10001 | 1 | 00000000 | 00010 |
| 10 | 10010 | 000000 | 10000 | 0 | 00000000 | 10110 |
| 00 | 1 | 1000 | 010 | 0000000000000000000010 | | |
| 10 | 01000 | 000010 | 00000 | 1 | 00000000 | 10000 |
| 10 | 01001 | 000000 | 10000 | 0 | 00000000 | 10001 |

F 3-0X0000

F 2-0X0004

F 3-0X0008

F 3-0X000C

F 3-0X0010

F 3-0X0014

F 3-0X0018

F 2-0X001C   
F 3-0X0020  
F 3-0X0024  
F 2-0X0028  
F 3-0X002C   
F3-0X0030

|  |
| --- |
| B. |
| int main(){ |
| int a = 8; |
| int b = -10; |
|  |
| if(a!=b){ |
| Return c/8; |
| } |
| else{ |
| return b;  }  } |

DESARROLLO

a:%L0

b:%L1

c:%L2

|  |  |
| --- | --- |
| 0X0000 | MOV 8,%L0 |
| 0X0004 | MOV -10,%L1 |
| 0X0008 | CMP %L0,%L1 |
| 0X000C | BE a,ELSE |
| 0X0010 | SRL %L2,8,%L3 |
| 0X0014 | MOV %L3,%Oo |
| 0X0018 | BA a,EXIT |
| ELSE |  |
| 0X001C | MOV %L1,%O1 |
| EXIT |  |
| 0X0020 | NOP |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 10 | 10000 | 000010 | 00000 | 1 | 0000000001000 | |
| 10 | 10001 | 000010 | 00000 | 1 | 1111111110110 | |
| 10 | 10000 | 010100 | 10000 | 0 | 00000000 | 10001 |
| 00 | 1 | 0001 | 010 | 0000000000000000000100 | | |
| 10 | 10011 | 100100 | 10010 | 1 | 00000000 | 01000 | |
| 10 | 01000 | 000010 | 00000 | 0 | 00000000 | 10011 |
| 00 | 1 | 1000 | 010 | 0000000000000000000010 | | |
| 10 | 01001 | 000010 | 00000 | 0 | 00000000 | 10001 |
| 00 | 000000 | 100 | 00000000000000000000 | | | |

F 3-0X0000

F 3-0X0004

F 3-0X0008

F 2-0X000C

F 3-0X0010

F 3-0X0014

F 2-0X0018

F 3-0X001C

F 2-0X0020

|  |
| --- |
| C. |
| int main(){ |
| int a=-21800 ;  Return a; |
| } |

DESARROLLO

a= -21800

00000000000000000101010100101000 => 21800

Utilizamos complemento a 2

00000000000000000101010100101000  
11111111111111111010101011010111   
 + 1  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
11111111111111101010101101011000

Cogemos los 22 bits mas significativos y dejamos aparte los otros 10bits

1111111111111110101010 ---------------------------1101011000

Volvemos a hacer complemento a 2 al numero de 22 bits y el numero de 10 bits lo pasamos a decimal

1111111111111110101010  
0000000000000001010101   
+ 1  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

0000000000000001010110 => 86 --------------- 1101011000=>856

a:%L1

|  |  |
| --- | --- |
| 0X0000 | SETHI 86,%L1 |
| 0X0004 | OR %L1,856,%L1 |
| 0X0008 | MOV %L1,%O1 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 00 | 10001 | 100 | 1111111111111110101001 | | |
| 10 | 10001 | 000010 | 00000 | 1 | 0001101011000 |
| 10 | 01001 | 000010 | 00000 | 0 | 00000000 | 10001 |

F 2-0X0000  
F 3-0X0004  
F 3-0X0008

|  |
| --- |
| D. |
| int main(){ |
| int a=6,b=4;  for(int i=0,i<=3,i++){  c=(a+b)/8  }  Return c; |
| } |

DESARROLLO

a:%L0   
 b:%L1   
 c:%L2   
 i:%L3

|  |  |
| --- | --- |
| 0X0000 | MOV 6,%L0 |
| 0X0004 | MOV 4,%L1 |
| 0X0008 | MOV 0,%L3 |
| 0X000C | CMP %L3,3 |
| 0X0010 | BG a, EXIT |
| 0X0014 | ADD %L0,%L1,%L4 |
| 0X0018 | SRL %L4,8,%L2 |
| 0X001C | ADD %L3,1,%L3 |
| EXIT |  |
| 0X0020 | MOV %L2,%Oo |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 10 | 10000 | 000010 | 00000 | 1 | 0000000000110 | |
| 10 | 10001 | 000010 | 00000 | 1 | 0000000000100 | |
| 10 | 10011 | 000010 | 00000 | 1 | 0000000000000 | |
| 10 | 10011 | 010100 | 10011 | 1 | 0000000000011 | |
| 00 | 1 | 1010 | 010 | 000000000000000000100 | | |
| 10 | 10100 | 000000 | 10000 | 0 | 00000000 | 10001 |
| 10 | 10010 | 100100 | 10100 | 1 | 00000000 | 01000 |
| 10 | 10011 | 000000 | 10011 | 1 | 0000000000001 | |
| 10 | 01000 | 000010 | 00000 | 0 | 00000000 | 10010 |

**8. Convierta el siguiente código a lenguaje de maquina SPARCV8**

Int ejemplo(int x, int y, int z){

Int a;   
 a=x-y + z\*8 ;  
 return a+2;

}

Int main(){   
 int x=4, y=2, z=-128;   
 int c=0;   
 int c=ejemplo(x,y,z);   
 return c+45;

}

DESARROLLO

X=%I0 C=%L0   
Y=%I1 A=%L1  
Z= %I2

|  |  |
| --- | --- |
| EJEMPLO |  |
| 0X0000 | SUB %IO, %I1, %L2 |
| 0X0004 | SRL %I2,8, %L3 |
| 0X0008 | ADD %L2, %L3, %L1 |
| 0X000C | JMPL %O7,8, %G0 |
| 0X0010 | ADD %L1,2, %O1 |
| MAIN |  |
| 0X0014 | MOV 4, %I0 |
| 0X0018 | MOV 2, %I1 |
| 0X001C | MOV -128, %I2 |
| 0X0020 | CALL EJEMPLO |
| 0X0024 | MOV O, %L0 |
| 0X0028 | ADD %L0,45,%O0 |

**9. Implemente una función mul en lenguaje de alto nivel, lenguaje de bajo nivel SPARCV8 y lenguaje de máquina SPARCV8 que realice la multiplicación de dos enteros sin signo usando solo sumas.**

**A:%I0 C:%L0**

**B:%I1 I:%L1**

int multiplicación(int a, int b){

int c=0;

for(int i=0, i<b,i++){  
 c= a+c;

}

Return c;

}

Int main(){

Int a=2;

Int b=3;

Multiplicación (a,b)

}

|  |  |
| --- | --- |
| multiplicacion |  |
| 0X0000 | MOV 0,%L0 |
| 0X0004 | BA FOR |
| 0X0008 | MOV 0,%L1 |
| FOR |  |
| 0X000C | CMP %L1,%I1 |
| 0X0010 | BGE A, EXIT |
| 0X0014 | ADD %L0,%I0,%L0 |
| 0X0018 | BA FOR |
| 0X001C | ADD %L1,1%L1 |
| EXIT |  |
| 0X0020 | JMPL %O7,8,%G0 |
| 0X0024 | MOV %L0,%O0 |
| MAIN |  |
| 0X0028 | MOV 2,%I0 |
| 0X002C | CALL muliplicacion |
| 0X0030 | MOV 3, %I1 |

**10.** **Implemente la función pot en lenguaje de alto nivel, lenguaje de bajo nivel SPARCV8 y lenguaje de máquina SPARCV8 que realice la potencia de dos números enteros sin signo realizando llamados a la función desarrollada en el punto 9.**

**A:%I0 C:%L0 D:%I2**

**B:%I1 I:%L1 J:5L2**

int multiplicación(int a, int b){

int c=0;

for(int i=0, i<b,i++){  
 c= a+c;

}

Return c;

}

Int main(){

Int a=2;

Int b=3;   
int d=1;

For(int j=0,j<b,j++){

d=Multiplicación (a,d)

}

Return d;

}

|  |  |
| --- | --- |
| multiplicacion |  |
| 0X0000 | MOV 0,%L0 |
| 0X0004 | BA FOR1 |
| 0X0008 | MOV 0,%L1 |
| FOR1 |  |
| 0X000C | CMP %L1,%I2 |
| 0X0010 | BGE A, EXIT |
| 0X0014 | ADD %L0,%I0,%L0 |
| 0X0018 | BA FOR1 |
| 0X001C | ADD %L1,1%L1 |
| EXIT |  |
| 0X0020 | JMPL %O7,8,%G0 |
| 0X0024 | MOV %L0,%O0 |
| MAIN |  |
| 0X0028 | MOV 2,%I0 |
| 0X002C | MOV 3,%I1 |
| 0X0030 | MOV 0, %I2 |
| 0X0034 | BA FOR |
| 0X0038 | MOV 1,%L2 |
| FOR |  |
| 0X003C | CMP %L2,%I1 |
| 0X0040 | BGE A, FALSE |
| 0X0044 | CALL multiplicacion |
| 0X0048 | BA FOR |
| 0X004C | ADD %L2,1,%L2 |
| FALSE |  |
| 0X0050 | MOV %I2,%O1 |

11. **Implemente una función pol en lenguaje de alto nivel, lenguaje de bajo nivel SPARCV8 y lenguaje de máquina SPARCV8 que calcule un polinomio de grado n, en donde los coeficientes del polinomio se encuentran almacenados en un vector k de n+1 posiciones, de la siguientes forma:**

**pol(x,k,n)=i=0nkixi**

**En donde el ki se encuentra en la posición k[i] del vector k.**

Int mul(int x, int y){ **x:%I5 y:%I6 i:%L6 r1:%L7**

int r1=0;

for(int i=0, i<y,i++){  
 r1= r1+x;

}

Return r1;

}

Int pot(int x,int y){ **x:%I3 y:%I4 i:%L4 r2:%L7**

Int i=0;  
Int r2=1;

For(i,i<y,j++){

r2=mul(r2,x)

}

Return r2;

}

Int pol(int x,int k[],int n){

Int i=0,r3=0,j=n,y=0,p;

For(i,i<=n,i++){

r3=pot(x,y); **x:%I0 k:%I1 n:%I2 i:%L0 r3:%L1** p=mul(k[i],r);y= y + p;  **y:%L2 j:%L3 r1:%L4**  
 j--;   
 }

Return y;

}

|  |  |
| --- | --- |
| **Mul** |  |
| **0X0000** | **MOV 0,%L6** |
| **0X0004** | **BA FOR1** |
| **0X0008** | **MOV 0,%L7** |
| **FOR 1** |  |
| **0X000C** | **CMP %L6,%I5** |
| **0X0010** | **BGE A, EXIT** |
| **0X0014** | **ADD %L7,%I4,%L7** |
| **0X0018** | **BA FOR1** |
| **0X001C** | **ADD %L6,1,%L6** |
| **EXIT** |  |
| **0X0020** | **JMPL %O7,8,%G0** |
| **0X0024** | **MOV %L7,%O0** |
| **POT** |  |
| **0X0028** | **MOV 0,%L4** |
| **0X002C** | **BA FOR 2** |
| **0X0030** | **MOV 1,%L5** |
| **FOR 2** |  |
| **0X0034** | **CMP %L4,%I4** |
| **0X0038** | **BGE A, EXIT2** |
| **0X003C** | **MOV %O7,%G1** |
| **0X0040** | **CALL MUL** |
| **0X0044** | **ADD %L4,1,%L4** |
| **EXIT2** |  |
| **0X0048** | **JMPL %G1,8,%G0** |
| **0X004C** | **MOV %L5,%O1** |
| **POL** |  |
| **0X0050** | **MOV 0,%L0** |
| **0X0054** | **MOV 0,%L1** |
| **0X0058** | **MOV %I2,%L3** |
| **0X005C** | **BA FOR3** |
| **0X0060** | **MOV 0,%L2** |
| **FOR 3** |  |
| **0X0064** | **CMP %L0,%I2** |
| **0X0068** | **BG A, EXIT3** |
| **0X006C** | **CALL POT** |
| **0X0070** | **NOP** |
| **0X0074** | **CALL MUL** |
| **0X0078** | **NOP** |
| **0X007C** | **ADD %L2,%L4,%L2** |
| **0X0080** | **BA FOR3** |
| **0X0084** | **SUB %L3,1,%L3** |
| **EXIT3** |  |
| **0X008C** | **MOV %L2,%O2** |